Efficient Real-Time Support for Automotive Applications: A Case Study

Gurulingesh R., Neera Sharma, Krithi Ramamritham and Sachitanand Malewar
Indian Institute of Technology Bombay
Email: {guru, sachit}@it.iitb.ac.in; {neeras, krithi}@cse.iitb.ac.in

Abstract

The number of computer-controlled functions in an automobile is increasing at a rapid rate and so is the number of microprocessors implementing and controlling these functionalities. Therefore, there is a need to minimize the computing power provided without affecting the performance and safety of the applications. The latter is especially important since intelligent automotive applications deal with critical data and involve deadline bound computations on data gathered from the automobiles' environment. These applications have stringent requirements on the freshness of data items and completion time of the tasks. Our work studies one such safety-critical application, namely Adaptive Cruise Control (ACC). We take a task+data centric approach for designing and implementing this application.

As our contributions we have (i) identified the data and task characteristics of ACC and shown how to map them on a real-world (robotic) platform, (ii) facilitated a real-time approach towards designing ACC by the application of mode-change and real-time data repository concepts for reducing CPU capacity requirements and (iii) provided the scheduling strategies to meet the timing requirements of the tasks. Experiments demonstrate that the CPU capacity requirement can be reduced without compromising the real-time guarantees for safety-critical applications.

I. Introduction

Computer-controlled functions in a car are increasing at a rapid rate and today's high-end vehicles have up to 80-100 microprocessors implementing and controlling various parts of the functionalities [1]. Some of the features currently controlled by microprocessors include electronic door control, cruise control, anti-lock braking system, etc. Sophisticated features like collision-avoidance, lane-keeping and by-wire systems are on the verge of becoming a reality. The practice of implementing each feature as an independent *black-box* excludes any possibility of sharing the applications among the microprocessors. Increasing the microprocessors in relation to individual features will be a difficult proposition both in terms of cost and integration complexity. Hence, the microprocessors must be effectively used to progress in this area to make *fully electronically controlled vehicle* a reality. Features like *Collision Avoidance or Adaptive Cruise Control* are safety-critical, having stringent timing requirements apart from having specific functional behavior. Hence, it is necessary to provide real-time guarantees for such features.

Minimizing the number of microprocessors in an automobile by effectively using them and providing real-time guarantees to the safety-critical applications is our goal. Three important requirements for such high integrity applications are:

- Functional and timing correctness
- Efficient utilization of resources
- Stability and performance analysis of the controller

The focus of this paper is on efficient utilization of resources without affecting the performance and safety of such applications. To this end, we discover efficient methods to allow sharing of multiple functions among the processors and at the same time guarantee real-time responses. This would be a critical need, for the cost-effective development of *fully electronically controlled vehicle* in the future and, in this paper, we have attempted to address this need. The stability analysis of the designed controller is planned in the future work. We have chosen ACC, a safety-critical application as the case-study for our approach.

The rest of the paper is structured as follows. Section II introduces ACC and the current practice followed to develop this application. The issues that need to be addressed to provide computational support for the ACC features are described in Section III. The mainstay of our approach, namely, a two-level real-time data repository and mode-change design, are detailed in Section IV and V, respectively. The experimental setup is described in Section VI and the results of the evaluations from the prototype model are shown under Section VII. Section VIII presents the related work followed by conclusions and future work.

II. Adaptive Cruise Control: An Overview

ACC is an intelligent feature that automatically adjusts vehicle speed to *maintain the safe distance* from the vehicle moving ahead in the same lane (a.k.a. *leading vehicle*). When there is no vehicle ahead, it tries to *maintain the safe*

speed set by the driver. The safe *Distance of Separation (DoS)* that needs to be maintained from the leading vehicle is a function of *host vehicle velocity* and a driver specified parameter, *timegap* (a.k.a. time separation) and it is given by [2]:

$$V_h * T_g + \delta \quad (1)$$

where V_h is the host vehicle speed and T_g is the timegap and δ is the separation distance when the leading vehicle is at standstill. This ensures that safety distance requirement is not violated under the extreme stopping condition where the leading vehicle can come to a halt instantaneously.

ACC continuously monitors the host vehicle's speed, status of cruise control, driver's inputs, and leading vehicle's speed and distance. Using all these data, ACC determines the desired acceleration of the host vehicle and achieves the safe DoS. The controller is typically designed in a hierarchical manner consisting of an *upper-level controller* and a *lower-level controller* [3]. The upper-level controller computes the desired acceleration and the lower-level controller achieves the desired acceleration by manipulating brake and engine outputs.

