5th Annual IEEE Conference on Automation Science and Engineering
Bangalore, India, August 22-25, 2009

A DECOMSYS Based Tool-Chain for Analyzing FlexRay based Automotive Control Applications

Dip Goswami, Pradeep Seshadri, Unmesh D. Bordoloi, and Samarjit Chakraborty

Abstract— **It is now widely believed that FlexRay will emerge as predominant protocol for in-vehicle automotive communication systems. As a result, there has been a lot of recent interest in timing and predictability analysis techniques that are specifically targeted towards FlexRay based electronic control units (ECU) networks. In this work, the DECOMSYS tool [1] is used for mapping the tasks on different ECUs, configuring the scheduling policy used at each ECU, and specifying the FlexRay parameters (e.g. slot sizes and message priorities). The overall system requirements are simulated by flashing such FlexRay based ECU network specifications into the DECOMSYS node. The design tools such as DECOMSYS mostly rely on simulations. As a result, they are time-consuming to use and unable to provide formal performance guarantees, which are important in the automotive domain. In our previous works, we have proposed a Real-Time Calculus (RTC) based compositional performance analysis framework for an ECU network that communicates via FlexRay bus [2]. The RTC based framework allows computation of performance analysis parameters such as the maximum end-to-end delay experienced by any message, the amount of buffer required at each communication controller and the utilization of the different ECUs and the bus. In this paper, we have plugged in the RTC based framework into the DECOMSYS tool. The combined setup consisting of DECMSYS and RTC based framework can be utilized to obtain hard performance guarantees, which can then be cross-validated using simulation.**

I. INRODUCTION

In modern automotive systems, various functions are implemented in ECUs which generally consist of one or more micro-controllers and a set of sensors and actuators. Different automotive functionalities are implemented in distributed fashion and various components of a task are implemented on different ECUs with messages and signals being exchanged between them. Such communication between multiplexed ECUs essentially leads to development of ECU network over one or more shared buses. Such ECU networks naturally gave rise to the need for different communication protocols specifically targeting automotive communication systems. The communication protocols are broadly classified in three groups: time-triggered, event-triggered and hybrid (i.e., combination of both time-triggered and event-triggered). The time-triggered protocols (time-division multiple access or TDMA [3]) have predictable temporal behavior at the cost of efficiency in utilization of communication bandwidth and flexibility. On the other hand, the event-triggered protocols (Controller Area Network or CAN [4]) have efficient bandwidth utilization and flexibility at the expense of predictability in temporal behavior. Hybrid protocols (TTCAN [5], FTT-CAN [6] and FlexRay [7]) are supposed to exploit the advantages of both time-triggered and event-triggered protocols. Consequently, there has been considerable emphasis on timing and predictability analysis of the hybrid protocols, especially FlexRay protocol.

Communication cycles are periodic in the FlexRay protocol. Such communication cycles are combination of a time-triggered or static (ST) segment and an event-triggered or dynamic (DYN) segment. The ST segment uses a TDMA scheme. Thus, the ST segment has predictable timing properties which also come with low bandwidth utilization and inflexibility. On the contrary, the DYN segment uses *Flexible TDMA* which is efficient in bandwidth utilization with the usual shortcomings of an event-triggered paradigm. Complexities in temporal performance analysis of FlexRay protocol, arose due to the presence of both ST and DYN segments, is the motivation of most of the research being performed on FlexRay based automotive systems.

The DECOMSYS tool-chain is an integrated prototyping platform for the emerging FlexRay communication standard for automotive electronics. This technology enables evaluation and analysis of the FlexRay standard, and facilitates application testing in the laboratory as well as in prototype cars. The DECOMSYS tool-chain is used for prototyping the FlexRay based ECU networks. These stand-alone tools mainly realize the actual FlexRay based network behaviors. However, these tools are unable to provide any formal performance guarantees such as maximum end-to-end delay suffered by any message being transmitted over the FlexRay bus or the maximum buffer requirement for the message. In our previous reported works [2], we have proposed a *compositional* modeling framework which can answer a wider variety of performance-related questions and is also helpful for synthesizing a FlexRay schedule (i.e. determine the slot sizes and message priorities) when maximum end-to-end delays or buffer requirements are provided as design constraints. The framework is based on basic Real-Time Calculus [8] and is implemented using a combination of Java and MATLAB. In this work, we have plugged in the RTC based framework into the DECOMSYS based setup. The combined framework can provide hard performance guarantee as well as can simulate the actual behavior of FlexRay-ECU networks. Thus, the combined setup allows one to validate the hard performance parameters obtained from RTC based framework in a real FlexRay network.

