A Multiple Stack Architecture for Intelligent Vehicles

Nicholas F. Maxemchuk, Shou-pon Lin and Yitian Gu
Department of Electrical Engineering
Columbia University
New York, USA
{nick, shouponlin, yitian}@ee.columbia.edu

***Abstract*—We present a general architecture for intelligent vehicles, especially collaborative driving applications. Intelligent vehicles are similar to communication networks as both types of systems interact with physical world using devices that are rapidly evolving, while intelligent vehicles are more complicated than communication network because they interact with the physical world in not only, one but several ways, and because many of the interactions have severe time constraints. The proposed architecture adopts multiple layered stack structures to address the issues. We also present a collaborative merge protocol as an instantiation of the proposed architecture, and discuss the advantage of the architecture in terms of the feasibility of model checking and conformance testing.**

I. Introduction

An architecture is used to decompose a complex problem into smaller, more manageable pieces. Each component of the architecture has a well defined set of inputs and provides a service in the form of a well defined set of outputs. The architecture of a computer program consists of subroutines or function calls that provide services that may be accessed from different locations in a program, rather than reproducing the code.

Architectures in communications networks have isolated the physical devices that perform communications from the routines that use communications to collaborate. Without architectures, programmers have to write routines for computer programs to communicate. When the communications routines are part of a program, a program that is written for one computer or communications device may have to be rewritten to operate on another computer or communications technology.

Communications architectures have evolved from subroutines that provide an interface between application programs and a communication channel to layered architectures that perform more sophisticated communications functions, such as routing or flow control. The subroutines made it possible to change the communications technology without changing the application, while the layered architecture made it possible to change parts of the communications functions without changing others. The Internet is a testament to the success of the layered architecture. Over the years the communication technologies have undergone significant advance, while the services that use the communications, such as email and WEB browsers have continued to operate without much change. To a large extent, the rest of the Internet communications stack has also survived the changes.

Using layered architectures, rather than the more general, modular architectures that are common in computer programs, also simplies the problem of verifying that a system operates correctly. In a layered architecture we can independently verify each module, by proving that each layer provides the service to the next higher layer, given that it receives the proper service from the lower layer. In a more general, modular architecture, in which the output from a module directly, or indirectly feeds back to a module that it receives data from, we must simultaneously prove that all of the modules operate correctly.

Intelligent vehicles are similar to communications networks, whether they collaborate using communications channels or not. They interact with the physical world using devices that are rapidly evolving. The design of intelligent vehicle architectures involves the integration of new features into existing vehicle architectures under the guarantee of robustness. The problem is defined as what functional characteristics should be added to make vehicles intelligent, and how to do the integration. Different intelligent vehicle projects have their own designed architectures to meet the specific objectives, but in common, they inherit general characteristics of all automotive systems.

Modular architecture view is very common in the research area of intelligent vehicles. The system is composed of several modules with each module representing a major functional component. The modules are interconnected, and the connections represent dataflow or inter-dependency between modules, depending on the design choice of architecture enigineers.

We surveyed several previous works on cooperative driving systems, particularly cooperative adaptive cruise control (CACC) systems, which are systems that require communications between cooperating vehicle and and active coordination of maneuvers. SARTRE is a European Commission co-funded project [1], which aims to develop strategies and technologies that enable vehicle platoons to operate on normal public highways, and the 2011 Grand Cooperative Driving Challenge (GCDC) [2] was the first competition to simulate the realistic heterogeneous scenario, in which passenger cars, vans, trucks, and buses are included in the same platoon, sharing roads with manually driven cars.

The architectures that were proposed to realize such systems

[1], [3]–[7] are more or less as hoc in a way that they are independent and tailored for the specific applications. In general these modular architectures are composed of functions such as wireless communications, environmental sensing, vehicle controllers, cordination controllers or platoon controllers, and human-machine interface.

The importance of a general architecture for cooperative driving has been recognized in [8]. A reference architecture for collaborative driving application is proposed. It provides a clear definition of the required services, but the architecture is still modular in nature.

