A Case Study on the Model-Based Design and Integration of Automotive Cyber-Physical Systems

Di Shang, Emeka Eyisi, Zhenkai Zhang, Xenofon Koutsoukos, Joseph Porter, Gabor Karsai, Janos Sztipanovits
Institute for Software Integrated Systems (ISIS)
EECS Department
Vanderbilt University, Nashville, TN 37235 USA
{di.shang, emeka.eyisi, zhenkai.zhang, xenofon.koutsoukos}@vanderbilt.edu

***Abstract*—Cyber-physical systems (CPS), such as automotive systems, are very difficult to design due to the tight interactions between the physical dynamics, computational dynamics and communication networks. In addition, the evaluation of these systems at the early design stages is very crucial and challenging. Model-based design (MBD) approaches have been applied in order to manage the complexities due interactions. In this paper, we present a case study to demonstrate the systematic design, analysis and evaluation of an integrated automotive control system. The system is composed of two independently designed controllers, a lane keeping controller and an adaptive cruise controller, which interact as a result of the integration. The integrated system is deployed on a hardware-in-the-loop simulator for evaluation under realistic scenarios. We present experimental results that demonstrate the effectiveness of the approach.**

I. Introduction

Cyber-physical systems (CPS) are complex systems whose behaviors emerge from the tight coupling and interactions between the physical dynamics, computational dynamics, and communication networks. Automotive systems are classical examples of CPS. Current automotive systems employ up to 100 electronic control units (ECUs) exchanging more than 2500 signals over up to 5 different bus systems [1] [2]. These ECUs control and monitor many subsystems of a vehicle such as chassis control, vehicle stability and engine control etc. Most of these subsystems are safety-critical and hence a failure could result in catastrophic consequences.

Recently, automotive systems have been gaining increased attention due to the increased pressure to integrate as many functionalities on as few ECUs as possible, in addition to the persistent effort for low production costs and tight time-to-market. These economic factors have fueled various emerging challenges to the design of automotive applications. Fueled by these drivers, automotive control applications are typically developed independently. Afterwards, global system objectives, for example autonomous driving, are achieved through the integration of the independently designed control applications, each performing a specific sub-objective towards the global goal. The work in [3] discussed a strategy involving the combined design of lateral and longitudinal controllers. This approach, although it could be beneficial in regards to improving overall system performance, can potentially lead to scalability challenges given the increasing integration of more and more control functionalities in vehicles.

In the integration of these control applications, various interactions from the cyber and physical domains, which may not be accounted for during design can manifest as a result of the composition of these components. Additionally, the independently designed control application might have objectives which result in conflicts during operation. Due to the tight coupling in CPS, the complex interactions within and between the cyber and physical domains of CPS affects the overall behavior of the integrated system and can result in unintended behaviors if not properly handled. Additionally, most issues with the integration of control applications are typically discovered in the final phases of the development cycle and at these later phases, correcting the issues is very expensive as it involves the modification of specifications, requirements and design. Hence, a systematic design, analysis and realistic testing of such system integration of automotive control applications early in the development cycle is very crucial.

A very well-known technique that is very beneficial in addressing these challenges is the model-based design (MDB) approach [4]. However, the lack of a sound approach, for the integration of components from the control design phase with the components from software generation and deployment on actual platform/network, makes the model driven approach very challenging because the tight interactions between the design phases often manifest during integration. In the current state-of-art, ad-hoc methods are often adopted with the goal of "making the system work". These ad hoc methods are becoming unpractical as the complexity of the system increases.

In an effort towards addressing the challenges with system integration in CPS, this paper demonstrates a model-based design and integration of cyber-physical systems with a complex case study application of an automotive CPS. We present a top-down model-based design and integration of two independently designed automotive control applications, a lane keeping control (LKC) application for the control of a vehicle's lateral distance and an adaptive cruise control (ACC) application for the control of vehicle's longitudinal velocity, for the overall objective towards autonomous driving. Our approach, using well-defined models, aims to evaluate and address the interactions from the cyber and physical domains that manifest as a result of the integration. Our model-based development process utilizes a model-based tool-chain, Embedded Systems Modeling Language (ESMoL) [5], which streamlines control design with software modeling, code generation and deployment on platform/network, providing detailed models for the various design layers in order to constrain the resulting interactions. In order to evaluate the system integration, we employ an experimental platform introduced in [6] which is based on the time-triggered paradigm [7]. We present experimental results from the hardware-in-the-loop simulation of the integrated system on the experimental platform.

The rest of the paper is organized as follows. In Section II, we formulate the problem addressed in this paper. Section III presents the control software design flow. In the section IV, the design of the independent controllers, LKC and ACC as well as the supervisory controller for their integration is described. Section V presents an evaluation of the integrated control system as well as the experimental results highlighting the impact of the interactions due to the integration. The conclusion of the paper is provided in Section VI.

II. Problem Statement

In [8], three fundamentally different design layers of CPS are introduced. The three layers are composed of the physical layer, the platform layer and the computation/communication layer. The physical layer represents physical components whose behavior are typically described in continuous (physical) time using, for example ordinary differential equations (ODEs). The platform layer represents the hardware components of CPS and includes the network architecture and computation platform that interact with the physical components through sensors and actuators. While executing the software components on processors and transferring data on communication links, their abstract behavior is "translated" into physical behavior. The computation/communication layer represents the software components executing on platform, with behavior expressed in logical time. A detailed description of these three layers are provided in [8]

This paper is based on an instantiation of the three fundamental design layers, specifically for an automotive CPS as depicted in Figure 1. In this figure, the physical layer includes vehicle chassis, together with the engine, transmission, brakes and tires. The physical objects are interconnected by physical components (e.g steering wheel) or cyber-physical objects (e.g. steer by wire). The platform layer is comprised of the electronic control units (ECUs) on which