Dip Goswami and Pradeep Seshadri are with Department of Computer Science, National University of Singapore, Singapore {goswami, pradeeps}@comp.nus.edu.sg
Unmesh D. Bordoloi is with Verimag Labs, France unmesh.d.bordoloi@verimag.fr
Samarjit Chakraborty is with the Institute for Real-Time Computer Systems, TU Munich, Germany samarjit@tum.de

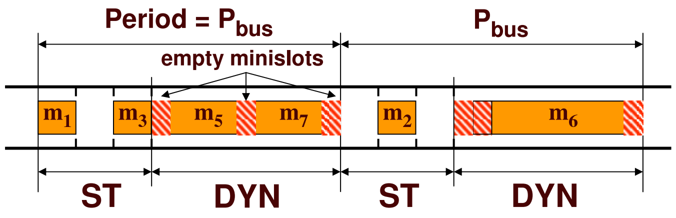


Fig. 1. Two typical FlexRay communication cycles.

The rest of this paper is organized as follows. The Section II briefly discusses the FlexRay protocol. In Section III, we have provided an overview of basic RTC. The Section IV has provided an overview of our framework and how that is adopted from RTC. The Section V has described the DECOMSYS tool-chain. How to plugin RTC based framework into the DECOMSYS tool-chain is explained in Section VI while some preliminary results are presented in Section VII. The conclusions are drawn in Section VIII.

II. FLEXRAY: AN OVERVIEW

As mentioned in the previous section, each FlexRay communication cycle is partitioned into ST and DYN segments. The lengths of these segments need not be equal, but are fixed over different cycles. The ST segment is further partitioned into a fixed number of equal-length *slots*. Each slot is allocated to a specific node and tasks within that node is allowed to send a message only during its allocated slot. If a node has no messages to send, then its slot goes empty (i.e. other nodes are not allowed to use it).

The DYN segment is also partitioned into equal-length slots, but slot size is much smaller and is referred to as a *minislot*. The tasks which send messages through the DYN segment are assigned fixed priorities. At the beginning of each DYN segment, the highest priority task is allowed to send a message. The length of such messages can be arbitrarily long (i.e. can occupy an arbitrary number of minislots), but has to fit within one DYN segment. However, if the task has no message to send, then only one minislot goes empty. In either case, the bus is then given to the next highest-priority task and the same process is repeated till the end of the DYN segment. Further, when its turn comes, a task is allowed to send only one message per communication cycle. For further details of this protocol, we refer the reader to the excellent description in [9] or to the full specification [7].

As an example, consider eight tasks $T_1, \ldots, T_8$ mapped onto different ECUs, which send messages on the FlexRay bus. Any message sent by a task T_i is indicated as m_i. Tasks T_1, T_2 and T_3 send messages over the ST segment and T_4 to T_8 over the DYN segment. For the DYN segment, the priorities of the tasks decrease from T_4 to T_8. Figure 1 shows two consecutive FlexRay communication cycles resulting from this mapping. In the first cycle, task T_2 has no message to send (hence the corresponding slot in the ST segment is empty) and in the second cycle T_1 and T_3 have nothing to send.

Similarly, in the first cycle, tasks T_5, T_6 and T_7 have messages to send, but not T_4 and T_8. Hence, there is one empty minislot corresponding to T_4 in the DYN segment, followed by the message m_5. The size of m_6 is bigger than the remaining length of the DYN segment, hence it is not sent; instead there is one empty minislot in its place. This is followed by m_7 and another empty minislot resulting out of no message from T_8. In the second cycle, T_4 and T_5 have no messages to send, which results in two empty minislots. These are followed by m_6 which could not be sent in the first cycle. The DYN segment ends with one empty minislot which might either be because T_7 had nothing to send or its message was longer than one minislot.

It may be noted that (i) the ST and DYN segments are independent of each other, and (ii) techniques for analyzing the timing behavior of the ST segment are already known (because it uses a TDMA scheme) [3]. Hence, major part of the complexities arises from the temporal behavior analysis of the messages being transmitted over the DYN segment of the FlexRay.