The various components of the ACC system are shown in Figure 1. We description of each of these components is skipped due to space constraint and can be found in [4].

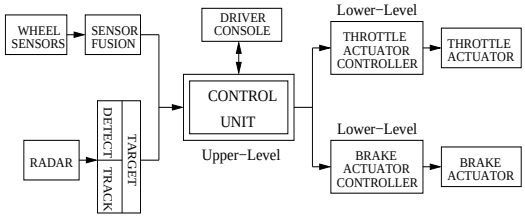

Fig. 1. Block diagram of ACC components

Real-Time Issues in ACC

One of the challenges faced when trying to maintain safe distance is that both the vehicles in the environment are moving. The task of the control system installed in the host vehicle is to continuously track the leading vehicle and adapt its velocity accordingly. The tracking rate should be fast enough to have accurate information about the leading vehicle and slow enough to ensure that the system is not overloaded with redundant operations, i.e., operations that do not lead to any change of control actions. The goal is to achieve correct functionality and meet timing requirements, while using the resources optimally. However, there are several factors which can make it very difficult to achieve these requirements. These include the rate at which vehicles are approaching or separating, vehicles cutting in from adjacent lanes to the host cars lane, and weather conditions.

In current ACC systems, all the sensors sense the data *periodically*. Also, these sensors provide the *raw data* which should be processed to convert them to application specific data known as *derived data*. The raw items reflect the external environment and the derived items are derived from raw items and/or other derived items i.e., each derived item 'd' has a read set denoted R(d) that can have members from both the raw and the derived set [5]. For instance, host velocity is derived from the angular velocity of the wheels obtained from four wheel sensors. The tasks which process raw data to get derived items are also run periodically in the existing systems. The periodicities of all the tasks in the system are set to handle the worst-case scenario.

III. Our Goals and Our Approach

Our goal is to develop solutions that address the following issues:

Effective tracking of dynamically varying data. A data item reflects the status of an entity only for a limited amount of time. When this time expires, the data item is considered to be stale and not valid anymore. This validity interval of a data item is not necessarily fixed during the execution of the system. For instance, consider a leading vehicle that accelerates for a certain amount of time and then travels with uniform velocity. The validity interval (and hence the sampling period) for the data item *leading distance* will be small when the leading vehicle is accelerating and it can be large when it is moving with a uniform velocity. To have a fixed sampling time as in existing systems requires a worse-case design, leading to over-sampled data and ineffective use of the processor.

Timely updates of derived data. The data derivation should be complete before the derived item's read set becomes invalid. The derived item needs to be updated only when one or more data items from its read set changes more than a threshold value. For example, the host velocity is derived only when the angular velocity of wheels changes more than the threshold value. In existing systems, the derived values are updated periodically causing unnecessary updates, leading to over sampling and hence inefficient use of the processing power.

Handling mode specific task sets. ACC system performs different tasks while following a close vehicle when compared to following a vehicle which is far away. For instance, we can have a task that determines how much time is left before the safety criteria are violated when the DoS is small. Similarly, we can have a task that can adjust the driver-set parameters - safe speed and timegap depending on the weather and road conditions when the DoS is large. The task characteristics like periodicity may also vary in different modes. Such changes in the modes of operation affect the task timing requirements, their dependencies, and execution times. In current approaches, all the tasks are executed at all the times. This leads to poor CPU utilization and scheduling overhead.

Our approach to address the above mentioned issues exploits two well known design techniques from real-time system domain: mode-change protocol and real-time data update protocols. Both the approaches help to design the application that leads to effective utilization of the CPU capacity by understanding the needs of the system's task and data characteristics. The mode-change protocol is a *task-centric* approach that allows the designer to vary the task sets and characteristics over a period of time. At any point of time the system will have and schedule only the necessary tasks without wasting the CPU capacity on unnecessary tasks. The real-time data-repository model is a *data-centric* approach that decides the task characteristics from the freshness requirements of base and derived data items. Certain periodic tasks from the mode-change approach are made aperiodic to facilitate the *on-demand updates to data items*.

