

# A Formal Framework for the Correct-by-construction and Verification of Distributed Time Triggered Systems

S Ramesh, P Vignesh V Ganesan, Gurulingesh Raravi<sup>†</sup>  
India Science Lab, General Motors Research  
ramesh.s@gm.com, vignesh.ganesan@acm.org, guru@it.iitb.ac.in

**Abstract**—In modern day automobiles, several critical vehicle functions are handled by ECS (Electronics and Control Software) applications. These are distributed, real-time, embedded systems and due to the level of criticality in their operation, they require a high-integrity development and verification process. Model-based development of such applications aims to increase the integrity of these applications through the usage of explicit models employed in clearly defined process steps leading to correct-by-construction artifacts.

Ensuring consistency and correctness of models across the various design steps is crucial in model-based development methodologies. This work proposes a formal framework for this purpose. The proposed framework captures models in two stages of development – an initial abstract, centralized functional model and a subsequent model of the distributed system consisting of several ECUs (Electronic Control Units) communicating over a common bus. The proposed framework can be used both for the correct-by-construction of distributed models from the abstract functional model and for the verification of existing distributed implementations against a given functional model. We have demonstrated the utility and effectiveness of the proposed framework with two case studies – one using a conventional cruise control system and the other using a brake-by-wire sub-system.

## I. INTRODUCTION

### A. Background

Electronics and Control Software (ECS) is an area of significant growth in the automotive sector. Several key vehicle functions, such as vehicle dynamics and stability control, engine and powertrain control, are handled by software. The features currently governed by ECS applications range from a simple door locking module to adaptive cruise control, anti-lock braking systems and hybrid powertrain management. Sophisticated features like collision-avoidance systems and steer- and brake-by-wire systems are on the verge of becoming a reality.

ECS applications are high-integrity, distributed, real-time applications; they place very stringent requirements on the development processes and supporting deployment infrastructures. As an implementation platform for such applications, time-triggered communication infrastructures, through protocols such as TTP (Time Triggered Protocol) and FlexRay, are

fast gaining popularity [1], [2]. The attractiveness of time-triggered communication infrastructures is primarily due to the real-time guarantees and predictable performance they provide. Given the development of these enablers, there still exists considerable shortfalls in the development process followed for ECS applications. It is widely accepted that model-based methodologies go a long way in filling this gap. In model-based methodologies, one starts with a high level functional model (say, in Matlab Simulink/Stateflow) which is then refined to a distributed implementation through a series of iterative steps. An important high-integrity requirement is that the correctness of the functional model is preserved across the refinement step. This correctness includes not only functional correctness but also timing correctness; for many control functions, correct outputs need to be produced within specified time bounds. the primary goal of this work is to provide a framework and a methodology to establish this correctness.

### B. Current Work

There are two approaches to ensuring correctness: (a) *correct-by-construction* development of the distributed tasks and messages along with their schedules and (b) conventional development followed by *posteriori verification* of task sets, messages and their schedules. We have developed a set of *sufficient conditions* on the task and message characteristics and their schedules; these conditions are presented in the form of linear constraints. These conditions can be used in the context of both the approaches described above – that is, (a) to automatically obtain *correct-by-construction* distributed implementations from functional models we solve for these constraints using linear programming techniques and (b) to verify the correctness of a given distributed implementation against a functional model, we check to see if the constraints are satisfied.

In order to arrive at the constraints we have developed a novel framework that can formally represent both the centralized functional models and their distributed implementation. The framework can accept information about task characteristics such as period, deadline, offset, WCET (Worst Case Execution Time), along with similar characteristics for messages. We assume a schedulable time-triggered common

<sup>†</sup>Currently at Embedded Real-Time Systems Lab, IIT Bombay.

shared bus. Although, our framework is not limited by the number of nodes or buses connecting them, it is, in its current form, limited to time-triggered infrastructures. Since our primary goal is to satisfy the gap in the methodology for engineering time-triggered systems, this paper will not address the challenges in porting this framework to an arbitrated common bus.

The framework can also be used to automatically obtain correct distributed implementations from given functional models. In summary, the contributions of this paper are:

- 1) A novel framework for modeling time-triggered systems and establishing consistency between their centralized and distributed implementations
- 2) A set of constraints for the verification of the task and message schedules in a time-triggered system
- 3) A proposed mechanism for automatic synthesis of task and message schedules in a time-triggered system (given the task distribution)
- 4) Case studies demonstrating the application of our approach – one using a conventional cruise control system and the other using a brake-by-wire subsystem

### C. Existing Work

We have evaluated and refined our framework through several case studies (two of which are presented here) and many interactions with engineers and system designers in our organization. The designers found the framework easy to use – primarily due to its lightweight approach and the fact that the level of abstraction used is what they are accustomed to while building control models. These features helped the engineers to go back and forth between high level models and implementations. This is essential in an iterative development environment where any unsatisfactory artifacts in the implementation and/or performance need to be easily traced to the models and contribute to the redesign of models. This was one of the reasons why many of the recent proposals on the efficient and automatic development of implementations from high level models recently reported could not be easily used by our designers. Some of these proposals [3], [4], [5] are indeed quite advanced, utilizing concrete formalisms and efficient automation of allocation and scheduling of time-triggered tasks. But they used less well-known formalisms, modeling languages and notations, however sound and powerful. Another feature of these proposals that come in the way of immediate adoption is that they recommend complete end-to-end automation of the design process with not-so-easy interfaces and access to the intermediate products for fine tuning to the local needs of the industries. We believe that complete automation of the development of distributed embedded control system is a still long way off and till this happens development methodologies and tools need to enable human intervention for fine tuning the products developed.

