Applying architectural hybridization in networked embedded systems

A. Casimiro, J. Rufino, L. Marques, M. Calha, and P. Veríssimo

FC/UL*
{casim,ruf,lmarques,mjc,pjv}@di.fc.ul.pt

Abstract

Building distributed embedded systems in wireless and mobile environments is more challenging than if fixed network infrastructures can be used. One of the main issues is the increased uncertainty and lack of reliability caused by interferences and fading in the communication, dynamic topologies, and so on.

When predictability is an important requirement, then the uncertainties created by wireless networks become a major concern. The problem may be even more stringent if some safety critical requirements are also involved.

In this paper we discuss the use of hybrid models and architectural hybridization as one of the possible alternatives to deal with the intrinsic uncertainties of wireless and mobile environments in the design of distributed embedded systems. In particular, we consider the case of safety-critical applications in the automotive domain, which must always operate correctly in spite of the existing uncertainties. We provide the guidelines and a generic architecture for the development of these applications in the considered hybrid systems. We also refer to interface issues and describe a programming model that is "hybridization-aware". Finally, we illustrate the ideas and the approach presented in the paper using a practical application example.

1 Introduction

Over the last decade we have witnessed an explosive use of wireless technologies to support various kinds of applications.

Unfortunately, when considering real-time systems, or systems that have at least some properties whose correctness depends on the timely and reliable communication, the communication delay uncertainty and unreliability characteristic of wireless networks becomes a problem. It is not possible ignore uncertainty and simply wait until a message arrives, hoping it will arrive soon enough.

* Faculdade de Ciências da Universidade de Lisboa. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by FCT through the Multiannual Funding and the CMU-Portugal Programs.

Our approach to address this problem is considering a hybrid system model, in which a part of the system is asynchronous, namely the part that encompasses the wireless networks and the related computational subsystems, and another part that is always timely, with well defined interfaces to the asynchronous subsystem. In this paper we discuss applicability aspects of this hybrid model, considering in particular safety-critical applications in the automotive domain.

In vehicles, safety-critical functions related to automatic speed and steering control are implemented using real-time design approaches, with dedicated controllers that are connected to car sensors and actuators through predictable networks. Despite all the advances in wireless communication technologies, relying on wireless networks to collect information from external sources and using this information in the safety-critical control processes, seems to be too risky. We argue that this may be possible if a hybrid system model and architecture are used. The advantage is the following: with the additional information it may be possible to improve some quality parameters of the control functions, possibly optimizing speed curves and fuel consumption or even improving the overall safety parameters.

One fundamental aspect to make the approach viable is to devise appropriate interfaces between the different parts of the architecture. On the other hand, special care must be taken when programming safety-critical applications, as we illustrate by providing the general principles of a "hybridization-aware" programming model.

The presented ideas and principles have been explored in the HIDENETS European project [9], in which a proof-of-concept prototype platooning application has been developed. We use this example to briefly exemplify the kind of benefits that may be achieved when using a hybrid system and architecture to build a networked embedded system.

The paper is structured as follows. Some related work is address in the next section. Then, Section 3 motivates the idea of using hybrid distributed system models and highlights their main advantages. In Section 4 we discuss the applicability of the model in the automotive context and in Section 5 address interface issues and introduce the hybrid-aware programming model. The platooning example case is then provided in Section 6 and we end the paper with some conclusions and future prospects.

2 Related work

The availability of varied and increasingly better technologies for wireless communication explains the pervasiveness of these networks in our everyday life. In the area of vehicular applications, new standards like the one being developed by IEEE 802.11p Task Group for Wireless Access in Vehicular Environments (WAVE) will probably become the basis on many future applications.

The 802.11p standard provides a set of seven different logical communication channels among which one is a special dedicated control channel which specifically aims at allowing some more critical vehicular applications to be developed [1]. In fact, improving the baseline technologies and standards in one of the ways to be able to implement safety-critical systems that operate over wireless networks. And there is a large body of research concerned with studying and proposing solutions to deal with the reliability and temporal uncertainties of wireless communication.