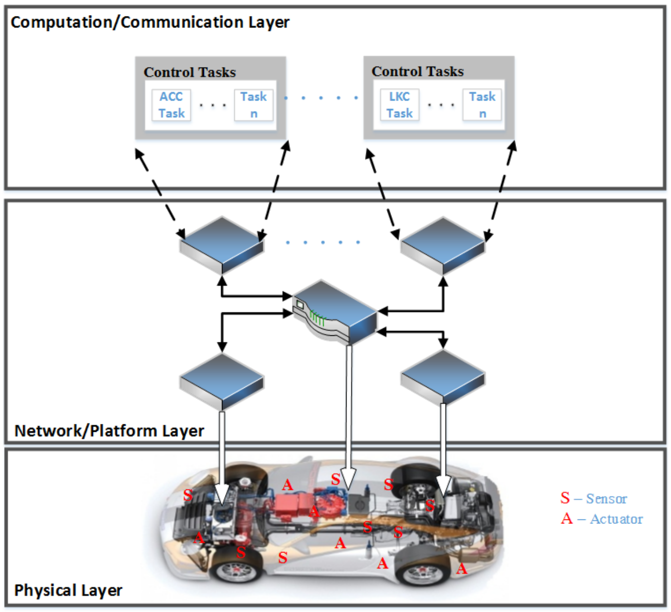


Figure 1. Design Flow in CPS Design Layers

the control software applications are deployed, together with the communication network over which the ECUs send and receive data. The computation/communication layer are comprised of software components such as the lane keeping control (LKC), adaptive cruise control (ACC), and engine control etc. that implement various functionalities in a vehicle.

The emerging behavior of automotive CPS results from the complex coupling and interactions within and across the components in the three design layers. Therefore, understanding these interactions and how they impact the overall system behavior is very crucial. These interactions can typically be grouped into two main categories:

1) **Physical Interactions** represent interactions that manifest as a result of composition of physical objects as well as changes in their dynamics and environment. Examples of such interactions are effects of changes in physical structure such as mass, suspension type, engine type etc. These interactions also include changes in the environment such as changes in road geometry, curves, road grade, banking, frictional surfaces etc.
2) **Cyber Interactions** represent interactions that manifest as a result of composition of cyber components or changes in the cyber components of the CPS. These involve changes in network/platform as well as the computational/communication layers. These changes include variation in network and computation component capacities and speed, deployment model, shared resources, task allocation as well as timing specifications and scheduling etc. These are often attributed to implementation effects and can have adverse impact on the overall system behavior if not handled

appropriately.

In this work, we assume the components of the physical layer are specified by a given physical vehicle dynamic model. We also assume the network/platform layer is specified based on a given set of computational nodes and communication network. The main research problem we address is handling both cyber and physical interactions that manifest in the integration of components in computation/communication layer of an automotive CPS. Specifically, we consider the integration of two independently designed automotive control applications, a lane keeping controller and an adaptive cruise controller for the control of the lateral and longitudinal vehicle dynamics respectively. We aim to address the challenges impacting the behavior of the overall system as a result of cyber and physical interactions due to the integration.

III. Control Software Design Flow

In this section, we describe the proposed high-confidence automotive control software design flow and tool-chain which uses our model-based tool, ESMoL, to integrate two commercial tools, Matlab/Simulink and TTE tools [9]. ESMoL, which is built in GME [10], is a multi-aspect embedded software design environment, which provides a mechanism for automating high-confidence distributed embedded control system design [5]. Figure 2 shows the design flow used in our automotive control software development process. The design flow is composed of eight steps as denoted in Figure 2, describing the top-down software process from control model importation to deployment on the experimental platform for system evaluation.

Step 1 specifies the control functionality in the MATLAB/Simulink environment. In our framework, a controller for a specific vehicle feature is typically designed in Matlab/Simulink and validated using simulations. Subsequently, the Real-Time Workshop (RTW) [11], an automatic code generator in Simulink, is used to generate the equivalent C code of the designed controller. In order to provide detailed implementation details for the controller, we utilize ESMoL. The Matlab/Simulink model of the controller is automatically imported into the ESMoL. The imported model is a structural replica of the Matlab/Simulink controller model in the ESMoL modeling environment. This imported model then becomes the functional specification for instances of software components. **Step 2** involves the specification of the logical software architecture which captures data dependencies between software component instances independent of their distribution over different nodes. **Step 3** defines the hardware platforms hierarchically as nodes with communication ports interconnected by networks. **Step 4** defines the deployment model by mapping software components to nodes, and data messages to communication ports. **Step 5** establishes a timing model by attaching timing parameter blocks to components and messages. **Step 6** translates the

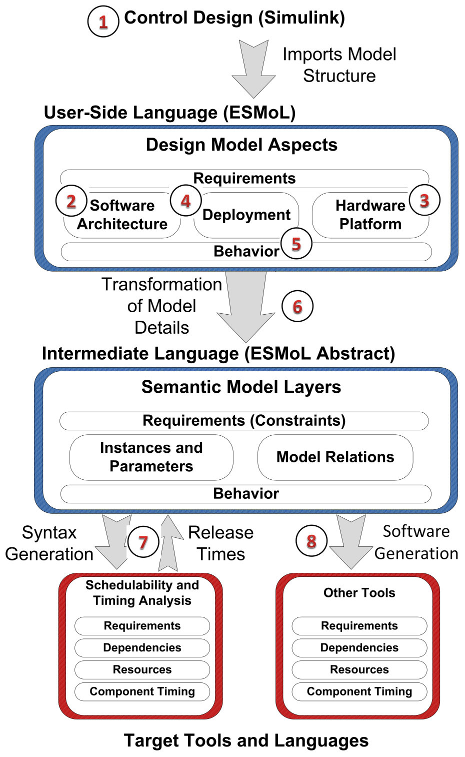


Figure 2. Embedded Control Software Design Flow Supported by the ESMoL Language and Modeling Tools.

ESMoL model into the simpler ESMoL Abstract model using the Stage1 interpreter of ESMoL. The model in this intermediate language is flattened and the relationships implied by structures in ESMoL are represented by explicit relation objects in ESMoL Abstract. **Step 7** takes the scheduling problem specification generated from the ESMoL Abstract model and uses a tool in ESMoL called SchedTool to solve the scheduling problem. The results are imported back into ESMoL model and written to the appropriate objects. A detailed description of the first 7 steps can be found in [5]. **Step 8** generates the control software which is compiled and then deployed on the experimental platform.

IV. Control Design

In this section, we describe the independently designed lane keeping and adaptive cruise controllers as well as the integration of the two controllers.