The problem of arriving at task schedules in the general context of uni-processor or multi-processor systems [6], [7] and in the context of real-time embedded systems [8], [9], [10], [11] is well-studied. In all these works, the emphasis has been

on efficient utilization of resources. The issue of correctness is often trivial as there is no refinement of high level functional models to lower level task sets. A few scheduling approaches exist, primarily in vendor supplied tools which attempt to solve this problem. These tools enable development of TT systems, but leave several of the steps described above to the designers and the issue of correctness not considered at all.

The organization of the paper is as follows. In the next section, we motivate the problem, providing a brief overview of time-triggered systems (Section II-A) and model-based development (Section II-B). In Section III, we present our formal framework, with a clear definition of the syntax and semantics of our models. Section IV arrives at our core result – a theorem which guarantee correctness across the translation from a functional model to a distributed implementation; the constraints governing this theorem are described along with their rationale in Section IV-A. In Section V, we illustrate the framework using two case studies – a cruise control system (Section V-A) which shows the usage of the model from the verification standpoint and a prototype brake-by-wire subsystem (Section V-B) showing the utility of the framework from the correct-by-construction standpoint. Section VI concludes the paper and provides some motivation for future directions of research.

## II. MOTIVATION OF THE PROBLEM

### A. Time-Triggered Systems

The authors in [1] describe a time-triggered (TT) system consisting of ECUs connected by a common time-triggered bus. In the context of our work, the usage of time-triggered communication infrastructures is primarily due to the temporal properties it guarantees – ECUs communicate during statically allocated time slots, as per the TDMA (Time Division Multiple Access) principle. This is achieved with a globally synchronized clock and static scheduling of tasks and messages. This temporal guarantee is what makes it possible for this framework to make the simplifying assumption of considering the communication bus as another processor.

### B. Model-based development

Model-based development is a well-known development paradigm for developing embedded control software. In this paradigm, the control software is developed in stages, starting with a high level functional model. The functional model is validated using simulation and then refined to software tasks that can run on implementation platforms. Matlab Simulink/Stateflow is one of the well-known and often used tools supporting model-based development of control software. For more information on model-based design with Simulink, please refer [12], [13].

For our work, we assume that the functional model is represented as a set of interconnected functional blocks. Each functional block can send/receive one or more signals (carrying values of different data types) to/from other blocks; each functional block computes certain functions over the input values and produces one or more outputs, which then feed into

other blocks. This computation is repeated periodically, with the *period* decided by the application under design. System designs commonly have components with different periods. Such systems are referred to as *multi-rate* systems, with the rate being  $(1/\text{period})$ . Certain blocks are input blocks capable of producing values without any inputs (sources) while a few others are output blocks consuming signals (sinks). These I/O blocks model the behavior of the sensors and actuators.

At specific points during a period, called *offsets* the input blocks in the functional model *fire*, producing fresh inputs which trigger the firing of the blocks reading these inputs. Any intermediate block is fired when all the blocks producing inputs to it have fired. Finally the output blocks fire and produce the outputs of the system. The computation time of each block and communication time between blocks are assumed to be zero. The application functionality requirement (the end-to-end timing constraints) is used to compute the rates of the system.

Now, given such a functional model (say in Simulink), its implementation on a time-triggered distributed platform involves many steps: (1) Partitioning the model into functions and further into sub-functions (2) Implementing the functions/sub-functions as tasks (3) Allocation of the tasks to different ECUs in the network (4) Assignment of communication slots to tasks within ECUs (5) Scheduling tasks within each ECU.

The tasks, along with their communication, are periodic with a fixed period. It is here that a relationship is established between the application cycle (the period of the application tasks) and the communication cycle (the period of the messages on the communication bus). Typically, the application cycle is an positive integral multiple of the communication cycle. It must be realized that unlike in the case of functional models (where all communication between and computation of blocks is instantaneous), the execution times of tasks and communication times between tasks are non-zero. The challenge, therefore, is to arrive at a schedule of tasks and messages across the network of ECUs, given the functional model, such that no properties of the functional model are violated – for instance, such that tasks execute on the right data at right time and the end-to-end timing constraints of the application are met.

### III. FORMAL MODELS

We will now describe the formal framework developed – the formalism used to describe the functional model and its final distributed implementation, the constraints and how we use the constraints to arrive at the solution along with theorems and proofs of how to use these constraints and ensure correctness of the model across the development process. The functional model is described in what we call the Centralized Control Model (CCM). The CCM captures the fact all development activities during this stage are performed on a model running on a single processor. The distributed version of the functional model, called the Distributed Control Model (DCM) captures the execution of the model after the functional blocks (now

tasks) are allocated to and distributed over a network of communicating ECUs.

#### A. Centralized Control Model

The Centralized Control Model (CCM) describes the functional model, from the point of view of a control engineer. The CCM closely resembles models created in Math works' Matlab Simulink environment. This is done with the intention of deploying the framework we have created within an industrial setting, where Simulink is almost the de-facto standard for control system modeling and simulation. Some of the features of the CCM are given below:

- 1) It captures the different functional blocks and their ordering relationships and constraints
- 2) The content of computation – what exactly is computed in a block – and the actual communication data between blocks are abstracted out
- 3) The computation and communication times are also abstracted out. All computation and communication is assumed to be instantaneous.