A line of research consists in devising new protocols for the MAC level using specific support at the physical level (e.g., Dedicated Short-Range Communications, DSRC) [17] or adopting decentralized coordination techniques, such as using rotating tokens [6]. In fact, the possibility of characterizing communication delays with a reasonable degree of confidence is sufficient for a number of applications that provide safety-related information to the driver, for instance to avoid collisions [5,14]. However, these applications are not meant to autonomously control the vehicles and therefore the involved criticality levels are just moderate. In general, and in spite of all improvements, we are still a few steps away of ensuring the levels of reliability and timeliness that are required for the most critical systems.

A recent approach that indeed aims at dealing with safety requirements and allow autonomous control of vehicles in wireless and mobile environments is proposed in [3]. The approach relies on the cooperation and coordination between involved entities, and defines a coordination model that builds on a real-time communication model designated as the Space Elastic model [2]. The Space Elastic model is actually defined to represent the temporal uncertainties associated real wireless communication environments. The work presented in [13] also addresses the coordination of automated vehicles in platoons. In particular, it focuses on the feasibility of coordination scenarios where vehicles are able to communicate with their closest neighbors. The authors argue that in these scenarios, communication between a vehicle and its leader/follower is possible, as supported by simulation results presented in [7]. In contrast with these works, we consider a hybrid system model, which accommodates both the asynchrony of the wireless environments and the predictable behavior of the embedded control systems and local networks.

In the area of wireless sensor networks efforts have also been made in devising architectures and protocols to address the temporal requirements of applications. One of the first examples is the RAP architecture [11], which defines query and event services associated to new network scheduling policies, with the objective of lowering deadline miss ratios. More recent examples include VigilNet, for real-time target tracking [8] in large-scale sensor networks, and TBDS [10], a mechanism for node synchronization in cluster-tree wireless sensor networks. Our focus is at a higher conceptual level, abstracting from the specific protocols, network topologies and wireless technologies that are used.

3 Hybrid system models

Classical distributed system models range from purely asynchronous to fully synchronous, assume different failure models, from crash to Byzantine. But independently of the particular synchrony or failure model that is assumed, they are typically homogeneous, meaning that the assumed properties apply to the entire system, and do not change over time. However, in many real systems and environment, we observe that synchrony or failure modes are not homogeneous: they vary with time or with the part of the system being considered.

Therefore, in the last few years we have been exploring the possibility of using hybrid distributed system model approaches, in which different parts of the system have different sets of properties (e.g. synchronism [16] or security [4]). Using hybrid models has a number of advantages when compared to approaches based on homogeneous models. The main advantages include more expressiveness with respect to reality, the provision of a sound theoretical basis for crystal-clear proofs of correctness, the possibility of being naturally supported by hybrid architectures and, finally, the possibility of enabling concepts for building totally new algorithms.

One example of a hybrid distributed system model is the Wormholes model [15]. In essence, this model describes systems in which it is possible to identify a subsystem that presents exceptional properties allowing overcoming fundamental limitations of overall system if seen as a whole. For instance, a distributed system in which nodes are connected by a regular asynchronous network, but in which there is also a separate real-time network connecting some synchronization subsystem in each node, can be well described by the Wormholes model. Another very simple example, in which the wormhole subsystem is only local to a node, is a system equipped with a watchdog. Despite the possible asynchrony of the overall system, the watchdog is synchronous and always resets the system in a timely manner whenever necessary.

We must note that designing systems based on the Wormholes model is not just a matter of assuming that uncertainty is not ubiquitous or does not last forever. The design philosophy also builds on the principle that predictability must be achieved in a proactive manner, that is, the system must be built in order to make predictability happen at the right time and right place.

4 Application in automotive context

The wormhole concept can in fact be instantiated in different ways, and here we discuss the possible application of the concept to car systems. Therefore, we first provide an overview of system components that are found in modern cars, and then we explain how an hybrid architecture can be projected over these systems.