We have found two types of architectures with layers similar to those in architectures for communications networks. One type of the architectures organize the functions that process sensors measurements to determine the positions of neighboring vehicles and obstacles [9]–[11]. The sensing technology can change without changing the applications that process and use the data. The other type of architectures organizes the applications needed for platooning [12]–[17]. Complex intelligent vehicles applications are realized by using the functions in lower layers, and they can be verified assuming that the simpler functions deliver their prescribed functions.

However, architectures with a single stack cannot compose the whole picture of intelligent vehilces. Different from communications networks, intelligent vehicles interact with the physical world in several ways, rather than one, and because many of the interactions are time critical. In addition to communications, intelligent vehicles have sensors that detect the surroundings, measuring devices that monitor the operation of the vehicle, and devices that control the operation of the vehicle. Each type of the layered architectures described above only deals with one aspect of the physical domain. While it is necessary to accommodate changes in all of the technologies, a simple layered architecture is not sufficient, which motivates the adoption of multiple stacks. The architecture of HAVEit project [18], though mostly consists of modules, has perception module divided into layers. Its perception module is subdivided into sensor hardwares and data fusion function.

The objective of this paper is to present a multiple stack architecture that supports the complex interactions between intelligent vehicles and the physical world, and accounts for time critical operations. The details of the architecture will be presented in section II. In section III we present an example instantiation of the multiple stack architecture on a collaborative merging protocol. The paper concludes in section IV.

II. An architecture for collaborative driving systems

Our objective in this section is to describe a target architecture for intelligent driving applications that has characteristics that are similar to those that have been successfully applied to communications networks. Engineering, verifying and testing intelligent driving systems are more complex in intelligent driving applications than in communications networks for several reasons.

1) Intelligent automobiles interact with the physical world in more ways than communications protocols. They sense the external environment, monitor the operation of the vehicle, and control the operation of the vehicle, as well as cooperating with adjacent vehicles and obtaining information from the infrastructure. Each Interaction with the physical world has its own failure modes, and may detect infrequent or unexpected events that occur in the physical world. For instance, a communications channel may lose messages, while vehicle sensors may generate inaccurate measurements or may detect unexpected obstacles on a roadway.
2) Many automotive applications, are time critical. The Internet is a ”best effort” network, and there arent any guarantees on the time that it takes to deliver a message. In cooperative driving scenario, when a message is delivered to multiple agents, they may react to the message at different times, resulting in many possible sequences of events. All of the sequences must result in safe operations.
3) More participants cooperate in intelligent vehicle applications than in most communications applications that have been verified. The platoon in CACC systems usually include more than three vehicles [2]. The merge protocol described in the next section involves three cooperating vehicles.

Intelligent driving systems interact with the physical world in several ways. We organize the interactions into separate stacks, so that we can isolate the hardware that controls or monitors the physical world from the software that uses the information or issues commands, as shown in Fig.1. A precursor of this architecture is described in [19].

The arrows represent service provide/use relationship between module. For example, in each stack there are arrows pointing upward between layers, which indicates that module(s) in a lower layer provides services to the module(s) that resides one layer higher than it. To state it in another way, module(s) in a higher layer uses on the services provided by the module(s) one layer beneath them.

Besides providing services to the layer directly above, modules in a layer can also provide services to components residing in other stacks. For instance, the coordination layer in the sensor stack requires group communication service from the local communications stack to exchange sensor information with nearby vehicle, and clock synchronization layer also requires communications. A collaborative driving application may also depend on the services in different stacks, as long as the dependencies do not constitute to directed cycles. An example of merge protocol will be described in the next section for illustrating purpose.

Creating multiple stacks in the architecture of intelligent vehicles is motivated by similar reasons that create layers in communication networks, that is, the stacks make it possible to separate the higher level application from interacting with physical domains. By having well defined interfaces between the layers in the stacks we can change a layer without affecting

