Trends in Automotive Communication Systems

NICOLAS NAVET, YEQIONG SONG, FRANÇOISE SIMONOT-LION, AND CÉDRIC WILWERT

Invited Paper

The use of networks for communications between the electronic control units (ECU) of a vehicle in production cars dates from the beginning of the 1990s. The specific requirements of the different car domains have led to the development of a large number of automotive networks such as Local Interconnect Network, J1850, CAN, TTP/C, FlexRay, media-oriented system transport, IDB1394, etc. This paper first introduces the context of in-vehicle embedded systems and, in particular, the requirements imposed on the communication systems. Then, a comprehensive review of the most widely used automotive networks, as well as the emerging ones, is given. Next, the current efforts of the automotive industry on middleware technologies, which may be of great help in mastering the heterogeneity, are reviewed. Finally, we highlight future trends in the development of automotive communication systems.

Keywords*—Car domains, in-vehicle embedded systems, fieldbuses, middlewares (MWs), networks, real-time systems.*

I. Automotive Communication Systems: Characteristics and Constraints

From Point-to-Point to Multiplexed Communications: Since the 1970s, one observes an exponential increase in the number of electronic systems that have gradually replaced those that are purely mechanical or hydraulic. The growing performance and reliability of hardware components and the possibilities brought by software technologies enabled implementing complex functions that improve the comfort of the vehicle's occupant as well as their safety. In particular, one of the main purposes of electronic systems is to assist the driver to control the vehicle through functions related to the steering, traction (i.e., control of the driving torque) or braking such as the antilock braking system (ABS), electronic stability program (ESP), electric power steering (EPS), active suspensions, or engine control. Another reason for using electronic systems is to control devices in the body of a vehicle such as lights, wipers, doors, windows, and, recently, entertainment and communication equipment (e.g., radio, DVD, hands-free phones, navigation systems).

In the early days of automotive electronics, each new function was implemented as a stand-alone electronic control unit (ECU), which is a subsytem composed of a microcontroller and a set of sensors and actuators. This approach quickly proved to be insufficient with the need for functions to be distributed over several ECUs and the need for information exchanges among functions. For example, the vehicle speed estimated by the engine controller or by wheel rotation sensors has to be known in order to adapt the steering effort, to control the suspension, or simply to choose the right wiping speed. In today's luxury cars, up to 2500 signals (i.e., elementary information such as the speed of the vehicle) are exchanged by up to 70 ECUs [1]. Until the beginning of the 1990s, data was exchanged through point-to-point links between ECUs. However this strategy, which required an amount of communication channels of the order of n^2 where n is the number of ECUs (i.e., if each node is interconnected with all the others, the number of links grows in the square of n), was unable to cope with the increasing use of ECUs due to the problems of weight, cost, complexity, and reliability induced by the wires and the connectors. These issues motivated the use of networks where the communications are multiplexed over a shared medium, which consequently required defining rules—protocols—for managing communications and, in particular, for granting bus access. It was mentioned in a 1998 press release (quoted in [2]) that the replacement of a "wiring harness with LANs in the four doors of a BMW reduced the weight by 15 kilograms." In the mid–1980s, the third part supplier Bosch developed the Controller Area Network (CAN), which was first integrated in Mercedes production cars in the early 1990s. Today, it has become the most widely used network in automotive systems and it is estimated [3] that the number of CAN nodes sold per year is currently around 400 million (all application fields). Other communication networks, providing different services, are now being integrated in automotive applications. A description of the major networks is given in Section II.

Manuscript received September 6, 2004; revised March 11, 2005.

N. Navet, Y. Song, and F. Simonot-Lion are with the Loria Laboratory, Vandoeuvre-lés-Nancy 54506, France (e-mail: nicolas.navet@loria.fr; song@loria.fr; simonot@loria.fr).

C. Wilwert is with PSA Peugeot Citröen, La Garenne-Colombes Cedex 92256, France (e-mail: cedric.wilwert@mpsa.com).

Digital Object Identifier 10.1109/JPROC.2005.849725

Car Domains and Their Evolution: As all the functions embedded in cars do not have the same performance or safety needs, different QoSs (e.g., response time, jitter, bandwidth, redundant communication channels for tolerating transmission errors, efficiency of the error detection mechanisms, etc.) are expected from the communication systems. Typically, an in-car embedded system is divided into several functional domains that correspond to different features and constraints [4]. Two of them are concerned specifically with real-time control and safety of the vehicle's behavior: the "powertrain" (i.e., control of engine and transmission) and the "chassis" (i.e., control of suspension, steering, and braking) domains. The third, the "body," mostly implements comfort functions. The "telematics" (i.e., integration of wireless communications, vehicle monitoring systems and location devices), "multimedia," and "human–machine interface" (HMI) domains take advantage of the continuous progress in the field of multimedia and mobile communications. Finally, an emerging domain is concerned with the safety of the occupant.

The main function of the powertrain domain is controlling the engine. It is realized through several complex control laws with sampling periods of a magnitude of some milliseconds (due to the rotation speed of the engine) and implemented in microcontrollers with high computing power. In order to cope with the diversity of critical tasks to be treated, multitasking is required and stringent time constraints are imposed on the scheduling of the tasks. Furthermore, frequent data exchanges with other car domains, such as the chassis (e.g., ESP, ABS) and the body (e.g., dashboard, climate control), are required.

The chassis domain gathers functions such as ABS, ESP, ASC (Automatic Stability Control), 4WD (4 Wheel Drive), which control the chassis components according to steering/braking solicitations and driving conditions (ground surface, wind, etc). Communication requirements for this domain are quite similar to those for the powertrain but, because they have a stronger impact on the vehicle's stability, agility and dynamics, the chassis functions are more critical from a safety standpoint. Furthermore, the "x-by-wire" technology, currently used for avionic systems, is now being introduced to execute steering or braking functions. "X-by-wire" is a generic term referring to the replacement of mechanical or hydraulic systems by fully electrical/electronic ones, which led and still leads to new design methods for developing them safely [5] and, in particular, for mastering the interferences between functions [6]. Chassis and powertrain functions operate mainly as closed-loop control systems and their implementation is moving toward a time-triggered approach [7]–[9], which facilitates composability (i.e., ability to integrate individually developed components) and deterministic real-time behavior of the system.

Dashboard, wipers, lights, doors, windows, seats, mirrors, and climate control are increasingly controlled by software-based systems that make up the "body" domain. This domain is characterized by numerous functions that necessitate many exchanges of small pieces of information among themselves. Not all nodes require a large bandwidth, such as the one offered by CAN; this lead to the design of low-cost networks such as Local Interconnect Network (LIN) and TTP/A (see Section II). On these networks, only one node, termed the master, possesses an accurate clock and drives the communication by polling the other nodes—the slaves—periodically. The mixture of different communication needs inside the body domain lead to a hierarchical network architecture where integrated mechatronic subsystems based on low-cost networks are interconnected through a CAN backbone. The activation of body functions is mainly triggered according to the driver and passengers' solicitation (e.g., opening a window, locking doors, etc).

Telematics functions are becoming more and more numerous: hands-free phones, car radio, CD, DVD, in-car navigation systems, rear seat entertainment, remote vehicle diagnostics, etc. These functions require a lot of data to be exchanged within the vehicle but also with the external world through the use of wireless technology (see, for instance, [10]). Here, the emphasis shifts from messages and tasks subject to stringent deadline constraints to multimedia data streams, bandwidth sharing, multimedia QoS where preserving the integrity (i.e., ensuring that information will not be accidentally or maliciously altered) and confidentiality of information is crucial. HMI aims to provide HMIs that are easy to use and that limit the risk of driver inattention [11].

Electronic-based systems for ensuring the safety of the occupants are increasingly embedded in vehicles. Examples of such systems are impact and rollover sensors, deployment of airbags and belt pretensioners, tire pressure monitoring, or adaptive cruise control (ACC) (in which the car's speed is adjusted to maintain a safe distance from the car ahead). These functions form an emerging domain usually referred to as "active and passive safety."

Different Networks for Different Requirements: The steadily increasing need for bandwidth[1] and the diversification of performance, costs and dependability[2] requirements lead to a diversification of the networks used throughout the car. In 1994, the Society for Automotive Engineers (SAE) defined a classification for automotive communication protocols [13]–[15] based on data transmission speed and functions that are distributed over the network. *Class A* networks have a data rate lower than 10 kb/s and are used to transmit simple control data with low-cost technology. They are mainly integrated in the "body" domain (seat control, door lock, lighting, trunk release, rain sensor, etc.). Examples of class A networks are LIN [16], [17] and TTP/A [18]. *Class B* networks are dedicated to supporting data exchanges between ECUs in order to reduce the number of sensors by sharing information. They operate from 10 to 125 kb/s. The J1850 [19] and low-speed CAN [20] are the main representations of this class. Applications that need high speed real-time communications require *class C* networks

[1]For instance, in [6], the average bandwith needed for the engine and the chassis control is estimated to reach 1500 kb/s in 2008 while it was 765 kb/s in 2004 and 122 kb/s in 1994.

[2]Dependability is usually defined as the ability to deliver a service that can justifiably be trusted; see [12] for more details.

(speed of 125 kb/s–1 Mb/s) or *class D* networks[3] (speed over 1 Mb/s). Class C networks, such as high-speed CAN [21], are used for the powertrain and currently for the chassis domains, while class D networks are devoted to multimedia data (e.g., media-oriented system transport (MOST) [22]) and x-by-wire applications that need predictability and fault tolerance (e.g., TTP/C [23] or FlexRay [24], [25] networks).

It is common, in today's vehicles, that the electronic architecture include four different types of networks interconnected by gateways. For example, the Volvo XC90 [3] embeds up to 40 ECUs interconnected by a LIN bus, a MOST bus, a low-speed CAN, and a high-speed CAN. In the near future, it is likely that a bus dedicated to occupant safety systems (e.g., airbag deployment, crash sensing) such as the "safe-by-wire plus" [26] will be added.

Event Triggered Versus Time Triggered: One of the main objectives of the design step of an in-vehicle embedded system is to ensure a proper execution of the vehicle functions, with a predefined level of safety, in the normal functioning mode but also when some components fail (e.g., reboot of an ECU) or when the environment of the vehicle creates perturbations (e.g., electromagnetic interference (EMI) causing frames to be corrupted). Networks play a central role in maintaining the embedded systems in a "safe" state, since most critical functions are now distributed and need to communicate. Thus, the different communication systems have to be analyzed in regard to this objective; in particular, messages transmitted on the bus must meet their real-time constraints, which mainly consist of bounded response times and bounded jitters.

There are two main paradigms for communications in automotive systems: time triggered and event triggered. Event triggered means that messages are transmitted to signal the occurrence of significant events (e.g., a door has been closed). In this case, the system possesses the ability to take into account, as quickly as possible, any asynchronous events such as an alarm. The communication protocol must define a policy to grant access to the bus in order to avoid collisions; for instance, the strategy used in CAN (see Section II-A1) is to assign a priority to each frame and to give the bus access to the highest priority frame. Event-triggered communication is very efficient in terms of bandwidth usage since only necessary messages are transmitted. Furthermore, the evolution of the system without redesigning existing nodes is generally possible, which is important in the automotive industry where incremental design is a usual practice. However, verifying that temporal constraints are met is not obvious and the detection of node failures is problematic.

When communications are time triggered, frames are transmitted at predetermined points in time, which is well suited for the periodic transmission of messages as required in distributed control loops. Each frame is scheduled for transmission at one predefined interval of time, usually termed a slot, and the schedule repeats itself indefinitely. This medium access strategy is referred to as time-division multiple access (TDMA). As the frame scheduling is statically defined, the temporal behavior is fully predictable; thus, it is easy to check whether the timing constraints expressed on data exchanges are met. Another interesting property of time-triggered protocols is that missing messages are immediately identified; this can serve to detect, in a short and bounded amount of time, nodes that are presumably no longer operational.

The first negative aspect is the inefficiency in terms of network utilization and response times with regard to the transmission of a periodic messages (i.e., messages that are not transmitted in a periodic manner). A second drawback of time-triggered protocols is the lack of flexibility even if different schedules (corresponding to different functioning modes of the application) can be defined and switching from one mode to another is possible at runtime. Finally, the unplanned addition of a new transmitting node on the network induces changes in the message schedule and, thus, necessitateS the update of all other nodes. TTP/C [23] is a purely time-triggered network but there are networks, such as TTCAN [27], FTT-CAN [28], and FlexRay [24], [25], that can support a combination of both time-triggered and event-triggered transmissions. This capability to convey both types of traffic fits in well with the automotive context, since data for control loops as well as alarms and events has to be transmitted.

Several comparisons have been done between event-triggered and time-triggered approaches; the reader can refer to [1], [28], [29] for good starting points.

II. In-Car Embedded Networks

The different performance requirements throughout a vehicle, as well as competition among companies of the automotive industry, have led to the design of a large number of communication networks. The aim of this section is to give a description of the most representative networks for each main domain of utilization.