4.1 In-vehicle components

Modern cars include a wide set of functions to be performed by electronics and microcontrollers complementing and in many cases totally replacing the traditional mechanical and/or hydraulic mechanisms. These functions include both hardware and software components and are usually structured around Electronic Control Units (ECUs), using the terminology of the automotive industry, which are subsystems composed of a microcontroller complemented with an appropriate set of sensors and actuators.

The functions being replaced by these components are quite diverse and aim to assist the driver in the control of the vehicle. They range from critical functions associated to the control of the power train (e.g. engine and transmission related functions), traction (e.g. driving torque), steering or braking, to less critical ones to control the different devices in the body of the vehicle, such as lights, wipers, doors and windows, seats, climate systems, just to name a few. Recently, a slightly different set of functions is also being incorporated. They are related to information, communication and entertainment (e.g. navigation systems, radio, audio and video, multimedia, integrated cellular phones, etc).

The implementation of these functions is supported in specialized ECUs. However, many of these functions are distributed along the car infrastructure. Thus, there is a need for those functions to be distributed over several ECUs that exchange information and communicate through in-vehicle networking. Furthermore, there may be required to exchange information between ECUs implementing different functions. For example, the vehicle speed obtained from a wheel rotation sensor may be required for gearbox control or for the control of an active suspension subsystem, but it may also be useful for other subsystems.

Given the different functional domains have different requirements in terms of safety, timeliness and performance guarantees, the interconnection of the different ECUs is structured along several networks, classified according to their available bandwidth and function. There are four classes of operation, including one (Class C) with strict requirements in terms of dependability and timeliness, and another (Class D) for high speed data exchanges such as those required for mobile multimedia applications.

The combination of the functions typically provided by each one of those four networking classes involves network interconnection through gateways, as illustrated in Figure 1.

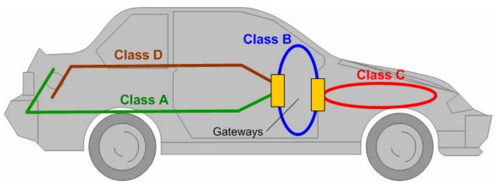

Fig. 1. Typical In-Vehicle Networking.

The in-vehicle ECUs provide support for the different functions implemented in nowadays cars. Each ECU is composed of a computing platform where the ECU software is executed. The computing platform is typically complemented with some specific hardware, a set of sensors, for gathering data from the system under control and a set of actuators which allows to act over the given car subsystem. The support of drive by wire functions integrating a set of sensors (e.g. proximity sensor) and actuators (e.g. speed and brake control) are just one example with relevance for the platooning application that we refer in Section 6.

Others ECUs may exhibit a slightly different architecture because they are intended to support different functions. One example is illustrated in Figure 2, intended to support the integration of infotainment functions. In this case, the architecture of the computing platform is designed to interface and to integrate the operation of multiple gadgets (radio, cellular phone) and technologies.

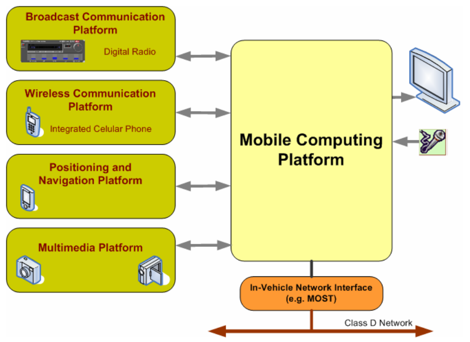
Broadcast Communication Platform
Digital Radio
Wireless Communication Platform
Integrated Celular Phone
Positioning and Navigation Platform
Multimedia Platform
Mobile Computing Platform
In-Vehicle Network Interface (e.g. MOST)
Class D Network

Fig. 2. Example of In-Vehicle Infotainment Functions.