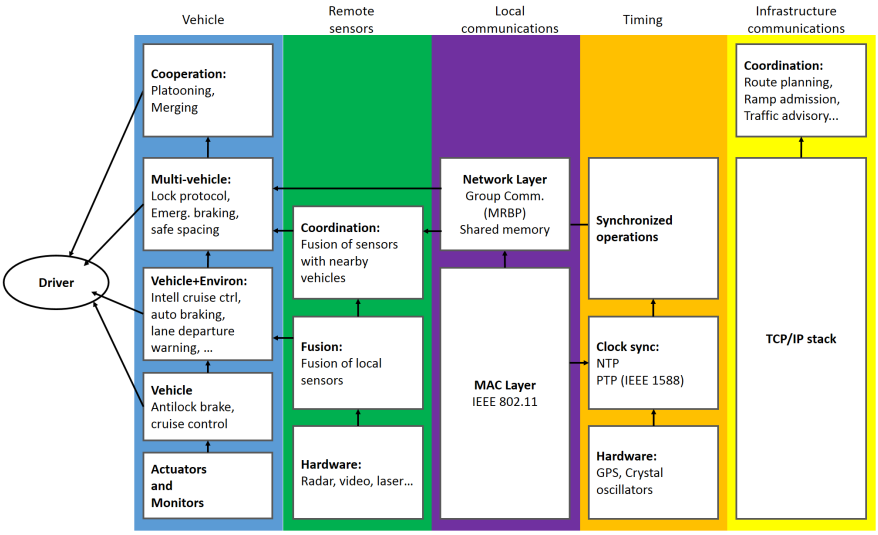

Fig. 1: Multiple stack layered architecture for collaborative driving system

the entire stack. For instance, we can change the data fusion algorithm we use to combine the reading from onboard sensors without changing an intelligent cruise control protocol that uses the distance to reduce the target speed as long as the new algorithms complies with the specified interface.

The layered architecture with services interfaces allows us to apply model checking and conformance testing to individual components rather than the whole system. To check that a component C would not cause an accident for any combination of failures, we assume that the components that provide their service to C have prescribed properties, and we then examine every possible execution sequence of C using model checking techniques [20]. The interfaces between layers also eliminates physical constraints when generating testing sequence [21] for each component. The test sequences can be injected into the components through the well-defined interface, and the outputs can be easily observed from the interface.

Care must be taken if some components rely on the services provided by other stacks than the one it resides on to avoid creating cyclic dependency. For instance, if the synchronization is acquired using MRBP rather than IEEE 802.11, and MRBP uses synchronized operations, then there is a directed cycles MRBP→PTP→synchronized operations→MRBP. We cannot verify nor test individual component on a directed cycles assuming the service that the component relies on, but we have to consider all the components on the cycles.

A. The vehicle stack

The vehicle stack is similar to the stacks in the PATH project [16], the Dolphin project [14], [15], and the Auto21 CDS project [17]. It has been shown that, by using stack, each layer can be designed individually without explicitly considering the implementation details of other layers, and higher level applications are allowed to use the services provided by lower level applications. Complicated applications such as platoon maneuvers can be designed and verified more easily.

The lowest layer directly interacts the vehicle dynamics. It consists of devices that monitor the operation of the vehicle and the actuators that control throttle, brake, and steering of a vehicle.

The applications in the second layer use measurement provided by the monitoring devices and controls the actuators. For instance, antilock braking monitors tire rotation and individually actuates the brakes to avoid skids. Cruise control monitors the vehicle speed and controls the throttle, and possibly the brakes.

The applications residing in the third layer use information from the onboard sensors that detect nearby vehicles and obstables. An automatic braking system monitors the distance to the lead vehicle to actuate the antilock brakes in the event that a rear end collision will occur. An intelligent cruise control system determines the speed set by cruise control in order to maintain safe following distance. A lane departure warning system detects the line markers on the highway and notifies the driver and/or controls the steering wheel when the vehicle deviates from the lane.

The applications in the fourth layer exchange information with adjacent vehicles to better control the operation of a single vehicle. The sensor measurements from neighboring vehicles can be used to further improve the estimates from the onboard sensors. Notification that the lead vehicle is braking allows an earlier response for an emergency braking system. And, measurements on the stopping distance for this vehicle and the vehicles in front of it, under the current road conditions and load in the vehicles, allows for a safer and more accurate calculation of the safe space beteen vehicles. A lock protocol [22] forms mutually-exclusive groups of vehicles that enable further cooperation.

Finally, the fifth layer applications communicate with nearby vehicles to coordinate their operation. Platooning systems create convoys of trucks on highways. Collaborative merge protocol assists driver to change lanes.