IV. Specifics of the Dual Mode System

A mode change will typically lead to either:

- adding/deleting a task or
- increasing/decreasing the execution time of a task or
- increasing/decreasing the frequency of execution of a task

For instance, ACC performs different tasks while *following* a *close* leading vehicle compared to one that is *far*. In different modes, we can have the sensing tasks execute at different frequencies to deal with dynamically varying data and we can have different set of tasks active in different modes. Hence, we do not need to have all the tasks active at all the times. The design of ACC application with this approach requires answers to the following questions: (i) How many modes, should the design have? (ii) What condition/event should trigger mode change in the system? (iii) When can we switch modes and how much time can mode change operation take? and (iv) How should the tasks be scheduled to meet their timing requirements? We have explained below how these issues are handled while designing ACC application.

(i)Two mutually exclusive phases of operation for ACC. Non-Critical Mode (NC Mode): In this mode, the environment status does not change rapidly. For instance, when the host vehicle is following a leading vehicle at uniform velocity, the parameters like DoS, leading vehicle velocity do not change rapidly. The rate at which the parameters of the system change decides the periodicity of the tasks. The sensor tasks in this mode can execute less frequently. **Safety-Critical Mode (SC Mode)**: In contrast to NC mode, here the system parameters vary rapidly. For example, consider the case when the leading vehicle is applying maximum brake (say $4.9m/s^2$) and host vehicle under ACC is decelerating (say $2m/s^2$). In this case the DoS between the two vehicles is reducing at a rapid rate. Hence, the radar task which senses this separation should begin to execute more frequently to give the controller fresh data, helping it to determine the right decision at the right time. The system is classified into these two modes of operation based on following parameters:

- Distance between the two vehicles (can take the values FAR, NEAR, FOLLOW).
- Rate of change of Distance - RoD (can take the values Incr-Fast, Incr-Slow, Decr-Fast, Decr-Slow).

Task sets in different modes are shown in Table I. All these tasks are *periodic* in nature. The tasks in NC-mode perform non-critical operations whereas the tasks in SC-mode carry out certain critical operations. The details of task sets are skipped here due to space constraint and can be found in [4].

The regions FAR, FOLLOW and NEAR are characterized in Figure 2 and their description can be found in [4].

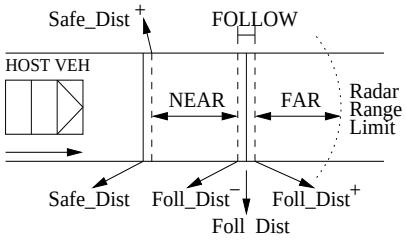

Fig. 2. Environment partitioned into regions

Mode	Task Set
NC Only	WeatherT, FrictionT, AdaptT, EDrT
SC only	TimeLeftT, SuggestT, AdjLaneT
Both Modes	WheelT, SpeedT, CruiseT, AccT, RadarT, LeadVelT, DistT, DriverT, BrakeT, ThrottleT, SwitchT, ExceptionT

TABLE I. Task sets in different modes of ACC system

(ii) Details of the modes. The decision on the current mode of the system is taken based on two parameters DoS and the RoD and change in their values triggers the mode-change phenomenon. Since the controller should have fresh information about the environment, considering only the DoS for switching modes will not yield good results e.g., when the leading vehicle is in the FAR region and decelerating fast. To tackle this case, we also consider RoD as one of the parameters while deciding mode switch condition. The FOLLOW region is used as hysteresis to avoid the *chattering* phenomenon. Table II shows all the possible situations for the system to operate in each of the modes. The details of the conditions to be met by the system in each of the modes are described in [4].

(iii) Switching modes. A mode change request is generated from either the radar task, when DoS parameter satisfies the condition for mode change or another periodic

LeadDist	RoD	mode
FAR	Decr-Fast	SC
FAR	Incr-Fast	NC
FAR	Decr-Slow	NC
FAR	Incr-Slow	NC
NEAR	—	SC
FOLLOW	—	Retain Mode

TABLE II. The state of the system in different modes

task, which tracks the RoD to satisfy the condition. Once the mode-switch condition is satisfied, the mode change process involves deleting the tasks in the current mode and creating the new tasks ensuring schedulability at any given point of time throughout this process. The periodic task *SwitchT* performs these operations in our implementation. The mode-switch operation is initiated upon the termination of the task that makes the mode switch condition true.

Mode change delay, defined as the delay between the time at which mode-change is requested and the time at which all the tasks that need to be deleted have been deleted and their allocated processor capacity becomes available, should be small. As we initiate the mode-change operation once the current task finishes its execution, the delay in reclaiming the processor capacity is bounded by the period of low priority task in the system.