A. Lane Keeping Control

Motivated by the need to overcome dynamic traffic congestion problems and driving safety issues, the lane keeping control of vehicles has become a very active research area. The lane keeping control is a driver-assitance vehicle feature that automatically controls a vehicle's lateral distance in order to keep the vehicle between lane markings while keeping other parameters such as lateral acceleration within a comfortable driving range. The lane keeping controller

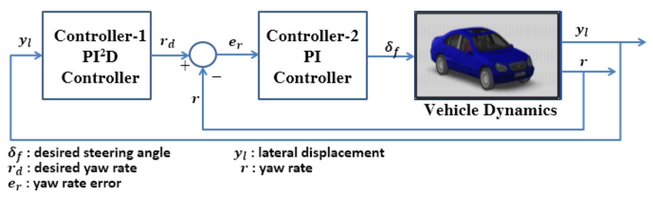

y_l
Controller-1 PI²D Controller
r_d
e_r
+
−
r
Controller-2 PI Controller
δ_f
y_l
r
Vehicle Dynamics
δ_f : desired steering angle
r_d : desired yaw rate
e_r : yaw rate error
y_l : lateral displacement
r : yaw rate

Figure 3. Nested PID Controller

(LKC) executes this objective with the aid of magnetic trackers for detecting magnetic markers on the road or through an integrated vision system.

Over the years, there have been quite a few developed techniques for lane keeping control. An adaptive self-tuning regulator for lane keeping control was presented in [12]. The approach provides lane keeping with robustness to unknown system parameters but the complexity of the approach makes it quite challenging for an actual implementation on a digital platform. In [13], a self-tuning controller based on system parameter identification and pole placement control was introduced. The authors in [14] presented a performance evaluation of several lane keeping control techniques such as H-infinity control, fuzzy control and self-tuning regulator and discussed the relative trade-off with each approach. In this paper, we adopt the nested PID lane keeping controller structure introduced in [15]. The control strategy is to force the lateral displacement of the vehicle at a lookahead distance to zero.

Figure 3 shows the block diagram for the nested PID LKC. The nested PID LKC is composed of two controllers. The outer loop controller, denoted as Controller-1 in Figure 3, is a PID-type controller with an additive integral action on the lateral offset to reject the disturbances on the curvature which increase linearly with respect to time. Controller-1 computes a desired reference yaw rate based on the vehicle's lateral displacement. The control law for Controller-1 is as follows:

$$r_d = -K_{P2} y_l - K_{I2}\int_0^t y_l dt - K_{I3}\int_0^t\int_0^t y_l dt - K_d y_{Ld} \quad (1)$$

Then the y_{Ld} is given by:

$$y_{Ld} = -\frac{1}{\tau^2}\alpha + \frac{1}{\tau} y_l \quad (2)$$

$$\dot{\alpha} = -\frac{1}{\tau}\alpha + y_l \quad (3)$$

where τ is the filter time constant and is set to a value of 0.01, K_{P2}, K_{I2}, K_{I3} and $K_d y_{Ld}$ are the controller gains.

The inner loop controller, denoted as Controller-2 in Figure 3, is a PI-type controller and computes the desired steering angle required for achieving zero lateral distance at the lookahead distance. The control law is described as follows

$$\delta_f = -K_{P1}(r - r_d) - K_{I1}\int_0^t (r - r_d)dt \quad (4)$$

The parameter gains for both controllers used in this paper are presented as follows

$$\begin{gathered} K_{P1} = 12; \quad K_{I1} = 10; \\ K_{P2} = 1.6; \quad K_{I2} = 0.12; \quad K_{I3} = 0.01; \\ K_d = 0.0005; \quad \tau = 0.01 \end{gathered} \quad (5)$$

B. Adaptive Cruise Controller

The adaptive cruise control (ACC) system is an active safety and driver-assistance vehicle feature that automatically controls a vehicles longitudinal velocity in a dynamic traffic environment. ACC enables an ACC-equipped vehicle to follow a leading forward moving vehicle while maintaining a desired distance from the leading vehicle as determined by the vehicles velocity and a specified time gap or headway.

Figure 4 shows a block diagram of the ACC system. The

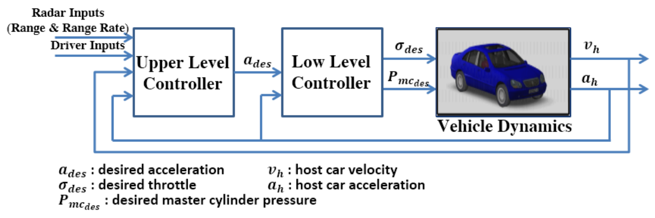

Radar Inputs (Range & Range Rate)
Driver Inputs
Upper Level Controller
a_{des}
Low Level Controller
σ_{des}
$P_{mc_{des}}$
v_h
a_h
Vehicle Dynamics
a_{des} : desired acceleration
σ_{des} : desired throttle
$P_{mc_{des}}$: desired master cylinder pressure
v_h : host car velocity
a_h : host car acceleration

Figure 4. Adaptive Cruise Control System

ACC system is based on our previous work in [6], where two hierarchical levels of control is applied. The upper level controller depicted in Figure 4 uses the driver inputs, the radar measurements and the current distance and velocity of the ACC-equipped vehicle relative to a leading vehicle, to compute the desired acceleration that is required to achieve the desired spacing or velocity. On the other hand, the main objective of the low level controller is two-fold. First, using the desired acceleration command from the upper level controller, the lower level controller determines whether to apply braking control or throttle control. Secondly, the required control command is applied to the vehicle in order to achieve the desired acceleration. The applied control command is either throttle angle command, or master cylinder pressure command. A more detailed description of the ACC depicted in Figure 4 can be found in [6].

C. Integrated Control System

In this section, we consider the integration of the previously described lane keeping controller and adaptive cruise controller. Although the two controllers modify the behavior of two seemingly different dynamics of the vehicle, with the ACC controlling the longitudinal dynamics while the

LKC controls the lateral dynamics, there exists physical interactions in the both the lateral and longitudinal dynamics of the vehicle. Not only that, changes in the physical environment such as geometry of vehicle path or road curvature highlights certain conflicts in the operation of the two distinctive controllers. For example, the ACC on detecting a leading vehicle dynamically adjusts the speed of the ACC-equipped vehicle to the lead vehicle's speed. On a curved road, the ACC in an effort to track the leading vehicle, can attain a vehicle speed that might be too fast, such that it can potentially obstruct the LKC's ability to maintain the desired lateral distance resulting in a conflict. This type of conflict can potentially result in undesired and unintended behavior of the overall system. In order to address these types of conflicts, we integrate a supervisory controller whose main objective is to restrict the regions of operation of the integrated system in a safe desirable manner. The overall integrated system is depicted in Figure 5.