B. The communications stacks

There are two communications stacks in the architecture, a local communications stack, and an infrastructure communications stack. The former is used to communicate with nearby vehicles, and the latter is used for communications with the infrastructure such as roadside systems. The TCP/IP stack, used in the Internet, is currently the dominant architecture, and is recommended for the infrastructure stack. It is important to note that the message delivery in the Internet does not have a delay guarantee, while some transportation applications have timing constraints, thus it is required to contruct another overlay on the TCP/IP stack to provide additional service guarantees.

The infrastructure stack is used to gather information for route planning and traffic advisory. It can be used to regulate the admission of ramp traffic when combined with merge protocol. In a roadway drivers tend to alternate at a ramp entry, but on an intelligent highway it is possible to control which vehicles merge next in order to optimize the throughput of the highway.

The lower layer in the local communication stack is the MAC layer, at which there are active research efforts on the standardization of communication protocol. IEEE 802.11p is a variant of IEEE 802.11 protocol that address communications in vehicular environment. IEEE 802.11p and IEEE 1609.4, along with several accompanying standards, are collectively known as the WAVE standard [23]

The higher layer of the local communication stack supports group communications protocols that are specifically designed for collaborative driving applications. The MRBP protocol [24] is a scheduled token passing protocol with bounded message delay. If a group member cannot recover any scheduled message, it stop transmitting which in turns stop everyone in the group from transmitting. With this mechanism, the communication failure can be detected and the merge protocol can safely abort upon such events.

C. The timing stack

The maneuvers performed by intelligent vehicles, whether autonomous or collaborative, are time critical. Timers are often used in time-critical applications, but timers that are set over an unreliable communications channel can result in many possible sequences of execution.

If timers are set by a message that is transmitted over an unreliable communications channels, then there can be different sequences of timer expiry. If two participants in a protocol set identical timers when they receive a message from a third participant, and the first recipient receives the message on the first transmission attempt but the second participant requires two transmission attempts, the first participants timer will fire first. However, if the second participant receives the message on the first attempt, and the first participant receives the message on the second attempt, then the second participants timer will fire first.

Applications that coordinate the operation of independent entities usually use absolute time. Commercialized GPS units and advances in synchronization protocols make it possible to use absolute time to coordinate intelligent vehicles. Our objective is to simplify timed protocols by allowing operations to occur simultaneously, and to reduce the number of sequences that we must consider when verifying the protocol. The timing stack synchronizes the clocks in nearby vehicles. An application in this stack determines when all of the vehicles will perform the coordinated sequence of operations.

Synchronized clocks are used to coordinate actions among a group of agents, and result in an unambiguous sequence of operation. Consider three battalions led by a general and his two lieutenants. The general dispatches messengers to give orders to his lieutenants. The time taken by the messengers to deliver the orders may vary due to road condition. If the general orders each battalion to attack a certain time after the messenger arrives, there will be a range of attack times for the two remote battalions, which result in a range of differences in the time between attacks. There will be six different possible sequences of attack. It is nearly impossible to have all battalions attack simultaneously. However, if the general and lieutenants synchronize their watches, and the general orders them to attack at specific times, that are greater than the time that it takes the messengers to arrive, then the general can specify a unique sequence of attacks, with a specified time between the attacks.

In the case where the messengers could be intercepted by the enemy, which corresponds to message loss in communication channel. We still need a protocol to guarantee that the messages have been received or to initiate an alternative plan when there is uncertainty. This problem is addressed by a communication protocol, such as the MRBP protocol described previously.

We can generally assume that intelligent vehicles are equipped with global positioning system (GPS) receivers as commercial receivers are ubiquitous nowadays. GPS receivers are often thought of as providing navigation information, but it also solves for timing offset with respect to UTC besides positional coordinates [25]. A commercial GPS receiver is able to extract timing information with an accuracy of tens to hundreds of nanoseconds with respect to UTC [25]. Crystal oscillators allow the clock frequency to be maintained accurately when GPS signal is temporarily unavailable [26]. When the GPS signal is not available for an extended period, the synchronization protocols that extend IEEE 1588 (PTP) over wireless links can synchronize clocks in nearby vehicles [27], [28]. These techniques provide clocks that are accurate within a few milliseconds, which is sufficient in most collaborative driving applications.