We now proceed to define a CCM

*Definition 1:* A CCM is a tuple

$$\langle S, <_c, p, offset_c, deadline_c \rangle \text{ where}$$

- $S$  is a finite set of functional blocks; it represents the basic blocks in a functional model; to allow multi-rate systems, we allow multiple instances of the same blocks in  $S$  but the different instances are distinguished for simplicity.
- $<_c$  is a binary relation over  $S$ , denoting the *firing* order of the blocks in  $S$ .  $<_c$  represents the producer-consumer relationship between blocks; if  $a <_c b$  then the data produced by  $a$  is consumed by  $b$ .
- $p$ , a positive real number, denotes the period of the cycle of execution of the blocks in  $S$ . For the sake of simplicity, we assume that all the blocks have the same period. As mentioned earlier, this is not a restriction since any network of blocks with multiple periods can always be converted into a network with a single period; the single period is the least common multiple of all the periods in the system.
- $offset_c : S \rightarrow [0, p]$ , i.e.,  $offset_c$  is a function that maps all blocks to some specific real values in the interval between 0 and  $p$ . The off-set (along with the deadline function defined next) also modifies the data flow relationship between blocks defined by  $<_c$ . For instance, if two blocks  $a, b$  are such that  $a <_c b$  but  $offset_c(a) > deadline_c(b)$ , then it is interpreted that the data produced in the previous application cycle by  $a$  is consumed by the block  $b$  in the current application cycle.
- $deadline_c : S \rightarrow [0, p]$ . For a block  $b \in S$ , the relative deadline  $deadline_c(b)$ , indicates the time within which  $b$  should be *fired* in each cycle. While  $offset_c$  models the earliest start time for a block,  $deadline_c$  indicates the latest time before which the output of a block appears.

The deadline provides a flexibility in scheduling the blocks; blocks can be fired anywhere between  $offset_c$  and  $deadline_c$ .

The above model of CCM generalizes the standard task graph models used in the scheduling literature [14]. The task graphs capture only the dependency relation between the tasks. But CCMs include the required end-to-end timing relationship between inputs and outputs. We also capture, using very simple abstractions, ‘out-of-cycle’ communication. Further, as seen from the next section, clear semantic behaviors are associated with the CCM, playing a crucial role in the verification of the correctness.

### B. Semantics of CCM

We provide a formal semantics to the CCM assuming the following periodic model of execution: the execution of a CCM proceeds in a series of cycles. All cycles are identical. Each cycle consists of periodically *firing* the various blocks in the CCM, respecting the data flow relation  $<_c$ , the  $offset_c$  and  $deadline_c$  values associated with the blocks; each block  $b$  is executed between the interval  $[offset_c(b), deadline_c(b)]$ .

For defining the semantics, we use the following notations: Given a set  $X$  of finite strings or sequences,  $X^\omega$ , denotes the set of all infinite sequences obtained by repeated concatenation of elements from the set  $X$ . For example, suppose  $X = \{ab, cad\}$ . Then  $X^\omega$  contains the following infinite strings:  $abababababab\dots$ ,  $cadabcadabcadabcad\dots$ . Also, given a set  $Q$ ,  $Perm(Q)$  denotes the set of all permutations of  $Q$  and  $|X|$  denotes the length of the finite string  $X$ .

Formally, the semantics of a CCM,  $A$ , is given as follows:

*Definition 2:* Given a CCM  $A$ ,  $Sem(A)$  is given by the following set

$$\{X \in Perm(S) \mid \forall i, j, \text{ if } X(i) <_c X(j) \wedge \text{deadline}_c(X(i)) < \text{offset}_c(X(j)) \text{ then } i < j\}^\omega$$

The condition on  $X$  in the above definition allows only certain permutations of blocks in  $S$  to appear in the execution sequence; those block occurrences that agree with the offset and deadline values. Each sequence models a possible execution sequence of the CCM, capturing only the ordering relationship between the blocks. We consider only acyclic control systems, that is, no algebraic loops.

*Definition 3:* A CCM is *well-formed* if  $<_c^+$ , the transitive closure of  $<_c$ , is an irreflexive relation.

*Definition 4:* A CCM is *consistent* if the following conditions hold – for any block  $a$ ,  $offset_c(a) \leq deadline_c(a)$ .

In the rest of this report, we shall be concerned only with well-formed, consistent CCMs.

### C. Distributed Control Model

The Distributed Control Model (DCM) models the distributed implementation of the functional model. The key aspects of a DCM are:

- 1) Distributed control wherein all functionality is distributed across several hosts communicating over the time-triggered communication bus

- 2) Tasks<sup>1</sup> and messages are treated as uniform entities; both are “executed” on a “processing unit” (tasks on a ECU and messages on a bus). The time-triggered nature of the communication infrastructure allows us to treat message scheduling on a bus exactly as we would task scheduling on a processor.
- 3) Tasks and messages have a direct causality relationship – messages are triggered by tasks. As described in the formalism, this is, in certain cases, an asymmetrical relationship.

*Definition 5:* The DCM is a tuple

$$\langle E \cup B, S \cup M, <_d, distr, wcet, sched, p_d \rangle \text{ where}$$