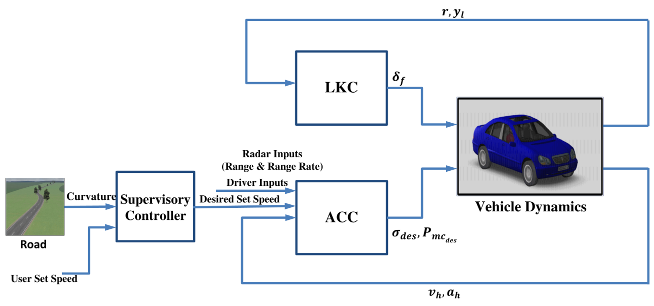


Figure 5. Integrated Control Systems

Figure 6 shows the model of the supervisory controller. The supervisory controller has two main modes and operates

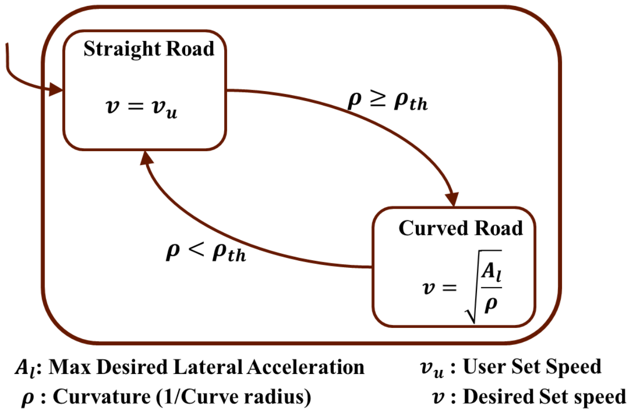


Figure 6. Supervisory Controller

by dynamically determining the desired longitudinal set-speed of the ACC based on the perception of the current road geometry, specifically the road curvature. The idea is that by restricting the allowable speed for the ACC based on the road curvature, the LKC can equally be able to achieve its desired objective of maintaining a desired lateral distance. Hence, on a relatively straight road, the ACC operates in its normal mode based on the user set-speed and radar inputs but on a curvy road, the supervisory controller modifies the user-set speed to a desired speed based on the radius of curvature. The underlying relationship between desired speed and road curvature is described by equation (6)

$$v = \sqrt{\frac{A_l}{\rho}} \tag{6}$$

where v, A_l and ρ are the desired set speed, maximum lateral acceleration and road curvature(the inverse of curve radius) respectively.

V. Evaluation

A. System Architecture

The system architecture for the TTEthernet-based hardware-in-the-loop simulator (HIL) is shown in Figure 7. The system architecture is based on time-triggered paradigm typically used to address the composability and predictability challenges by precisely defining the interfaces between components both in time and value domains [7]. Additionally, our choice of a time-triggered paradigm is in line with the current increased efforts towards the standardization of in-car communication networks, such as FlexRay and Time-Triggered Ethernet (TTEthernet or TTE), with the overall objective of guaranteeing highly reliable, deterministic, and fault-tolerant system performance [16]. We briefly describe the components of the HIL architecture.

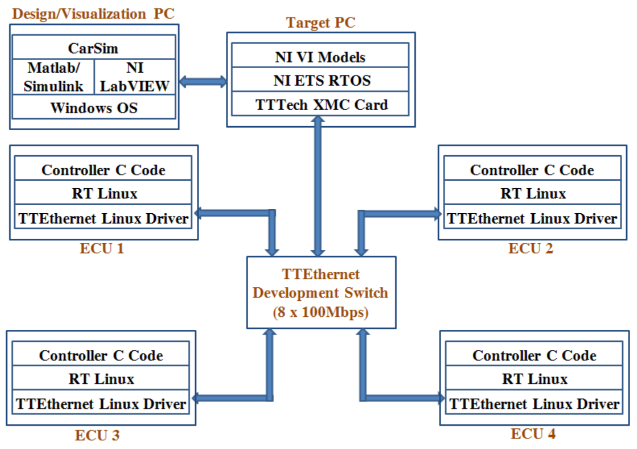


Figure 7. HIL Simulator Architecture

The *Design/Visualization PC* represents the computing platform, running Windows operating system, for the dynamic modeling of a vehicle using CarSim as well as the initial control design and testing using Matlab/Simulink. The design/visualization PC is also used for the visualization and reporting of results from various experiments.

The *TTEthernet Development Switch* is an 8-port 100Mbps system which supports 100 Base-TX Ethernet and enables hard real-time communication based on the TTEthernet protocol. In Fig. 7, the end systems communicate with each other through the switch that form a star or cascaded star network topology. The traffic classes supported in TTEthernet include time-triggered (TT), rate-constrained (RC), and best-effort (BE), which can be used in different mixed-criticality scenarios. In this work, we only use the TT traffic which guarantees the deterministic end-to-end communication delay. The switch operates based on user defined configurations based on an experimental scenario. The configurations are specified in our model-based tool, ESMoL, which is discussed in Section 3.

The *Target PC* is a NI LabView Real-Time Target running NI's Real-Time Module which provides a complete solution for creating reliable, stand-alone real-time systems [17]. In this HIL, the vehicle's physical dynamics modeled in CarSim is deployed on the Target PC during experiments. The Target PC is also integrated with a TTTech PCIe-XMC card [9] which enables the seamless integration and communication with ECUs on the Time-Triggered network supported by the TTEthernet switch.

In Fig. 7, the distributed system has four ECUs, but there could possibly be more or fewer number of ECUs connected at a time based on a specific configuration. In our framework, an ECU is an IBX-530W-ATOM box with an Intel Atom CPU running a Real-Time Linux (RT-Linux) operating system. Each ECU is integrated with a TTEthernet Linux driver using an implementation of the TTEthernet protocol to enable the communication with other end systems in the TTEthernet network. Controller software components are deployed on the ECUs for execution of automotive control applications. The controller software components that are deployed on each ECU are generated from the software design and deployment models for the controller specified in ESMoL.