(iv) Scheduling tasks in different modes. The modes in a system are characterized by number of active tasks. The tasks and their characteristics (periodicity, WCET) are known a priori. Hence, static priority scheduling is the obvious choice for system with different modes. Rate Monotonic Algorithm (RMA) is used to schedule the tasks in our implementation.

V. Specifics of the Real-Time Data Repository

In this section we describe our real-time data repository model, which consists of two levels of data store. The concept of two levels of data store is motivated by: (a) the presence of raw and derived data items and (b) the fact that a small change in raw data i.e. sensor values might not affect the action of the ACC controller. As we discussed in Section IV, the wheel sensor task *WheelT* periodically senses the angular velocity of the wheels which is used by another periodic task *SpeedT* to determine the linear velocity of the vehicle. We realize that the periodic execution of the task *SpeedT* may be skipped in some cases for instance, when the host vehicle is following a leading vehicle moving with a uniform velocity maintaining the safe DoS. In such cases, we can choose to skip the execution of *speedT*, until we observe a considerable change in the value sensed by the task *WheelT*. This approach of avoiding the unnecessary updates in the system would result in a best utilization of CPU capacity. Our real-time data repository model is a data centric approach, in which we explore the possibility of making some of the periodic tasks in our task set aperiodic, to reduce the number of unnecessary updates in the system. In this approach we describe the temporal characteristics of the data items in the system, which help us decide the temporal characteristics of the updating tasks associated with the data item.

We use two levels of data store in our approach: *Environment Data Repository (EDR)* and *Derived Data Repository (DDR)* as shown in Figure 2. EDR is an active entity, storing the data pertaining to the controlled ACC system. EDR contains base data items and the procedures for data derivation task. The second repository DDR in the model acts as a global database for the system. ACC controller communicates with DDR to get the current values of vehicle parameters.

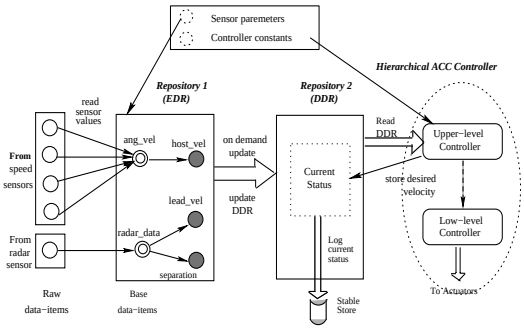

Fig. 3. Real-time data repository for ACC

*1) **Task Models**:* As shown in Figure 2, circular nodes represent data items and arrows acting on these nodes represent tasks operating on data items. We identify the following classes of tasks in the system:

- Sensor Reading (SR) Tasks: Data collected from sensors are *temporally consistent*. They are valid if:

$$CurrentTime - TS(d_i) \leq VI(d_i) \qquad (2)$$

  where, $TS(d_i)$ and $VI(d_i)$ denote time stamp and validity interval of data object d_i, respectively. The tasks *WheelT* and *RadarT* update EDR periodically. These tasks have a known period T_i, a computation time C_i, and a relative deadline D_i equal to its period.
- On-demand Update (OD) Tasks: The derived data items are calculated and updated in DDR only when one of the data items from the read set, $R(d)$ changes more than the threshold value, δ_{th}. The tasks $SpeedT$, $DistT$ and $LeadVelT$ fall under this category. The validity condition of data item d_i in DDR can be written as:

$$|v_i(inEDR) - v_i(inDDR)| \leq \delta_{th} \qquad (3)$$

  where v_i is the value of the data item d_i and δ_{th} is the threshold value.

- Lower Level Controller (LC) Tasks: They give command to the mechanical system of the vehicle to achieve the desired velocity. These tasks are performed by the hardware of our prototype model.
- Other Tasks: They are the lower priority tasks such as WeatherT and FrictionT for monitoring weather and road conditions. These tasks are executed only when there are no critical tasks pending in the system.

2) ***Scheduling of On-Demand tasks****:* The second repository update task and upper level controller tasks are modeled as on-demand aperiodic tasks. A well known conceptual framework to guarantee the service on aperiodic tasks is the *aperiodic server technique*. This models the behavior of aperiodic tasks by reserving a share of processor bandwidth for each of the aperiodic tasks and associates a controlling server thread to maintain each reserved bandwidth. We use Constant Bandwidth Server (CBS) [6] for scheduling aperiodic tasks, since it permits hard real-time guarantees. Since we do not know the utilization (bandwidth requirement) of on-demand aperiodic (e.g., SpeedT) task a priori, we implement CBS which adapts bandwidth dynamically using feedback control which is explained in [4].