III. REAL-TIME CALCULUS (RTC)

At the heart of the RTC [8] lies the modeling of (i) the triggering pattern of tasks (i.e. the event model) which generate an execution demand on an ECU and communication demand on the bus, and (ii) the service offered by an ECU (or the bus) to each task running on it (i.e. the resource model).

A. Event Model

The arrival rate of any event stream triggering a task is upper and lower-bounded by two functions $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$. Let $R(t)$ be the total number of events that arrive during the time interval $[0, t]$. Let $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$ denote the maximum and minimum number of events that might arrive within *any* interval of length Δ as shown in Equations (1) and (2). The timing properties of standard event models — like *periodic, periodic with jitter* and *sporadic* — as well as more arbitrary arrival patterns can be represented by an appropriate choice of $\alpha^u(\Delta)$ and $\alpha^l(\Delta)$.

$$\alpha^u(\Delta) = \max_{t\geq 0}\{R(t+\Delta) - R(t)\} \tag{1}$$

$$\alpha^l(\Delta) = \min_{t\geq 0}\{R(t+\Delta) - R(t)\} \tag{2}$$

B. Resource Model

Let $\beta^u(\Delta)$ and $\beta^l(\Delta)$ denote upper and lower bounds on the *service* available to a task. Let $S(t)$ be the number of activations of this task that were serviced during the time interval $[0, t]$. Then, $\beta^u(\Delta)$ and $\beta^l(\Delta)$ are as per the Equations (3) and (4). If there are multiple tasks running on an ECU, the service bounds $\beta^u(\Delta)$ and $\beta^l(\Delta)$ available to any task clearly depend on the scheduling policy being used.

$$\beta^u(\Delta) = \max_{t\geq 0}\{S(t+\Delta) - S(t)\} \tag{3}$$

$$\beta^l(\Delta) = \min_{t\geq 0}\{S(t+\Delta) - S(t)\} \tag{4}$$

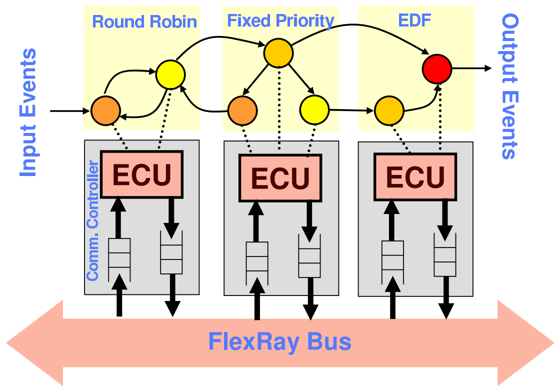


Fig. 2. A FlexRay based network of ECUs, with an application partitioned and mapped onto multiple ECUs.

C. Performance Model

Based on the event and resource model described above, the RTC framework computes the various performance parameters. The two important performance parameters are the worst case (or maximum) end-to-end delay suffered by any particular message while being transmitted over the FlexRay bus and the buffer requirement in the same scenario. Given $\alpha^u(\Delta)$ and $\beta^l(\Delta)$, the maximum *delay* experienced by a task before its activation is serviced and the maximum number of *backlogged* activations can be computed using (5) and (6) [8].

$$\text{delay} \leq \sup_{t\geq 0}\{\inf_{\tau\geq 0}\{\alpha^u(t) \leq \beta^l(t+\tau)\}\} \tag{5}$$

$$\text{backlog} \leq \sup_{t\geq 0}\{\alpha^u(t) - \beta^l(t)\} \tag{6}$$

D. The RTC Toolbox

The RTC Toolbox [10] is a MATLAB toolbox for system-level performance analysis of distributed real-time and embedded systems. A Java kernel carries out the computations on the curves based on the RTC framework while a set of MATLAB libraries connect the kernel to the MATLAB command line. Thus, in essence, the toolbox provides us with a library of MATLAB functions for compositional performance analysis.

IV. RTC BASED FRAMEWORK

The system architectures, we are interested in, consist of multiple ECUs communicating via FlexRay bus. One or more applications are partitioned into tasks, which are then mapped onto different ECUs. ECUs running multiple tasks use a scheduler to share the available processing resources as shown in Figure 2. Each task is activated at a certain rate or is triggered by an output from another task. Once activated, it needs to be processed and hence consumes a fixed number of processor cycles from the ECU on which it is running. In this section, we give an overview of our basic modeling framework for such networks. Our modeling technique [2] is adopted from RTC, where a mathematical framework was presented for analyzing the timing properties of multiprocessor embedded systems communicating via FlexRay bus.