- $E$  is the set of execution units, called, ECUs.
- $B$  is the set of time-triggered communication buses over which messages between tasks allocated to different nodes are transferred.
- $S$  is a set of computational tasks, each task denoted by  $\tau_i$
- $M$  is a set of messages exchanged between tasks in  $S$  running in different ECUs. Each message in  $M$  corresponds to at least one sender and one or more receiver tasks. We shall use  $\alpha$  to denote either a task or a message.
- $<_d \subset (S \times M) \cup (M \times S)$ , models the causal relationship between tasks and messages. Using this, we can model the communication between a pair of tasks in  $S$ . Consider two tasks,  $\tau_1$  and  $\tau_2$  and a message,  $m$ . Tasks  $\tau_1$  and  $\tau_2$  are *communicating tasks* if  $\exists m : \tau_1 <_d m \wedge m <_d \tau_2$ .
  - Note that for a task  $\tau$  and message  $m$ , it is possible that  $\tau <_d m$  holds and  $m <_d \tau'$  **does not** hold for any  $\tau'$ . This models the ‘out-of-cycle’ communication; the message sent by  $\tau$  is actually buffered by the receiving node’s controller within the same cycle and is consumed in the next cycle by the receiving task. However, the framework does not allow ‘sender-less’ communication.<sup>2</sup>
- $distr : (S \cup M) \rightarrow (E \cup B)$  is the distribution function which allocates tasks in  $S$  to an ECU and messages in  $M$  to a time-triggered communication bus, i.e.,  $distr(\alpha)$  belongs to  $E$  if  $\alpha$  is a task and to  $B$  otherwise.
- $wcet : (S \cup M) \rightarrow Time$ ; this captures the worst case execution time for tasks and messages. For a message, it is calculated as the sum queuing, transmission and propagation delays. Note that for a message, we round off the  $wcet$  to the next nearest slot. Also,  $wcet(\alpha) \geq 0$  for any  $\alpha$  in  $S$
- $sched : (S \cup M) \rightarrow (Time \times Time)$ ; assigns a pair of positive time points to each task (or) message  $\alpha$  in  $S \cup M$ . Given  $\alpha$ ,  $sched(\alpha) = (t_1, t_2)$ , where  $t_1, t_2$  are the start

<sup>1</sup>For the ease of exposition, while relating the CCM and the DCM, we will not distinguish between the CCM blocks and DCM tasks. We use the same notation  $S$  for denoting these entities and interchangeably use the terms blocks and tasks.

<sup>2</sup>In some sense, the messages are not merely messages but includes the task of sending and buffering in the receiver node. In contrast,  $m <_d \tau$  models only the act of reading the message from the local buffer, by task  $\tau$ .

and end time of execution of  $\alpha$ ; these two time points are denoted by  $begin(\alpha)$  and  $end(\alpha)$  respectively.

- $p_d$  represents the length of the communication cycle. We assume that the application cycle and communication cycle are synchronized.

#### D. Semantics of the DCM

In a manner similar to that of the CCM, we also define a formal semantics for the DCM.

*Definition 6:* Given a DCM  $D$ , the semantics of  $D$ , denoted by  $Sem(D)$ , is given by

$$\{X \in Perm(S) \mid \text{if } end(X(i)) \leq begin(X(j)) \\ \text{then } i < j, \text{ for each } i, j \leq |X|\}^w$$

*Definition 7:* A DCM is said to be *non-preemptive* provided  $(begin(\alpha_1), end(\alpha_1))$  and  $(begin(\alpha_2), end(\alpha_2))$  are not overlapping for all  $\alpha_1$  and  $\alpha_2$  such that  $distr(\alpha_1) = distr(\alpha_2)$ ; that is,  $\alpha_1, \alpha_2$  are allocated to the same execution unit (ECU or bus).

*Definition 8:* A DCM is well-formed if and only if

- 1) for any  $\alpha \in (S \cup M)$ ,  $end(\alpha) \leq p_d$ .
- 2) for any  $m \in M, \tau \in S$ , if  $m <_d \tau$  then  $\exists \tau' \in S$ , such that  $\tau' <_d m$ .

*Definition 9:* A DCM is *consistent* if the following conditions hold –

- 1)  $begin(\alpha) + wcet(\alpha) = end(\alpha)$  for any  $\alpha$  in  $S$ .
- 2) If  $\alpha_1 <_d \alpha_2$  then  $begin(\alpha_2) \geq end(\alpha_1)$  where  $\alpha_1, \alpha_2 \in (S \cup M)$ .

In the rest of our discussion, we will be concerned only with DCMs that are well-formed, non-preemptive and consistent. We now proceed to define the *correctness* of an implementation.

*Definition 10:* A DCM  $D$  *correctly implements* a CCM  $A$ , provided

- 1)  $\emptyset \subset Sem(D) \subseteq Sem(A)$
- 2)  $offset_c(\tau) \leq begin(\tau) \leq end(\tau) \leq deadline_c(\tau) \leq p$ , for each task  $\tau$  in  $S$

These conditions ensure that all data flow and timing relationships between the CCM and the DCM hold.

## IV. VERIFICATION AND SYNTHESIS

The correctness notion defined above forms the basis for verification/synthesis of the distributed implementation. Given a CCM  $A$  and DCM  $D$ , the verification involves checking that  $D$  is a correct implementation of  $A$ . On the other hand, synthesis involves, given  $A$ , automatically generating a  $D$  that is correct. Automatic synthesis of the entire DCM including allocation and scheduling is a very difficult problem. Hence we focus on a synthesis starting with a partial DCM in which all the components of the DCM except *sched* are known; the problem then is to obtain the function *sched* such that the DCM  $D$  when completed with *sched* correctly implements the CCM  $A$ . We now have the following theorem.

*Theorem 11:* Suppose a CCM  $A$  and DCM  $D$  are non-preemptive, well-formed and consistent, with  $p_d = p$ . If

- 1)  $offset_c(\tau) \leq begin(\tau) \leq end(\tau) \leq deadline_c(\tau) \leq p$  for each task  $\tau$  in  $S$  and

- 2)  $\forall \tau_i, \tau_j \tau_i <_c \tau_j$  and  $deadline_c(\tau_i) < offset_c(\tau_j)$  iff  $\tau_i, \tau_j$  are communicating tasks<sup>3</sup>.

Then,  $D$  is a correct implementation of  $A$ .

Definition 7 ensures that we have a non-empty  $Sem(D)$ . Definition 9.2 along with Condition 2 of Theorem 11 ensure that the dataflow relationships in the CCM  $A$  are consistent with those in the DCM  $D$ ; that is, they ensure  $Sem(D) \subseteq Sem(A)$ . The timing relationships between the CCM and the DCM are guaranteed by Definition 9.1 and Condition 1 of Theorem 11.