A. Priority Buses

To ensure at runtime the "freshness"[4] of the exchanged data and the timely delivery of commands to actuators, it is crucial that the Medium Access Control (MAC) protocol is able to ensure bounded response times of frames. An efficient and conceptually simple MAC scheme that possesses this capability is the granting of bus access according to the priority of the messages (the reader can refer to [30] and [31] for how to compute bounds on response times for priority buses). To this end, each message is assigned an identifier, unique to the whole system. This serves two purposes: giving priority for transmission (the lower the numerical value, the greater the priority) and allowing message filtering upon reception. The two main representatives of such "priority buses" are CAN and J1850.

[3]Class D is not formally defined, but it is generally considered that networks over 1 Mb/s belong to class D.

[4]The freshness property is verified if data has been produced recently enough to be safely consumed: the difference between the time when data is used and the last production time must be always smaller than a specified value.

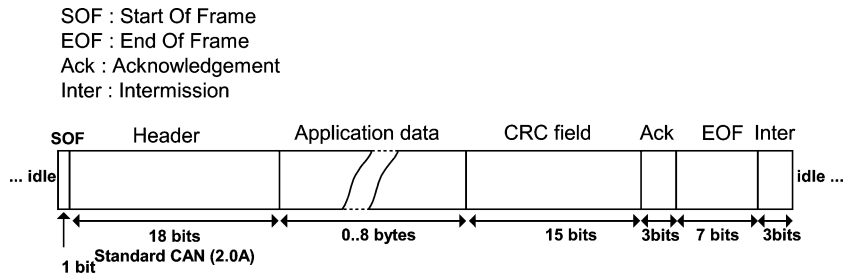

Fig. 1. Format of the CAN 2.0A data frame.

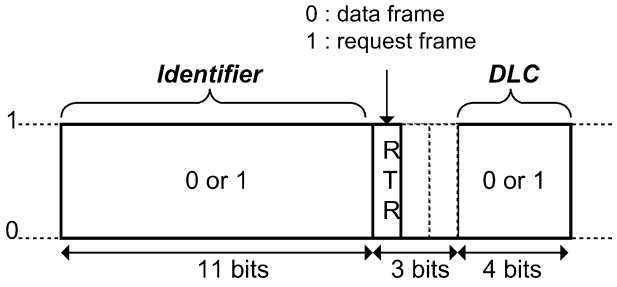

Fig. 2. Format of the header field of the CAN 2.0A data frame.

1) CAN: CAN is without a doubt the most widely used in-vehicle network. It was designed by Bosch in the mid–1980s for multiplexing communication between ECUs in vehicles and thus for decreasing the overall wire harness: length of wires and number of dedicated wires (e.g., the number of wires has been reduced by 40%, from 635 to 370, in the Peugeot 307, which embeds two CAN buses with regard to the nonmultiplexed Peugeot 306 [32]). Furthermore, it allows to share sensors among ECUs.

CAN on a twisted pair of copper wires became an ISO standard in 1994 [20], [21] and is now a *de facto* standard in Europe for data transmission in automotive applications, due to its low cost, its robustness, and the bounded communication delay (see [3], [33]). In today's car, CAN is used as an SAE class C network for real-time control in the powertrain and chassis domains (at 250 or 500 kb/s), but it also serves as an SAE class B network for the electronics in the body domain, usually at a data rate of 125 kb/s.

On CAN, data, possibly segmented in several frames, may be transmitted periodically, aperiodically, or on demand (i.e., client–server paradigm). A CAN frame is labeled by an identifier, transmitted within the frame (see Figs. 1 and 2), whose numerical value determines the frame priority. There are two versions of the CAN protocol differing in the size of the identifier: CAN 2.0A (or "standard CAN") with an 11-b identifier and CAN 2.0B (or "extended CAN") with a 29-b identifier. For in-vehicle communications, only CAN 2.0A is used, since it provides a sufficient number of identifiers (i.e., the number of distinct frames exchanged over one CAN network is lower than 2^{11}).

CAN uses nonreturn-to-zero (NRZ) bit representation with a bit stuffing of length 5. In order not to lose the bit time (i.e., the time between the emission of two successive bits of the same frame), stations need to resynchronize periodically, and this procedure requires edges on the signal. Bit stuffing is an encoding method that enables resynchronization when using NRZ bit representation where the signal level on the bus can remain constant over a longer period (e.g., transmission of 000 000...). Edges are generated into the outgoing bit stream in such a way to avoid the transmission of more than a maximum number of consecutive equal-level bits (five for CAN). The receiver will apply the inverse procedure and destuff the frame. CAN requires the physical layer to implement the logical "and" operator: if at least one node is transmitting the "0" bit level on the bus, then the bus is in that state regardless if other nodes have transmitted the "1" bit level. For this reason, "0" is termed the dominant bit value, while "1" is the recessive bit value.

The standard CAN data frame (CAN 2.0A; see Fig. 1) can contain up to 8 B of data for an overall size of, at most, 135 b, including all the protocol overheads such as the stuff bits. The sections of the frames are:

- The header field (see Fig. 2), which contains the identifier of the frame, the remote transmission request (RTR) bit that distinguishes between data frame (RTR set to zero) and data request frame (RTR set to 1) and the data length code (DLC) used to inform of the number of bytes of the data field.
- The data field, having a maximum length of 8 B.
- The 15-bit cyclic redundancy check (CRC) field, which ensures the integrity of the data transmitted.
- The Acknowledgment field (Ack). On CAN, the acknowledgment scheme solely enables the sender to know that at least one station, but not necessarily the intended recipient, has received the frame correctly.
- The end-of-frame (EOF) field and the intermission frame space, which is the minimum number of bits separating consecutive messages.

Any CAN node may start a transmission when the bus is idle. Possible conflicts are resolved by a priority-based arbitration process, which is said to be nondestructive in the sense that, in case of simultaneous transmissions, the highest priority frame will be sent despite the contention with lower priority frames. The arbitration is determined by the arbitration fields (identifier plus RTR bit) of the contending nodes. An example illustrating CAN arbitration is shown in Fig. 3. If one node transmits a recessive bit on the bus while another transmits a dominant bit, the resulting bus level is dominant due to the AND operator realized by the physical layer. Therefore, the node transmitting a recessive bit will observe a dominant bit on the bus and then will immediately stop transmitting. Since the identifier is transmitted "most significant bit first," the node with the numerically lowest identifier field

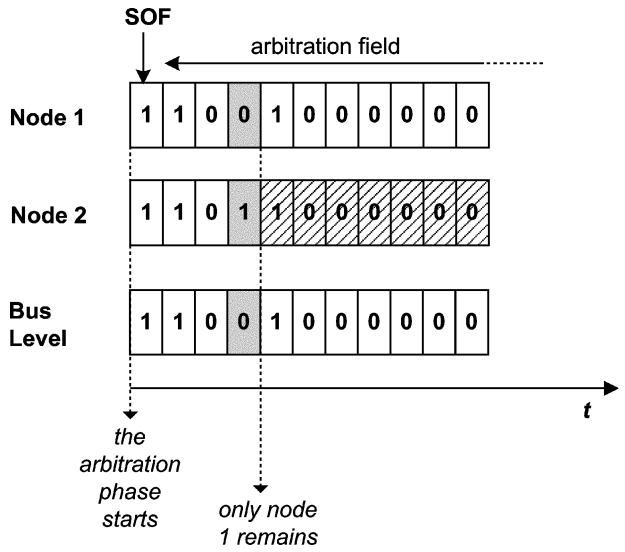

Fig. 3. CAN arbitration phase with two nodes starting transmitting simultaneously. Node 2 detects that a frame with a higher priority than its own is being transmitted when it monitors a level 0 (i.e., dominant level) on the bus while it has sent a bit with a level 1 (i.e., recessive level). Afterwards, Node 2 immediately stops transmitting.

will gain access to the bus. A node that has lost the arbitration will wait until the bus becomes free again before trying to retransmit its frame. CAN arbitration procedure relies on the fact that a sending node monitors the bus while transmitting. The signal must be able to propagate to the most remote node and return back before the bit value is decided. This requires the bit time to be at least twice as long as the propagation delay, which limits the data rate: for instance, 1 Mb/s is feasible on a 40-m bus at maximum, while 250 kb/s can be achieved over 250 m.

CAN has several mechanisms for error detection. For instance, it is checked that the CRC transmitted in the frame is identical to the CRC computed at the receiver end, that the structure of the frame is valid, and that no bit-stuffing error occurred. Each station which detects an error sends an "error flag," which is a particular type of frame composed of six consecutive dominant bits that allows all the stations on the bus to be aware of the transmission error. The corrupted frame automatically reenters into the next arbitration phase, which might lead it to miss its deadline due to the additional delay. The error recovery time, defined as the time from detecting an error until the possible start of a new frame, is 17–31 bit times. CAN possesses some fault-confinement mechanisms aimed at identifying permanent failures due to hardware dysfunctioning at the level of the microcontroller, communication controller, or physical layer. The scheme is based on error counters that are increased and decreased according to particular events (e.g., successful reception of a frame, reception of a corrupted frame, etc.). The relevance of the algorithms involved is questionable (see [34]), but the main drawback is that a node has to diagnose itself, which can lead to the nondetection of some critical errors. For instance, a faulty oscillator can cause a node to transmit continuously a dominant bit, which is one manifestation of the "babbling idiot" fault; see [35]. Furthermore, other faults such as the partitioning of the network into several subnetworks may prevent all nodes from communicating due to bad signal reflection at the extremities. Without additional fault-tolerance facilities, CAN is not suited for safety-critical applications such as some future x-by-wire systems. For instance, a single node can perturb the functioning of the whole network by sending messages outside their specification (i.e., length and period of the frames). Many mechanisms were proposed for increasing the dependability of CAN-based networks (see [36]–[42]), but as pointed out in [38], if each proposal solves a particular problem, they have not been conceived to be combined. Furthermore, the fault hypotheses used in the design of theses mechanisms are not necessarily the same, and the interactions between them remain to be studied in a formal way.

The CAN standard only defines the physical layer and data link layer (DLL). Several higher level protocols have been proposed, for instance, standardizing startup procedures, implementing data segmentation, or sending periodic messages (see OSEK/VDX communication in Section III-B1). Other higher level protocols standardize the content of messages in order to ease the interoperability between ECUs. This is the case for J1939, which is used, for instance, in Scania's trucks and buses [43].

2) Vehicle Area Network (VAN): VAN (see [44]) is very similar to CAN (e.g., frame format, data rate) but possesses some additional or different features that are advantageous from a technical point of view (e.g., no need for bit stuffing; in-frame response: a node being asked for data answers in the same frame that contained the request). VAN was used in production cars by the French carmaker PSA (Peugeot–Citroën) in the body domain, but, as it was not adopted by the market, it was abandoned in favor of CAN.

3) The J1850 Network: The J1850 [19] is an SAE class B priority bus that was adopted in the United States for communications with nonstringent real-time requirements, such as the control of body electronics or diagnostics. Two variants of the J1850 are defined: a 10.4-kb/s single-wire version and a 41.6-kb/s two-wire version. The trend in new designs seems to be the replacement of J1850 by CAN or a low-cost network such as LIN (see Section II-C1).

B. Time-Triggered Networks

Among communication networks, as discussed before, one distinguishes time-triggered networks, where activities are driven by the progress of time, and event-triggered ones, where activities are driven by the occurrence of events. Both types of communication have advantages, but one considers that, in general, dependability is much easier to ensure using a time-triggered bus (refer, for instance, to [7] for a discussion on this topic). This explains that, currently, only time-triggered communication systems are being considered for use in x-by-wire applications. In this category, multiaccess protocols based on TDMA are particularly well suited; they provide deterministic access to the medium (the order of the transmissions is defined statically at the design time), and thus bounded response times. Moreover, their regular

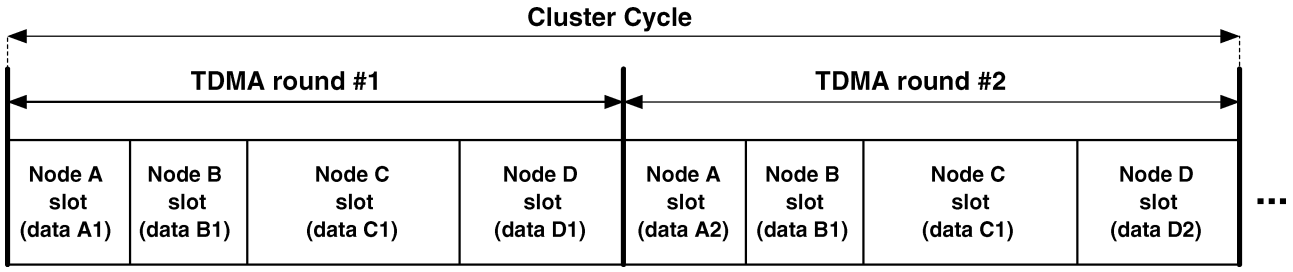

Fig. 4. Example of a TTP/C communication cycle with four nodes, A, B, C, and D.

message transmissions can be used as "heartbeats" for detecting station failures. The three TDMA based networks that are candidates for supporting x-by-wire applications are TTP/C (see Section II-B1), FlexRay (see Section II-B2) and TTCAN (see [27]). At the time of writing, FlexRay, which is backed by the world's automotive industry, seems to be in a very strong position for becoming the standard in the industry.