As already mentioned, ST segment of FlexRay uses TDMA scheme and its temporal behavior is known. Therefore, our modeling considerations in [2] are mainly focused on timing behavior analysis of the messages transmitted over the DYN segment of the FlexRay. In the DYN segment of FlexRay, the tasks are given access to the bus in decreasing order of their priorities. In other words, the task with the highest priority is offered access to the bus at the start of the DYN segment. Further, once given access to the bus, a task can occupy it till the end of the current DYN segment. Hence, the most straightforward approach would be to model the protocol as a fixed priority scheduler. Here, β is used to model the total service offered by the DYN segment and successive β's are computed from the message sizes and message generation rates of the different tasks. However, this approach does not work because of the following properties of FlexRay:

- One minislot is consumed from the available service each time a task is not ready to transfer a message, before the next lower priority task is allowed to send its message on the bus.
- A task can send at most one message in each DYN segment (where the maximum length of the message can be equal to the length of the DYN segment).
- If a DYN message is generated by its sender task after the slot has started, the message to wait until the next bus cycle starts in order to contend for the bus.
- A task is only allowed to send a message if it fits into the remaining portion of the DYN segment, i.e. a message cannot straddle two communication cycles.

Therefore, the service β offered by FlexRay is obtained by few transformations which are described in details in [2]. We have used the RTC Toolbox to perform the necessary transformations on β for the performance analysis of the FlexRay based ECU networks, and followed a similar combined MATLAB and Java based software architecture. The modified FlexRay model was then plugged into the existing RTC toolbox, thus creating a single unified framework for realizing our performance analysis models. Thus, our implementation framework can now model basic scheduling policies like fixed priority and TDMA as well the FlexRay scheduling policy.

V. DECOMSYS TOOL-CHAIN

DECOMSYS is a commercially available automotive prototyping platform for common network protocols such as CAN, LIN and FlexRay. Typically, the DECOMSYS comes with number of associated tools used for various purposes. The DECOMSYS tool-chain mainly consists of the following,

- Decomsys Node (ARM) kit: Automotive prototyping platform.
- Designer Pro: Software tool for creating configuration.
- Decomsys Bus Doctor2: Monitoring the FlexRay bus.

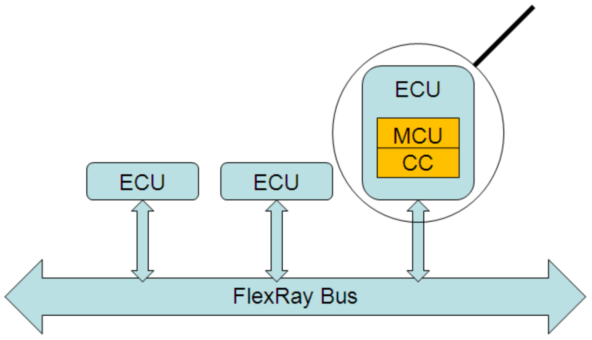
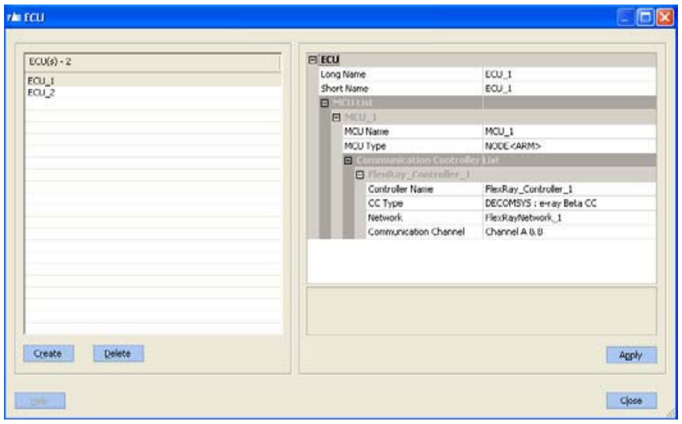


Fig. 4. FlexRay and ECU.