The timing stack is composed of three layers. The hardware layer has a clock maintained by crystal oscillator that is constantly augmented with timing information obtained by GPS receiver. The clock synchronization layer communicates with nearby vehicles to synchronize their clocks. The third layer maintains a list of times and generates messages when the current time matches those in the list. A protocol that uses synchronized clocks should respond to such messages.

All timed actions should be initiated from this routine, so that there are no timers or clocks in any of the other protocols. By removing time from the other protocols, it is possible to use verification and conformance testing techniques that have been developed for communications protocols, to prove that these protocols will operate properly.

D. The sensor stack

The sensor stack in this architecture is similar to the sensor stack described in [9]. There is a fusion layer that combines the readings from all of the sensors in the vehicle to produce a list, or possibly a mapping, of the vehicles and obstacles surrounding the vehicles. At the hardware level some preprocessing of sensor measurement may be done. The fusion layer allow new sensor to be incorporated without affecting the layers above fusion layer.

The new component in the sensor stack is a coordination layer. This layer communicates between vehicles to guarantee that they are using identical maps. Exchanging measurements with neighboring vehicles or the infrastructure (if available) offers the possibility of cooperative sensing. It also detects possible failures or inaccuracies in sensors. This coordination layer use the MRBP protocol, or other group communication protocols with delay guarantees, to communicate to ensure their measurements are sufficiently up-to-date.

III. Example: A Merge Protocol

The simple merge protocol described in this section, and in references [29], assists a driver who wants to merge between two cars in an adjacent lane as shown in Fig.2. The implementation in the stack architecture is particularly simple because the protocol uses functions provided by other intelligent driving functions, such as intelligent cruise control and intelligent braking, and guarantees provided by the MRBP protocol.

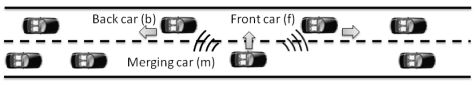

Fig. 2: The merge maneuver

The merge protocol is initiated, for example, when a driver of car m signals his/her intention to merge into the left lane. The merge protocol uses the lock protocol to ensure that car f and car b in that lane are capable of participating in a cooperative merge and will not participating in another maneuver for the duration of the merge. After obtaining the lock, the three cars join a communications group, and car m instructs cars f and b to create a safe gap. Cars f and b use their intelligent cruise control to create a safe gap for the merge, while car m instructs its intelligent cruise control to align itself with the gap that is being created. The driver is notified to change lanes when the gap is large enough and the merging car is aligned with the gap. The maneuver completes

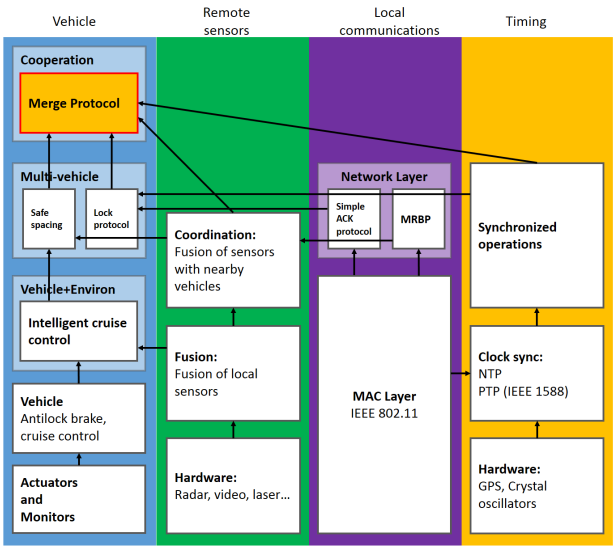

Fig. 3: Multiple stack layered architecture for collaborative driving system

when the driver finishes moving into the gap between the front car and the back car. If an emergency occurs at any time during the merge, such as hard braking in a lead car, the merge protocol is interrupted, the intelligent cruise control, intelligent braking, and lane maintenance systems are reset to the collision avoidance mode, the driver is notified that the merge is aborted, and the merge must be re-initiated at a later time.