1) The TTP/C Protocol: The time-triggered protocol TTP/C, which is defined in [23], is a central part of the Time-Triggered Architecture (TTA) (see [45]), and it possesses numerous features and services related to dependability, such as the bus guardian (components that prevent a node from transmitting outside its specification, for instance, at the wrong time or sending a larger frame), the group membership algorithm (knowledge of the set of stations that are functioning properly), and support for mode changes (i.e., specific operational phases of an application; see [46]). The TTA architecture and the TTP/C protocol have been designed and extensively studied at the Vienna University of Technology, Vienna, Austria. Hardware implementations of the TTP/C protocol, as well as software tools for the design of the application, are commercialized by the TTTech company[5] and available today.

On a TTP/C network, transmission relies on redundant channels and each channel transports its own copy of the same message. Although EMI is likely to affect both channels in quite a similar manner, the redundancy provides some resilience to transmission errors. TTP/C can be implemented with a bus topology or a star topology. The latter topology provides better fault tolerance, since, in the star topology, guardians are integrated into central star couplers and protect against errors that cannot be avoided by a local bus guardian. For instance, a star topology is more resilient to spatial proximity faults (i.e., faults that affect all components located in a given area, such as temperature peaks) and to faults due to a desynchronization of an ECU. To avoid a single point of failure, a dual star topology should be used with the drawback that the length of the wires is significantly increased. At the MAC level, the TTP/C protocol implements a synchronous TDMA scheme: the stations (or nodes) have access to the bus in a strict deterministic sequential order and each station possesses the bus for a constant period called a slot, during which it has to transmit one frame. The sequence of slots such that all stations have accessed the bus one time, is called a TDMA round. An example of a round is shown in Fig. 4.

The size of the slot is not necessarily identical for all stations in the TDMA round, but a slot belonging to one station is the same size in each round. Consecutive TDMA rounds may differ according to the data transmitted during the slots, and the sequence of all TDMA rounds is the "cluster cycle" which repeats itself in a cycle.

TTP/C defines three types of frames:

— The "cold start frame," solely used at the initialization of the network.
— The data frame with explicit C-State. The C-State is a field that indicates the internal state of the communication controller: current time, frame being transmitted, current functioning mode of the cluster, membership vector (i.e., the list of stations that are considered as being operational), etc. This information is needed by stations willing to integrate the cluster at startup or reintegrate it at runtime.
— The data frame with implicit C-State. In that case, the C-State is not explicitly transmitted, but the receiver can still detect if it disagrees with the sender on the C-State, since the CRC is computed on the fields of the frame plus the C-State.

A TTP/C frame is composed of a field for indicating mode change requests, of application data, of a CRC and, depending on the frame type, of the C-state. A data frame can carry a payload of up to 240 B [47] but, at the time of writing, the "compatibility layer" specification, which defines the exact format of the frame, is not publicly available for the latest version of the protocol [23].

In order to ease and speed up the design of fault-tolerant applications, TTP/C implements the main services for fault-tolerance. In particular, TTP/C offers a clique[6] avoidance algorithm and a membership algorithm that also provides data acknowledgment. These powerful algorithms have been formally verified (see, for instance, [48] and [49]). The assumptions on the faults that can be handled by the network (i.e., the fault hypothesis) used for the design of TTP/C are precisely stated. These assumptions are quite restrictive; for example, two successive faults must occur at least two rounds apart. In our opinion, future research should investigate whether the fault hypothesis considered in the TTP/C design are relevant in the context of automotive embedded systems where

[5]See http://www.tttech.com

[6]"Cliques" are sets of stations that disagree on the state of the system, for instance, on the set of nodes that are operational at a given time.

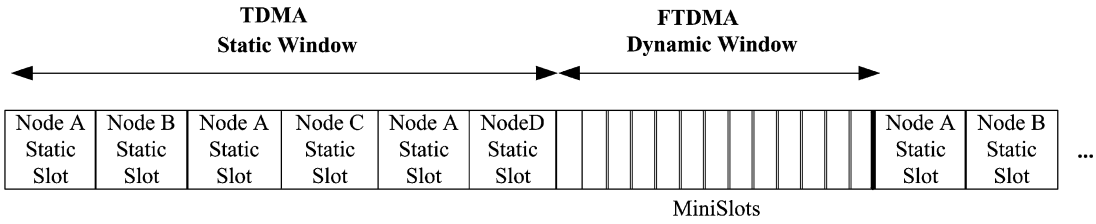

Fig. 5. Example of a FlexRay communication cycle with 4 nodes A, B, C and D.

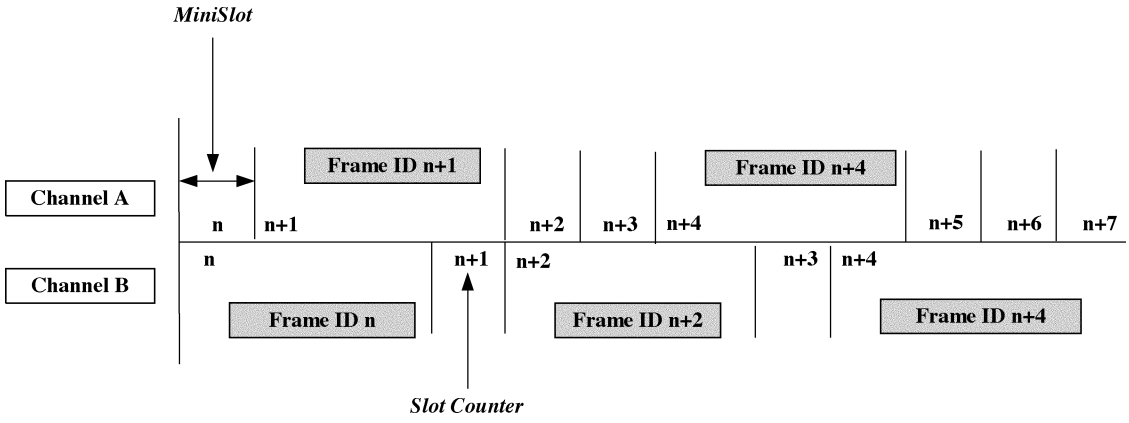

Fig. 6. Example of message scheduling in the dynamic segment of the FlexRay communication cycle.

the environment can be very harsh. Situations outside the fault hypothesis of TTP/C are treated using "never give up" (NUP) strategies which aim to continue operating in a degraded mode. For example, a usual method is that each node switches to local control according to the information still available, while trying to return to the normal mode.

2) FlexRay Protocol: A consortium of major companies from the automotive field is currently developing the FlexRay protocol. The core members are BMW, Bosch, DaimlerChrysler, General Motors, Motorola, Philips, and Volkswagen. The first publicly available specifications of the FlexRay protocol [24] have been recently released.

The FlexRay network is very flexible with regard to topology and transmission support redundancy. It can be configured as a bus, a star, or a multistar. It is not mandatory that each station possess replicated channels nor a bus guardian, even though this should be the case for critical functions such as steer-by-wire. At the MAC level, FlexRay defines a communication cycle as the concatenation of a time-triggered (or static) window and an event triggered (or dynamic) window. In each communication window, the size of which is set statically at design time, two distinct protocols are applied. The communication cycles are executed periodically. The time-triggered window uses a TDMA MAC protocol; the main difference with TTP/C is that a station in FlexRay might possess several slots in the time-triggered window, but the size of all the slots is identical (see Fig. 5). In the event-triggered part of the communication cycle, the protocol is Flexible TDMA (FTDMA): the time is divided into so-called minislots, each station possesses a given number of minislots (not necessarily consecutive), and it can start the transmission of a frame inside each of its own minislots. A minislot remains idle if the station has nothing to transmit which actually induces a loss of bandwidth (see [50] for a discussion on that topic). An example of a dynamic window is shown in Fig. 6: on channel B, frames have been transmitted in minislots n and $n+2$ while minislot $n+1$ has not been used. It is noteworthy that frame $n+4$ is not received simultaneously on channels A and B, since, in the dynamic window, transmissions are independent in both channels.

The FlexRay MAC protocol is more flexible than the TTP/C MAC, since in the static window nodes are assigned as many slots as necessary (up to 2047 overall) and since the frames are only transmitted if necessary in the dynamic part of the communication cycle. In a similar way as with TTP/C, the structure of the communication cycle is statically stored in the nodes; however, unlike TTP/C, mode changes with a different communication schedule for each mode are not possible.

The FlexRay frame consists of three parts: the header, the payload segment containing up to 254 B of data, and the CRC of 24 b. The header of 5 B includes the identifier of the frame and the length of the data payload. The use of identifiers allows to move a software component, which sends a frame X, from one ECU to another ECU without changing anything in the nodes that consume frame X. It has to be noted that this is no more possible when signals produced by distinct components are packed into the same frame for the purpose of saving bandwidth (see the description of frame packing in Section III-A).

From the dependability point of view, the FlexRay document [24] specifies solely the bus guardian and the clock synchronization algorithms. Other features, such as a mem-

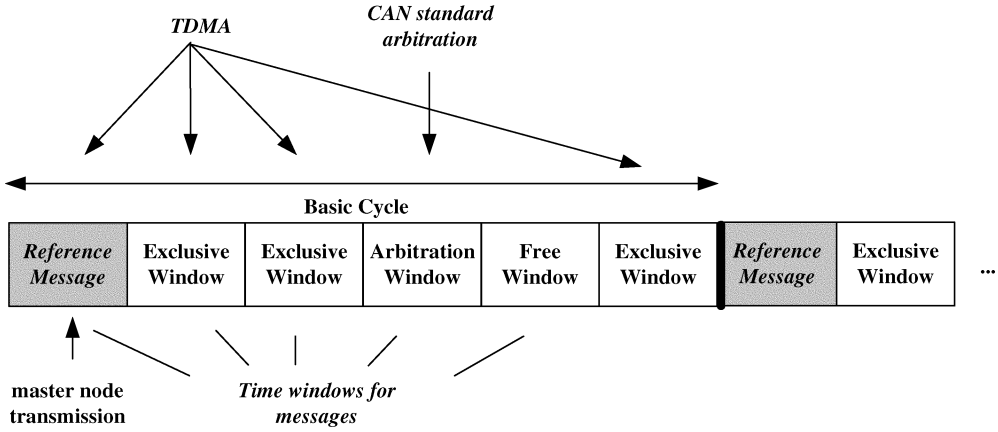

Fig. 7. Example of a TTCAN basic cycle.

bership service or mode management facilities, will have to be implemented in software or hardware layers on top of FlexRay. This will allow to conceive and implement exactly the services that are needed with the drawback that correct and efficient implementations might be more difficult to achieve in a layer above the communication controller.

In the FlexRay specification ([24], p. 8), it is argued that the protocol provides scalable dependability i.e., the "ability to operate in configurations that provide various degrees of fault tolerance." Indeed, the protocol allows for mixing links with single and dual transmission supports on the same network, subnetworks of nodes without bus guardians or with different fault-tolerance capability with regards to clock synchronization, etc. In the automotive context, where critical and noncritical functions will increasingly coexist and interoperate, this flexibility can prove to be efficient in terms of cost and reuse of existing components if missing fault-tolerance features are provided in a middleware (MW) layer such as OSEK FTCom [51] or the one currently under development within the automotive industry project AUTOSAR (see Section III-B).

3) Time-Triggered CAN (TTCAN) Protocol: TTCAN (see [27]) is a communication protocol developed by Robert Bosch GmbH on top of the CAN physical layer and DLL. TTCAN uses the CAN standard but, in addition, requires that the controllers must have the possibility to disable automatic retransmission of frames upon transmission errors and to provide the upper layers with the point in time at which the first bit of a frame was sent or received [52]. The bus topology of the network, the characteristics of the transmission support, the frame format, as well as the maximum data rate—1 Mb/s—are imposed by the CAN protocol. Channel redundancy is possible (see [53] for a proposal), but not standardized and no bus guardian is implemented in the node. The key idea is to propose, as with FlexRay, a flexible time-triggered/event-triggered protocol. As illustrated in Fig. 7, TTCAN defines a basic cycle (the equivalent of the FlexRay communication cycle) as the concatenation of one or several time-triggered (or "exclusive") windows and one event-triggered (or "arbitrating") window. Exclusive windows are devoted to time-triggered transmissions (i.e., periodic messages), while the arbitrating window is ruled by the standard CAN protocol: transmissions are dynamic and bus access is granted according to the priority of the frames. Several basic cycles that differ by their organization in exclusive and arbitrating windows and by the messages sent inside exclusive windows can be defined. The list of successive basic cycles is called the system matrix, which is executed in loops. Interestingly, the protocol enables the master node (i.e., the node that initiates the basic cycle through the transmission of the "reference message") to stop functioning in TTCAN mode and to resume in standard CAN mode. Later, the master node can switch back to TTCAN mode by sending a reference message.