VI. Robotic Vehicle: Experimental Setup

This section describes the implementation details of both hardware and software used to demonstrate the concept. Since our aim of this implementation is to prove the concept, we have implemented the essential parts of the system, abstracting out some real world factors which is scalable to accommodate these factors. The robot on which ACC was implemented is shown in Figure 4 which had the following features:

- Obstacle detection range: $2m$.
- Maximum speed: $0.50cm/s$.
- Maintains path through white-line following.
- Closed-loop controller(s).

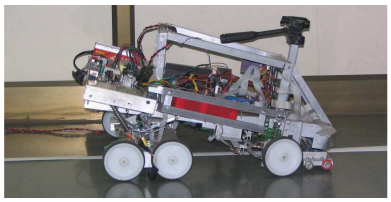

Fig. 4. Robotic Vehicle Platform

The robot was controlled by a PC running on RTLinux-3.1 platform. The PC performed all the computations and issued commands to the robot. The controller polled the data values from different sensors and performed computations to decide the action to be taken and drove the actuators to carry out the appropriate actions. The sensors were used to measure the host vehicle speed and leading vehicle distance. The task structure and data items in real-time repository are shown in Figure 5. The data items are represented by rectangular boxes and the tasks by circular boxes. In SC mode and NC mode, the tasks and data

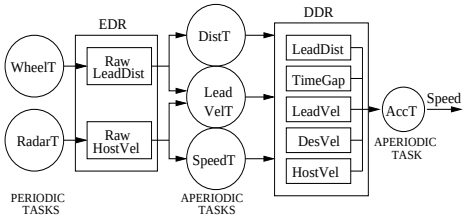

Fig. 5. Task and data structure in real-time repository implementation

items listed in Section IV exist (common tasks) along with an additional task $SwitchT$ to carry out mode change operation. The tasks shown in Table I that exist in one of the two modes are implemented as dummy tasks with different periodicities. In the SC mode, the common tasks execute at double the speed as compared to NC mode.

VII. Results and Observations

Experiments were conducted in three stages. Initial experiments were meant to observe whether the robot was achieving the desired functionalities of the ACC system. This basic implementation did not incorporate either the mode-change or the real-time repository concepts. Secondly, experiments were conducted to test the behavior of the two-level real-time data repository design. These experiments were used to observe the reduction in the number of update tasks executed. The second level tasks were executed only when necessary, as opposed to periodic execution in the initial design without the two-level repository. Finally, experiments were conducted to study the system behavior under mode-change design. We have also studied the effectiveness of CBS scheduling scheme used in real-time data repository approach to schedule the tasks in terms of bandwidth adaptation and scheduling error. Due to space constraint we have skipped the description of these experiments and can be found in [4].

A. Basic Experiments

Four different tests were carried out to observe the system behavior by logging the host velocity and DoS, with time. The experimental vehicle was able to maintain its speed equivalent to the set speed with tolerance of $\pm$3cm/s in cruise control case where there was no leading vehicle. In ACC case where leading vehicle was moving with uniform velocity at a constant DoS, a delay of 0.5s was observed in its velocity response to the changes in the environment which can be attributed to its physical charac-

teristics. The results of these experiments with description can be found in [4].

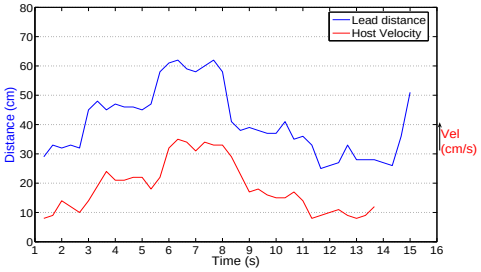

Fig. 6. Host Velocity: Leading vehicle with varying velocity

The next experiment simulated a scenario, where leading vehicle exhibits varying velocity. The DoS was increased gradually between time interval 1-6s, kept constant between 6-8s, then gradually decreased from 8-12s and again kept constant between 12-14s. Figure 6 shows the velocity response and Figure 7 shows the time separation maintained in this case.