The merge protocol uses the services of:

- safe spacing system which instructs the intelligent cruise control system to create gaps or maintain a safe following distance, rather than directly controlling the throttle and brakes.
- coordinated sensor component that provides an environmental mapping of the surroundings that is consistent to those used by participating vehicles, rather than processing all of the sensor readings from each participating vehicle.
- lock protocol to guarantee a consistent cooperating group during the merge.
- synchronized operations component to guarantee that all of the participants perform actions in a correct and timely manner, rather than maintaining its own clock or timers.

In the stack architecture, the merge protocol relies on the services provided by other components. The merge protocol and the services it depends on are summarized in Fig.3. The infrastructure communication stack is omitted because the merge protocol does not require any roadside systems. The arrows represents dependency relationship between components. Note that Fig.3 is in effect a module dependency graph, with arrows being the edges and components being the vertices.

It should be pointed out that Fig.3 does not contain any directed cycles. The acyclicity of the dependency graph implies that we can verify and test each component separately.

Suppose we are to verify that the merge protocol is safe against any combination of failures, we assume that the components on which merge protocol depends, namely the lock protocol, the safe spacing function, the sensor coordination function, and the synchronized operations component, provide their services and have their own failure modes. We specify the merge protocol with appropriate model such as finite automata, and examine every possible execution sequence with model checking technique for finite automata. With the merge protocol verified, we can generate test sequence for the merge protocol to determine if a merge protocol implementation is correct.

IV. Conclusion

A multiple stack architecture for intelligent vehicles is presented in this work. The layered structure in each stacks makes it possible for an system engineer to develop collaborative driving applications without being familiar with the details of physical interactions such as the feedback rules that govern the vehicle control, or the sensor fusion algorithms that produce mapping of objects in the surrounding environment. With multiple stacks, which extend the modular approaches in other architectures, an application can interact with physical world in different ways with sufficient abstraction. The introduction of timing stack makes it possible to coordinate multiple vehicles within demanding timing constraints. The stacks with well-defined interfaces between layers facilitate model checking and conformance testing of individual layers, as illustrated by the merge protocol example. Model checking and conformance testing are of particular importance since intelligent vehicles, unlike communication systems, involve human drivers, thus the safety of the systems is a major concern.

References

[1] http://www.sartre-project.eu/, "The SARTRE project," 2013.

[2] E. van Nunen, R. Kwakkernaat, J. Ploeg, and B. Netten, "Cooperative competition for future mobility," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 1018–1025, Sept. 2012.

[3] A. Geiger, M. Lauer, F. Moosmann, B. Ranft, H. Rapp, C. Stiller, and J. Ziegler, "Team AnnieWAY's entry to the 2011 grand cooperative driving challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1008–1017, 2012.

[4] L. Guvenc, I. Uygan, K. Kahraman, R. Karaahmetoglu, I. Altay, M. Senturk, M. Emirler, A. Karci, B. Guvenc, E. Altug, M. Turan, O. Tas, E. Bozkurt, U. zgner, K. Redmill, A. Kurt, and B. Efendioglu, "Cooperative adaptive cruise control implementation of team mekar at the grand cooperative driving challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1062–1074, 2012.

[5] R. Kianfar, B. Augusto, A. Ebadighajari, U. Hakeem, J. Nilsson, A. Raza, R. Tabar, N. Irukulapati, C. Englund, P. Falcone, S. Papanastasiou, L. Svensson, and H. Wymeersch, "Design and experimental validation of a cooperative driving system in the grand cooperative driving challenge," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 994–1007, 2012.

[6] K. Lidstrom, K. Sjoberg, U. Holmberg, J. Andersson, F. Bergh, M. Bjade, and S. Mak, "A modular CACC system integration and design," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1050–1061, 2012.

[7] J. Martensson, A. Alam, S. Behere, M. Khan, J. Kjellberg, K.-Y. Liang, H. Pettersson, and D. Sundman, "The development of a cooperative heavy-duty vehicle for the GCDC 2011: Team scoop," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 3, pp. 1033–1049, 2012.

[8] S. Behere, M. Trngren, and D.-J. Chen, "A reference architecture for cooperative driving," *Journal of Systems Architecture*, 2013.

[9] M. Aeberhard, S. Schlichtharle, N. Kaempchen, and T. Bertram, "Track-to-track fusion with asynchronous sensors using information matrix fusion for surround environment perception," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1717–1726, 2012.