B. Controller Software Implementation

The integrated control systems composed of the independently designed LKC and ACC controllers as well as the supervisory controller are designed in Matlab/Simulink and validated using simulations. Subsequently, the Simulink models are imported into the ESMoL environment, and then we follow the software design flow described in Section III in order to implement the integrated control system.

Figure 8 shows the logical interconnections of the controller components. Each component represents a task, and there are four tasks, which are *Supervisor*, *ACC*, *LKC*, and *Collection* respectively. Two tasks, *InputHandler* and *OutputHandler*, are used to represent the sensing and actuation of the CarSim.

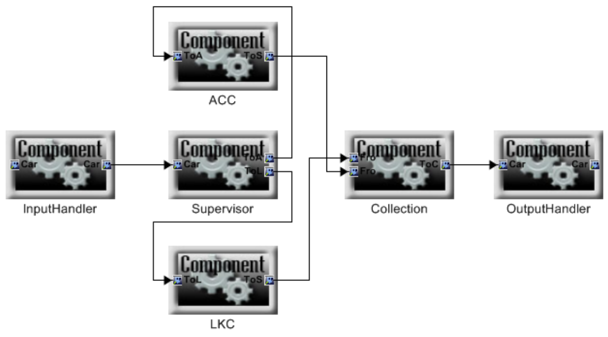


Figure 8. Logical Software Architecture of ACC And LKC Controller.

In Figure 9, the network/platform configurations are explicitly modeled in the ESMoL. Three ECUs are specified as ECU1, ECU2 and ECU3. RT-Target node is where the CarSim simulator runs. In order to represent the sensors and actuators of a vehicle, two virtual I/O devices are used. Specific parameters for TTEthernet need to be defined, such as hyperperiod, bandwidth, time slot size, clock synchronization cycle, and synchronization precision. These specified parameters can be used to generate the TTEthernet configuration script using an ESMoL interpreter.

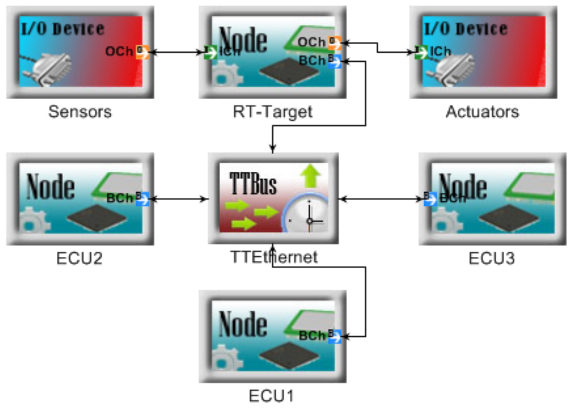


Figure 9. Network/Platform Representation.

Figure 10 shows the deployment model of control software. Dashed arrows represent assignment of components to their respective processors, and solid lines represent assignment of message instances (component ports) to communication channels (port objects) on the processor. We manually deploy *Supervisor* and *Collection* on ECU1, *ACC* on ECU2, and *LowLevelController* and *LKC* on ECU3.

In Figure 11, the timing and execution model for tasks and message transfers of the control system are shown. The control system runs at a period of 10ms. Since the TTEthernet provides a synchronized time base for communication, all the message transfers are attached with *TTExecInfo* to indicate time-triggered communication. For

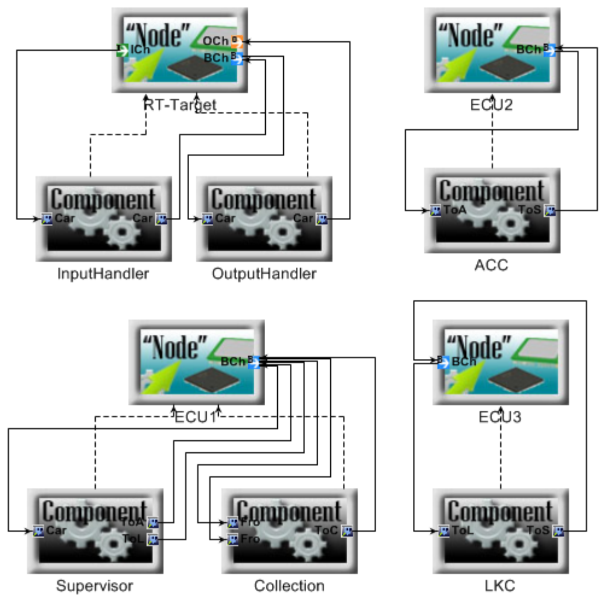


Figure 10. Platform Deployment Aspect of Control Software.

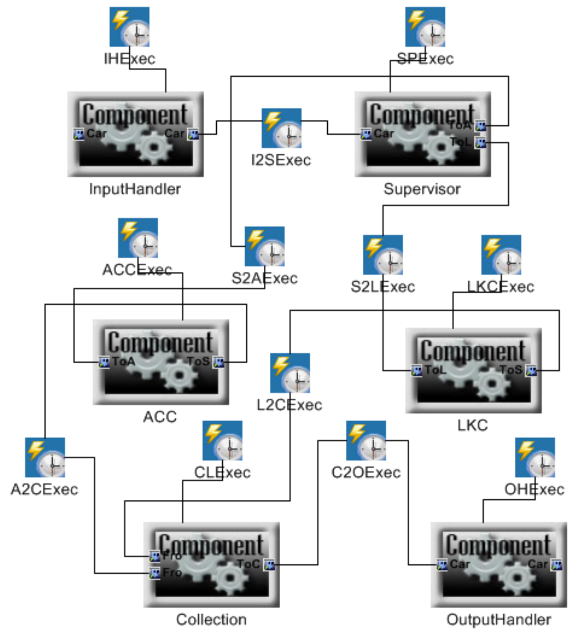


Figure 11. Timing/Execution Model of ACC Controller.

example, *S2AExec* is used to specify the timing for the message transferring from *Supervisor* to *ACC*. We set the execution period which is the hyperperiod of the TTEthernet configuration, desired offset which is used to specify the *initstart_ns* field of TT message in the generated TTEthernet configuration script, and worst case duration which is the worst case communication time of the TTEthernet. The TTEthernet driver on each ECU has a scheduler to take advantage of the synchronized time base, which can invoke the tasks according to a static schedule. Thus, all the tasks can be executed according to the time-triggered paradigm. We specify the *TTExecInfo* for each task. For instance, *LKCExec* specifies the execution time of *LKC* in the 10ms period. Before scheduling, we only need to provide the execution period and the task's worst case execution time, which is measured empirically.