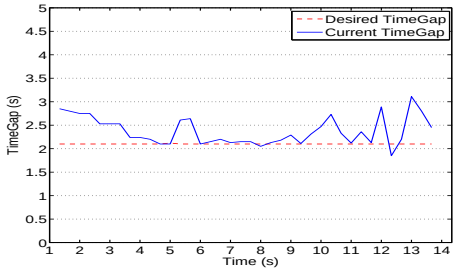

Fig. 7. Timegap: Leading vehicle with varying velocity

B. Two-level Real-Time Data Repository Experiments

The second set of experiments tested the two-level real-time data repository design. These experiments captured the system behavior when the data update tasks of the two-level repository design are executed only when necessary. The velocity response of the host vehicle with the use of the real-time repository where the leading distance increases in time intervals 0-5s and 6-7s, kept constant between 5-6s and 7-8s and then gradually reduced from 8s to 12s is shown in Figure 8.

We can observe from the graph that during the time interval 5-6s and 7-8s when the DoS was constant, the on-demand task is not invoked and during other time intervals, it is invoked whenever the DoS changes by threshold value (which was set to 5cm for this experiment). More experimental results can be found in [4]. Table III shows

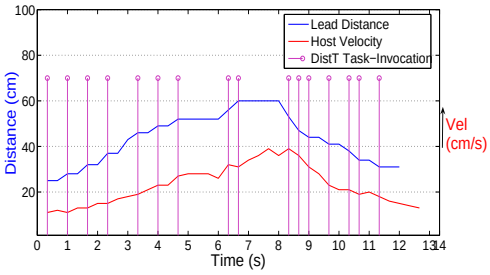

Fig. 8. Host Velocity: Leading vehicle with varying velocity case - *on-demand updates*

the DistT task's invocation with and without two-level real-time repository model. The periodicity of the task was set to 0.3s in the experiments discussed in Section VII-A. We can observe that this approach requires less processing power compared to the conventional approach.

		DistT Task Invocation	
Lead Dist	**Time Window**	**with 2-level**	**without 2-level**
Const	0 to 12	3	40
Incr-Decr	0 to 12	16	40

TABLE III. Number of invocations of lead-dist updating task in the two models

C. Mode-Change Experiments

Finally, we study the effect of mode-change delay on data freshness and safety of the system in terms of maintaining the desired time separation. The velocity response of the host vehicle and time separation maintained with the mode-change implementation, when leading vehicle is moving with varying velocity are shown in Figure 9 and 10 respectively. The DoS is kept as the criterion for changing the mode i.e., if *leading distance* $\leq$ *65cm* enter SC mode else NC mode. The periodicity of the tasks was set to 0.3s in SC mode and 0.6s in NC mode. We can observe from the graph that the system is operating in NC-mode between time interval 8-12s and then it enters SC-mode at time 13s and again switches back to NC-mode at 19s. The desired timegap is violated couple of times in Figure 10 which can be attributed to the inertia of the vehicle and mode-change delay. This suggests that a conservative approach should be taken while deciding the safe DoS or time separation by taking these two factors into account. In this approach too, we can observe that half the CPU capacity is saved when the system operates in NC-mode compared to conventional approach.

VIII. Related Work

Adaptive Cruise Control is a well studied research topic in control systems. The design techniques and simulation results for ACC equipped vehicles are reported in [7]. Prior

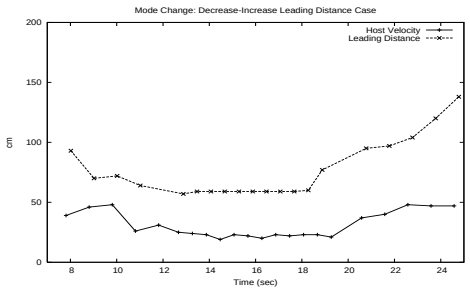

Fig. 9. Host Velocity: Mode-change case, Leading vehicle with varying velocity

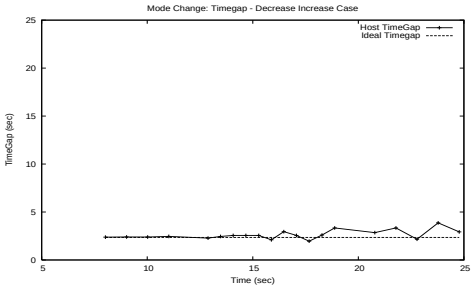

Fig. 10. Timegap: Mode-change case, Timgap in Leading vehicle with varying velocity