TTCAN is built on a well-mastered and low-cost technology, CAN, but, as defined by the standard, does not provide important dependability services such as the bus guardian, membership service, and reliable acknowledgment. It is, of course, possible to implement some of these mechanisms at the application or MW level but with reduced efficiency. Probably, carmakers might consider the use of TTCAN for some systems during a transition period until FlexRay technology is fully mature.

C. Low-Cost Automotive Networks

Several fieldbus networks have been developed to fulfill the need for low-speed/low-cost communication inside mechatronic-based subsystems generally made of an ECU and its set of sensors and actuators. Two representatives of such networks are LIN and TTP/A. The low-cost objective is achieved not only because of the simplicity of the communication controllers but also because the requirements set on the microcontrollers driving the communication are reduced (i.e., low computational power, small amount of memory, low-cost oscillator). Typical applications involving these networks include controlling doors (e.g., door locks, opening/closing windows) or controlling seats (e.g., seat position motors, occupancy control). Besides cost considerations, a hierarchical communication architecture, including a backbone such as CAN and several subnetworks such as LIN, enables reducing the total traffic load on the backbone.

Both LIN and TTP/A are master–slave networks where a single master node, the only node that has to possess a precise and stable time base, coordinates the communication on

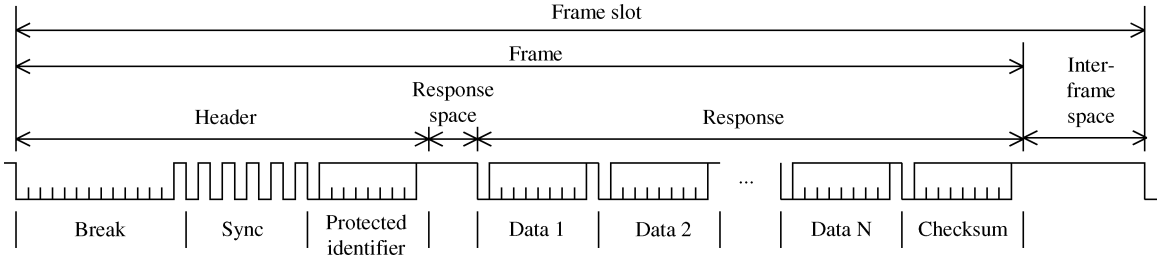

Fig. 8. Format of the LIN frame. A frame is transmitted during its "frame slot" which corresponds to an entry of the schedule table.

the bus: a slave is only allowed to send a message when it is polled. More precisely, the dialogue begins with the transmission by the master of a "command frame" that contains the identifier of the message whose transmission is requested. The command frame is then followed by a "data frame" that contains the requested message sent by one of the slaves or by the master itself (i.e., the message can be produced by the master).

1) LIN: LIN (see [16] and [17]) is a low-cost serial communication system used as SAE class A network, where the needs in terms of communication do not require the implementation of higher bandwidth multiplexing networks such as CAN. LIN is developed by a set of major companies from the automotive industry (e.g., DaimlerChrysler, Volkswagen, BMW, and Volvo) and is already widely used in production cars.

The LIN specification package (LIN version 2.0 [16]) includes not only the specification of the transmission protocol (physical layer and DLL) for master–slave communications but also the specification of a diagnostic protocol on top of the DLL. A language for describing the capability of a node (e.g., bit rates that can be used, characteristics of the frames published and subscribed by the node, etc.) and for describing the whole network is provided (e.g., nodes on the network, table of the transmissions' schedule, etc.). This description language facilitates the automatic generation of the network configuration by software tools.

A LIN cluster consists of one "master" node and several "slave" nodes connected to a common bus. For achieving a low-cost implementation, the physical layer is defined as a single wire with a data rate limited to 20 kb/s due to EMI limitations. The master node decides when and which frame shall be transmitted according to the schedule table. The schedule table is a key element in LIN; it contains the list of frames that are to be sent and their associated frame slots, thus ensuring determinism in the transmission order. At the moment a frame is scheduled for transmission, the master sends a header (a kind of transmission request or command frame) inviting a slave node to send its data in response. Any node interested can read a data frame transmitted on the bus. As in CAN, each message has to be identified: 64 distinct message identifiers are available. Fig. 8 depicts the LIN frame format and the period, termed a "frame slot," during which a frame is transmitted.

The header of the frame that contains an identifier is broadcast by the master node, and the slave node that possesses the identifier inserts the data in the response field. The "break" symbol is used to signal the beginning of a frame. It contains at least 13 dominant bits (logical value zero) followed by one recessive bit (logical value one) as a break delimiter. The rest of the frame is made of byte fields delimited by one start bit (value zero) and one stop bit (value one), thus resulting in a 10-b stream per byte. The "sync" byte has a fixed value (which corresponds to a bit stream of alternatively zero and one); it allows slave nodes to detect the beginning of a new frame and be synchronized at the start of the identifier field. The so-called protected identifier is composed of two subfields: the first 6 b are used to encode the identifier and the last 2 b the identifier parity. The data field can contain up to 8 B of data. A checksum is calculated over the protected identifier and the data field. Parity bits and checksum enable the receiver of a frame to detect bits that have been inverted during transmission.

LIN defines five different frame types: unconditional, event-triggered, sporadic, diagnostic, and user defined. Frames of the latter type are assigned a specific identifier value and are intended to be used in an application-specific way that is not described in the specification. The first three types of frames are used to convey signals. Unconditional frames are the usual type of frames used in the master–slave dialog and are always sent in their frame slots. Sporadic frames are frames sent by the master, only if at least one signal composing the frame has been updated. Usually, multiple sporadic frames are assigned to the same frame slot, and the higher priority frame that has an updated signal is transmitted. An event-triggered frame is used by the master willing to obtain a list of several signals from different nodes. A slave will only answer the master if the signals it produces have been updated, thus resulting in bandwidth savings if updates do not take place very often. If more than one slave answers, a collision will occur. The master resolves the collision by requesting all signals in the list one by one. A typical example of the use of the event-triggered transfer given in [16] is the doorknob monitoring in a central locking system. As it is rare that multiple passengers simultaneously press a knob, instead of polling each of the four doors, a single event-triggered frame can be used. Of course, in the rare event when more than one slave responds, a collision will occur. The master will then resolve the collision by sending one by one the individual identifiers of the list during the successive frame slots reserved for polling the list. Finally, diagnostic frames have a fixed size of 8 B fixed

value identifiers for both the master's request and the slaves' answers and always contain diagnostic or configuration data whose interpretation is defined in the specification.

It is also worth noting that LIN offers services to send nodes into a sleep mode (through a special diagnostic frame termed "go-to-sleep-command") and to wake them up, which is convenient, since optimizing energy consumption, especially when the engine is not running, is a real matter of concern in the automotive context.

2) The TTP/A Network: Like TTP/C, TTP/A [18] was initially invented at the Vienna University of Technology and is now commercially available from the TTTech company. TTP/A pursues the same aims and shares the main design principles as LIN, and it offers, at the communication controller level, some similar functionalities—in particular, in the areas of plug-and-play capabilities and online diagnostics services. TTP/A implements the classic master–slave dialogue, termed master–slave round, where the slave answers the master's request with a data frame having a fixed length data payload of 4 B. The "multipartner" rounds enable several slaves to send up to an overall amount of 62 B of data after a single command frame. A "broadcast round" is a special master–slave round in which the slaves do not send data; it is, for instance, used to implement sleep/wake-up services. The data rate on a single wire transmission support is, as for LIN, equal to 20 kb/s, but other transmission supports enabling higher data rates are possible. To the best of our knowledge, TTP/A is not currently in use in production cars.

D. Multimedia Networks

Many protocols have been adapted or specifically conceived for transmitting the large amount of data needed by emerging multimedia applications in automotive systems. Two major contenders in this category are MOST and IDB-1394.

1) MOST Network: MOST (see [22]) is a multimedia network development of which was initiated in 1998 by the MOST Cooperation (a consortium of carmakers and component suppliers). MOST provides point-to-point audio and video data transfer with a data rate of 24.8 Mb/s. This support end-user applications like radios, global positioning system (GPS) navigation, video displays, and entertainment systems. The MOST's physical layer is a plastic optical fiber (POF) transmission support which provides a much better resilience to EMI and higher transmission rates than classical copper wires. Current production cars from BMW and DaimlerChrysler employ a MOST network.

2) The IDB-1394 Network: IDB-1394 is an automotive version of IEEE 1394 for in-vehicle multimedia and telematic applications jointly developed by the IDB Forum[7] and the 1394 Trade Association.[8] The system architecture of IDB-1394 permits existing IEEE 1394 consumer electronics devices to interoperate with embedded automotive grade devices. IDB-1394 supports a data rate of 100 Mb/s over a twisted pair or POF, with a maximum number of embedded devices which are limited to 63 nodes. From the point of view of transmission rate and interoperability with existing IEEE 1394 consumer electronic devices, IDB-1394 is a serious competitor for MOST technology.

[7]See http://www.idbforum.org

[8]See http://www.1394ta.org

III. MW Layer

The design of automotive electronic systems has to take into account several constraints. First, these systems are moving to integrated electronics architectures in the sense that a tight cooperation between functions is increasingly needed. Second, they are produced through a complex cooperative multipartner development process. Finally, these new systems are subject to increasingly stringent requirements in terms of safety, availability, and fault tolerance. A classic approach for easing the integration of software-based components is to furnish an *MW layer* that provides common services and a common interface to application software components. In practice, an MW is made up of a set of existing communication protocols and carmarker-specific layers. Among the existing automotive protocols, some, such as OSEK-Com (see Section III-B) or ISO transport layer (see [54]), are communication oriented, but several others offer specialized services: diagnostic modules (e.g., ISO 15 765 [54]–[56]), calibration services (e.g., CCP [57]), or I/O abstraction layers (e.g., HIS consortium I/O library).

A. Functions of an Automotive MW

The main functions related to the communication services that have to be realized by an MW are listed below.

- **Hiding the distribution**. Ideally, communication services should be fully independent from the location of the involved entities. The same services and the same interface should be available for intra-ECU, inter-ECU, and interdomain communications whatever the underlying protocols.
- **Hiding the heterogeneity of the platforms**. A large diversity of microcontrollers, protocols, and operating systems (OSs) are used inside the same vehicle. The MW should encapsulate the OS services and provide an application programming interface (API) independent of the underlying protocols, of the CPU architecture (e.g., 8/16/32 b, endianness). As much as possible, the MW should provide common services to access I/O devices.
- **Providing high-level services**. The aim here is to diminish the development time and increase quality through the reuse of validated services. Examples of high-level services include working mode management, remote procedure call (RPC), redundancy management, membership service, downloading functionalities, etc. Application processes in the different ECUs of a vehicle exchange data, termed *signals* (e.g., the number of revolutions per minute, the speed of the vehicle, etc.) while *frames* are transmitted over the network; since control functions of a vehicle are subject to heavy timing constraints, many *signals* have

a limited lifetime, and the problem is to realize the communication in such a way as to ensure that these "freshness" constraints are met; so, in particular, one important service of an MW is to pack the signals that are sent by application processes into frames and to send the frames at the right point in time for ensuring the deadline constraint on each signal it contains. This function is generally called frame packing, and it is performed according to an offline generated configuration (this point will be discussed in Section III-C1).

— **Ensuring QoS**. It can be necessary for the MW to improve the QoS provided by the lower level protocols. For instance, if the Hamming distance of the CRC is too small with regards to the dependability objectives, an additional CRC can be transparently inserted by the MW in the data field of the MAC level frame. The MW can also serve to implement mechanisms for correcting "bugs" in the lower level protocols such as the "inconsistent message duplicate" of CAN (see [41] for such a proposal). In addition, the MW can offer QoS guarantees that are not considered by lower level communication layers. We saw previously that one of its roles at runtime is to realize the packing of signals into frames and the sending of frames according to freshness properties required on signals. Other typical examples include reliable acknowledgment service on CAN, supplying status information on the data that is consumed by the application-level software (e.g., data was refreshed since last reading; its freshness constraint was not respected), or providing filtering mechanisms (e.g., notify the application for each k reception or when the data value has changed in a significant way). Finally, an MW can implement algorithms to adapt, at runtime, the parameters of the communication protocols (e.g., priorities, transmission rate) according to the requirements of the application or changing environmental conditions (e.g., EMI level).