ECU(s) - 2
ECU_1
ECU_2

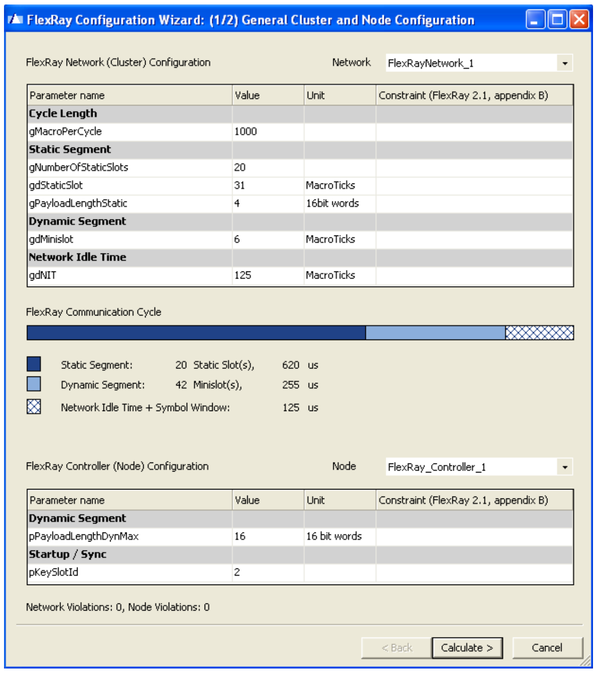
ECU	
Long Name	ECU_1
Short Name	ECU_1
MCU List	
MCU_1	
MCU Name	MCU_1
MCU Type	NODE <ARM>
Communication Controller List	
FlexRay_Controller_1	
Controller Name	FlexRay_Controller_1
CC Type	DECOMSYS : e-ray Beta CC
Network	FlexRayNetwork_1
Communication Channel	Channel A & B

Fig. 5. Overall ECU interface from Designer Pro software.

- Decomsys vision software (EB Treos Inspector): BusDoctor2 interfacing software.
- SIMTOOLS and SIMTARGET software: Modeling, simulating and application code generation.

In the next few paragraphs, we are going to discuss about the DECOMSYS tool-chain. Figure 3 shows the overall architecture of the DECOMSYS based tool-chain. An electronic control unit or ECU comprises of a Node ARM and Communication Controller (CC) (see Figure 4). The Node ARM is a kind of Micro-Controller Unit (MCU) that is used in various applications. The Decomsys Node or Node ARM acts as prototyping platform for various network protocols. The users can flash the application programs into the Node ARM through TFTP. Figure 5 shows the snapshot of the overall Node ARM from the Designer pro software. An ECU can simultaneously have for two CAN, one LIN and two FlexRay communication controllers. The MCUs send messages to the buses using specified CC.

The Designer Pro software is used for generating configuration code for the Node ARM. Various CC parameters are set using the Designer Pro. For FlexRay, Designer Pro software can essentially adjust communication cycle length, length of ST and DYN segments, duration of static slots, duration of minislots and the number of bits transmitted in a slot (Figure 6). Designer pro also allows users to create the configuration code based on the defined parameters. This configuration code can be included in the application code


FlexRay Configuration Wizard: (1/2) General Cluster and Node Configuration
< Back
Calculate >
Cancel

FlexRay Network (Cluster) Configuration   Network: FlexRayNetwork_1

Parameter name	Value	Unit	Constraint (FlexRay 2.1, appendix B)
Cycle Length			
gMacroPerCycle	1000		
Static Segment			
gNumberOfStaticSlots	20		
gdStaticSlot	31	MacroTicks	
gPayloadLengthStatic	4	16bit words	
Dynamic Segment			
gdMinislot	6	MacroTicks	
Network Idle Time			
gdNIT	125	MacroTicks	

FlexRay Communication Cycle

■	Static Segment:	20 Static Slot(s),	620	us
□	Dynamic Segment:	42 Minislot(s),	255	us
▨	Network Idle Time + Symbol Window:		125	us

FlexRay Controller (Node) Configuration   Node: FlexRay_Controller_1

Parameter name	Value	Unit	Constraint (FlexRay 2.1, appendix B)
Dynamic Segment			
pPayloadLengthDynMax	16	16 bit words	
Startup / Sync			
pKeySlotId	2		

Network Violations: 0, Node Violations: 0

Fig. 6. Configuration parameters.