An ESMoL interpreter called Stage1 is used to translate ESMoL models into an abstract intermediate language that contains explicit relation objects that represent relationships implied by structures in ESMoL. This translation is similar to the way a compiler translates concrete syntax first to an abstract syntax tree, and then to intermediate semantic representations suitable for optimization. Stage1 was implemented using the UDM model navigation API. The ESMoL_Abstract target model is the flattened ESMoL model and the source for the transformations implemented in Stage2. After model transformation, the ESMoL model becomes an ESMoL_Abstract model in the form of XML file.

An ESMoL_Abstract interpreter called Stage2 is used to generate the TTEthernet configuration script for network scheduling. This interpreter takes the parameters specified in the TTEthernet components of the network/platform model, and combines them with message specifications. Message specifications are generated for inter-processors message transfers, which can be deduced from the software architecture model and the deployment model. The desired offset field of TT message is from the timing model. When the configuration script is generated, a scheduler provided by TTTech [9] is used for network scheduling.

For task scheduling, we use a heuristic algorithm based on the bottom-level of the system task graph. The bottom-level represents the longest path from any task in the graph to the end of the task graph (we assume the graph is polar). We use a technique such as that described in [18], where the allocation of the tasks to the processors is known, but the message ordering must be determined. The algorithm places each task according to the bottom-level of its corresponding task graph vertex (in descending order). This ensures that all dependencies are met before any task can be scheduled. Our variant of the algorithm heuristically chooses the order of bus messages based on the bottom-level of their sending tasks, with ties broken by the sender whose graph vertex has the greatest out-degree. This heuristic can not guarantee optimality, but generally packs the schedule tightly resulting in a reasonable end-to-end latency. For optimality, other message permutations could produce shorter schedules, but the search for an optimal schedule is known to be NP-hard [18]. In this case of task scheduling, the critical path is simple and is: *InputHandler* $\mapsto$ *Supervisor* $\mapsto$ *ACC* $\mapsto$ *Collection* $\mapsto$ *OutputHandler*.

After network/task scheduling, the schedule information is updated into the ESMoL and ESMoL_Abstract models automatically. The interpreter uses the updated ESMoL_Abstract model to assemble all the codes for compilation. The network scheduling result is used by a tool called ^{TTE}Build from TTTech to generate the binary configuration files for TTEthernet switch and PCIe-XMC card, and C code configuration files for ECUs. The Stage2 interpreter assembles the C code files generated by RTW and ^{TTE}Build with glue code files and generates a Makefile automatically. The tasks are executed in RT-Linux kernel space in order to take advantage of the synchronized time base of TTEthernet. After compilation the kernel modules are deployed onto the respective ECUs as specified by the deployment model.

C. Experiments

In this section, we present two sets of experiments to illustrate our design approach. The first set of experiments presents the initial control design phase of the integrated system composed of the supervisory controller, the LKC and the ACC performed in Matlab/Simulink. This set of experiments highlights the potential impact of interactions and conflicts between the objectives of the ACC and LKC. We show the improved system behavior with the inclusion of the supervisory controller in order to restrict such interactions or conflicts. The second experiment involves the model-based software development of the integrated system with the three controllers based on the controller software implementation and the deployment on the experimental platform for a hardware-in-the-loop simulation. We present results from the hardware-in-the-loop simulation. In all the experiments, a vehicle dynamic model created and configured in CarSim is used to represent the host vehicle. The selected test track is a dynamic path with a combination of straight paths and curved roads with radii of 160m, 200m and 160m for the three curves respectively as seen in Figure 12.

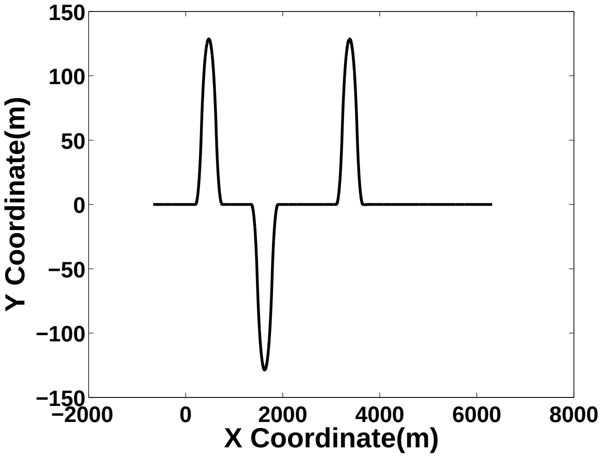
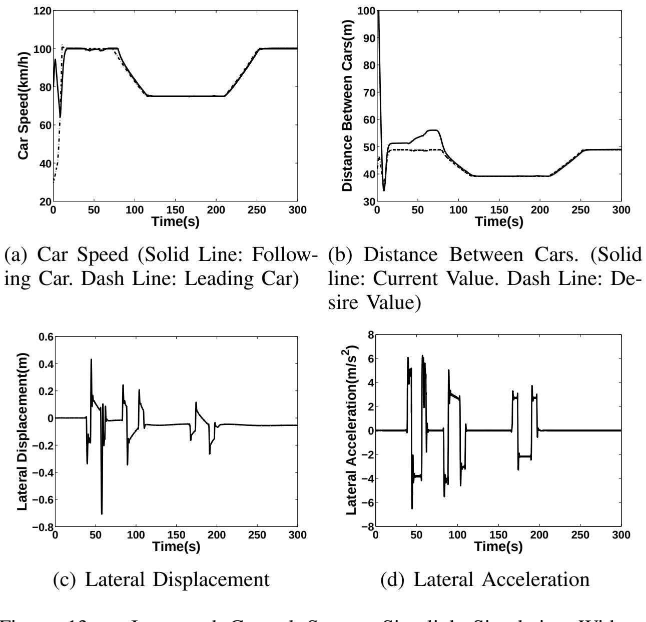


Figure 12. Integrated Control Simulation Track

1) Simulation of the Integrated Control System in the Control Design Phase: This set of experiments are designed to illustrate the importance of the supervisory controller in handling certain physical interactions that emerge as a result of integrating the independently designed LKC and ACC systems. Two scenarios are considered, in the first case we consider the integration of the two controllers without a supervisory controller and in the second case we consider the integration of the controllers with the aid of a supervisory controller.

