A Review of Embedded Automotive Protocols

Nicolas Navet[1], Françoise Simonot-Lion[2]

April 14, 2008

INRIA - RealTime-at-Work [1]
LORIA - Nancy Université [2]
Campus Scientifique, BP 239
54506 Vandoeuvre-lès-Nancy - France
Contact author: Nicolas Navet (Nicolas.Navet@loria.fr)

Abstract

The use of networks for communications between the Electronic Control Units (ECU) of a vehicle in production cars dates from the beginning of the 90s. The specific requirements of the different car domains have led to the development of a large number of automotive networks such as LIN, J1850, CAN, FlexRay, MOST, etc.. This paper first introduces the context of in-vehicle embedded systems and, in particular, the requirements imposed on the communication systems. Then, a review of the most widely used, as well as the emerging automotive networks 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, with a special focus on the proposals of the AUTOSAR consortium. Finally, we highlight future trends in the development of automotive communication systems.

1 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 occupants 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 ABS (Anti-lock Braking System), ESP (Electronic Stability Program), EPS (Electric Power Steering), 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 equipments (e.g., radio, DVD, hand-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 subsystem composed of a micro-controller 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 90s, 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 [31]) 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 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 [28] that the number of CAN nodes sold per year is currently around 400 millions (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 2.

Car domains and their evolution As all the functions embedded in cars do not have the same performance or safety needs, different Quality of Services (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 (see chapter Françoise Simonot-Lion). 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 micro-controllers with high computing power. In order to cope with the diversity of critical tasks to be treated, multi-tasking 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 slowly 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 [67] and, in particular, for mastering the interferences between functions [4]. Chassis and powertrain functions operate mainly as closed-loop control systems and their implementation is moving towards a time-triggered approach [56, 30, 49, 46], 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, 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 leads to the design of low-cost networks such as LIN and TTP/A (see section 2). 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 leads to a hierarchical network architecture where integrated mechatronic sub-systems based on low-cost networks are interconnected through a CAN backbone. The activation of body functions is mainly triggered by the driver and passengers' solicitations (e.g. opening a window, locking doors, etc).

Telematics functions are becoming more and more numerous: hand-free phones, car radio, CD, DVD, in-car navigation systems, rear seat entertainment, remote vehicle diagnostic, 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, [54]). Here, the emphasis shifts from messages and tasks subject to stringent deadline constraints to multimedia data streams, bandwidth sharing, multimedia quality of service where preserving the integrity (*i.e.,* ensuring that information will not be accidentally or maliciously altered) and confidentiality of information is cru-

cial. HMI aims to provide Human Machine Interfaces that are easy to use and that limit the risk of driver inattention [16].

Electronic-based systems for ensuring the safety of the occupants are increasingly embedded in vehicles. Examples of such systems are: impact and roll-over sensors, deployment of airbags and belt pretensioners, tyre pressure monitoring or Adaptive Cruise Control (or ACC - the car's speed is adjusted to maintain a safe distance with 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 [60, 59, 11] based on data transmission speed and functions that are distributed over the network. *Class A* networks have a data rate lower than 10 Kbit/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 [52, 33] and TTP/A [21]. *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 Kbit/s to 125 Kbit/s. The J1850 [61] and low-speed CAN [23] are the main representations of this class. Applications that need high speed real-time communications require *class C* networks (speed of 125Kbit/s to 1Mbit/s) or *class D* networks[3] (speed over 1Mb/s). Class C networks, such as high-speed CAN [25], are used for the powertrain and currently for the chassis domains, while class D networks are devoted to multimedia data (e.g., MOST [36]) and safety critical applications that need predictability and fault-tolerance (e.g., TTP/C [65] or FlexRay [10] networks) or serve as gateways between sub-systems (see [58]).

It is common, in today's vehicles, that the electronic architecture include

[1] For instance, in [4], the average bandwidth needed for the engine and the chassis control is estimated to reach 1500kbit/s in 2008 while it was 765kbit/s in 2004 and 122kbit/s in 1994.

[2] Dependability is usually defined as the ability to deliver a service that can justifiably be trusted, see [3] for more details.

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

four different types of networks interconnected by gateways. For example, the Volvo XC90 [28] 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 possible that a bus dedicated to Occupant Safety Systems (e.g. airbag deployment, crash sensing) such as the "Safe-by-Wire plus" [7] 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 pre-defined 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., 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 §2.1.1) 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 it it required in distributed control loops. Each frame is sched-

uled for transmission at one pre-defined interval of time, usually termed a slot, and the schedule repeats itself indefinitely. This medium access strategy is referred to as TDMA (Time Division Multiple Access). 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 aperiodic 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 run-time. 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 [65] is a purely time-triggered network but there are networks, such as TTCAN [27], FTT-CAN [15] and FlexRay, 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 have to be transmitted.

Several comparisons have been done between event-triggered and time-triggered approaches, the reader can refer to [29, 1, 15] for good starting points.

2 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.

2.1 Priority buses

To ensure at run-time 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 [63, 40] and Chapter ThomasNolte for how to compute bound 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.

2.1.1 The CAN network

CAN (Controller Area Network) is without a doubt the most widely used in-vehicle network. It was designed by Bosch in the mid 80's 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 that embeds two CAN buses with regard to the non-multiplexed Peugeot 306 [34]). Furthermore, it allows to share sensors among ECUs.

CAN on a twisted pair of copper wires became an ISO standard in 1994 [23, 25] 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 delays (see [28]). 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 500KBit/s), but it also serves as an SAE class B network for the electronics in the body domain, usually at a data rate of 125Kbit/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, whose numerical value determines the frame priority. CAN uses Non-Return-to-Zero (NRZ) bit representation with a bit stuffing of length 5. In order not

[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.

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 Non-Return-to-Zero (NRZ) bit representation where the signal level on the bus can remain constant over a longer period of time (e.g. transmission of '000000..'). 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 (5 for CAN). The receiver will apply the inverse procedure and de-stuff the frame. The standard CAN data frame (CAN 2.0A) can contain up to 8 bytes of data for an overall size of, at most, 135bits, including all the protocol overheads such as the stuff bits. The reader interested in the details of the frame format and the bus access procedure should refer to Chapter ThomasNolte. CAN bus access 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, 1Mbit/s is feasible on a 40 meter bus at maximum while 250Kbit/s can be achieved over 250 meters. To alleviate the data rate limit, and extend the lifespan of CAN further, car manufacturers are beginning to optimize the bandwidth usage by implementing "traffic shaping" strategies that are very beneficial in terms of response times, this is the subject of the Chapter MathieuGrenier in this book.

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 6 consecutive dominant bits that allows all the stations on the bus to be aware of the transmission error. The corrupted frame automatically re-enters 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 to 31 bit times. CAN possesses some fault-confinement mechanisms aimed at identifying permanent failures due to hardware dysfunctioning at the level of the micro-controller, 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 [18]) but the main drawback is that a node has to diagnose itself, which can lead to the non-detection 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 Chapter JuanPimentel. Furthermore, other faults such as the partitioning of the network into several sub-networks 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 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 Chapter JuanPimentel), but, if each proposal solves a particular problem, they have not necessarily 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, for standardizing startup procedures, implementing data segmentation or sending periodic messages (see OSEK/VDX and AUTOSAR in §3). 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 [66].

2.1.2 The VAN network

Vehicle Area Network (VAN, see [24]) 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 for years in production cars by the French carmaker PSA Peugeot-Citroën in the body domain (e.g, for the 206 model) but, as it was not adopted by the market, it was abandoned in favor of CAN.