B. State of the Art in Automotive MW

Some carmakers, such as DaimlerChrysler with the TITUS/DBKOM communication stack, possess proprietary MW that helps to integrate ECUs and software modules developed by their third-party suppliers. To the best of our knowledge, no publicly available precise description of such MW exists. Several cooperative projects aimed at the development of standard MW have been undertaken within the automotive industry (European ITEA EAST-EEA project[9] or, more recently, AUTOSAR[10]). The only results publicly available have been produced in the context of the OSEK/VDX consortium,[11] whose objective is to build a standard architecture for in-vehicle control units. Among the results of the OSEK/VDX group, two specifications are of particular interest: the communication layer [58] and the fault-tolerant communication layer [51]. Finally, we shall review the Volcano MW [59], which is a commercial product.

1) OSEK/VDX Communication: The OSEK/VDX consortium[12] specifies a communication layer [58] that defines common software interfaces and common behavior for internal and external communications between application processes. At the application layer level, processes exchange signals, or "messages" in OSEK/VDX terminology, that are stored in "message objects," while communicating OSEK/VDX entities exchange so-called interaction layer protocol data units (I-PDUs) that are a collection of messages. Each receiver for a message can specify it as queued (first-in, first-out (FIFO) buffer of fixed length) or unqueued (i.e., a new value overwrites the old one) and associate it with a filtering mechanism. How signals are packed into a frame is statically specified offline, and OSEK/VDX Communication automatically realizes the packing/unpacking at runtime. I-PDU and messages are described through the OSEK/VDX Implementation Language (see [60]).

The I-PDU containing a message can be transmitted each time the message is refreshed (message object has the "Triggered Transfer Property") or not (message object has the "Pending Transfer Property"). The transmission mode is a characteristic of the I-PDU; it can be "direct" (the I-PDU contains at least one message having the "Triggered Transfer Property" and a new value for this message is written), "periodic" (the I-PDU contains only messages with "Pending Transfer Property"), or "mixed" (the I-PDU is at least periodically transmitted but it can also be transmitted as in the direct mode). Two examples illustrate these mechanisms. In Fig. 9, M and N are message objects with the Triggered Transfer Property that are attached to an I-PDU named P, which possesses the direct transmission mode. Each time M or N is updated, the I-PDU is sent to the underlying layer. For instance, at time t_2, N is updated and P is sent with messages X_1 and Y_2. Fig. 10 illustrates a configuration where M is a message object with the Triggered Transfer Property and N a message object with the Pending Transfer Property. They are both attached to P, an I-PDU with mixed transmission mode. Each time a new value of M is provided by the application, P is sent (time t_1 and t_5 in the example) but P is also sent at a predefined rate (at time t_2 and t_4).

The transmission of messages is nonblocking for the application, so *OSEK/VDX Communication* includes notification mechanisms for informing the application on the status of a transmission or reception. In particular, it is possible for a sender to know if the time between a transmission request and the successful transmission over the network exceeds a given threshold ("Transmission Deadline Monitoring"). Similarly, a receiving node can be informed if a periodic message has not been received within a given time interval ("Reception Deadline Monitoring"). Finally, a minimum delay between two successive transmissions can be specified (i.e., transmission requests are postponed until the delay expires), which provides some resilience

[9]See http://www.east-eea.net

[10]See http://www.autosar.org and [6].

[11]Detailed information can be obtained at http://www.osek-vdx.org

[12]See http://www.osek-vdx.org

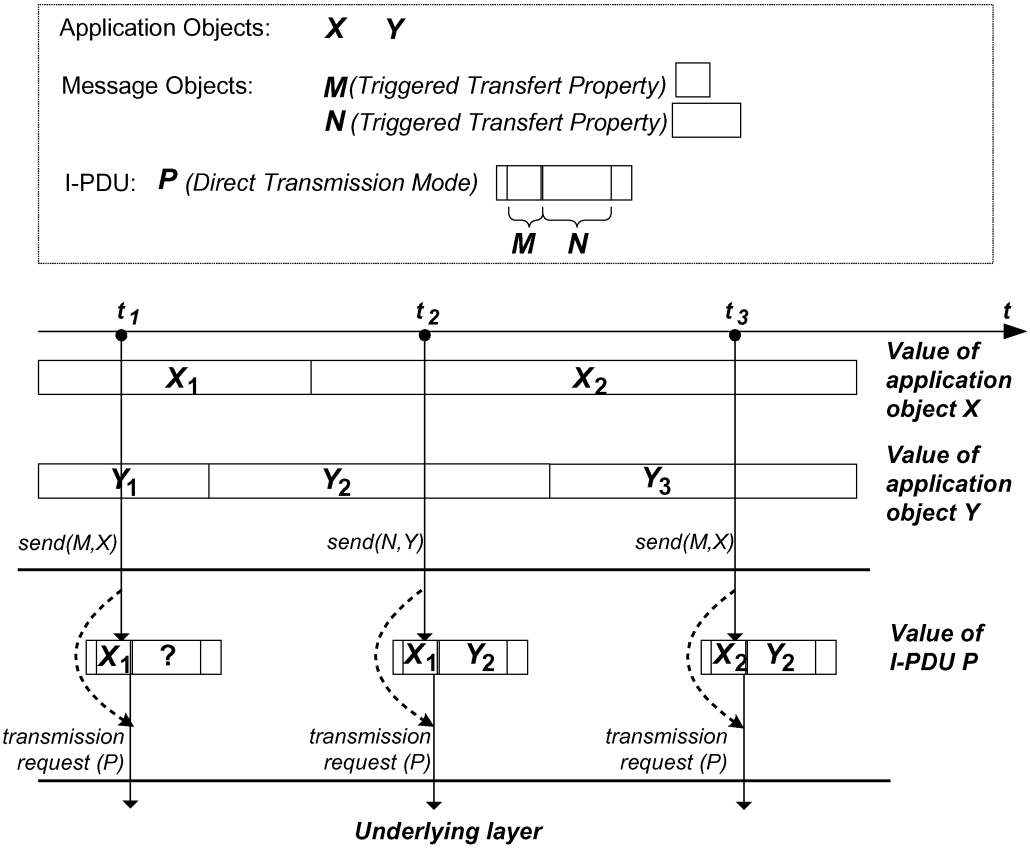

Fig. 9. I-PDU with direct transmission mode: the I-PDU is sent each time one of its message is updated.

against nodes transmitting outside their specification due to a software dysfunctioning.

OSEK/VDX Communication runs on top of a transport layer (e.g., ISO 15 765-2 [54] for CAN as DLL) that takes care of the I-PDU segmentation, and it can operate on any OS compliant with OSEK/VDX OS services for tasks, events, and interrupt management (see [61]). Some questions deserve to be raised. In particular, communications between application processes that are internal to one ECU or located in two distant ECUs do not obey exactly the same rules (see [62] for more details); thus, the designer has to take into account the distribution of the functions, which is a hindrance to portability.

2) Volcano: Volcano [59], [63] is a commercial product of Volcano Communications Technologies initially developed in partnership with Volvo Car Corporation (1994–1998). It consists of a communication layer and a tool chain which supports requirement capturing, variant and version handling, and facilities for a multipartner development process. Originally, this product aimed to support CAN and Volcano Lite (a Volvo-proprietary low-speed network based on a "single master–multiple slaves" protocol). It was then extended to fit LIN protocol and is being prepared to support FlexRay and MOST protocols.

The concept of Volcano is based on *signals*. These signals represent data produced and consumed by functions implemented on each ECU while at the communication layer, *frames* are transmitted over the network. The communication model of Volcano is "publisher–subscriber."

Volcano aimed to make an optimized usage of resources online, e.g., the CAN bandwidth, and to build solutions that would ensure timing properties on signals by using schedulability analysis techniques (see [30], [59] for CAN protocol). For this purpose, Volcano configuration tools include a "frame-packing" algorithm (see Section III-C1). Furthermore, Volcano offers an API that hides the communication on the nework from the application developers. The Volcano API provides four main services, termed "read," "write," "v_input," and "v_output." A "read" call returns the latest value of a signal that is stored in the Volcano layer to the application, and a "write" call updates the value of a signal. A "v_output" call copies the frames in the communication controller, while a "v_input" copies the received frames and makes the signal available to the application (through a "read" call). Fig. 11 shows an example that illustrates these mechanisms: s is a signal whose value is produced by an application process activated through an external event (e.g., a button pressed) on the ECU1 (publisher of s); this signal is consumed by an application process located on the ECU2 (subscriber of s) in order to apply an action (e.g., lights switch on); the different steps for the production, the transmission, and the consumption of s are:

1) the processing of the external event ($[t_1, t_2[$);
2) at t_2, the "write" call makes the value σ of s available at the Volcano layer;
3) at t_4, the frame F including the value σ of s is copied in the communication controller ("v_output" call);

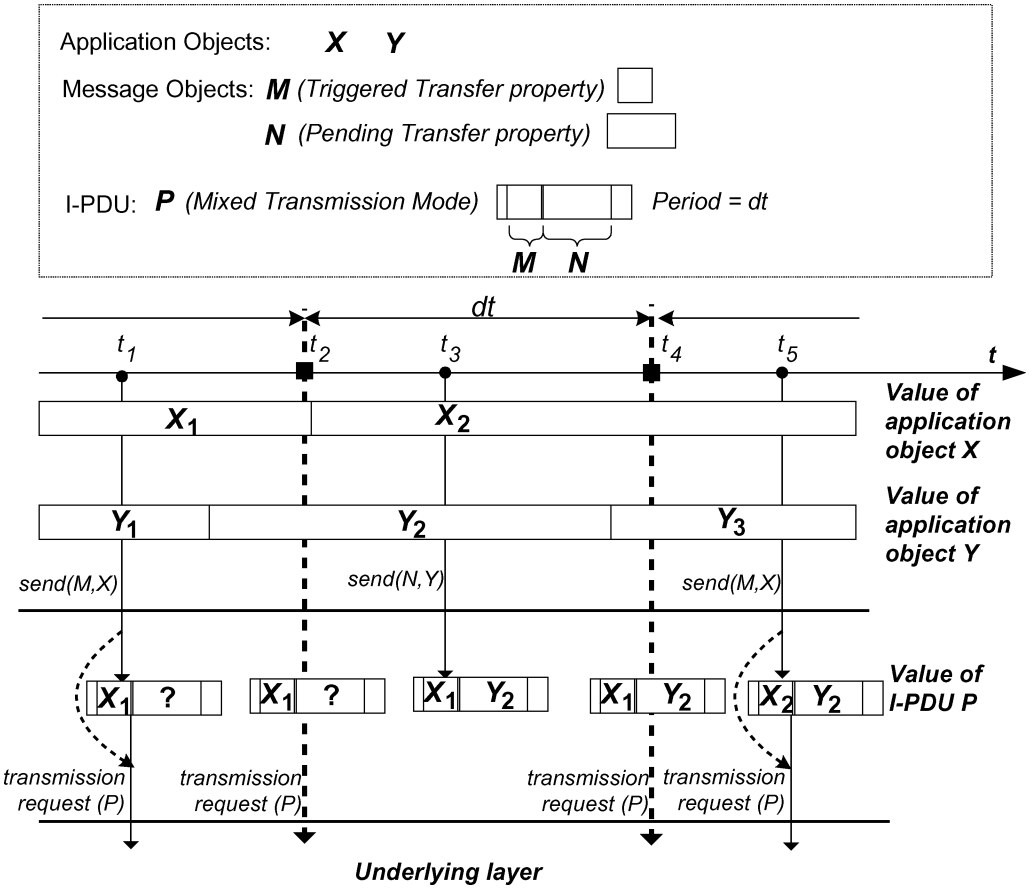

Fig. 10. I-PDU with mixed transmission mode: the I-PDU is sent with a predefined period plus each time a message with Triggered Transfer Property is updated.

4) the frame F is sent (e.g., on CAN bus, start of arbitration) at t_4;
5) the transmission of F ends at time t_5;
6) a "v_input" call done by the ECU2 copies the value σ of the signal s in the Volcano layer at time t_6;
7) a "read" call reads this value at the application level (time t_7) and processes it in order to apply the corresponding action $(]t_7, t_8])$.

According to their values, three kinds of signals are handled: integer signals (unsigned number, static length between 1 and 16 b), Boolean signals (1 b), and byte signals that define a sequence of bytes (between 1 and 8) not interpreted by the online Volcano part. As said previously, a signal is described by its "publisher" ECU and each of its "subscriber" ECUs. More precisely, at design time a database of signals is constituted where their intrinsic characteristics (e.g., size) as well as the "publishing" and "subscribing" nodes are recorded. Each signal is characterized by a set of attributes; some of them specify timing constraints that have to be respected online.