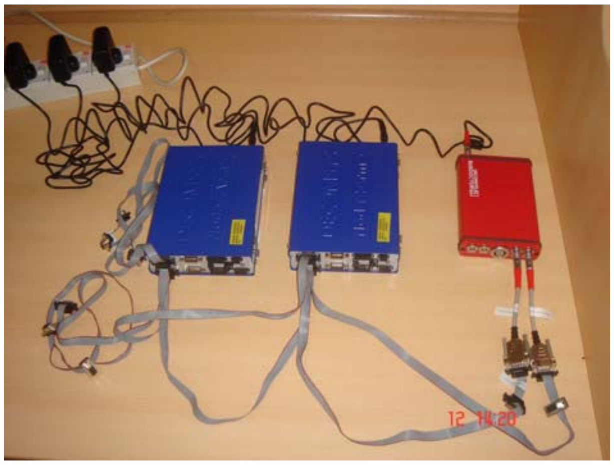


Fig. 7. Overall DECOMSYS setup: ECU (two boxes from left) and Bus Doctor (right most box).

and compiled using the GCC compiler provided with the Node ARM starter kit to binary file. With application code and the configuration code, the overall system requirements can be simulated by flashing the generated binary files into the respective ARM node. The Designer pro has options for exporting the generated configurations into a Fibex file and also for importing configurations into it using the predefined Fibex format files. The generated configuration files are stored as BOR file format by the Designer pro software.

The Decomsys vision software (EB Treos inspector) is the software tool that accompanies the Decomsys BusDoctor2 hardware. The BusDoctor2 hardware can be plugged into the CAN and FlexRay networks. The BusDoctor hardware sends the monitored values to the Decomsys Vision software

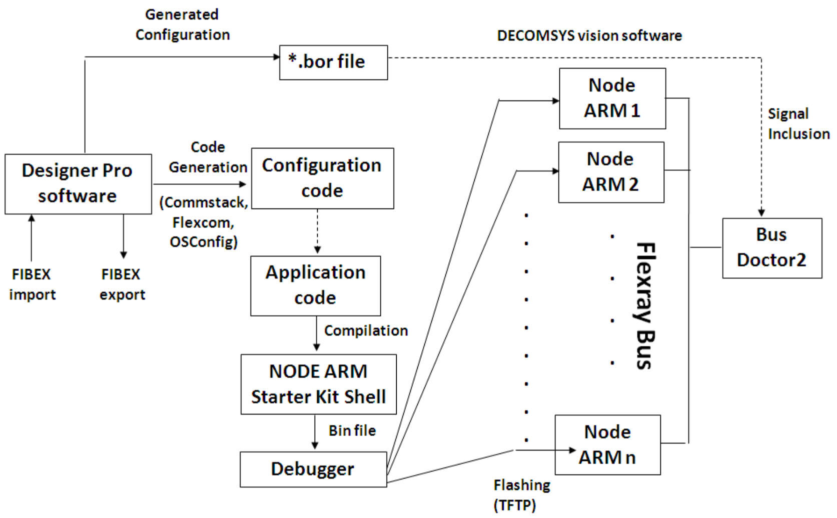


Fig. 3. DECOMSYS-based FlexRay Prototyping.

by an USB or an ethernet interface. The Decomsys Vision can import the signals to be monitored by means of the BOR file generated by the Designer Pro software tool. The Vision software can monitor and record the values of the signals in the bus over a period of time. The recorded values can be stored for later analysis.

SIMTOOLS ans SIMTARGET are software tools provided by Decomsys which combine MATLAB/Simulink and the Designer pro software to generate both application and configuration files. The combined tools with MATLAB eases the design process by combining the design, simulation and code generation into one tool.

VI. Coupling DECOMSYS with RTC based Framework

We propose a two layer performance analysis setup for the FlexRay based systems (Figure 8). First, configuration parameters such as cycle length, length of ST and DYN segments are chosen. The messages, generated from various tasks distributed among different ECUs, are mapped into the ST and DYN segments of the FlexRay cycle. The DECOMSYS is used for configuring FlexRay based systems as described in Section V. Second, the chosen configuration parameters are formally and compositionally evaluated in terms of certain performance parameters such as worst case end-to-end delay and the buffer requirement. The RTC based framework, described in Section IV, can formally analyze the performance of a set of configuration parameters with a specific task-mapping. Towards this, the configuration codes has to be extracted from the DECOMSYS for further analysis by RTC based framework. Designer Pro can export the configuration codes in Fibex format which can further be converted into intermediate graph-based/XML format. The graph-based representation should further generate MAT-LAB basedformat which acts as input to the RTC based framework for analysis.