2.1.3 The J1850 network

The J1850 [61] is an SAE class B priority bus that was adopted in the USA for communications with non-stringent real-time requirements, such as the control of body electronics or diagnostics. Two variants of the J1850 are defined: a 10.4Kbit/s single-wire version and 41.6Kbit/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 §2.3.1).

2.2 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 once 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 [56] 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, multi-access protocols based on TDMA (Time Division Multiple Access) 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 message transmissions can be used as "heartbeats" for detecting station failures. The three TDMA based networks that could serve as gateways or for supporting safety critical applications are TTP/C (see [65]), FlexRay (see §2.2.1) and TTCAN (see §2.2.2). FlexRay, which is backed by the world's automotive industry, is becoming the standard in the industry and is already used in the BMW X5 model since 2006 (see [58]). In the following, we choose not to discuss further TTP/C which, to the best of our knowledge, is no more considered for vehicles but is now used in aircraft electronic systems. However, the important experience gained over the years with TTP/C, in particular regarding fault-tolerance features (see [19]) and their formal validation (see Chapter HolgerPfeifer), will certainly be beneficial to FlexRay.

2.2.1 The FlexRay Protocol

A consortium of major companies from the automotive field is currently developing the FlexRay protocol. The core members are BMW, Bosch, Daimler, General Motors, NXP Semiconductors, Freescale Semiconductor and Volkswagen. The first publicly available specification of the FlexRay Protocol have been released in 2004, the current version of the specification [10] is available at `http://www.flexray.com`.

The FlexRay network is very flexible with regard to topology and transmission support redundancy. It can be configured as a bus, a star or multi-star. It is not mandatory that each station possesses replicated channels nor a bus guardian, even though this should be the case for critical functions such as the 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, 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 Figure 1). In the event-triggered part of the communication cycle, the protocol is FTDMA (Flexible Time Division Multiple Access): the time is divided into so-called mini-slots, each station possesses a given number of mini-slots (not necessarily consecutive) and it can start the transmission of a frame inside each of its own mini-slots. A mini-slot remains idle if the station has nothing to transmit which actually induces a loss of bandwidth (see [9] for a discussion on that topic). An example of a dynamic window is shown in Figure 2: on channel B, frames have been transmitted in mini-slots n and $n+2$ while mini-slot $n+1$ 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

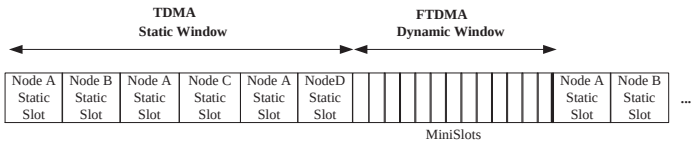

Figure 1: Example of a FlexRay communication cycle with 4 nodes A, B, C and D

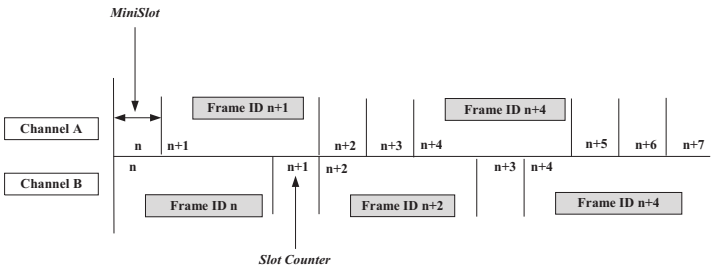

Figure 2: Example of message scheduling in the dynamic segment of the FlexRay communication cycle

for each mode are not possible.

The FlexRay frame consists of 3 parts : the header, the payload segment containing up to 254 bytes of data and the CRC of 24 bits. The header of 5 bytes 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 (i.e., which is refer to as frame-packing or PDU-multiplexing - see [57] for this problem addressed on CAN).

From the dependability point of view, the FlexRay standard specifies solely the bus guardian and the clock synchronization algorithms. Other features, such as mode management facilities or a membership service, will have to be implemented in software or hardware layers on top of FlexRay (see, for instance, [5] for a membership service protocol that could be used along with 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, 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 non-critical functions will increasingly co-exist and interoperate, this flexibility can prove to be efficient in terms of cost and re-use of existing components if missing fault-tolerance features are provided in a middleware layer, for instance such as the one currently under development within the automotive industry project AUTOSAR (see §3.3). The reader interested in more information about FlexRay can refer to Chapter Bernhard Schätz in this book, and to [50, 20] for how to configure the communication cycle.

2.2.2 The TTCAN protocol

TTCAN (Time Triggered Controller Area Network - see [27]) is a communication protocol developed by Robert Bosch GmbH on top of the CAN physical and data-link layers. TTCAN uses the CAN standard but, in addition, requires that the controllers 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 [55]. The bus topology of the network, the characteristics of the transmission support, the frame format, as well as the maximum data rate - 1Mbits/s - are imposed by CAN protocol. Channel redundancy is possible (see [35] 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 Figure 3, 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

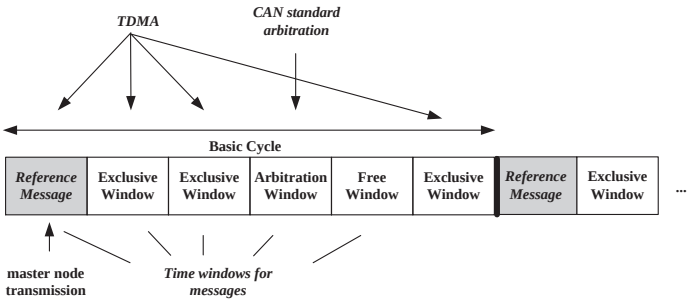

Figure 3: Example of a TTCAN Basic Cycle

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 middleware level but with reduced efficiency. Some years ago, it was thought that carmakers could be interested in using TTCAN during a transition period until FlexRay technology is fully mature but this was not really the case and it seems that the future of TTCAN in production cars is rather unsure.

2.3 Low-cost automotive networks

Several fieldbus networks have been developed to fulfill the need for low-speed / low-cost communication inside mechatronic based sub-systems 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 micro-controllers 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 sub-networks 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 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).

2.3.1 The LIN network

LIN (Local Interconnect Network, see [33, 52]) 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 2.1 [33]) includes not only the specification of the transmission protocol (physical and data link layers) for master-slave communications but also the specification of a diagnostic protocol on top of the data link layer. 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.). These 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 20Kbit/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

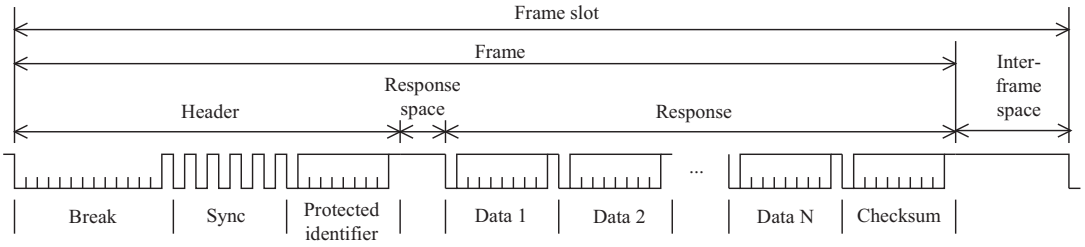

Figure 4: Format of the LIN frame. A frame is transmitted during its "frame slot" which corresponds to an entry of the schedule table

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. Figure 4 depicts the LIN frame format and the time 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 this 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 0) followed by one recessive bit (logical value 1) as a break delimiter. The rest of the frame is made of byte fields delimited by one start bit (value 0) and one stop bit (value 1), thus resulting in a 10-bit stream per byte. The "sync" byte has a fixed value (which corresponds to a bit stream of alternatively 0 and 1), 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 sub-fields: the first 6 bits are used to encode the identifier and the last two bits, the identifier parity. The data field can contain up to 8 bytes 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 [32] is the doors' knob 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 bytes, fixed value identifiers for both the master's request and the slave 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.