4.2 Architectural hybridization in vehicles

Given the description provided above, it is clear that there is a separation between what may be called a general computing platform, able to run general purpose local and distributed applications connected through wireless networks, and embedded systems dedicated to the execution of specific car functions. Interestingly, there exist gateways between these different subsystems, which allow for information to flow across their boundaries. For example, the information provided by a proximity sensor in the car electronic subsystem may be highly relevant for a driver warning application running in the general computing platform. However, sending information from the general purpose system to a critical control system is not so trivial, and as far as we know is typically avoided.

We argue that in this context it is interesting and useful to apply the wormholes hybrid model in order to explicitly assume the existence of a general (payload) system, asynchronous, but in which complex applications can be executed without special concern for timeliness, and a wormhole subsystem, which is timely, reliable, and in which it is possible to execute critical functions to support interactions with the payload system. The wormhole must provide at least one Timely Timing Failure Detection (TTFD) service, available to payload applications, to allow the detection of timing failures in the payload part or in the payload-wormhole interactions. This TTFD service must also be able to timely trigger the execution of fault handling operations for safety purposes. These handlers have to be implemented as part of the wormhole subsystem and will necessarily be application dependant.

With these settings it is possible to deal with information flows from the payload side to the critical subsystems, thus allowing developing applications that run in the general computing platform, which are able to exploit the availability of wireless communication, and which are still able to control critical systems in a safe way. Of course that in order for this to be possible, the applications must be programmed in a way that is "hybridization-aware", explicitly using the TTFD service provided by the wormhole subsystem and being complemented by safety functions that must be executed on predictable subsystems. In the following section we describe the architectural components that constitute the hybrid system, focusing on these interfacing and programming issues.

5 Designing applications in hybrid systems

5.1 Generic architecture

In the proposed approach for the design of safety-critical applications in hybrid systems, the system architecture must necessarily encompass the two realms of operation: the asynchronous payload and the synchronous real-time subsystem, as illustrated in Figure 3.

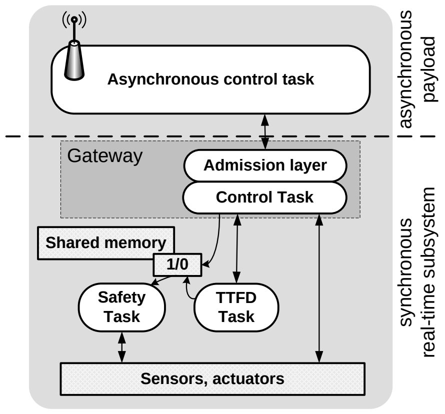

Fig. 3. System architecture for asynchronous control.

A so called asynchronous control task executes in the payload part, possibly interacting with external systems through wireless or other non real-time networks. Interestingly, this asynchronous control task can perform complex calculations using varied data sources in order to achieve improved control decisions. On the real-time (or wormhole) part of the system, several tasks will be executed in a predictable way, always satisfying (by design) any required deadline.

In order to exploit the synchronism properties of the wormhole part of the system, the interface to access wormhole services must be carefully designed. The solution requires the definition of a wormhole gateway, much like the gateways between the different network classes that are defined in car architectures.

This wormhole gateway includes an admission layer, which restricts the patterns of service requests as a means to secure the synchrony properties of the wormhole subsystem (we assume that the payload system can be computationally powerful, and the number of requests sent to the wormhole subsystem is not bounded a priori). Some service requests may be delayed, rejected or simply not executed because of lack of resources. This behavior is admissible because from the perspective of the asynchronous system, no guarantees are given anyway.

Several interface functions may be made available, some of which specifically related to the application being implemented (e.g., functions for control commands to be sent to actuators or ECUs, and for sensor information to be read). At least it is necessary to provide a set of functions to access and use the TTFD service. The role of the TTFD service is fundamental: in simple terms, it is a kind of "enhanced watchdog" programmed by the payload application, and it works as a switching device that gives control to the wormhole part when the payload becomes untimely. A more detailed description of the TTFD service and how it must be used is provided in Section 5.2.

A control task is defined within the gateway, which will implement the specific functions and will also interact with the TTFD service, forwarding start and stop commands received from the payload. The task may also decide whether an actuation command can effectively be applied or not, depending on the timeliness status of the payload.