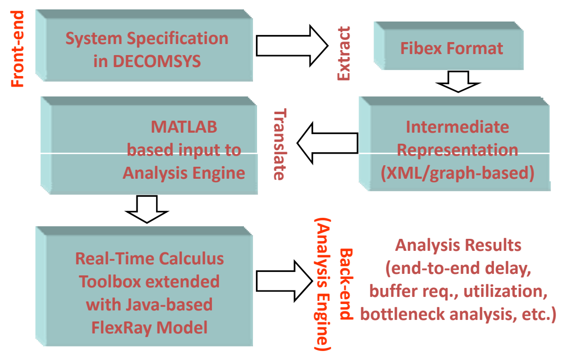

Front-end
System Specification in DECOMSYS
Extract
Fibex Format
MATLAB based input to Analysis Engine
Translate
Intermediate Representation (XML/graph-based)
Real-Time Calculus Toolbox extended with Java-based FlexRay Model
Back-end (Analysis Engine)
Analysis Results (end-to-end delay, buffer req., utilization, bottleneck analysis, etc.)

Fig. 8. Performance Analysis of FlexRay based ECU network using DECOMSYS and RTC based framework.

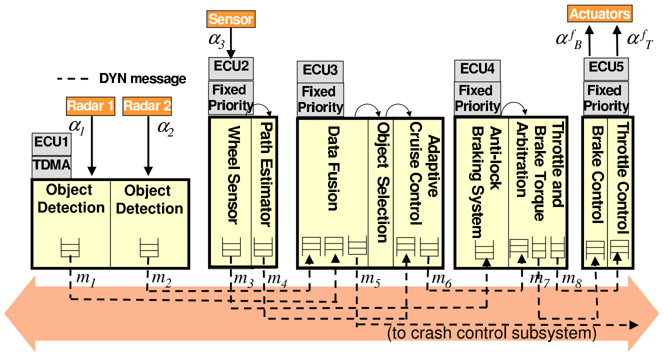


Fig. 9. The system architecture of an ACC subsystem.

VII. Adaptive Cruise Control Application: A Case Study

We will now show the utility of the framework by modeling an adaptive cruise control (ACC) application. In what follows, we first describe the system architecture of ACC application. Subsequently, we shall illustrate how our model can be used for formal performance analysis and debugging of an architecture consisting of multiple ECUs communicating via FlexRay bus.

1) System Description:: As shown in Figure 9, the ACC subsystem consists of five ECUs communicating via a FlexRay bus. The bus communication cycle is 16 ms long. The length of the DYN segment is of 14 ms and consists of 560 minislots, and the length of ST segment is 2 ms[1]. Each minislot in the DYN segment can accommodate 4 bytes of data. $ECU1$ receives data from two radar sensors periodically every 50 ms, and $ECU2$ periodically receives data from a wheel sensor every 50 ms.

The data received by $ECU1$ from each radar is processed by an *Object Detection* task. The processed data streams m_1 and m_2 are sent over the FlexRay bus to $ECU3$ to be processed by the *Data Fusion*, *Object Selection* and *Adaptive Cruise Control* tasks. The periodic data received by $ECU2$ from each radar is processed by the task *Wheel Sensor*. The data processed by this task is sent over the bus as the message stream m_3, which triggers the task *Anti-lock Braking System* at $ECU4$. The task *Adaptive Cruise Control* running at $ECU3$ also receives a message, m_4, from the task *Path Estimator* running on $ECU2$. The resulting data stream from $ECU3$, m_6, is transmitted over the the bus to $ECU4$ which runs the *Throttle and Brake Arbitration* task. The output from the *Throttle and Brake Arbitration* task is fed into the *Brake Control* and *Throttle Control* tasks ($ECU5$) via the messages m_7 and m_8, which in turn send their outputs to two different actuators. These final output control signals are bounded by the functions α_B^f and α_T^f respectively. Finally, $ECU3$ also transmits a message stream m_4 to a *Crash Control* subsystem via the DYN segment of the bus.

In Figure 9, the dashed lines represent messages transmitted via the DYN segment of the FlexRay bus (m_1 has the highest priority, followed by m_2 and so on). The arrows between tasks in $ECU2$, $ECU3$, and $ECU4$ represent data dependencies (i.e. data from the incoming arrow flows into the task pointed to by the arrow). It may be noted that $ECU1$ uses a TDMA policy to schedule the tasks running on it, and the rest use a fixed-priority scheduler. Finally, Table I shows the lengths of the different messages and the execution times of the various tasks running on the different ECUs. The delay and buffer requirements of all the different message streams computed by the RTC based framework are listed in Table II.