2.3.2 The TTP/A network

As TTP/C, TTP/A [21] was initially invented at the Vienna University of Technology. 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 on-line 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 bytes. The "Multi-partner" rounds enable several slaves to send up to an overall amount of 62 bytes 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 20Kbit/s, but other transmission supports enabling higher data rates are possible. To our best knowledge, TTP/A is not currently in use in production cars.

2.4 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 prominent protocols in this category are MOST and IDB-1394.

2.4.1 The MOST network

MOST (Media Oriented System Transport, see [36]) 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 different possible data rates. This supports end-user applications like radios, GPS navigation, video displays and entertainment systems. 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 cooper wires. Current production cars from BMW and DaimlerChrysler employ a MOST network, and MOST has now become the de-facto standard for transporting audio and video within vehicles (see [37, 38]). At the time of writing, the third revision of MOST has been announced with, as a new feature, the support of a channel that can transport standard Ethernet frames.

2.4.2 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 (see `http://www.idbforum.org`) and the 1394 Trade Association (see `http://www.`

1394ta.org). 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 100Mbps over 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 was at some time considered a serious competitor for MOST technology but, despite a few early implementations at Renault and Nissan, as far as we know the protocol did not reach wide acceptance on the market.

3 Middleware layer

3.1 Rationale for a middleware

The design of automotive electronic systems has to take into account several constraints. First, the performance, quality and safety of a vehicle depend on functions that are mainly implemented in software and moreover depend on a tight cooperation between these functions (see chapter Françoise Simonot-Lion). Second, in-vehicle embedded systems are produced through a complex cooperative multi-partner development process shared between OEMs and suppliers. In order to increase the efficiency of the production of components and their integration, two important problems have to be solved: 1) the portability of components from one Electronic Control Unit to another one enabling some flexibility in the architecture design, and 2) the reuse of components between platforms which is a keypoint especially for ECU suppliers. So the cooperative development process raises the problem of interoperability of components. A classic approach for easing the integration of software components is to implement a *middleware layer* that provides application programs with common services and a common interface. In particular, the common interface allows the design of an application disregarding the hardware platform and the distribution, and therefore enables the designer focusing on the development and the validation of the software components and the software architecture that realize a function.

Among the set of common services usually provided by a middleware, those that related to the communication between several application components are crucial. They have to meet several objectives:

- *Hide the distribution* through the availability of services and interfaces that are the same for intra-ECU, inter-ECU, inter-domain communications whatever the underlying protocols,

- *Hide the heterogeneity* of the platform (i.e., micro-controllers, protocols, Operating Systems, etc.) by providing an interface independent of the underlying protocols and of the CPU architecture (e.g., 8/16/32 bits, endianness),

- *Provide high-level services* in order to shorten the development time and increase quality through the re-use of validated services (*e.g.* working mode management, redundancy management, membership service, etc.). A good example of such a function is the"frame-packing" (sometimes also called "signal multiplexing") that enables application components to exchange *signals* (e.g. the number of revolutions per minute, the speed of the vehicle, the state of a light, etc.) while, at run-time, *frames* are transmitted over the network; so, the "frame-packing" service of a middleware consists in packing the signals into frames and sending the frames at the right points in time for ensuring the deadline constraint on each signal it contains,

- *Ensure QoS properties required by the application*, in particular, it can be necessary to improve the QoS provided by the lower-level protocols as, for example, by furnishing an additional CRC, transparent to the application, if the Hamming distance of the CRC specified by the network protocol is not sufficient in regard to the dependability objectives. Other examples are the correction of "bugs" in lower level protocols such as the "inconsistent message duplicate" of CAN (see [48] for such a proposal and chapter JuanPimentel), the provision of a reliable acknowledgment service on CAN, the status information on the data consumed by the application components (e.g., data was refreshed since last reading, its freshness constraint was not respected, etc.) or filtering mechanisms (e.g., notify the application for each k reception or when the data value has changed in a significant way).

Note that a more advanced features would be to come up with adaptive communication services, thanks to algorithms that would modify at run-time the parameters of the communication protocols (e.g., priorities, transmission fre-

quencies, etc.) according to the current requirements of the application (e.g., inner-city driving or highway driving) or changing environmental conditions (e.g., EMI level). For the time being, to the best of our knowledge, not such feature exists in automotive embedded systems. In fact, this point requires a coordinated approach for the design of function (as the definition of control law parameters, the identification of the parameters acting on the robustness of the function, etc.) and the deployment of the software architecture that implements the function (specifically the communication parameters). By increasing the efficiency and the robustness of the application, such an adaptive strategy would certainly ease the re-usability.

3.2 Automotive middlewares prior to AUTOSAR

Some carmakers possess a proprietary middleware (MW) that helps to integrate ECUs and software modules developed by their third-party suppliers.For instance, the TITUS/DBKOM communication stack, is a proprietary middleware (MW) of Daimler that standardizes the cooperation between components according to a client/server model. *Volcano* [8, 53, 51] is a commercial product of Mentor Graphics, initially developed in partnership with Volvo. The Volcano Target Package (VTP) consists of a communication layer and a set of off-line configuration tools for application distributed on CAN and / or LIN. It is aimed to provide the mapping of signals into frames under network bandwidth optimization and ensure a predictable and deterministic real time communication system thanks to schedulability analysis techniques (see [63, 8]). To the best of our knowledge, no publicly available technically precise description of TITUS and Volcano exists.

The objective of the OSEK/VDX consortium (`http://www.osek-vdx.org`) is to build a standard architecture for in-vehicle control units. Among the results of this group, two specifications are of particular interest in the context of this chapter: the *OSEK/VDX Communication* layer [43] and the *Fault-Tolerant Communication* layer [42]. The OSEK/VDX consortium (`http://www.osek-vdx.org`) specifies a communication layer [43] that defines common software interfaces and common behavior for internal and external communications between application components. At the application layer, these components exchange signals, termed "messages" in OSEK/VDX terminology, while communicating OSEK/VDX entities exchange so-called I-PDUs (Interaction Layer Protocol Data Unit) that are collections of mes-

sages. Each consumer of a message can specify it as queued or unqueued (i.e. a new value overwrites the old one) and associate it with a filtering mechanism. The emission of an I-PDU onto the network can be specified as triggered by the sending of a message that it contains or not. In the latter case, the emission of the I-PDU is asynchronous with the sending of the message. How signals are packed into a frame is statically defined off-line and the *OSEK/VDX Communication* layer automatically realizes the packing / unpacking at run-time. The characteristic of I-PDU and messages are specified through the OSEK/VDX Implementation Language (see [45]).