A safety task must also be in place, which will be in charge of ensuring a safe control whenever the asynchronous control task is prevented from taking over the control. This safe control will be done using only the locally available information, collected from local sensors. This control task can be designed to keep the system in a safe state, but this will be a pessimistic control in the sense that it will be based only on local information. The effective activation of this task is controlled by the TTFD service, using a status variable in a shared-memory structure, or some equivalent implementation. Quite naturally, each specific application must have its own associated safety task. Therefore, although the architecture is generically applicable to safety-critical applications in hybrid systems, some components must be instantiated on a case-by-case basis.

In Figure 3 we also represent the sensors and actuators, which are necessarily part of the real-time subsystem.

5.2 Using the TTFD service

A fundamental idea underlying the approach is to use the TTFD service to monitor the timeliness of a payload process. The TTFD service provides the following functions: `startTFD`, `stopTFD` and `restartTFD`. The `startTFD` function specifies a start instant to start the monitoring of a timed action and a maximum duration for that action. The handling functions that are executed when a timing failure is detected must be programmed a priori as part of the wormhole. A specific handler may be specified when starting a timing failure monitoring activity. The `stopTFD` function stops the on-going monitoring activity, returning an indication of whether the timed execution was timely terminated or not. The `restartTFD` function allows the atomic execution of a `stopTFD` request followed by a `startTFD` request.

Before starting a timed execution, the TTFD service is requested to monitor this execution and a deadline is provided. If the execution is timely, then the TTFD monitoring activity will be stopped before the deadline. Otherwise, when the deadline is reached the TTFD service will raise a timing fault condition (possibly a boolean variable in the shared memory, as shown in Figure 3).

From a programmers view perspective, and considering that we are concerned with the development of asynchronous control applications, there are two important issues to deal with: a) determining the deadline values provided to the TTFD service; b) use the available functions in a way that ensures that either the execution is timely (thus allowing control commands to be issued) or else a timing failure is always detected (and safety handler can at least be executed).

The deadline must be such that the application is likely able to perform the necessary computations within that deadline. In control, there is a trade-off between reducing the duration of the control cycle and the risk of not being able to compute a control decision within the allowed period. On the other hand, specifying large deadlines will have a negative influence on the quality of control. The other restriction for the deadline is determined by safety rules and by the characteristics of a fail-safe real-time control task that will be activated when the deadline is missed. The deadline must be such that the fail-safe control task, when activated, is still able to fulfill the safety requirements.

The second issue concerns the way in which interactions between the payload and the wormhole must be programmed, which we discuss in what follows.

5.3 Payload-wormhole interactions

In the proposed architecture, TTFD requests are in fact directed to the control task, along with actuation commands. That is, when the asynchronous control task sends an actuation command, it is implicitly finishing an on-going timed action, and it must explicitly start a new one by specifying a deadline for the next actuation instant.

The idea is the following: when an actuation command is sent from the payload to the wormhole, it is supposed to be sent before a previously specified deadline. Therefore, when the command is received by the control task, this task first has to stop the on-going TTFD monitoring activity. Depending on the returned result, the control task will know if the actuation command is timely (and hence can be safely used and applied to actuators) or if it is untimely (in which case, it will just be discarded). In the latter case, the TTFD must already have triggered the failure indication. In fact, this indicator is used by the safety task to decide if it should indeed become active and take over the control of the system. As soon as a timing failure occurs, the indicator is activated, and the safety task will take over the next time it is released. This means that a late command received from the payload will be ignored, and it will be ensured that the safety task will already be in control.

In a steady state, the asynchronous control task will be continuously sending commands to the the wormhole, timely stopping the on-going TTFD monitoring activity, atomically restarting the TTFD for a future point in time (the next actuation deadline), and applying the control command.

6 Platooning example

Let us consider the example of a platooning application, in which the objective is to achieve a better platoon behavior (keep cars close together at the maximum possible speed), using not only the information available from local car sensors, but also other information collected from other cars or from fixed stations along the road. The hybrid architecture will encompass an asynchronous platooning control task running in some on-board general purpose computer, processing all the available information and producing control decisions that must be sent to the vehicle ECUs.