— Type of information: it can be a *state* signal (each production carries a complete description of the information and some signal values can be missed by a subscriber of such a signal) or *state change* signal (this signal carries an incremental description of data, e.g., an OFF–ON button pressed; each production of these signals has to be consumed).
— Generation rule: it can be *periodic* (produced at regular intervals) or *sporadic* (produced by the application in response to some external events).
— Latency: for the publisher of a signal, it represents the time interval between the occurrence of the event that generates a value of this signal (external event or timeout) and the first "v_output" call copying the frame containing this value (e.g., $[t_1, t_3]$ in Fig. 11); for a subscriber of the signal, it is the estimated delay between two events, its availability to the application and the effective application consumption (e.g., $[t_6, t_8]$ in Fig. 11).
— Intervals between signal instances: it includes two attributes, *min_interval* and *max_interval*; for the publisher of a signal, the *min_interval* defines the minimum time between two "write" requests; for a subscriber of a signal, this optional attribute corresponds to a constraint that defines the minimum time interval between two consecutive arrivals of the signal. The *max_interval* attribute is not useful for a published signal (already captured through *min_interval* and *latency* attributes), while for a subscribed signal, it is the maximum interval between two consecutive consumptions of the signal by application functions.
— Freshness constraint: the *max_age* attribute (see Fig. 11) concerns only subscribed signals and represents the maximum acceptable age of the signal at

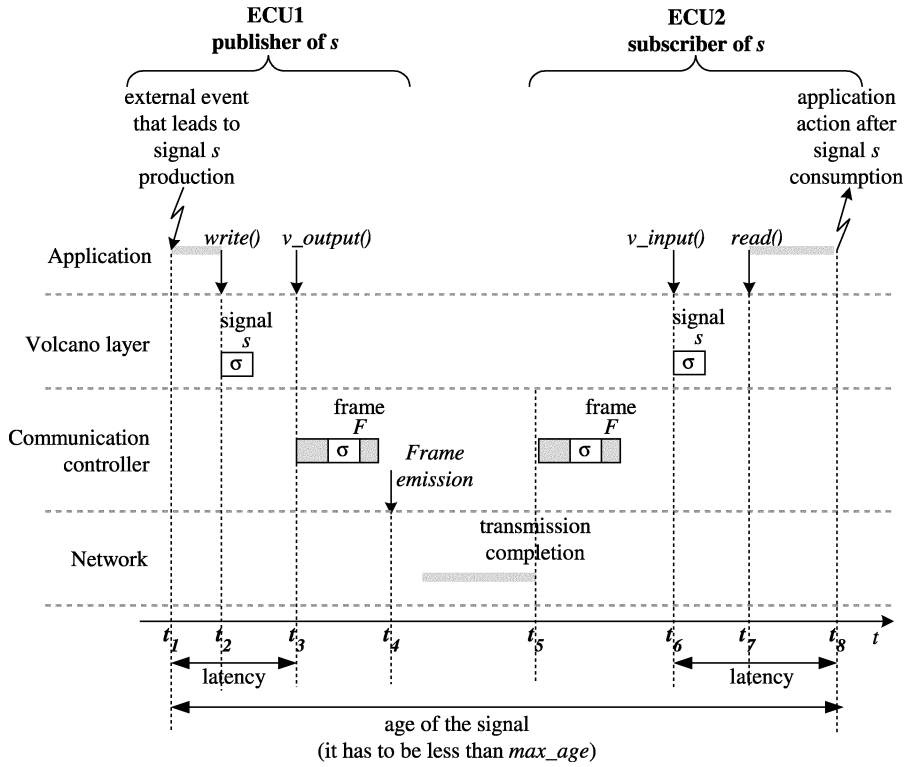

Fig. 11. Volcano services.

its consumption by an application function (on the Fig. 11, the "age" of signal s is $[t_1, t_8]$ and this interval has to be less than the specified *max-age*).

From this description, a set of frames is produced by the so-called "frame compiler" with the main objective of meeting temporal constraints. Detailed specifications of offline configuration strategies are, to the best of our knowledge, not publicly available.

At runtime, the Volcano communication layer forms the frames according to the specifications derived from the configuration step and it checks that the assumptions made offline are met. Furthermore, at runtime, several published signals that are updated at different rates can be placed in the same frame. So a special mechanism is provided by Volcano in order to indicate which signal of a received frame has been updated from the last transmission of the same frame. This mechanism is based on the so-called update bit attached to each signal composing a frame. An update bit is set for a published signal each time this signal has been written ("write" call) and reset, each time the frame containing the signal is sent on the network. On the receiving side, the Volcano layer can determine which signal in the received frame has been modified and can propagate this information to the application thanks to a "flag" that is set when the update bit is set and reset explicitly by the application. Finally, in order to verify on line that timing constraints are respected, "timeouts" can be handled by the Volcano layer. A timeout can be attached to a subscribed signal and is associated to a given time interval and to a default value of the signal. Each time the concerned signal is not received within the given time interval, the signal is set to the given default value. The timeout reset is done after reception of a frame containing the updated signal.

More information about Volcano concepts can be found at Volcano Automotive Group.[13]

3) OSEK/VDX Fault-Tolerant Communication (FTCom): OSEK/VDX Communication, like the current version of Volcano, is not intended to be used on top of a time-triggered network. However, higher level services are still needed on top of FlexRay or TTP/C for facilitating the development of fault-tolerant applications. OSEK/VDX FTCom (see [51]) is a proposal that pursues this objective. It mainly provides message handling but also supports clock synchronization services, lifesign update, and start-up functions. An optional service is the membership status of the nodes, which can be implemented at the FTCom level when the underlying network does not provide it, as is the case with FlexRay. In the remainder of this section, we describe message handling and synchronization services.

a) Message Handling: OSEK/VDX FTCom architecture is structured in several layers, as illustrated in Fig. 12. The *interaction layer* is in charge of the communication to and from the Communication Network Interface (CNI, a memory area shared by the controller and the host computer where data is written), of the packing and unpacking of the frames (according to an offline generated configuration) and, finally, of the byte-order conversion that may be necessary on the local CPU.

The *fault-tolerant layer* contains mechanisms for handling the redundancy of the communication: message instances can be replicated and transmitted in several distinct frames.

[13]See http://www.volcanoautomotive.com

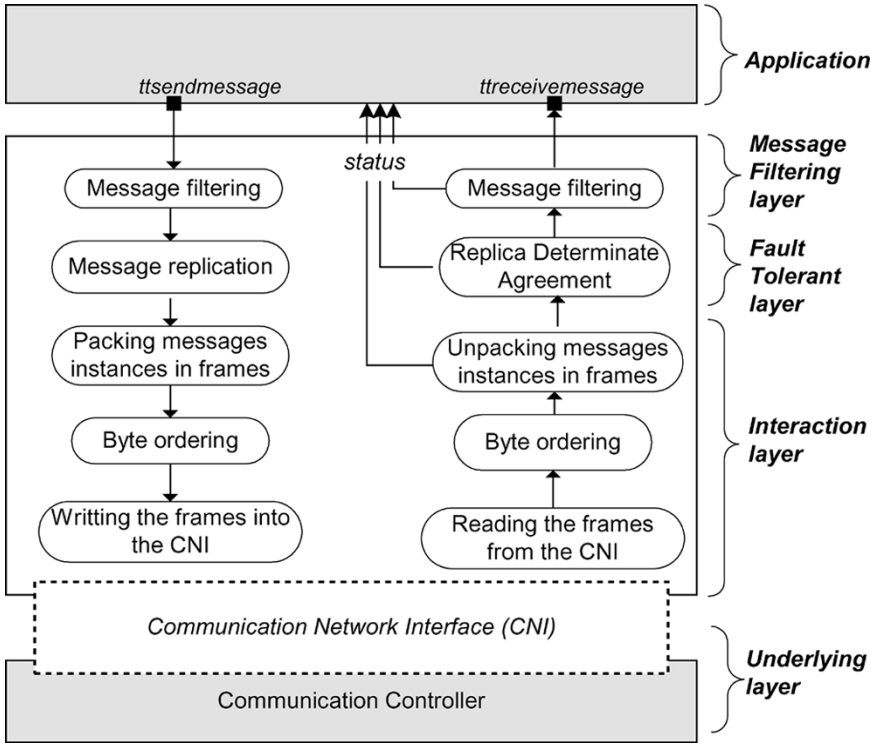

Fig. 12. Structure and services of OSEK/VDX FTCom.

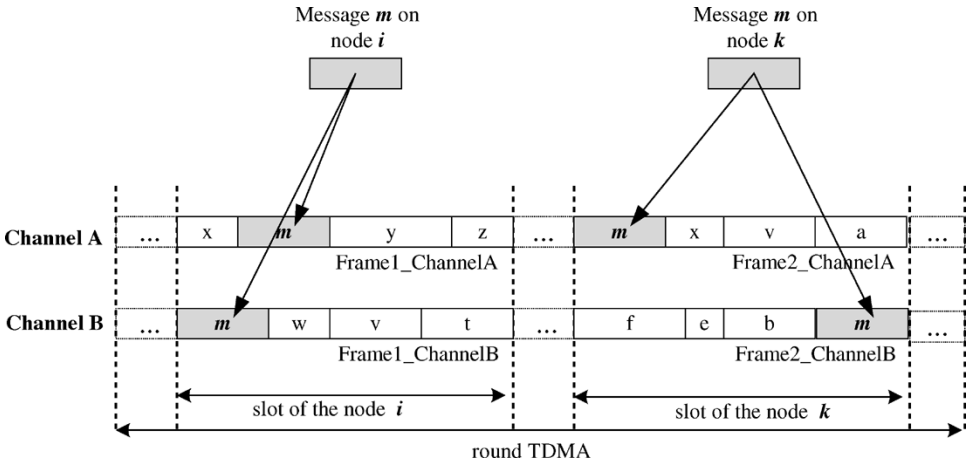

Fig. 13. Fault tolerance through replicated channels and redundant nodes: the same message m is transmitted, on both communication channels, by each of the two nodes (i and k) that compose the fault-tolerant unit.

The replication of nodes is a second way of ensuring fault tolerance: the same information can be produced by a set of replicated nodes called a fault-tolerant unit (FTU). Fig. 13 shows an example where two nodes i and k composing a FTU send four instances of the same message m. The frame packing can be different for each node and each communication channel, as in the example shown in Fig. 13. One of the main functions of OSEK/VDX FTCom, implemented in the *fault-tolerant layer*, is to manage the redundancy of data needed for achieving fault-tolerance. In the example of Fig. 13, four message instances are received by the interaction layer during each TDMA round. From an implementation standpoint, it is usually preferable to present only one copy of data to the application located on a receiver node. This simplifies the application code and keeps it independent from the level of redundancy which, subsequently, facilitate the portability of the application. In OSEK/VDX terminology, the algorithm responsible for the choice of the value that will be transmitted to the application is termed the Replica Determinate Agreement (RDA). Many agreement strategies are possible: pick-any (fail-silent node), average-value, pick-a-particular-one, majority vote, etc. FTCom provides some predefined algorithm and, when none of them can be applied, it provides a generic way for specifying the agreement strategy of replicated data.

The *message filtering layer* consists of mechanisms for passing only "significant" data to the application or to the fault-tolerant layer (on a receiver node and a producer node,

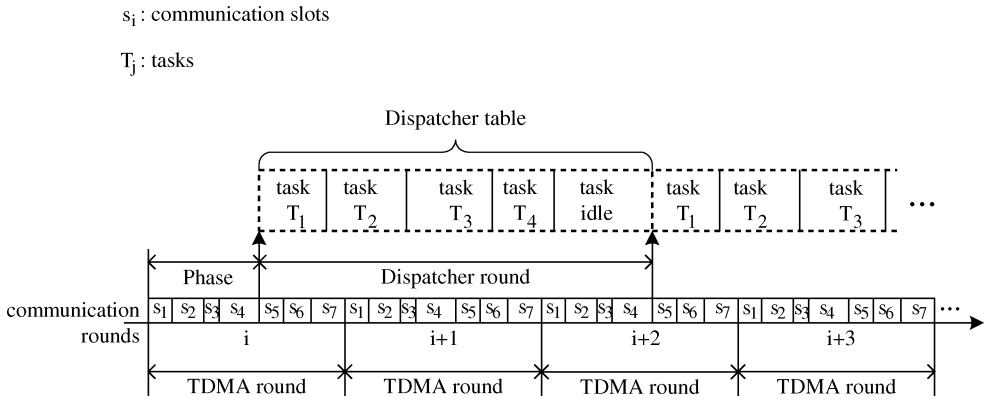

Fig. 14. OSEK/VDX FTCom synchronization service on a time-triggered OS.

respectively). Several generic algorithms, as well as a framework for user-defined ones, are available. Among the predefined filtering algorithms, we can cite the following: passing messages whose value has been modified, passing messages whose value is included in a predefined interval, etc. The filtering mechanisms are statically defined at the configuration step.

b) Synchronization Service: On a time-triggered OS such as OSEK Time [38], the scheduling of tasks is specified in an offline generated time table called the dispatcher table. OSEK/VDX FTCom provides a service, used by the OS, to synchronize the start of the task schedule defined in the dispatcher table, to a particular point in time in the message schedule (i.e., the TDMA round). Since the message schedule is the same over the whole network, this service can be used to synchronize applications running on different ECUs.