#### A. Framework Constraints

To summarize, we restate the constraints on a given DCM that must hold for Theorem 11 to be true.

- 1) **Non-preemptive:**  $(begin(\alpha_1), end(\alpha_1))$  and  $(begin(\alpha_2), end(\alpha_2))$  do not overlap for any  $\alpha_1, \alpha_2$  in  $S$  such that  $distr(\alpha_1) = distr(\alpha_2)$
- 2) **Consistency:**
  - a)  $p_d = p$
  - b) For all  $\alpha$  in  $S$   $\alpha = begin(\alpha) + wcet(\alpha)$
  - c) If  $\alpha_1 <_d \alpha_2$  then  $begin(\alpha_2) \geq end(\alpha_1)$  for all  $\alpha_1, \alpha_2 \in (S \cup M)$ .
- 3) **Correctness:**
  - a) For all  $\tau$  in  $S$   $offset_c(\tau) \leq begin(\tau) \leq end(\tau) \leq deadline_c(\tau) \leq p$
  - b) For all  $\tau_i, \tau_j \tau_i <_c \tau_j$  and  $deadline_c(\tau_i) < offset_c(\tau_j)$  iff  $\tau_i, \tau_j$  are communicating tasks<sup>4</sup>.

For a given DCM  $D$ , all the above constraints except Constraint 1 are simple linear constraints. Constraint 1 has a disjunction which might exponentially increase the complexity of finding a solution. These constraints form the basis of the solution to the verification and synthesis problem. For verification, given the distributed implementation of a functional model and the end-to-end timing requirements, one simply constructs the CCM and DCM models and checks whether the above constraints of Theorem 11 are satisfied.

The constraints developed here are similar to those developed by other researchers, albeit for different environments – [15] shows similar models and constraints for systems in a uni-processor setting with asynchronous communication with no communication costs. [16] also shows similar constraints, but not within a model-based development methodology.

The existence of a solution to the synthesis problem depends on the feasibility of solving these constraints. If they are feasible, we arrive at several solutions and need an objective function to arrive at the one best suited to the application. For example, we can consider objective functions based on extensibility [17]. Several heuristic approaches can be considered :

<sup>3</sup>In cases where a task communicates to several other tasks via the same message, this condition shall include the receiver task with the earliest deadline ( $deadline_c$ )

<sup>4</sup>In cases where a task communicates to several other tasks via the same message, this condition shall include the receiver task with the earliest deadline ( $deadline_c$ )

Earliest Deadline First (EDF) [11], Earliest Start Time First (ESTF) (considered here) and Least Laxity First (LLF) [10].

## V. CASE STUDIES

In this section, we illustrate the utility of our framework by considering two automotive applications – a simple Cruise Control application and a real world Brake-By-Wire (Electromechanical Braking EMB) sub-system. The data for representing the EMB system is quite realistic, as it was provided by engineers within our organization who worked on its implementation.

### A. Cruise Control

In this case study, we consider a conventional cruise control system. This system is described as follows. The role of the conventional cruise control in the vehicle to maintain the vehicle at a driver desired speed. The system is shown in Figure 1.

1) *CCM of the cruise control system*: The cruise control system under consideration here is a multi-rate system. The *MODE* block (accepting input from the driver – Driver Set Mode, Desired Vehicle Speed, not shown in Figure 1) is shown here to be running at 12.500ms. The *WSS* (Sensor) blocks run at 3.125ms, feeding the *CC* block with wheel speed data. The *CC* (Cruise Control) block accepts the input from the *WSS* and *MODE* and runs at a rate of 12.500ms. The *CC* block sends the computed control signal to the *ACT* (Actuator) block which also runs at the same rate of 12.500ms. The first step in the representing this system in our framework is to convert the system to a single-rate system. To achieve this, multiple instances of blocks with smaller periods are created. For instance, four instances of *WSS* blocks - *WSS1*, *WSS2*, *WSS3* and *WSS4* are created as shown in Figure 2. The converted model can be described by the CCM as given below.

- **S**:  $WSS_1, WSS_2, WSS_3, WSS_4, MODE, CC, ACT$
- **I**:  $WSS_1, MODE$
- **O**:  $ACT$
- $<_c$  is defined such that:  $WSS_1 <_c CC, WSS_2 <_c CC, WSS_3 <_c CC, WSS_4 <_c CC, MODE <_c CC, CC <_c ACT$
- **p**: 12.500ms

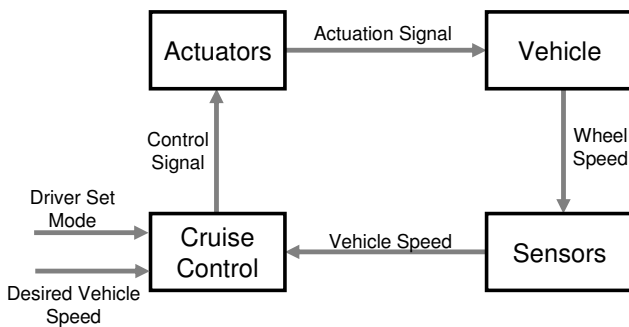


Fig. 1. Block diagram of the cruise control system

- **offset<sub>c</sub>** (in ms):  $WSS_1 = 1.200, WSS_2 = 4.325, WSS_3 = 7.450, WSS_4 = 10.575, MODE = 6.000, CC = 11.1, ACT = 11.9$
- **deadline<sub>c</sub>** (in ms):  $WSS_1 = 3.125, WSS_2 = 6.250, WSS_3 = 9.375, WSS_4 = 12.500, MODE = 12.500, CC = 12.500, ACT = 12.500$

The offset values are chosen based upon some real life instances of this problem and the deadlines are computed from the periods of the blocks.