The information exchanged between vehicles (through the wireless network) includes time, position and speed values. This information is relevant for the platooning control application, since it will know where to position each other car in a virtual map and hence what to do regarding the own car speed. Clocks are assumed to be synchronized through GPS receivers and accelerations (positive and negative) are bounded. In this way, worst case scenarios can be considered when determining the actuation commands.

Every car in the platoon periodically retrieves the relevant information from local sensors (through the wormhole interface), disseminates this information, and hopefully receives the same information from the other cars. In the platooning application case, failures in the communication will not have serious consequences. In fact, if a car does not receive information from the preceding car, it will use old information and will "see" that car closer than it is in reality. The consequence is that the follower car will stop, even if not necessary.

Given the periodic nature of the payload message exchanges, the asynchronous control tasks may become aware of lost or very delayed messages (if timeouts are used) and refrain from sending actuation commands to the wormhole. In this case, or if the payload becomes to slow (remember that this is a general purpose computing environment), the actuation commands expected by the wormhole will not be received or will arrive too late, and meanwhile the safety task is activated to take over the control of the car.

From the platooning application perspective, the proposed implementation provides some clear improvements over a traditional implementation. The latter is pessimistic in the sense that it must ensure larger safety distances between cars, in particular at high speeds, since no information is available about the surrounding environment and in particular about the speed of the preceding car. On the other hand, in the prototype we implemented it is possible to observe that independently of the platoon speed, the distance between every two cars is kept constant because follower cars are able to know the distance to the preceding car, and its speed also.

We implemented a prototype of this platooning application, which was demonstrated using emulators for the physical environment and for the wireless network. Figure 4 illustrates some of the hardware and a graphical view of the platoon in the physically emulated reality. The interested reader can refer to [12], which provides additional details about this demonstration.

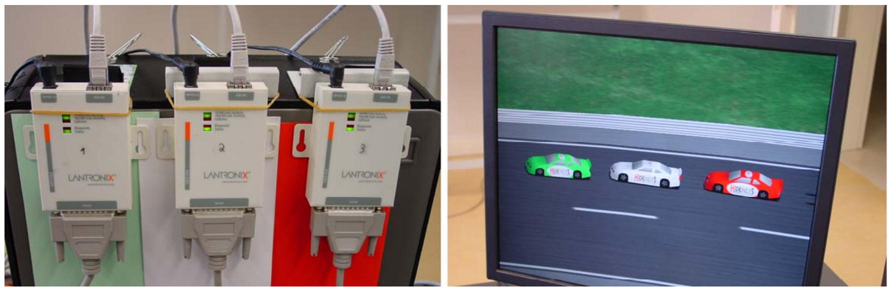

Fig. 4. Platooning demonstration.

7 Conclusions

The possibility of using wireless networks for car-to-car or car-to-infrastructure interactions is very appealing. The availability of multiple sources of information can be used to improve the quality of control functions and implicitly the safety or the fuel consumptions. The problem that we addressed in this paper is concerned with the potential lack of timeliness and with the unreliability of wireless networks, which make it difficult to consider their use when implementing real-time applications or safety-critical systems.

We propose an approach that is based on the use of a hybrid system model and architecture. The general idea is to allow applications to be developed on a general purpose part of the system, typically asynchronous, and provide the means to ensure that safety-critical properties are always secured. Since we focus on applications for the vehicular domain, typically control applications, we first explain why the considered hybrid approach is very reasonable in this context. Then we provide the guidelines for designing asynchronous control applications, explaining in particular how the interactions between the payload and the wormhole subsystems should be programmed.

From the experience we had in the development of the platooning example application and from the observations we made while executing our demonstration system, we conclude that the proposed approach constitutes a potentially interesting alternative for the implementation of optimized safety-critical systems in wireless environments.