work discusses control aspects of the application not the real-time aspects. A datacentric approach to the architectural design of performance critical vehicular applications has been examined before in [8] and [9]. In particular, Gustafsson and Hansson [8] address the issues in the design and implementation of an active realtime database system for EECU (Electronic Engine Control Unit) software. A set of ondemand updating algorithms: OnDemand Depth First Traversal (ODDFT) and OnDemand Top Bottom (ODTB) are presented in [5]. These algorithms optimistically skip unnecessary updates and hence provide increased performance. Methods for the specification and runtime treatment of mode changes are discussed in [10]. An approach to handle mode changes in the time triggered, strictly periodic, pre run-time scheduled system MARS, is studied in [11]. Sha et al [12] attempt to address the problem of analyzing a priori a single processor system, scheduled according to the rate monotonic scheduling policy, with tasks able to lock and unlock semaphores according to the priority ceiling protocol.

IX. Conclusions and Further Work

In this paper, we have presented the issues involved in developing real-time support for ACC. By using a two-level real-time data repository model to update the derived data only when necessary and designing ACC with different modes, each containing different task sets with different characteristics, we have utilized processor capacity effectively, compared to existing approaches. We have shown that these approaches can enable system designers and developers to build a safe and predictable system making effective use of the CPU capacity even if there are demanding timing requirements to be satisfied by the applications.

We are working on possible extensions to the research described here. First, more analysis of the system design is being carried out with different conditions for mode switching, periodicity of the tasks in different modes and conditions for triggering second level update tasks. Second, application needs are being mapped to a distributed platform (as it is the case in the real-world) and the real-time communication issues between the processors are being studied using FlexRay and CAN like communication infrastructures. Third, the impact of mode-change and real-time data repository design concepts on the controller's stability and performance from control theoretical perspective is being studied. Fourth, the stability analysis of the designed controller is being carried out.

References

[1] H. Kopetz, "Automotive electronics," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, June 1992, pp. 132–140.

[2] J. Zhou and H. Peng, "Range policy of adaptive cruise control vehicles for improved flow stability and string stability," in *IEEE Transactions on Intelligent Transportation Systems*, vol. 6, June 2005, pp. 229–237.

[3] R. Rajamani and C. Zhu, "Semi-autonomous adaptive cruise control systems," *IEEE Transactions on Vehicular Technology*, pp. 1491–1501, Sept 2002.

[4] G. Goud, N. Sharma, K. Ramamritham, and S. Malewar, "An efficient real-time support for automotive applications: A case study," IIT Bombay, Tech. Rep. IITB/KReSIT/10, April 2006.

[5] T. Gustafsson and J. Hansson, "Dynamic on-demand updating of data in real-time database systems," in *Proceedings of the ACM Symposium on Applied computing*, NY, USA, 2004, pp. 846–853.

[6] L. Abeni and G. Buttazzo, "Integrating multimedia applications in hard real-time systems," in *RTSS '98: Proceedings of the IEEE Real-Time Systems Symposium*, vol. 19. Washington, DC, USA: IEEE Computer Society, 2-4 Dec. 1998, pp. 4–13.

[7] P. A. Ioannou and C.-C. Chien, "Autonomous intelligent cruise control," in *IEEE Trans. on Vehicular Technology*, June 1993, pp. 657–672.

[8] T. Gustafsson and J. Hansson, "Data management in real-time systems: a case of on-demand updates in vehicle control systems." in *Proceedings of the 10th IEEE RTAS*, 2004, pp. 182–191.

[9] D. Nyström, A. Tesanovic, C. Norström, J. Hansson, and N.-E. Bånkestad, "Data management issues in vehicle control systems: A case study." in *Proceedings of the 16th Euromicro Conference on Real-Time Systems*, 2002, pp. 249–256.

[10] G. Fohler, "Flexibility in statically scheduled hard real-time systems," Ph.D. dissertation, Technische Universität Wien, Institut für Technische Informatik, Vienna, Austria, 1994.

[11] ——, "Realizing changes of operational modes with pre run-time scheduled hard real-time systems," in *Proceedings of the Second International Workshop on Responsive Computer Systems*. Saitama, Japan: Springer Verlag, October 1992.

[12] L. Sha, R. Rajkumar, J. Lehoczky, and K. Ramamritham, "Mode change protocols for priority-driven preemptive scheduling," Amherst, MA, USA, Tech. Rep., 1989.