As illustrated in Fig. 14, a dispatcher table (or dispatcher round) is defined on a given ECU. Its length has to be a multiple of the TDMA round (that is why, as shown in the example, an idle task completes the dispatcher round). The synchronization service configuration requires the identification of the TDMA round during which the dispatcher table will start, the length of the round and the phase (or offset) between the beginning of the TDMA and the dispatcher rounds.

C. MW Configuration

In the previous paragraphs, we identified the main services that are to be provided by an automotive MW and the solutions that are available or under development. Additionally, an important issue is the generation of an MW optimized for the needs of a given application, or even for the particular needs of a given ECU. This step is crucial because it will have an direct impact on the performances and implementation costs.

Two main problems can be identified. The first one is to minimize the overheads, in terms of memory and CPU usage, induced by the MW on each node. After the minimum set of services needed have been identified and, assuming the MW has a modular design, the most compact implementation of the MW can be generated. Another issue is to make technical implementation choices that are efficient with respect to the platform (i.e., OS, hardware, networks) and the workload (CPU, messages). In particular, one has to decide the characteristics of the MW tasks: segmentation in tasks and interrupt routines, allocation of task priorities, activation pattern (e.g., a task is periodic or invoked by the application), etc.

The second major configuration problem is to set the characteristics of the communication in such a way as to meet the freshness constraints of the signals and to minimize the bandwidth utilization. The latter point is important for enabling the use of low-cost electronic components and for facilitating an incremental design process. Knowing the transmission protocols, the set of ECUs, and, for each ECU, the set of signals that are to be sent over the network, their size, their deadline, and their production period, the problem consists of building the set of frames and choosing the characteristics of transmission. In particular, on a priority bus such as CAN, the periods and the priorities of the frames have to be chosen while on a TTP/C network; the problem is to define the cluster cycle: the number of transmissions of each frame per cluster cycle and the location of the transmissions inside the cycle. These latter two points are discussed in Section III-C1 and Section III-C2, respectively.

1) Frame Packing Over a Priority-Based Network: On a priority-based protocol, the packing of the signals into frames and the sending of the frames at the right point in time is done by the MW. The whole problem is to constitute, offline, the set of frames from a given set of signals in such a way as to respect the deadline constraints and minimize bandwidth consumption. Each signal is characterized by its sending station, its production period (or minimum interarrival for sporadic signals) on this station, its size, and its relative deadline. The classic frame-packing method is to bind, statically, each signal to a frame. The frames are then periodically transmitted even if some signals having lower frequencies than the frame have not been generated. The results of the frame-packing step is a set of frames where each frame is characterized by the transmitting station, the priority on the network, the signals composing it, and the transmission period (which is the smallest period of the signals contained in the frame).

The frame-packing problem, which is closely related to "bin packing," was shown to be NP-complete in [64] and that it cannot be solved using an exhaustive approach even for a very small number of signals and/or ECUs (see [65] for

more details). The solution is thus to find efficient heuristics. A solution is implemented in the configuration tools of the Volcano MW (see Section III-B2), but the algorithms implemented in this commercial product are not published. Several heuristics are presented in [64] to build a set of frames over CAN that minimizes the bandwidth consumption but without explicitly searching for a feasible solution. In particular, the problem of deciding the priorities of the frames is not addressed. Two heuristics are proposed in [65]. The first one is a greedy algorithm inspired by bin-packing approaches; its complexity allows using it on large size problems. The second heuristic explores the solution's space more extensively; it can only be applied to problems of limited size but, in this case, it is slightly more efficient. More recently, a greedy algorithm that proved to be efficient on the authors' experiments was proposed in [66].

2) Frame Packing Over a Time-Triggered Protocol: In TTP/C, the transmission order inside a round can be freely chosen by the application designer. Among the criteria for constructing the TDMA round, applicative constraints like computation times and sampling rates have to be taken into account. But, as shown in [67], [68], the robustness of a TDMA-based system against transmission errors depends heavily on the location of the slots inside the round.

In automotive systems, one observes that transmission errors can be correlated: there occur perturbations that corrupt several consecutive frames (so-called bursts of errors). Should two frames that belong to the same FTU (see Section III-B3) be transmitted just one after the other, then a single perturbation could corrupt both of the frames. The objective to pursue depends on the status of the FTU with regard to the concept of "fail-silence" (basically, a node is said fail-silent if one can safely consume the data that we have received from this node; see [45] and [69] for more details on that subject). For FTUs composed of a set of fail-silent nodes, the successful transmission of one single frame for the whole set of replicas is sufficient, since the value carried by the frame is necessarily correct. In this case, the objective to achieve with regard to the robustness against transmission errors is minimizing the probability that all frames of the FTU (carrying data corresponding to the same production cycle) be corrupted. This probability is denoted *P_all* in [67]. In practice, replicated sensors may return slightly different observations and, without extra communication for an agreement, replicated nodes of a same FTU may transmit different data. If a decision, such as a majority vote, has to be taken by a node with regard to the value of the transmitted data, the objective is to maximize probability that at least one frame of each FTU is successfully transmitted during a production cycle. If the production cycle is equal to one round then it comes back to minimizing *P_one*, the probability that one or more frames of an FTU have become corrupted. It was shown in [67], with some reasonable assumptions on the error model, that the optimal solution to minimize *P_all* is to "spread" the different frames of a same FTU uniformly over the TDMA round. An algorithm that ensures the optimal solution is provided for the case where the FTUs have, at most, two different cardinalities (for instance, one FTU is made of two replicas, and other FTUs are made of three replicas). For the other cases, a low-complexity heuristic is proposed [67], and it was proven to be close to the optimal on simulations that were performed. In [68], it was demonstrated that, under very weak assumptions on the error model and whatever the number of FTUs and their cardinalities, the clustering together of the transmission of all the frames of an FTU minimizes *P_one* when the production cycle of the data sent is equal to the length of a TDMA round. These two results, for the fail-silent case and non-fail-silent case, provide simple guidelines for the application designer in designing the schedule of transmission.

IV. Open Issues for Automotive Communication Systems

A. Optimized Networking Architectures

The traditional partitioning of the automotive application into several distinct functional domains with their own characteristics and requirements is useful in mastering the complexity, but this leads to the development of several independent subsystems with their specific architectures, networks, and software technologies.

Some difficulties arise from this partitioning, since more and more cross-domain data exchanges are needed. This requires implementing gateways whose performances in terms of CPU load and impact on data freshness have to be carefully assessed. For instance, an ECU belonging, from a functional point of view, to a particular domain can be connected, for wiring reasons, onto a network of another domain. For example, the diesel particulate filter (DPF) is connected onto the body network in some vehicles even though it belongs, from a functional standpoint, to the powertrain. This can raise performance problems, since the DPF needs a stream of data with strong temporal constraints coming from the engine controller located on the powertrain network. Numerous other examples of cross-domain data exchanges can be cited such as the engine controller (powertrain) that takes input from the climate control (body) or information from the powertrain displayed on the dashboard (body). There are also some functions that one can consider as being cross domains such as the immobilizer, which belongs both to the body and powertrain domains. Upcoming x-by-wire functions will also need very tight cooperation between the ECUs of the chassis, the powertrain, and the body.

A current practice is to transfer data between different domains through a gateway usually called the "central body electronic," belonging to the body domain. This subsystem is recognized as being critical in the vehicle: it constitutes a single point of failure, its design is overly complex, and performance problems arise due to an increasing workload.

An initial foreseeable domain of improvement is to further develop the technologies needed for the interoperability between applications located on different subnetworks. In particular, much work remains to be done in the area of MW, since existing ones are far from the desirable characteristics listed in Section III-A.

Future work should also be devoted to optimizing networking architectures. This implies rethinking the current practice that consists of implementing networks on a per-domain basis. The use of technologies that could fulfill several communication requirements (e.g., high-speed, event-triggered, and time-triggered communication, all with FlexRay) with scalable performances is certainly one possible direction for facilitating design.

B. System Engineering

The verification of the performances of a communication system is twofold. On the one hand, some properties of the communication system services can be proved independently of the application. For instance, the correctness of the synchronization and the membership and clique avoidance services of TTP/C have been studied using formal methods in [48] and [70].

There are other constraints whose fulfillment cannot be determined without a precise model of the system. This is typically the case for real-time constraints on tasks and signals where the patterns of activations and transmissions have to be identified. Much work has already been done in this field during the last ten years: schedulability analysis on priority buses [30], joint schedulability analysis of tasks and messages [71], probabilistic assessment of the reliability of communications under EMI [34], [72], etc. What is now needed is to extend these analyzes to take into account the peculiarities of the platforms in use (e.g., overheads due to the OS and the stack of communication layers) and to integrate them in the development process of the system. The problem is complicated by the development process being shared between several partners (the carmaker and various third-party suppliers). Ways have to be found to facilitate the integration of components developed independently and to ensure their interoperability.

In terms of the criticality of the involved functions, future automotive x-by-wire systems can reasonably be compared with flight-by-wire systems in the avionic field. According to [73], the probability of encountering a critical safety failure in vehicles must not exceed $5 \cdot 10^{-10}$ per hour and per system, but other studies consider 10^{-9}. It will be a real challenge to reach such dependability, in particular because of the cost constraints. It is certain that the know-how gathered over the years in the avionic industry can be of great help, but design methodologies adapted to the automotive constraints have to be developed.

The first step is to develop technologies able to integrate different subsystems inside a domain (see Section III), but a real challenge is to shift the development process from subsystem integration to a complete integrated design process. The increasing amount of networked control functions inside in-car embedded systems leads to developing specific design processes based, among others, on formal analysis and verification techniques of both dependability properties of the networks and dependability requirements of the embedded application.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their helpful criticisms and suggestions.

REFERENCES

[1] A. Albert, "Comparison of event-triggered and time-triggered concepts with regards to distributed control systems," presented at the Embedded World Conf. 2004, Nürnberg, Germany, 2004.
[2] G. Leen and D. Heffernan, "Expanding automotive electronic systems," *IEEE Comput.*, vol. 35, no. 1, pp. 88–93, Jan. 2002.
[3] K. Johansson, M. Törngren, and L. Nielsen, *Handbook of Networked and Embedded Control Systems*, D. Hristu-Varsakelis and W. S. Levine, Eds. Boston, MA: Birkhäuser, 2005.
[4] F. Simonot-Lion, "In-car embedded electronic architectures: how to ensure their safety," presented at the 5th IFAC Int. Conf. Fieldbus Systems and Their Applications (FeT 2003), Aveiro, Portugal, 2003.
[5] C. Wilwert, N. Navet, Y.-Q. Song, and F. Simonot-Lion, "Design of automotive X-by-Wire systems," in *The Industrial Communication Technology Handbook*, R. Zurawski, Ed. Boca Raton, FL: CRC, 2004.
[6] M. Ayoubi, T. Demmeler, H. Leffler, and P. Köhn, "X-by-Wire functionality, performance and infrastructure," presented at the Convergence Conf. 2004, Detroit, MI.
[7] J. Rushby, "A Comparison of Bus Architecture for Safety-Critical Embedded Systems," NASA/CR, Tech. Rep. NASA/CR-2003-212161, Mar. 2003.
[8] M. Krug and A. V. Schedl, "New demands for in-vehicle networks," in *Proc. 23rd EUROMICRO Conf. '97*, pp. 601–605.
[9] S. Poledna, W. Ettlmayr, and M. Novak, "Communication bus for automotive applications," presented at the 27th Eur. Solid-State Circuits Conf., Villach, Austria, 2001.
[10] K. Ramaswamy and J. Cooper, "Delivering multimedia content to automobiles using wireless networks," presented at the Convergence Conf. 2004, Detroit, MI.
[11] Ford Motor Co. (2001) Ford to study in-vehicle electronic devices with advanced simulators. [Online]. Available: http://media.ford.com/article_display.cfm?article_id=7010
[12] A. Avizienis, J. Laprie, and B. Randell, "Fundamental concepts of dependability," in *Proc. 3rd Information Survivability Workshop*, 2000, pp. 7–12.
[13] "J2056/2 survey of known protocols," in *SAE Handbook*. Warrendale, PA: Soc. Automotive Eng. (SAE), 1994, vol. 2.
[14] "J2056/1 class C application requirements classifications," in *SAE Handbook*. Warrendale, PA: Soc. Automotive Eng. (SAE), 1994.
[15] Intel Corp.. (2004) Introduction to in-vehicle networking. [Online]. Available: http://support.intel.com/design/auto/autolxbk.htm
[16] LIN Consortium. (2003, Sep.) LIN Specification Package, Revision 2.0. [Online]. Available: http://www.lin~subbus.org/
[17] A. Rajnák, *The Industrial Communication Technology Handbook*, R. Zurawski, Ed. Boca Raton, FL: CRC, 2005.
[18] H. Kopetz *et al.*, *Specification of the TTP/A Protocol*. Vienna, Austria: Univ. Technol. Vienna, 2002.
[19] *Class B Data Communications Network Interface—SAE J1850 Standard—Rev. Nov. '96*.
[20] *Road Vehicles—Low Speed Serial Data Communication—Part 2: Low Speed Controller Area Network*, ISO 11 519-2, 1994.
[21] *Road Vehicles—Interchange of Digital Information—Controller Area Network for High-Speed Communication*, ISO 11 898, 1994.
[22] MOST Cooperation. (2004, Aug.) MOST Specification Revision 2.3. [Online]. Available: http://www.mostnet.de
[23] TTTech Computertechnik GmbH. (2003, Nov.) Time-Triggered Protocol TTP/C, High-Level Specification Document, Protocol Version 1.1. [Online]. Available: http://www.tttech.com
[24] FlexRay Consortium. (2004, Jun.) FlexRay Communication System, Protocol Specification, Version 2.0. [Online]. Available: http://www.flexray.com
[25] D. Millinger and R. Nossal, *The Industrial Communication Technology Handbook*, R. Zurawski, Ed. Boca Raton, FL: CRC, 2005.
[26] P. Bühring, "Safe-by-wire plus: bus communication for the occupant safety system," presented at the Convergence Conf. 2004, Detroit, MI.
[27] *Road Vehicles—Controller Area Network (CAN)—Part 4: Time-Triggered Communication*, ISO 11 898-4, 2000.
[28] J. Ferreira, P. Pedreiras, L. Almeida, and J. A. Fonseca, "The FTT-CAN protocol for flexibility in safety-critical systems," *IEEE Micro (Special Issue on Critical Embedded Automotive Networks)*, vol. 22, no. 4, pp. 46–55, July–Aug. 2002.
[29] P. Koopman, "Critical embedded automotive networks," *IEEE Micro (Special Issue on Critical Embedded Automotive Networks)*, vol. 22, no. 4, pp. 14–18, July–Aug. 2002.