OSEK/VDX Communication runs on top of a transport layer (e.g., [26]) that takes care mainly 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 [44]). 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 [13] for more details); thus, the designer has to take into account the distribution of the functions which is a hindrance to portability. Finally, OSEK/VDX Communication does not obey to a Time-Triggered approach and is not intended to be used on top of a Time-Triggered (TT) network, as for example TTP/C or FlexRay. These networks already implement some features that were specified in OSEK/VDX Communication, as the time-triggered sending of I-PDU, while some that are offered by this middleware are not compatible with the TT paradigm, as the direct transmission of an I-PDU as soon as a message that it contains is sent by the application. 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* (Fault-Tolerant Communication, see [42]) is a proposal whose objective is to complete OSEK/VDX for Time-Triggered distributed architectures. One of its main functions is to manage the redundancy of data needed for achieving fault-tolerance (i.e., the same information can be produced by a set of replicated nodes) by presenting only one copy of data to the receiver application according to the agreement strategy specified by the designer. Two other important services of the FTCom, that are also provided by OSEK Communication, are 1) to manage the packing/unpacking of messages [57], which is needed if the network bandwidth has to be optimized (see §4.1), and 2) to provide message filtering mechanisms for passing only "significant" data

to the application. OSEK/VDX FTCom was developed to run on top of a time-triggered operating system (OS) such as OSEK Time [41]. In this OS, the scheduling of tasks is specified in a time table called the dispatcher table that is generated off-line. OSEK/VDX FTCom allows the OS to synchronize the start of the task schedule defined in the dispatcher table to a particular point in time in the I-PDU schedule (i.e. the TDMA round). As this point is shared by all the ECUs connected on the same network, this service can be used to synchronize distant applications.

Between 2001 and 2004, a European cooperative project aimed at the specification of an automotive MW within the automotive industry (ITEA EAST-EEA project - see `http://www.east-eea.net`) was undertaken. To the best of our knowledge, the ITEA EAST-EEA project was the first important initiative targeting the specification of both the services to be ensured by the middleware and the architecture of the middleware itself in terms of components and architecture of components. Similar objectives guide the work done in the AUTOSAR consortium, see `http://www.autosar.org` and [14, 17], that gathers most the key players in the automotive industry (see Chapter StefanVoget) . The specifications produced by the consortium become quickly de-facto standards for the cooperative development of in-vehicle embedded systems (see, for instance, the migration to AUTOSAR at PSA Peugeot-Citröen [12]).

3.3 AUTOSAR

AUTOSAR specifies the software architecture embedded in an ECU. More precisely, it provides a reference model which is comprised of three main parts:

- the application layer,
- the basic software (middleware software components),
- and the Run Time Environment (RTE) that provides standardized software interfaces to the application software.

One of AUTOSAR's main objective is to improve the quality and the reliability of embedded systems. By using a well suited abstraction, the reference model supports the separation between software and hardware, it

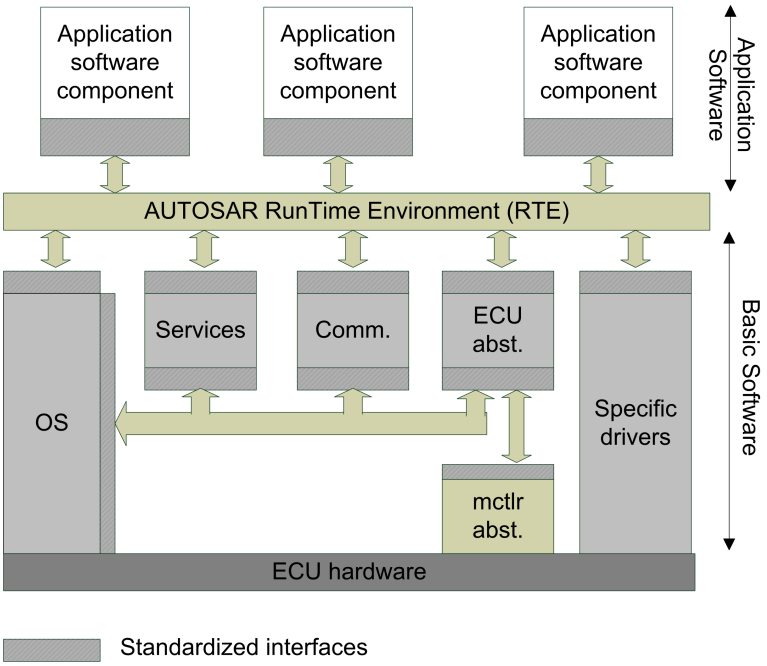

Figure 5: AUTOSAR reference architecture

eases the mastering of the complexity, allows the portability of application software components and therefore the flexibility for product modification, upgrade and update, as well as the scalability of solutions within and across product lines. The AUTOSAR reference architecture is schematically illustrated in figure 5. An application software component is compliant with AUTOSAR if its code only calls entry points defined by the RTE. Furthermore, a basic software component used at the middleware layer has to be of one of the type defined in AUTOSAR; it is AUTOSAR compliant if it provides the services and the interface formally defined in the specification of its type. The generation of an AUTOSAR middleware is done from the basic software components, generally provided by suppliers, and the specification of the application itself (description of applicative-level tasks, signals sent or received, events, alarms, etc.). Therefore its deployment can be optimized for each ECU.

One of the main objectives of the AUTOSAR middleware is to hide the characteristic of the hardware platform as well as the distribution of the application software components. Thus the inter- or intra-ECU communication services are of major importance and are thoroughly described in the docu-

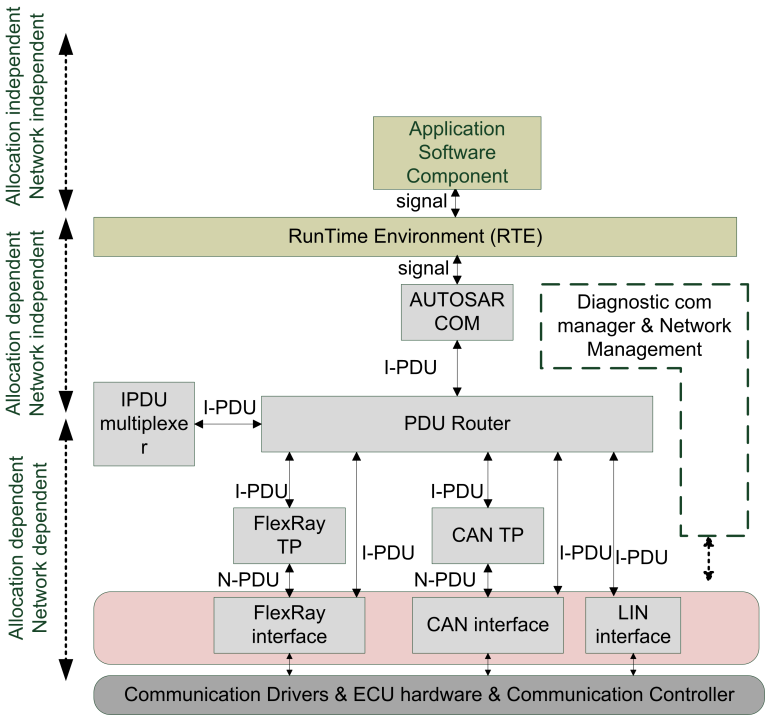

Figure 6: Communication software components and architecture

ments provided by the AUTOSAR consortium (see figure 6 for an overview of the different modules). The role of these services is crucial for the behavioral and temporal properties of an embedded and distributed application. So, their design and configuration have to be precisely mastered and the verification of timing properties becomes an important activity. The problem is complex because the objects (e.g., signals, frames, I-PDU, etc.) that are handled by services at one level are not the same objects that are handled by services at another level. Nevertheless each object is strongly dependent of one or several objects handled by services belonging to neighboring levels. The AUTOSAR standard proposes two communication models:

- "sender-receiver" used for passing information between two application software components (belonging to the same task, to two distinct tasks on the same ECU or to two remote tasks),
- "client-server" that supports function invocation.