Figure 13 shows the simulation results for the integrated system without a supervisory controller. The lookahead distance of the LKC controller is $5m$. The desired time-gap of the ACC is set to $1.5s$. The leading vehicle starts at an initial position of $0, 0$ with an initial speed of $30km/h$ while the host vehicle, equipped with the integrated control system, starts at an initial position of $(-800, 0)$ with an initial speed of $80km/h$.

From Figure 13, it can be seen that the ACC performs it's desired objectives effectively by dynamically tracking the leading vehicle's speed as shown in Figure 13a while at same time maintaining a safe vehicle distance as shown in Figure13b. On the other hand, the performance of the LKC is deteriorated due to the resulting conflict and interactions with the ACC. The lateral displacement as shown in Figure13c deviates from the desired lane of the vehicle with a peak value of about $-0.7m$. This amount of deviation can result in potentially catastrophic consequence. Although, the curves in the paths are very aggressive, the lateral acceleration exceeds $4m/s^2$ for the most part in the curved roads.



(a) Car Speed (Solid Line: Following Car. Dash Line: Leading Car)
(b) Distance Between Cars. (Solid line: Current Value. Dash Line: Desire Value)
(c) Lateral Displacement
(d) Lateral Acceleration

Figure 13. Integrated Control System Simulink Simulation Without Supervisor

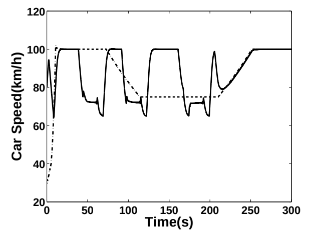
Figure 14 shows the simulation results for integrated system with a supervisory controller as well as a comparison of the lateral performance. The supervisory controller dynamically modifies the set-speed input to the ACC based on the perceived road geometry/curvature. The specified controller and system parameters are the same as in the previous scenario. From Figure14a and b, it can be observed that the set speed input to the ACC is modified by the supervisory controller based on the curvature of the road.

Figure 14c and d compares the lateral performance of the system with a supervisory controller and without a supervisory controller. It can be seen in Figure 14 that, the lateral displacement for the case with a supervisory controller is limited to a peak value of about $-.34m$ as compared to $-0.7m$ in the case without a supervisory controller. Likewise the lateral acceleration is also reduced in the aggressive curves as compared to the case without the supervisory controller. These two scenarios highlights the importance of the supervisory controller in the integration of the two independently designed controllers specifically in handling interactions emerging from the physical layer of the CPS.

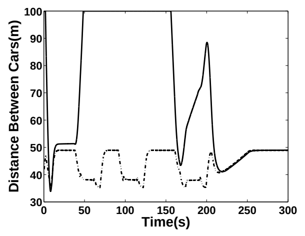
2) Hardware-in-the-loop Simulation of the Integrated Control System: In this experiment, we present the experimental results from testing the integrated control system with the supervisory controller on the experimental platform. The controller and system parameters as well as the path to track for this experiment are identical to the case for the Simulink simulations in the control design phase. We evaluate the impact of platform effects on the performance of the integrated control system as a result of deployment on to the experimental platform.

The results from the execution of the integrated control system on the experimental platform is present in Figure 15. The vehicle speed and distance plots in Figure 15a and b respectively are similar to the speed and distance plots from the integrated system simulation in the control design phase presented in Figure 14a and b.

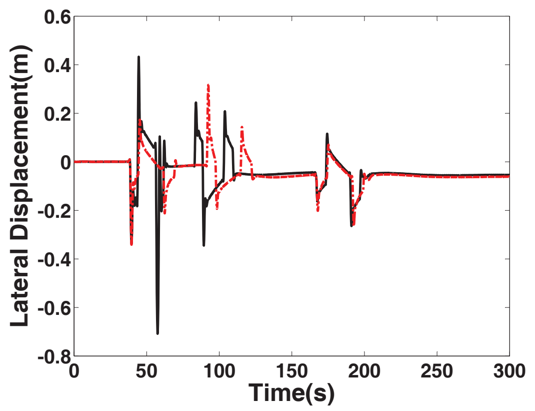
Taking a closer look at the results in Figure15d, we can observe some oscillations in the results from the HIL simulation. This resulting difference can be attributed to platform effects as a result of the deployment of the integrated control system on the experimental platform. Specifically, this implementation limitation is due to the fact that the computation on the RT-Target is not synchronized with the network communication in the platform. The synchronization issue also leads to difference in the lateral distance offset as can be seen in Figure 15c.



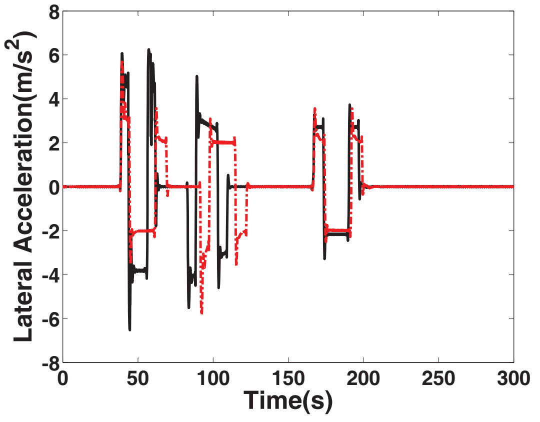
(a) Car Speed (Solid Line: Following Car. Dash Line: Leading Car)



(b) Distance Between Cars. (Solid line: Current Value. Dash Line: Desire Value)



(c) Lateral Displacement (Solid Line: Without Supervisor. Dot Line: With Supervisor)