[30] K. Tindell, A. Burns, and A. J. Wellings, "Calculating Controller Area Network (CAN) message response times," *Control Eng. Practice*, vol. 3, no. 8, pp. 1163–1169, 1995.
[31] N. Navet and Y.-Q. Song, "Validation of real-time in-vehicle applications," *Comput. Ind.*, vol. 46, no. 2, pp. 107–122, Nov. 2001.
[32] Y. Martin, "L'avenir de l'automobile tient à un fil," *L'argus de l'automobile*, vol. 3969, pp. 22–23, Mar. 2005.
[33] CAN in Automation. (2005) Challenges in automotive applications. [Online]. Available: http://www.can-cia.org/applications/passengercars/challenge.html
[34] B. Gaujal and N. Navet, "Fault confinement mechanisms on CAN: analysis and improvements," *IEEE Trans. Veh. Technol.*, to be published.
[35] M. Barranco, G. Rodriguez-Navas, J. Proenza, and L. Almeida, "CANcentrate: an active star topology for can networks," presented at the 5th Int. Workshop Factory Communication System, Vienna, Austria, 2004.
[36] G. Lima and A. Bums, "Timing-independent safety on top of CAN," presented at the 1st Int. Workshop Real-Time LAN's in the Internet Age , Vienna, Austria, 2002.
[37] ——, "A consensus protocol for CAN-based systems," in *Proc. 24th Real-Time Systems Symp.*, 2003, pp. 420–429.
[38] G. Rodriguez-Navas, M. Barranco, and J. Proenza, "Harmonizing dependability and real time in CAN networks," presented at the 2nd Int. Workshop Real-Time LANs in the Internet Age, Porto, Portugal, 2003.
[39] J. Ferreira, L. Almeida, J. Fonseca, G. Rodriguez-Navas, and J. Proenza, "Enforcing consistency of communication requirements updates in FTT-CAN," presented at the Int. Workshop Dependable Embedded Systems, Florence, Italy, 2003.
[40] G. Rodriguez-Navas and J. Proenza, "Clock synchronization in CAN distributed embedded systems," presented at the 3rd Int. Workshop Real-Time Networks, Catania, Italy, 2004.
[41] L. M. Pinho and F. Vasques, "Reliable real-time communication in can networks," *IEEE Trans. Comput.*, vol. 52, no. 12, pp. 1594–1607, Dec. 2003.
[42] L.-B. Fredriksson, "CAN for critical embedded automotive networks," *IEEE Micro*, vol. 22, no. 4, pp. 28–35, July–Aug. 2002.
[43] M. Waern, "Evaluation of protocols for automotive systems," M.S. thesis, KTH Machine Design, Stockholm, Sweden, 2003.
[44] *Road Vehicles—Low Speed Serial Data Communication—Part 3: Vehicle Area Network (VAN)*, ISO 11 519-3, 1994.
[45] H. Kopetz, *Real-Time Systems: Design Principles for Distributed Embedded Applications*. Norwell, MA: Kluwer, 1997.
[46] H. Kopetz, R. Nossal, R. Hexel, A. Krüger, D. Millinger, R. Pallierer, C. Temple, and M. Krug, "Mode handling in the time-triggered architecture," presented at the IFAC-DCCS '97, Seoul, Korea, 1997.
[47] TTA Group. (2004) TTP—Frequently asked questions. [Online]. Available: http://www.ttagroup.org/technology/faq.htm
[48] G. Bauer and M. Paulitsch, "An investigation of membership and clique avoidance in TTP/C," in *Proc. 19th IEEE Symp. Reliable Distributed Systems*, 2000, pp. 118–124.
[49] H. Pfeifer, "Formal verification of the TTP group membership algorithm," presented at the FORTE/PSTV 2000, Pisa, Italy.
[50] G. Cena and A. Valenzano, "Performance analysis of byteflight networks," in *Proc. IEEE Workshop Factory Communication Systems 2004*, pp. 157–166.
[51] OSEK Consortium. (2001, Jul.) OSEK/VDX Fault-Tolerant Communication, Version 1.0. [Online]. Available: http://www.osek-vdx.org/
[52] Robert Bosch GmbH. (2004) Time Triggered Communication on CAN. [Online] . Available: http://www.can.bosch.com/content/TT_CAN.html
[53] B. Müller, T. Führer, F. Hartwich, R. Hugel, and H. Weiler, "Fault tolerant TTCAN networks," presented at the 8th Int. CAN Conf. (iCC), Las Vegas, NV, 2002.
[54] *Road Vehicles—Diagnostics on CAN—Part 2: Network Layer Services*, ISO 15 765-2, 1999.
[55] *Road Vehicles—Diagnostics on CAN—Part 1: General Information*, ISO 15 765-1, 1999.
[56] *Road Vehicles—Diagnostics on CAN—Part 3: Application Layer Services*, ISO 15 765-3, 1999.
[57] CAN calibration protocol, version 2.1, ASAP Task Force, 1999.
[58] OSEK/VDX Communication, Version 3.0.3, OSEK Consortium. (2004, Jul.). [Online]. Available: http://www.osek-vdx.org/
[59] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg, "Volcano—A revolution in on-board communications," Volvo, Tech. Rep. 98-12-10, 1999.
[60] OSEK Consortium. (2004, Jul.) OSER/VOX System Generation—OIL: OSEK Implementation Language, Version 2.5. [Online]. Available: http://www.osek-vdx.org/
[61] ——, (2004, July) OSEK/VDX Operating System, Version 2.2.2. [Online]. Available: http://www.osek-vdx.org
[62] P. Feller, "Real-time application development with OSEK—A review of OSEK standards," Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, Tech. Note CMU/SEI 2003-TN-004, 2003.
[63] A. Rajnák and M. Ramnefors, "The volcano communication concept," presented at the Convergence Conf. 2002, Detroit, MI.
[64] C. Norström, K. Sandström, and M. Ahlmark, "Frame packing in real-time communication," Mälardalen Real-Time Res. Ctr., Tech. Rep. 00-07-25, 2000.
[65] R. Santos Marques, N. Navel, and F. Simonot-Lion, "Frame packing under real-time constraints," in *Proc. 5th IFAC Int. Conf. Fieldbus Systems and Their Applications—FeT'2003*, pp. 185–192.
[66] R. Saket and N. Navet. (2003) Frame packing algorithms for automotive applications. INRIA. [Online]. Available: http://www.inria.fr/rrrt/rr-9998.html
[67] B. Gaujal and N. Navet. (2002) Maximizing the robustness of TDMA networks with application to TTP/C. INRIA. [Online]. Available: http://www.inria.fr/rrrt/rr-9619.html
[68] ——, "Optimal replica allocation for TTP/C based systems," presented at the 5th IFAC Int. Conf. Fieldbus Systems and Their Applications—FeT'2003, Aveiro, Portugal.
[69] S. Poledna, *Fault-Tolerant Real-Time Systems: The Problem of Replica Determinism*. Norwell, MA: Kluwer, 1996.
[70] H. Pfeifer and F. W. von Henke, "Formal analysis for dependability properties: the time-triggered architecture example," in *Proc. 8th IEEE Int. Conf. Emerging Technologies and Factory Automation (ETFA 2001)*, pp. 343–352.
[71] K. Tindell and J. Clark, "Holistic schedulability analysis for distributed hard real-time systems," *Microprocess. Microprogram.*, vol. 40, pp. 117–134, 1994.
[72] N. Navet, Y.-Q. Song, and F. Simonot, "Worst-case deadline failure probability in real-time applications distributed over CAN (controller area network)," *J. Syst. Arch.*, vol. 46, no. 7, pp. 607–618, 2000.
[73] "X-by-wire—Safety related fault tolerant systems in vehicles (final report)," X-by-Wire Project, Brite-EuRam 111 Program, Rep. No. XByWire-DB-6/6-24, 1998.

Nicolas Navet received the M.S. degree in computer science from the University of Berlin, Berlin, Germany, in 1994 and the Ph.D. degree in computer science from the University of Nancy, Nancy, France, in 1999.

Before joining the INRIA, LORIA Laboratory, Nancy, France, in November 2000, he was a Research Scientist at Gemplus Software. His research interests include scheduling theory, the design of communication protocols for real-time and fault-tolerant data transmission, and probabilistic risk evaluation when transient faults may occur (e.g., electromagnetic interference). More information on his work can be found at http://www.loria.fr/~nnavet.

YeQiong Song received the B.S. degree in telecommunications and computer science from Beijing University of Posts and Telecommunication, Beijing, China, in 1984, the M.Sc. degree in telecommunications and computer science from University of Paris 6, Paris, France, in 1988, the Ph.D. degree in telecommunications and computer science from the Institut National Polytechnique de Lorraine, Lorraine, France, in 1991, and the Habilitation to Lead Research (HdR) degree in telecommunications and computer science from University of Henri Poincaré Nancy 1, Nancy, France, in 2004.

Since 1992, he has been an Associate Professor at University of Henri Poincaré Nancy 1 and a Member of the TRIO research group, LORIA laboratory, Vandoeuvre-lés-Nancy, France, since 1988. He was also a Full-Time Researcher from 2001 to 2003 at INRIA Lorraine, Lorraine, France. His research interests include on the one hand the modeling and performance evaluation of networks and real-time distributed systems using queueing analysis, network calculus, and scheduling theory, and on the other hand, the implementation of real-time QoS mechanisms in fieldbuses, in-vehicle networks, switched Ethernet, IP networks, and power line communication networks.

Françoise Simonot-Lion received the Habilitation to Lead Researches (HdR) degree in computer science from the University of Nancy, Nancy, France, in 1999

Since 1997, she has been the scientific team leader of TRIO, an INRIA research project team (Real Time and InterOperability), at LORIA Laboratory, Nancy, France, and was for four years (2001–2004) responsible of CARAMELS, a Technological Research Team, granted by the ministry for Research and Technology and associated with PSA Peugeot Citroën. She participated in the French project Embedded Electronic Architecture (AEE 1999–2001) and in the European project ITEA EAST-EEA (2001–2004), whose purpose was to define a layered software architecture focused on a middleware concept and a common architecture description language supporting the formal description of in-vehicle embedded systems (EAST-ADL). She is currently Professor in Computer Science at the Institut National Polytechnique de Lorraine (INPL), Nancy, France. Since 2004, she is responsible for the "Safe Design for Embedded and Ambient Systems" cursus at Ecole des Mines de Nancy—INPL. Her main research topics are, on the one hand, modeling and verification techniques for the design of optimized real time distributed applications under safety constraints, and on the other hand, specification of embedded services ensuring a real-time QoS (scheduling of tasks and messages, real-time middleware, frame packing).

Cédric Wilwert received the Ph.D. degree in Computer Science at the LORIA Laboratory (Nancy, France) in 2005 for work on the dependability of x-by-wire automotive applications.

He is now working for the French carmaker PSA Peugeot Citroën, La Garenne-Colombes Cedex, France, on the dependability of electronic engine control systems.