Two communication modes are supported for the "sender-receiver" communication model:

- the "explicit" mode is specified by a component that makes explicit calls to the AUTOSAR middleware for sending or receiving data,
- the "implicit" mode means that the reading (resp. writing) of data is automatically done by the middleware before the invocation (resp. after the end of execution) of a component consuming (resp. producing) the data without any explicit call to AUTOSAR services.

AUTOSAR identifies three main objects regarding the communication: signal exchanged between software components at application level, I-PDU (Interaction Layer Protocol Data Unit) that consists of a group of one or several signals, and the N-PDU (Data Link Layer Protocol Data Unit) that will actually be transmitted on the network. Precisely AUTOSAR defines:

- *signals* at application level that are specified by a length and a type. Conceptually a signal is exchanged between application software components through ports disregarding the distribution of this component. The application needs to precise a *Transfer Property* parameter that will impact the behavior of the transmission:
  - the value "*triggered*" for this parameter indicates that each time the signal is provided to the middleware by the application, it has to be transmitted on the network (as we will see later, this means that the sending of the frame containing this signal is directly done after the emission of the signal by the application component),
  - on the contrary, the value "*pending*" for a signal indicates that its actual transmission on the network depends only on the emission rule of the frame that contains the signal.

  Furthermore, when specifying a signal, the designer has to indicate if it is a *data* or an *event*. In the former case, incoming data are not queued on the receiver side: when a new value arrives, it erases the previous value of the same signal. The latter case specifies that signals are queued on the receiver side and therefore, it ensures that for each transmission of the signal, a new value will be made available to the application. The handling of buffers or queues is done by the RTE.
- I-PDU are built by the AUTOSAR COM component. Each I-PDU is made of one or several signals and is passed via the PDU Router

to the communication interfaces. The maximum length of an I-PDU depends on the maximum length of the L-PDU (i.e., Data Link Layer PDU) of the underlying communication interface: for CAN and LIN the maximum L-PDU length is 8 bytes while for FlexRay the maximum L-PDU length is 254 bytes. AUTOSAR COM ensures a local transmission when both components are located on the same ECU, or by building suited objects and triggering the appropriate services of the lower layers when the components are remote. This scheme enables the portability of components and hide their distribution. The transformation from signals to I-PDU and from I-PDU to signals is done according to an off-line generated configuration. Each I-PDU is characterized by a behavioral parameter, termed *Transmission Mode* with different possible values:

  - "direct" indicates that the sending of the I-PDU is done as soon as a "triggered" signal contained in this I-PDU is sent at application layer,
  - "periodic" means that the sending of the I-PDU is done only periodically - it imposes that the I-PDU does not contain "triggered" signals,
  - "mixed" means that the rules imposed by the "triggered" signals contained in the I-PDU are taken into account, and additionally the I-PDU is sent periodically if it contains at least one "pending" signal,
  - "none" characterizes I-PDUs whose emission rules depend on the underlying network protocol (e.g., FlexRay) and no transmission is initiated by AUTOSAR COM in this mode.

- an N-PDU is built by the basic components CAN TP (Transport Protocol) or FlexRay TP. It consists of the data payload of the frame that will be transmitted on the network and protocol control information. Note that the use of a transport layer is not mandatory and I-PDUs can be transmitted directly to the lower layers (see figure 6). When a transport layer is used, an N-PDU is obtained by:

  - splitting the I-PDU so as to obtain several N-PDUs that are compliant with the frame data payload length,

– or assembling several I-PDUs into one N-PDU.

The RTE (Run Time Environment) implements the AUTOSAR middleware interface and the corresponding services. In particular, the RTE handles the *implicit*/*explicit* communication modes and the fact that the communication involves *events* (queued) or *data* (unqueued). Figure 7 illustrates how the transmission of a signal S between two remote application components (ASC-S on the sender side and ASC-R on the receiver one) is handled by the RTE and the COM components. Signal S is assumed to be a data, therefore it is not queued, and it is received explicitly (explicit mode). On each ECU, the RTE is generated according to the specification of the signal exchanges between applicative-level components. Thus, in particular, on the receiver side, a buffer is defined in the RTE for each data that is received by this ECU. At the initialization of the system, the value of signal S at the receiver end is set to a statically defined value (in the example of figure 7, the initial value is 0). The buffer contains value 0 between $t1$ and $t3$, and value 20 from time $t3$ on. The value returned by a read call done by the application software component ASC-R on the receiver side is 0 thus at time $t2$ and 20 at time $t4$.

In figure 8, a similar example is given but this time signal S is an *event* and so, it is queued by the RTE on the receiving ECU. At time $t1$, the queue for S is initialized (queue is empty). Value 20 is queued at time $t3$, at $t4$ a read call done by the receiver application component returns 20 and the queue becomes empty. At time $t2$ and $t5$ such a read call returns an code indicating that the queue is empty.

The AUTOSAR COM component is responsible for several functions: on the sender side, it ensures the transmission and notifies the application about its outcome (success or error). In particular, AUTOSAR COM can inform the application if the transmission of an I-PDU did not take place before a specified deadline (i.e., deadline monitoring). On the receiver side, it also notifies the application (success or error of a reception) and supports the filtering mechanism for signals (dispatching each signal of a received I-PDU to the application or to a gateway). Both at the sending and receiving end, the endianness conversion is taken in charge. An important role of the COM component is to pack/unpack *signals* into/from *I-PDU*s. Note that, as the maximal length of an I-PDU depends on the underlying networks, the design