References

1. IEEE P802.11p/D3.0, Part 11: Wireless LAN Medium Access Contrl (MAC) and Physical Layer (PHY) Specifications: Amendment: Wireless Access in Vehicular Environments (WAVE), July 2007. Draft 3.0.

2. M. Bouroche, B. Hughes, and V. Cahill. Building reliable mobile applications with space-elastic adaptation. In *WOWMOM '06: Proceedings of the 2006 International Symposium on on World of Wireless, Mobile and Multimedia Networks*, pages 627–632, Washington, DC, USA, 2006. IEEE Computer Society.
3. M. Bouroche, B. Hughes, and V. Cahill. Real-time coordination of autonomous vehicles. pages 1232–1239, Sept. 2006.
4. M. Correia, P. Veríssimo, and N. F. Neves. The design of a COTS real-time distributed security kernel. In *Proc. of the Fourth European Dependable Computing Conference*, Toulouse, France, October 2002.
5. Tamer Elbatt, Siddhartha K. Goel, Gavin Holland, Hariharan Krishnan, and Jayendra Parikh. Cooperative collision warning using dedicated short range wireless communications. In *VANET '06: Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, pages 1–9, New York, NY, USA, 2006. ACM.
6. M. Ergen, Duke Lee, Raja Sengupta, and P. Varaiya. Wtrp - wireless token ring protocol. *Vehicular Technology, IEEE Transactions on*, 53(6):1863–1881, 2004.
7. S. Halle, J. Laumonier, and B. Chaib-Draa. A decentralized approach to collaborative driving coordination. pages 453–458, Oct. 2004.
8. Tian He, Pascal Vicaire, Ting Yan, Liqian Luo, Lin Gu, Gang Zhou, Radu Stoleru, Qing Cao, John A. Stankovic, and Tarek Abdelzaher. Achieving real-time target tracking usingwireless sensor networks. In *RTAS '06: Proceedings of the 12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 37–48, Washington, DC, USA, 2006. IEEE Computer Society.
9. HIDENETS. Web site: http://www.hidenets.aau.dk/.
10. Anis Koubâa, André Cunha, Mário Alves, and Eduardo Tovar. Tdbs: a time division beacon scheduling mechanism for zigbee cluster-tree wireless sensor networks. *Real-Time Syst.*, 40(3):321–354, 2008.
11. Chenyang Lu, Brian M. Blum, Tarek F. Abdelzaher, John A. Stankovic, and Tian He. Rap: A real-time communication architecture for large-scale wireless sensor networks. In *Eighth IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 55–66, Washington, DC, USA, 2002. IEEE Computer Society.
12. L. Marques, A. Casimiro, and M. Calha. Design and development of a proof-of-concept platooning application using the hidenets architecture. In *DSN '09: Proceedings of the International Conference on Dependable Systems and Networks (DSN'09), to appear*, 2009.
13. F. Michaud, P. Lepage, P. Frenette, D. Letourneau, and N. Gaubert. Coordinated maneuvering of automated vehicles in platoons. *Intelligent Transportation Systems, IEEE Transactions on*, 7(4):437–447, Dec. 2006.
14. James A. Misener and Raja Sengupta. Cooperative collision warning: Enabling crash avoidance with wireless. In *12th World Congress on ITS*, New York, NY, USA, November 2005.
15. P. Verissimo. Travelling through wormholes: a new look at distributed systems models. *SIGACT News*, 37(1):66–81, 2006.
16. P. Veríssimo and A. Casimiro. The Timely Computing Base model and architecture. *IEEE Transactions on Computers - Special Issue on Asynchronous Real-Time Systems*, 51(8), August 2002. A preliminary version of this document appeared as Technical Report DI/FCUL TR 99-2, Department of Computer Science, University of Lisboa, Apr 1999.
17. Qing Xu, Tony Mak, Jeff Ko, and Raja Sengupta. Vehicle-to-vehicle safety messaging in dsrc. In *VANET '04: Proceedings of the 1st ACM international workshop on Vehicular ad hoc networks*, pages 19–28, New York, NY, USA, 2004. ACM Press.