2) *DCM of the cruise control system*: Given the allocation of tasks to ECUs (*distr*), the messages communicated (*M*) and the worst case execution time (*wcet*) of the various tasks and messages, we compute the message and task schedule using the framework.

- **E ∪ B**:  $E_1, E_2, E_3, B$
- **S ∪ M**:  $WSS_1(\tau_1), WSS_2(\tau_2), WSS_3(\tau_3), WSS_4(\tau_4), MODE(\tau_5), CC(\tau_6), ACT(\tau_7), m(\tau_1, \tau_6), m(\tau_2, \tau_6), m(\tau_3, \tau_6), m(\tau_4, \tau_6), m(\tau_5, \tau_6), m(\tau_6, \tau_7)$
- $<_d$ :  $WSS_1 <_d m(\tau_1, \tau_6), WSS_2 <_d m(\tau_2, \tau_6), WSS_3 <_d m(\tau_3, \tau_6), WSS_4 <_d m(\tau_4, \tau_6), MODE <_d m(\tau_5, \tau_6), CC <_d m(\tau_6, \tau_7), m(\tau_1, \tau_6) <_d CC, m(\tau_2, \tau_6) <_d CC, m(\tau_3, \tau_6) <_d CC, m(\tau_4, \tau_6) <_d CC, m(\tau_5, \tau_6) <_d CC, m(\tau_6, \tau_7) <_d ACT$
- **distr**:  $(WSS_1, WSS_2, WSS_3, WSS_4) \mapsto E_1, (MODE, CC) \mapsto E_2, (ACT_1, ACT_2) \mapsto E_3, m(\tau_i, \tau_j) \mapsto B \forall m$
- **wcet** (in ms):  $WSS = 0.300, MODE = 0.032, CC = 0.350, ACT = 0.400, m(\tau_i, \tau_6) = 0.100$  (where  $1 \leq i \leq 5$ ),  $m(\tau_6, \tau_7) = 0.200$
- **sched**: To be determined
- **p<sub>d</sub>**: 12.500ms

The schedule using the ESTF heuristic for the above task set is shown in Table I (all values in ms). Since the heuristics we have used to generate a schedule are different from those used by the developers of the system our verification process involves ensuring that the schedules are similar – start and end times across both are maintained to meet deadlines. For the purposes of this case study we simply show the schedule we have generated in Table I. We can see from the table, all the constraints stated in Section IV-A are satisfied. Hence, according to Theorem 11, the DCM of cruise control system is verified to be a correct implementation of its CCM.

### B. Brake-by-wire Subsystem (Electromechanical Braking, EMB)

This case study was on a brake-by-wire subsystem based on electromechanical braking (EMB) that has been implemented as a research prototype within our organization. The framework was provided to engineers who worked on the subsystem. With some help they were able to use the framework and the associated prototype scheduling tool to arrive at a task and message schedule. The practical implementation of the subsystem was on several controllers (represented here as the ECUs) connected over a FlexRay bus.

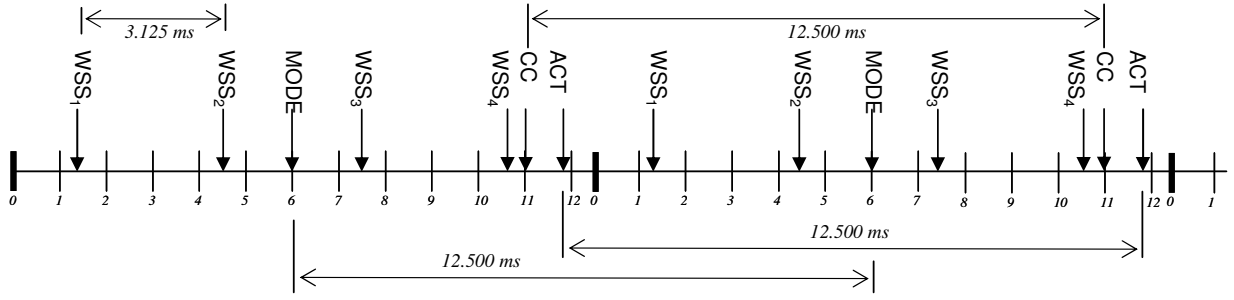


Fig. 2. Expanded single-rate CCM representation of the cruise control functional model

1) *Control System and Functional Architecture*: The brake-by-wire subsystem (Figure 3) accepts an input from the driver in the form of the brake pedal position from the Brake Pedal block (IP in the framework). This input is then given to the control logic (Braking Logic in Figure 3, BL in the framework) and at the same time, provided to the local version of the control logic (Local Logic, LL). This is a replicated block for the purposes of fault-tolerance which computes a local value of the torque request. The brake manager (BM) governs the actual value of the torque and force requests sent to the brake calipers (not shown in Figure 3). It accepts the input from both the brake logic and then local logic (for the torque value) and decides, based on some criteria, which one to select. This is sent to the brake calipers along with the force value supplied by the vehicle manager (VM) supplied force value. The vehicle manager calculates the force value based on several aspects of vehicle dynamics and the torque request from the braking logic.