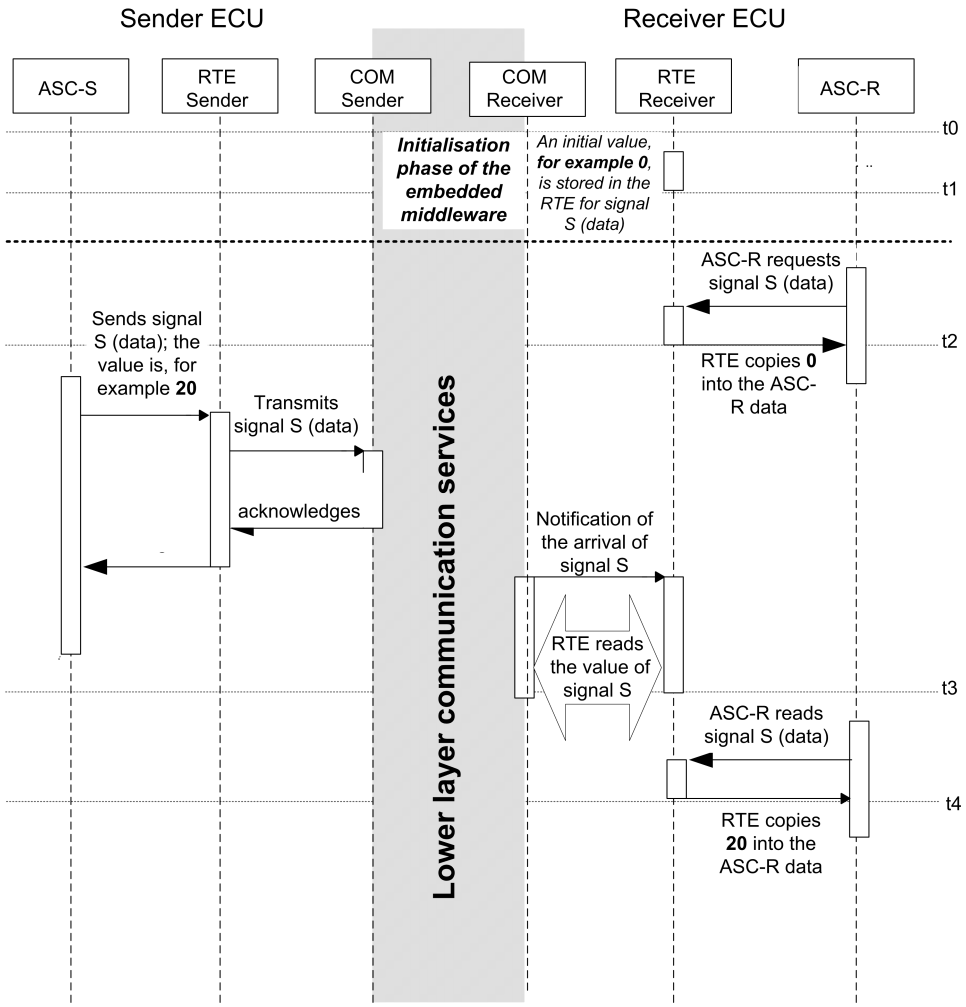

Figure 7: Sender-receiver communication model for a data signal according to the explicit communication mode

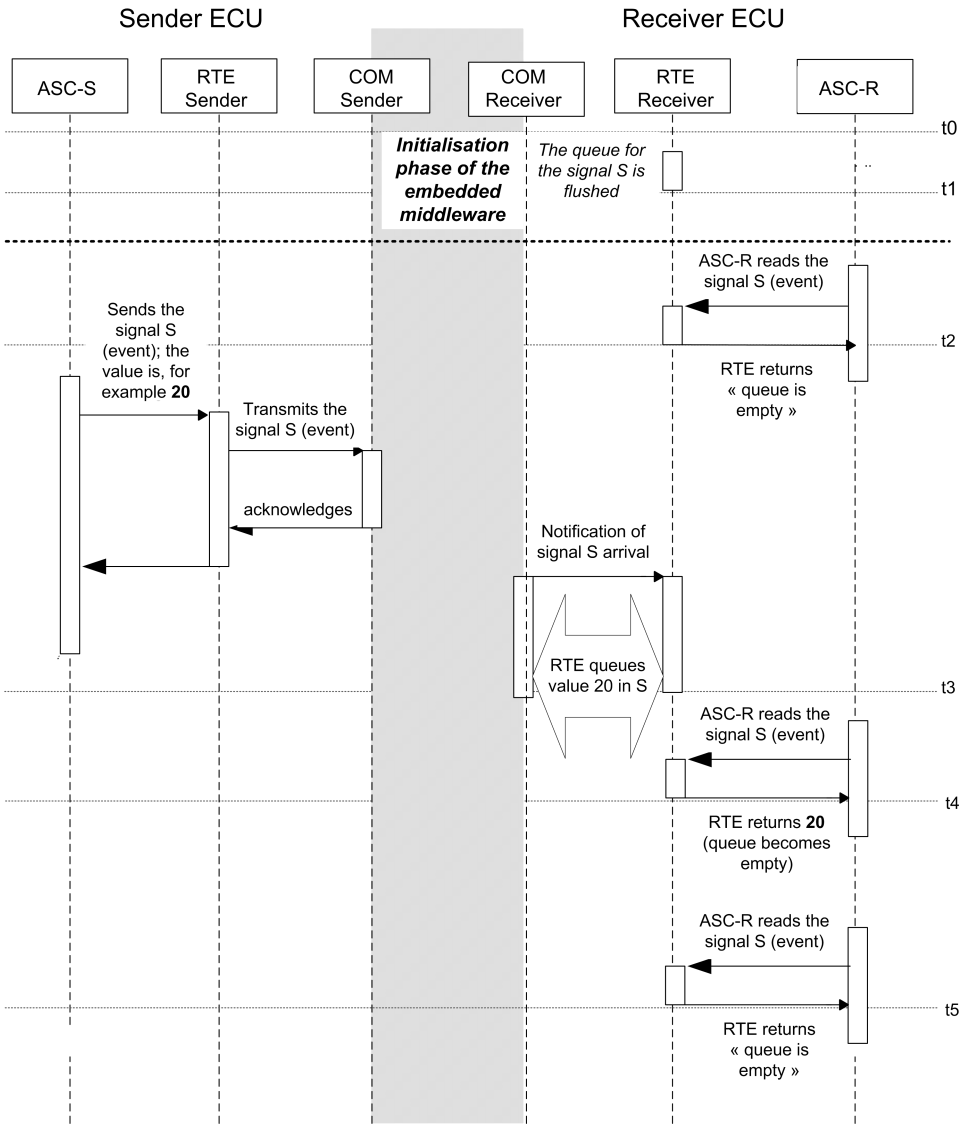

Figure 8: Sender-receiver communication model for an event signal according to the explicit communication mode

Transfer Property of the signals / Transmission mode of the I-PDU	All the signals in the I-PDU are *Triggered*	All the signals in the I-PDU are *Pending*	At least one signal is *Triggered* and one is *Pending* in the I-PDU
Direct	The transmission of the I-PDU is done each time a signal is sent		This configuration could be dangerous: if no emission of triggered signals occurs, the pending signals will never be transmitted
Periodic	The transmission of the I-PDU is done periodically	The transmission of the I-PDU is done periodically	The transmission of the I-PDU is done periodically
Mixed	The transmission of the I-PDU is done each time a signal is sent and at each period	The transmission of the I-PDU is done periodically	The transmission of the I-PDU is done each time a Triggered signal is sent and at each period

Table 1: Transmission Mode of an I-PDU vs Transfer Property of its signals

of a COM component has to take into account the networks and therefore it is not fully independent of the hardware architecture. The COM component has also to determine the points in time where to send the I-PDUs. This is based on the attributes *Transmission Mode* of an I-PDU and on the attribute *Transfer Property* of each signal that it contains. Table 1 summarizes the combinations that are possible. Note that the "none" *Transmission Mode* is not indicated in this table, in that case the transmission is driven by the underlying network layers.

As can be seen in table 1, the actual sending of an I-PDU and therefore of the signals that it contains is relevant to several rules. The figure 9 illustrates how a *Direct* I-PDU containing two signals S1 and S2 *Triggered* is transmitted. At time $t1$, $t2$, $t3$, $t4$, the sending of signal S1 or of signal S2 to the COM component triggers the emission of the I-PDU to the lower layer. In figure 10, we consider an I-PDU in which are packed signal S1 (*Triggered*) and signal S2 (*Pending*). The transmission mode of the I-PDU is set to *mixed* with period δt. Each time a new value of S1 is provided to the RTE, the I-PDU is passed to the lower layer (times *t1* and *t6*). In addition, the I-PDU is also transmitted every δt (times *t4* and $t5$). Note that, in this configuration, some values of S2 may not be transmitted, as for example the value of S2 provided at time $t2$.