[10] E. Bertolazzi, F. Biral, M. Da Lio, A. Saroldi, and F. Tango, "Supporting drivers in keeping safe speed and safe distance: The SASPENCE subproject within the european framework programme 6 integrating project PReVENT," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 525–538, 2010.

[11] A. Amditis, E. Bertolazzi, M. Bimpas, F. Biral, P. Bosetti, M. Da Lio, L. Danielsson, A. Gallione, H. Lind, A. Saroldi, and A. Sjgren, "A holistic approach to the integration of safety applications: The INSAFES subproject within the european framework programme 6 integrating project PReVENT," *IEEE Transactions on Intelligent Transportation Systems*, vol. 11, no. 3, pp. 554–566, 2010.

[12] P. Varaiya and S. Shladover, "Sketch of an IVHS systems architecture," in *Vehicle Navigation and Information Systems Conference, 1991*, vol. 2, pp. 909–922, 1991.

[13] R. Horowitz and P. Varaiya, "Control design of an automated highway system," *Proceedings of the IEEE*, vol. 88, no. 7, pp. 913–925, 2000.

[14] S. Kato, S. Tsugawa, K. Tokuda, T. Matsui, and H. Fujii, "Vehicle control algorithms for cooperative driving with automated vehicles and intervehicle communications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 3, no. 3, pp. 155–161, 2002.

[15] S. Tsugawa, S. Kato, T. Matsui, H. Naganawa, and H. Fujii, "An architecture for cooperative driving of automated vehicles," in *2000 IEEE Intelligent Transportation Systems, 2000. Proceedings*, pp. 422–427, 2000.

[16] http://www.path.berkeley.edu/, "California PATH," 2013.

[17] S. Halle, J. Laumonier, and B. Chaib-draa, "A decentralized approach to collaborative driving coordination," in *The 7th International IEEE Conference on Intelligent Transportation Systems, 2004. Proceedings*, pp. 453–458, 2004.

[18] http://www.haveit-eu.org/, "HAVEit," 2013.

[19] S.-p. Lin and N. Maxemchuk, "An architecture for collaborative driving systems," in *2012 20th IEEE International Conference on Network Protocols (ICNP)*, pp. 1–2, Oct. 2012.

[20] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, and P. Schnoebelen, *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer Publishing Company, Incorporated, 1st ed., 2010.

[21] D. Lee and M. Yannakakis, "Principles and methods of testing finite state machines-a survey," *Proceedings of the IEEE*, vol. 84, pp. 1090 –1123, Aug. 1996.

[22] Shou-pon Lin and Nicholas F. Maxemchuk, "The fail safe operation of collaborative driving systems," *Journal of Intelligent Transportation Systems, to be published*.

[23] R. Uzcategui and G. Acosta-Marum, "Wave: A tutorial," *IEEE Communications Magazine*, vol. 47, pp. 126 –133, May 2009.

[24] N. Maxemchuk, P. Tientrakool, and T. Willke, "Reliable neighborcast," *Vehicular Technology, IEEE Transactions on*, vol. 56, pp. 3278 –3288, Nov. 2007.

[25] Lombardi, Michael A., Lisa M. Nelson, Andrew N. Novick, and Victor S. Zhang, "Time and frequency measurements using the global positioning system," *Cal Lab: International Journal of Metrology*, vol. 8, no. 3, pp. 26–33, 2001.

[26] J. Davis and J. Furlong, "Report on the study to determine the suitability of GPS disciplined oscillators as time and frequency standards traceable to the UK national time scale UTC(NPL)," 1997.

[27] J. C. Eidson and K. Lee, "Sharing a common sense of time," *IEEE Instrumentation & Measurement Magazine*, vol. 6, pp. 26– 32, Mar. 2003.

[28] H. Abubakari and S. Sastry, "IEEE 1588 style synchronization over wireless link," in *IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication, 2008. ISPCS 2008*, pp. 127 –130, Sept. 2008.

[29] B. Kim and N. Maxemchuk, "A safe driver assisted merge protocol," in *Systems Conference (SysCon), 2012 IEEE International*, pp. 1–4, Mar. 2012.