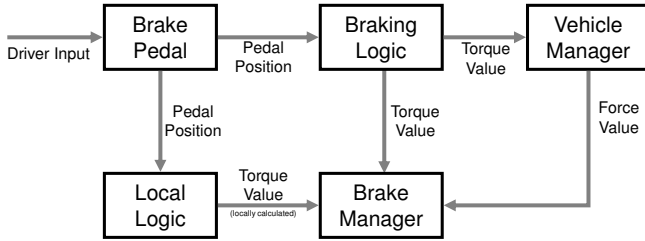


Fig. 3. Functional Architecture of brake-by-wire subsystem

TABLE I

DCM REPRESENTATION OF THE CRUISE CONTROL SYSTEM

S	begin	deadline <sub>c</sub>	M	begin	deadline <sub>c</sub> *
WSS <sub>1</sub>	1.200	3.125	$m(1,6)$	1.550	11.100
WSS <sub>2</sub>	4.325	6.250	$m(2,6)$	4.675	11.100
WSS <sub>3</sub>	7.450	9.375	$m(3,6)$	7.800	11.100
WSS <sub>4</sub>	10.575	12.500	$m(4,6)$	10.900	11.100
MODE	6.000	12.500	$m(5,6)$	6.500	11.100
CC	11.100	12.500	$m(6,7)$	11.500	11.900
ACT	11.900	12.500	-	-	-

\* - deadline for messages is equal to the offset of the receiver task

2) *CCM of the brake-by-wire subsystem*: The brake-by-wire subsystem, much like the previous case study, is also a multi-rate system. We have replicated the IP, BM, BL and VM blocks to arrive as a single rate for the system.

- **S**:  $IP_1, IP_2, LL, BM_1, BM_2, BL_1, BL_2, VM_1, VM_2$
- **I**:  $IP_1, IP_2$
- **O**:  $BM_1, BM_2$
- $<_c$  is defined such that:  $IP_1 <_c LL, IP_1 <_c BL_1, BL_1 <_c BM_1, BL_1 <_c VM_1, VM_1 <_c BM_1, IP_2 <_c BL_2, BL_2 <_c BM_2, BL_2 <_c VM_2, VM_2 <_c BM_2$
- **p**: 12.50ms
- **offset<sub>c</sub>** (in ms):  $IP_1 = 1.20, IP_2 = 7.45, LL = 2.20, BM_1 = 5.75, BM_2 = 12.00, BL_1 = 3.30, BL_2 = 9.55, VM_1 = 5.00, VM_2 = 11.25$
- **deadline<sub>c</sub>** (in ms):  $IP_1 = 6.25, IP_2 = 12.50, LL = 12.50, BM_1 = 6.25, BM_2 = 12.50, BL_1 = 6.25, BL_2 = 12.50, VM_1 = 6.25, VM_2 = 12.50$

3) *DCM of the brake-by-wire subsystem*: The brake-by-wire subsystem, is spread over four ECUs – the brake pedal ECU sends out a pedal position message, the brake logic resides on a separate ECU (usually common across all the four corner brakes), the brake manager and local logic reside on each corner brake and the vehicle supervisor control manager is contained in a separate ECU.

- **E ∪ B**:  $E_1, E_2, E_3, E_4, B$
- **S ∪ M**:  $IP_1(\tau_1), IP_2(\tau_2), LL(\tau_3), BM_1(\tau_4), BM_2(\tau_5), BL_1(\tau_6), BL_2(\tau_7), VM_1(\tau_8), VM_2(\tau_9), m(\tau_1, \tau_3), m(\tau_6, \tau_8), m(\tau_8, \tau_4), m(\tau_2, \tau_7), m(\tau_7, \tau_9), m(\tau_9, \tau_5)$ <sup>5</sup>.
- $<_d$ :  $IP_1 <_d m(\tau_1, \tau_3), m(\tau_1, \tau_3) <_d LL, m(\tau_1, \tau_3) <_d BL_1, BL_1 <_d m(\tau_6, \tau_8), m(\tau_6, \tau_8) <_d VM_1, m(\tau_6, \tau_8) <_d BM_1, VM_1 <_d m(\tau_8, \tau_4), m(\tau_1, \tau_3) <_d BM_1, IP_2 <_d m(\tau_2, \tau_7), m(\tau_2, \tau_7) <_d BL_2, BL_2 <_d m(\tau_7, \tau_9), m(\tau_7, \tau_9) <_d VM_2, m(\tau_7, \tau_9) <_d BM_2, VM_2 <_d m(\tau_9, \tau_5), m(\tau_9, \tau_5) <_d BM_2$
- **dist<sub>r</sub>**:  $(IP_1, IP_2) \mapsto E_1, LL \mapsto E_2, (BM_1, BM_2) \mapsto E_2, (BL_1, BL_2) \mapsto E_3, (VM_1, VM_2) \mapsto E_4$
- **wcet** (in ms):  $IP = 0.30, LL = 1.00, BM = 0.30, BL = 0.25, VM = 0.40, m(\tau_i, \tau_j) = 0.10 \forall m$
- **sched**: To be determined

<sup>5</sup>Multi-cast messages are considered only once, since all receiving tasks can read the message from the same slot on the bus; for example, since the same message from the Brake Pedal( $\tau_1$ ) is read by both the Local Logic ( $\tau_3$ ) and Braking Logic ( $\tau_6$ ), we represent it once as  $m(\tau_1, \tau_3)$ , leaving out  $m(\tau_1, \tau_6)$

- $p_d$ : 12.50ms

TABLE II  
DCM REPRESENTATION OF THE BRAKE-BY-WIRE SUBSYSTEM

S	begin	deadline <sub>c</sub>	M	begin	deadline <sub>c</sub> *
IP <sub>1</sub>	1.20	6.25	$m(1,3)$	1.60	2.20
IP <sub>2</sub>	7.45	12.50	$m(6,8)$	3.65	5.00
LL	2.20	12.50	$m(8,4)$	5.50	5.75
BM <sub>1</sub>	5.75	6.25	$m(2,7)$	7.85	9.55
BM <sub>2</sub>	12.00	12.50	$m(7,9)$	9.90	11.25
BL <sub>1</sub>	3.30	6.25	$m(9,5)$	11.75	12.00
BL <sub>2</sub>	9.55	12.50	-	-	-
VM <sub>1</sub>	5.00	6.25	-	-	-
VM <sub>2</sub>	11.25	6.25	-	-	-