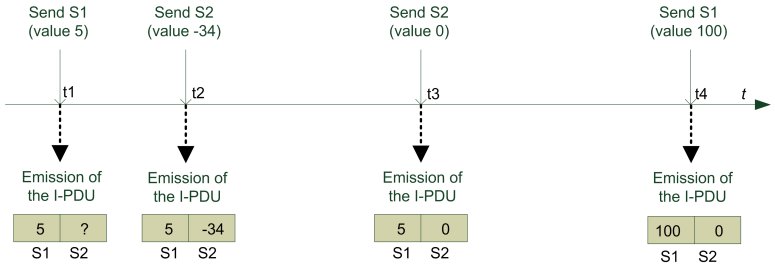

Figure 9: Transmission of an I-PDU in *Direct* mode with two *Triggered* signals

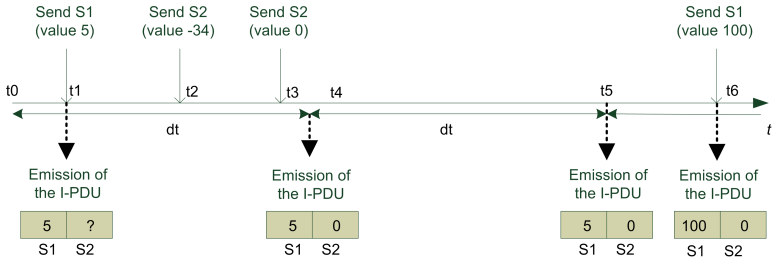

Figure 10: Transmission of an I-PDU in *mixed* mode which contains a *Triggered* signal (S1) and a *Pending* signal (S2)

The COM component is generated off-line on the basis of the knowledge of the signals, the I-PDUs and the allocation of application software components on the ECUs. The AUTOSAR PDU Router (see figure 6), according to the configuration, dispatches each I-PDU to the right network communication stack. This basic component is statically generated off-line as soon as the allocation of software components and the operational architecture is known. Other basic software components of the communication stack are responsible for the segmenting/reassembling of I-PDU(s) when needed (FlexRay TP, CAN TP) or for providing an interface to the communication drivers (FlexRay Interface, CAN Interface, LIN Interface).

4 Open issues for automotive communication systems

4.1 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 sub-systems 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 (see, for instance, [62]). 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 sub-networks. With the AUTOSAR project, significant progresses in the area of MW have been achieved over the last years and we are coming closer to the desirable characteristics listed in §3.1.

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 possible with FlexRay) with scalable

performances is certainly one possible direction for facilitating the design. Certainly, software tools, such as our tool NETCAR-Analyzer (see `http://www.realtimeatwork.com`), will be helpful to master the complexity and come up with cost and dependability-optimized solutions. The use of software along the development cycle will be facilitated by the advent of the ASAM FIBEX standard [2], in the process of being adopted by AUTOSAR, which enables to fully describe the networks embedded in a vehicle (CAN, LIN, FlexRay, MOST and TTCAN protocols), the frames that are exchanged between ECUs and the gatewaying strategies.

4.2 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 [6, 47, 5].

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 10 years: schedulability analysis on priority buses [63], joint schedulability analysis of tasks and messages [64, 22], probabilistic assessment of the reliability of communications under EMI [39, 19, 18], etc. What is now needed is to extend these analyses 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-part 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 [68], 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 4.1) 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.

References

[1] A. Albert. Comparison of event-triggered and time-triggered concepts with regards to distributed control systems. In *Proceedings of Embedded World 2004*, Nürnberg, February 2004.

[2] ASAM. FIBEX - field bus exchange format, version 3.0. Available at `http://http://www.asam.net/`, January 2008.

[3] A. Avizienis, J. Laprie, and B. Randell. Fundamental concepts of dependability. In *Proceedings of the 3rd Information Survivability Workshop*, pages 7–12, 2000.

[4] M. Ayoubi, T. Demmeler, H. Leffler, and P. Köhn. X-by-Wire functionality, performance and infrastructure. In *Proceedings of Convergence 2004*, Detroit, Michigan, 2004.

[5] R. Barbosa and J. Karlsson. Formal specification and verification of a protocol for consistent diagnosis in real-time embedded systems. In *To appear at the Third IEEE International Symposium on Industrial Embedded Systems (SIES'2008)*, June 2008.

[6] G. Bauer and M. Paulitsch. An investigation of membership and clique avoidance in TTP/C. In *Proceedings of the 19th IEEE Symposium on Reliable Distributed Systems*, 2000.

[7] P. Bühring. Safe-by-Wire Plus: Bus communication for the occupant safety system. In *Proceedings of Convergence 2004*, Detroit, Michigan, 2004.

[8] L. Casparsson, A. Rajnak, K. Tindell, and P. Malmberg. Volcano - a revolution in on-board communications. Technical report, Volvo, 1999.

[9] G. Cena and A. Valenzano. Performance analysis of Byteflight networks. In *Proceedings of the 2004 IEEE Workshop of Factory Communication Systems (WFCS 2004)*, pages 157–166, September 2004.

[10] FlexRay Consortium. FlexRay communications system - protocol specification - version 2.1. Available at `http://www.flexray.com`, December 2005.

[11] Intel Corporation. Introduction to in-vehicle networking. Available at `http://support.intel.com/design/auto/autolxbk.htm`, 2004.

[12] P.H. Dezaux. Migration strategy of in-house automotive real-time applicative software in AUTOSAR standard. In *Proceedings of the 4th European Congress Embedded Real Time Software (ERTS 2008)*, Toulouse, France, 2008.

[13] P. Feiler. Real-time application development with OSEK - a review of OSEK standards, 2003. Technical Note CMU/SEI 2003-TN-004.

[14] H. Fennel, S. Bunzel, H.Heinecke, J. Bielefeld, S. Fürstand K.P. Schnelle, W. Grote, N. Maldenerand T. Weber, F. Wohlgemuth, J. Ruh, L. Lundh, T. Sandén, P. Heitkämper, R. Rimkus, J. Leflour, A. Gilberg, U. Virnich, S. Voget, K. Nishikawa, K. Kajio, K. Lange, T. Scharnhorst, and B. Kunkel. Achievements and exploitation of the AUTOSAR development partnership. In *Convergence 2006*, Detroit, USA, October 2006.

[15] 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*, 22(4):46–55, July-August 2002.

[16] Ford Motor Company. Ford to study in-vehicle electronic devices with advanced simulators. Available at url `http://media.ford.com/article_display.cfm?article_id=7010`, 2001.

[17] S. Fürst. AUTOSAR for Safety-Related Systems: Objectives, Approach and Status. In *2nd IEE Conference on Automotive Electronics*, London, UK, March 2006. IEE.

[18] B. Gaujal and N. Navet. Fault confinement mechanisms on CAN: Analysis and improvements. *IEEE Transactions on Vehicular Technology*, 54(3):1103–1113, May 2005.

[19] B. Gaujal and N. Navet. Maximizing the robustness of TDMA networks with applications to TTP/C. *Real-Time Systems*, 31(1-3):5–31, December 2005.

[20] M. Grenier, L. Havet, and N. Navet. Configuring the communication on FlexRay: the case of the static segment. In *ERTS Embedded Real Time Software 2008*, 2008.

[21] H. Kopetz et al. *Specification of the TTP/A Protocol.* University of Technology Vienna, September 2002.

[22] R. Henia, A. Hamann, M. Jersak, R. Racu, K. Richter, and R. Ernst. System level performance analysis - the SymTA/S approach. *IEE Proceedings Computers and Digital Techniques*, 152(2):148–166, march 2005.

[23] International Standard Organization. *ISO 11519-2, Road Vehicles - Low Speed serial data communication - Part 2: Low Speed Controller Area Network.* ISO, 1994.

[24] International Standard Organization. *ISO 11519-3, Road Vehicles - Low Speed serial data communication - Part 3: Vehicle area network (VAN).* ISO, 1994.