TABLE I
THE WORKLOAD ON THE BUS AND THE ECUS FOR THE ACC SUBSYSTEM.

Bus		**ECUs**		
Message	*# Bytes*	*Task*	*WCET*	*BCET*
m_1	128	Data Fusion	8 ms	6 ms
m_2	128	Object Selection	3 ms	1 ms
m_3	64	Adaptive Cruise Control	6 ms	4 ms
m_4	64	Arbitration	7 ms	5 ms
m_5	128	Path Estimation	12 ms	10 ms
m_6	64	Brake Control	4 ms	2 ms
m_7	32	Throttle Control	4 ms	2 ms
m_8	32	Anti-Lock Braking System	8 ms	6 ms
		Wheel Sensor	6 ms	4 ms
		Object detection	6 ms	4 ms

TABLE II
DELAY AND BUFFER REQUIREMENT OF EACH MESSAGE STREAM ON THE FLEXRAY BUS.

Message	*Delay*	*Buffer*
m_1	16.80 ms	128 Bytes
m_2	17.575 ms	128 Bytes
m_3	17.95 ms	64 Bytes
m_4	18.325 ms	64 Bytes
m_5	19.55 ms	256 Bytes
m_6	25.85 ms	192 Bytes
m_7	27.225 ms	64 Bytes
m_8	20.20 ms	64 Bytes

VIII. CONCLUSIONS

FlexRay, which is backed by world's automotive industry, is most likely going to become the standard protocol in the automotive industry. As such, of late there has been lot of interest in performance analysis of FlexRay based networks. In this paper, we have presented a performance analysis scheme for a network of heterogeneous ECUs communicating via a FlexRay bus. In particular, the DECOMSYS-based design specifications of a FlexRay-based ECU network are analyzed and evaluated using the RTC based analysis framework. The combined framework may be exploited for performance debugging and thus assist the designer in choosing the optimal set of system parameters for a given set of design constraints.

[1]The FlexRay communication cycle contains also a *symbol window* and *network idle time* [7], but their size is ignorable and therefore not considered in our analysis.

Acknowledgements: This work is funded by General Motors through the project *Performance Modeling and Tool-Support for Designing FlexRay-based ECU Networks* (National University of Singapore project number R-252-000-345-597).

REFERENCES

[1] "DECOMSYS - Dependable Computer Systems, Hardware und Software Entwicklung GmbH." www.decomsys.com.

[2] A. Hagiescu, U. D. Bordoloi, S. Chakraborty, P. Sampath, P. Vignesh, V. Ganesan, , and S. Ramesh, "Performance analysis of flexray-based ecu networks," in *Design Automation Conference (DAC)*, (San Diego, USA), 2007.

[3] P. Pop, P. Eles, and Z. Peng, "Schedulability-driven communication synthesis for time-triggered embedded systems," *Real-Time Systems*, vol. 26, no. 3, pp. 297–325, 2004.

[4] "CAN Specification, Ver 2.0, Robert Bosch GmbH." www.semiconductors.bosch.de/pdf/can2spec.pdf, 1991.

[5] "ISO/CD11898-4, Road Vehicles Controller Area Network (CAN) Part 4: Time-Triggered Communication, International Standards Organization, Geneva," 2000.

[6] J. Ferreira, P. Pedreiras, L. Almeida, and J. A. Fonseca, "The FTT-CAN protocol for flexibility in safety-critical systems," *IEEE Micro*, vol. 22, no. 4, pp. 46–55, 2002.

[7] "The FlexRay Communications System Specifications, Ver. 2.1." www.flexray.com.

[8] S. Chakraborty, S. Künzli, and L. Thiele, "A general framework for analysing system properties in platform-based embedded system designs," in *Design, Automation and Test in Europe (DATE)*, 2003.

[9] T. Pop, P. Pop, P. Eles, Z. Peng, and A. Andrei, "Timing analysis of the FlexRay communication protocol," in *18th Euromicro Conference on Real-Time Systems (ECRTS)*, 2006.

[10] E. Wandeler and L. Thiele, "Real-Time Calculus (RTC) Toolbox." http://www.mpa.ethz.ch/Rtctoolbox.