\* - deadline for messages is equal to the offset of the receiver task

The schedule arrived at using the ESTF heuristic for the above task set is shown in Table II (all values in ms). This was generated with the help of an early prototype of the tool we are developing. This tool accepts a description of the system at both the level of the functional model and the distributed representation. It then populates an instance of our framework with this information. Given a schedule, it can verify the constraints provided by the framework and report on the correctness of the schedule (task and message). It can also be used to find a schedule, as in this particular case study. Currently we have limited options of schedulers (and constraint solvers) we can use to find a schedule. We hope to better these options in future versions of the tool. As we can see from the table, all the constraints stated in section IV-A are satisfied. Hence, according to Theorem 11, the DCM of brake-by-wire is verified to be a correct implementation of its CCM.

## VI. CONCLUSIONS AND FURTHER WORK

The paper presents a novel framework for the development of distributed, real-time, embedded systems for control applications in the automotive domain. The framework captures both functional models and their distributed implementations providing uniform treatment of both tasks and messages. The functional model generalizes the standard task graph model used in the scheduling literature by including end-to-end constraints and associating semantic behavior with the functional model. This semantic behavior forms the basis for checking the correctness of implementation. A set of linear constraints have been developed satisfaction of which ensures that the implementation is correct. The set of constraints can also be used to automatically synthesize correct implementations, using one of several linear programming methodologies and associated solvers. A prototype tool was also developed to populate the framework with information from commonly used modeling tools like Matlab Simulink. The paper demonstrated the effectiveness of the framework using two real-life automo-

tive case studies - a cruise control system and brake-by-wire subsystems.

Possible extensions to the work involve –

- Optimizing the payload carried by each slot. Currently, the framework does not restrict placing more than one message in a particular slot. This can be further enhanced with constraints on the slot size and the amount of data that can be packed into it. Providing this as a set of constraints leads to potentially using an optimization tool to find an optimal solution.
- Synthesizing distribution of tasks to nodes, that is, synthesizing a solution to the allocation problem.
- Furthering the development of the tool to involve several schedulers and scheduling policies. Currently, the framework is poised to incorporate other third party heuristics based solvers and optimization tools to find a correct, and in certain cases, an optimal schedule.

## REFERENCES

- [1] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 112–126, Jan 2003.
- [2] N. Navet, Y. Song, F. Simonot-Lion, and C. Wilwert, "Trends in automotive communication systems," *Proceedings of the IEEE*, vol. 93, no. 6, pp. 1204–1223, 2005.
- [3] P. Caspi, A. Curic, A. Maignan, C. Sofronis, S. Tripakis, and P. Niebert, "From Simulink to SCADE/Lustre to TTA: a layered approach for distributed embedded applications," in *LCTES*, 2003, pp. 153–162.
- [4] S. Tripakis, C. Sofronis, N. Scaife, and P. Caspi, "Semantics-preserving and memory-efficient implementation of inter-task communication on static-priority or EDF schedulers," in *EMSOFT*, 2005, pp. 353–360.
- [5] W. Damm, A. Metzner, F. Eisenbrand, G. Shmonin, R. Wilhelm, and S. Winkel, "Mapping task-graphs on distributed ecu networks: Efficient algorithms for feasibility and optimality," in *RTCSA*, 2006, pp. 87–90.
- [6] M. R. Garey and D. S. Johnson, "Strong NP-completeness results: Motivation, examples, and implications," *J. ACM*, vol. 25, no. 3, pp. 499–508, 1978.
- [7] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. R. Kan, "Optimization and approximation in deterministic sequencing and scheduling: A Survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.
- [8] M. Caccamo, T. Baker, A. Burns, G. Buttazzo, and L. Sha, *Real-Time Scheduling for Embedded Systems*, ser. Handbook of Networked and Embedded Systems. Boston: Birkhauser, 2005, pp. 173–195.
- [9] F. Balarin, L. Lavagno, P. Murthy, and A. Sangiovanni-vincentelli, "Scheduling for embedded real-time systems," *IEEE Des. Test*, vol. 15, no. 1, pp. 71–82, 1998.
- [10] K. Ramamritham, J. A. Stankovic, and P. F. Shiah, "Efficient scheduling algorithms for real-time multiprocessor systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 1, no. 2, pp. 184–194, 1990.
- [11] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *Journal of the ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [12] J. Lagenwalter and T. Erkinen, "Embedded steer-by-wire development," in *Embedded World, Nuremberg, Germany*, 2004.
- [13] Mathworks, *The Mathworks Simulink and Stateflow User's Manual*. The Mathworks, 2007. [Online]. Available: <http://www.mathworks.com>
- [14] W. Zheng, J. Chong, C. Pinello, S. Kanajan, and A. L. Sangiovanni-Vincentelli, "Extensible and scalable time triggered scheduling," in *ACSD*, 2005, pp. 132–141.
- [15] R. Gerber, S. Hong, and M. Saksena, "Guaranteeing real-time requirements with resource-based calibration of periodic processes," *IEEE Trans. Software Eng.*, vol. 21, no. 7, pp. 579–592, 1995.
- [16] C. Ekelin, "An optimization framework for scheduling of embedded real-time systems," Ph.D. dissertation, Department of Computer Engineering, Chalmers University of Technology, Sweden, 2004.
- [17] A. Saroop, R. K. P. V. Ganesan, and R. Sethu, "Extensible scheduling in embedded real-time systems with hard deadlines," General Motors Research, Tech. Rep., Feb 2006, available on request.