[25] International Standard Organization. *ISO 11898, Road Vehicles - Interchange of Digital Information - Controller Area Network for high-speed Communication.* ISO, 1994.

[26] International Standard Organization. *15765-2, Road Vehicles - Diagnostics on CAN - Part 2: Network Layer Services.* ISO, 1999.

[27] International Standard Organization. *11898-4, Road Vehicles - Controller Area Network (CAN) - Part 4: Time-Triggered Communication.* ISO, 2000.

[28] K. Johansson, M. Törngren, and L. Nielsen. *Handbook of Networked and Embedded Control Systems*, chapter Vehicle Applications of Controller Area Network. D. Hristu-Varsakelis and W.S. Levine editors, Birkhäuser, Boston, Massachussets, 2005.

[29] P. Koopman. Critical embedded automotive networks. *IEEE Micro, Special Issue on Critical Embedded Automotive Networks*, 22(4):14–18, July-August 2002.

[30] M. Krug and A. V. Schedl. New demands for in-vehicle networks. In *Proceedings of the 23rd EUROMICRO Conference'97*, July 1997.

[31] G. Leen and D. Heffernan. Expanding automotive electronic systems. *IEEE Computer*, 35(1), January 2002.

[32] LIN Consortium. *LIN Specification Package, version 1.3*, December 2002. Available at `http://www.lin-subbus.org/`.

[33] LIN Consortium. *LIN Specification Package, revision 2.1*, November 2006. Available at `http://www.lin-subbus.org/`.

[34] Y. Martin. L'avenir de l'automobile tient à un fil. *L'argus de l'automobile*, 3969:22–23, March 2005.

[35] B. Müller, T. Führer, F. Hartwich, R. Hugel, and H. Weiler. Fault tolerant TTCAN networks. In *Proceedings of the 8th International CAN Conference (iCC)*, Las Vegas, Nevada, 2002.

[36] MOST Cooperation. *MOST Specification Revision 2.3*, August 2004. Available at `http://www.mostnet.de`.

[37] H. Muyshondt. Consumer and automotive electronics converge: Part 1 - Ethernet, USB, and MOST. Available at `http://www.automotivedesignline.com/`, February 2007.

[38] H. Muyshondt. Consumer and automotive electronics converge: Part 2 - a MOST implementation. Available at `http://www.automotivedesignline.com/`, March 2007.

[39] N. Navet, Y. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over CAN (Controller Area Network). *Journal of Systems Architecture*, 46(7):607–617, 2000.

[40] N. Navet and Y.-Q. Song. Validation of real-time in-vehicle applications. *Computers in Industry*, 46(2):107–122, November 2001.

[41] OSEK Consortium. *OSEKtime OS, Version 1.0*, July 2001. Available at `http://www.osek-vdx.org/`.

[42] OSEK Consortium. *OSEK/VDX Fault-Tolerant Communication, Version 1.0*, July 2001. Available at `http://www.osek-vdx.org/`.

[43] OSEK Consortium. *OSEK/VDX Communication, Version 3.0.3*, July 2004. Available at `http://www.osek-vdx.org/`.

[44] OSEK Consortium. *OSEK/VDX Operating System, Version 2.2.2*, July 2004. Available at `http://www.osek-vdx.org/`.

[45] OSEK Consortium. *OSEK/VDX System Generation - OIL: OSEK Implementation Language, Version 2.5*, July 2004. Available at `http://www.osek-vdx.org/`.

[46] M. Peteratzinger, F. Steiner, and R. Schuermans. Use of XCP on FlexRay at BMW. Translated reprint from HANSER Automotive 9/2006, available at url `https://www.vector-worldwide.com/vi_downloadcenter_en,,223.html?product=xcp`, 2006.

[47] H. Pfeifer and F.W. von Henke. Formal Analysis for Dependability Properties: the Time-Triggered Architecture Example. In *Proceedings of the 8th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2001)*, pages 343–352, October 2001.

[48] L.M. Pinho and F. Vasques. Reliable real-time communication in can networks. *IEEE Transactions on Computers*, 52(12):1594–1607, 2003.

[49] S. Poledna, W. Ettlmayr, and M. Novak. Communication bus for automotive applications. In *Proceedings of the 27th European Solid-State Circuits Conference*, September 2001.

[50] T. Pop, P. Pop, P. Eles, and Z. Peng. Bus access optimisation for FlexRay-based distributed embedded systems. In *Proceedings of the conference on Design, Automation and Test in Europe (DATE '07)*, pages 51–56, San Jose, CA, USA, 2007. EDA Consortium.

[51] A. Rajnák. *The Embedded Systems Handbook*, chapter Volcano - Enabling Correctness by Design. CRC Press, August 2005. R. Zurawski editor, ISBN 0-8493-2824-1.

[52] A. Rajnák. *The Industrial Communication Technology Handbook*, chapter The LIN Standard. CRC Press, January 2005. R. Zurawski editor, ISBN 0-8493-3077-7.

[53] A. Rajnák and M. Ramnefors. The Volcano communication concept. In *Proceedings of Convergence 2002*, 2002.

[54] K. Ramaswamy and J. Cooper. Delivering multimedia content to automobiles using wireless networks. In *Proceedings of Convergence 2004*, Detroit, Michigan, 2004.

[55] Robert Bosch GmbH. Time triggered communication on CAN: TTCAN. Available at `http://www.semiconductors.bosch.de/en/20/ttcan/index.asp`, 2008.

[56] J. Rushby. A comparison of bus architecture for safety-critical embedded systems. Technical report, NASA/CR, March 2003.

[57] R. Saket and N. Navet. Frame packing algorithms for automotive applications. *Journal of Embedded Computing*, 2:93–102, 2006.

[58] A. Schedl. Goals and Architecture of FlexRay at BMW. Slides presented at the Vector FlexRay Symposium, March 2007.

[59] Society of Automotive Engineers. J2056/1 class C application requirements classifications. In *SAE Handbook*. 1994.

[60] Society of Automotive Engineers. J2056/2 survey of known protocols. In *SAE Handbook*, volume 2. 1994.

[61] Society of Automotive Engineers. Class B data communications network interface - SAE J1850 standard - rev. nov96, 1996.

[62] J. Sommer and R. Blind. Optimized resource dimensioning in an embedded CAN-CAN gateway. In *IEEE Second International Symposium on Industrial Embedded Systems (SIES'2007)*, pages 55–62, July 2007.

[63] K. Tindell, A. Burns, and A.J. Wellings. Calculating Controller Area Network (CAN) message response times. *Control Engineering Practice*, 3(8):1163–1169, 1995.

[64] K. Tindell and J. Clark. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessors and Microprogramming*, 40:117–134, 1994.

[65] TTTech Computertechnik GmbH. *Time-Triggered Protocol TTP/C, High-Level Specification Document, Protocol Version 1.1*, November 2003. Available at `http://www.tttech.com`.

[66] M. Waern. Evaluation of protocols for automotive systems. Master's thesis, KTH Machine Design, Stockholm, 2003.

[67] C. Wilwert, N. Navet, Y.-Q. Song, and F. Simonot-Lion. *The Industrial Communication Technology Handbook*, chapter Design of Automotive X-by-Wire Systems. CRC Press, January 2005. R. Zurawski editor, ISBN 0-8493-3077-7.

[68] X-by-Wire Project, Brite-EuRam 111 Program. X-By-Wire - safety related fault tolerant systems in vehicles, final report, 1998.