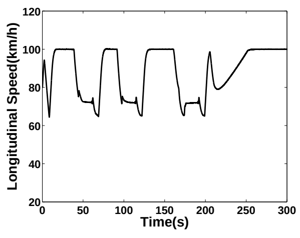


(d) Lateral Acceleration(Solid Line: Without Supervisor. Dot Line: With Supervisor)

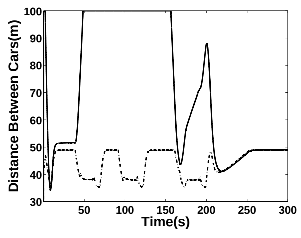
Figure 14. Integrated Control System Simulink Simulation with Supervisor

VI. Conclusion

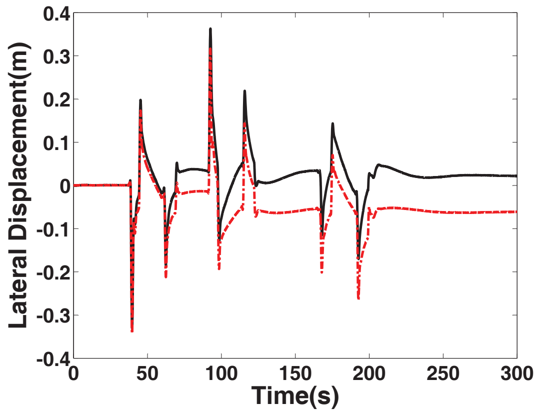
CPS, such as automotive systems, are complex systems that require systematic methodologies such as model-based design in to address CPS design challenges. We present a MBD approach that facilitates the design and integration of time-triggered automotive control systems. We present an automotive control system case study for the integration of two independently designed controllers, the LKC and the ACC. Cyber and physical interactions emerge as a result of this integration. A supervisory controller is integrated to handle certain physical interactions such as the impact of road curvature on the behavior of the two controllers. The designed integrated control system is evaluated using a HIL simulator.



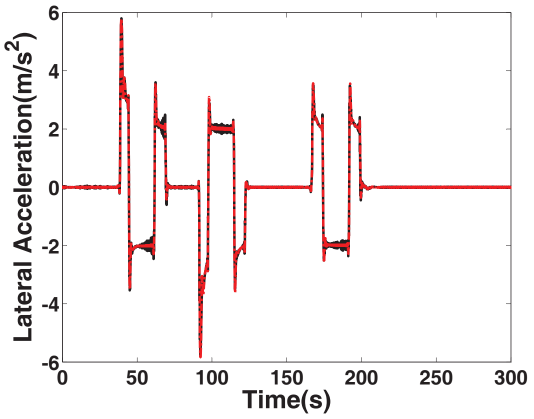
(a) Following Car Speed



(b) Distance Between Cars. (Solid line: Current Value. Dash Line: Desire Value)



(c) Lateral Displacement (Solid Line: HIL Simulation. Dot Line: Simulink Simulation)



(d) Lateral Acceleration (Solid Line: HIL Simulation. Dot Line: Simulink Sim ulation)

Figure 15. Integrated Controller HIL Simulation

REFERENCES

[1] J. Mossinger, "An insight into the hardware and software complexity of ecus in vehicles," *Advances in Computing and Information Technology, Communications in Computer and Information Science*, vol. 198, pp. 99–106, 2011.

[2] ——, "Software in automotive systems," *IEEE Software*, vol. 27, no. 2, pp. 92 –94, march-april 2010.

[3] E. H. Lim and J. K. Hedrick, "Lateral and longitudinal vehicle control coupling for automated vehicle operation," in *Proceedings of 1999 American Control Conference*, vol. 5. IEEE, 1999, pp. 3676–3680.

[4] T. Stahl and M. Volter, *Model-driven software development*. John Wiley and Sons, 2006.

[5] J. Porter, G. Hemingway, H. Nine, C. VanBuskirk, N. Kottenstette, G. Karsai, and J. Sztipanovits, "The esmol language and tools for high-confidence distributed control systems design. part 1: Language, framework, and analysis," Tech. Rep., Sept. 2010.

[6] E. Eyisi, Z. Zhang, X. Koutsoukos, J. Porter, G. Karsai, and J. Sztipanovits, "Model-based control design and integration of cyber-physical systems: An adaptive cruise control case study," *Journal of Control Science and Engineering*, 2013.

[7] H. Kopetz and G. Bauer, "The time-triggered architecture," *Proc. of the IEEE*, vol. 91, no. 1, pp. 112 – 126, jan 2003.

[8] J. Sztipanovits, X. Koutsoukos, G. Karsai, N. Kottenstette, P. Antsaklis, V. Gupta, B. Goodwine, J. Baras, and S. Wang, "Toward a science of cyber-physical system integration," *Proc. of the IEEE*, vol. 100, no. 1, pp. 29 –44, jan. 2012.

[9] "Ttethernet," http://www.tttech.com/en/products/ttethernet/.

[10] A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The generic modeling environment," in *Workshop on Intelligent Signal Processing*, 2001.

[11] "Mathworks - real-time workshop," http://www.mathworks.com/products/rtw/.

[12] M. Netto, S. Chaib, and S. Mammar, "Lateral adaptive control for vehicle lane keeping," in *Proceedings of the 2004 American Control Conference*. Piscataway, NJ, USA: IEEE Press, 2004, pp. 2693–2698.

[13] V. Bobál, J. Böhm, J. Fessl, and J. Machácek, *Digital Self-tuning Controllers: Algorithms, Implementation and Applications*, ser. Advanced Textbooks in Control and Signal Processing Series. Springer-Verlag London Limited, 2005.

[14] S. Chaib, M. Netto, and S. Mammar, "H infinity, adaptive, pid and fuzzy control: comparison of controllers for vehicle lane keeping," in *2004 IEEE intelligent Vehicles Symposium*. Piscataway, NJ, USA: IEEE Press, 2004, pp. 139–144.

[15] R. Marino, S. Scalzi, G. Orlando, and M. Netto, "A nested pid steering control for lane keeping in vision based autonomous vehicles," in *Proceedings of the 2009 conference on American Control Conference*, ser. ACC'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 2885–2890.

[16] 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, june 2005.

[17] "National instruments," http://www.ni.com/.

[18] O. Sinnen, *Task Scheduling for Parallel Systems*. Wiley-Interscience, 2007.