UPCOMING AUTOMOTIVE STANDARDS FOR FAULT-TOLERANT COMMUNICATION: FLEXRAY AND OSEKTIME FTCOM.*

C. KÜHNEL AND M. SPICHKOVA

Institut für Informatik,
Bolzmannstr. 3,
D-85748 Garching, Germany
E-mail: {kuehnelc,spichkov}@in.tum.de

A safety-critical system needs fault-tolerant communication between its components. This is especially important for automotive domain, as it consists of distributed real-time systems that are based on the results of the communication. To realize distributed systems with predictable time behavior the time-triggered paradigm is used. According to this paradigm, a time-triggered communication protocol, FlexRay, and an operating system OSEKtime with corresponding communication layer FTCom for the fault-tolerant communication are introduced. In this paper we present the formal specifications of FlexRay and FTCom that allow us to argue about their properties in a precise, formal manner and also infer the collaboration between their properties.

1. Introduction

The trend in the automotive industry to shift functionality from machanics and eletronics to software has been going on for several years now, but progress seems to have slowed down. Most of the manufacturers have presented drive-by-wire prototypes, but none of them has entered mass production. The experiences with increased software in the infotainment domain have shown severe quality issues. To overcome this, new technology for distributed fault-tolerant systems, is required.

One major problem today is reliable, deterministic communication for distributed automotive systems. In this domains two standards for automotive systems have established: FlexRay, a fault-tolerant communication

*This work was partially funded by the german federal ministry of education and technology (bmbf) in the framework of the verisoft project under grant 01 is c38. the responsibility for this article lies with the authors.

network, and OSEK FTCom, a fault-tolerant communication layer for the OSEKtime OS operating system.

The specified system towards the time-triggered paradigm – both, the communication protocol and the operating system, are time-triggered. In a time-triggered system all actions are executed at predefined points in time. This provides a time behavior that is deterministic: task execution times and their order, as well as message transmission times are deterministic. This property is important for distributed real-time systems, because for such kind of systems it is possible to prove their time properties with reasonable effort.

The FlexRay and the FTCom form, together with OSEKtime OS, the verification framework[3] that provides the methodology for the verification of application properties. We abstract here from the detailed specification of the OS, as well as from the application components to concentrate on the representation of the fault-tolerant communication between application components via FlexRay and FTCom.

To make a formal analysis for FlexRay and FTCom possible, several aspects of the systems have been formalized[11]. In this paper, the major aspects of the formalization are described.

1.1. *FOCUS*

This paper is based on the formal language FOCUS[4]. It was chosen since it provides means for modeling concurrent, distributed system and allows to specify them in formal manner.

The central concept in FOCUS are *streams*, that represent communication histories of *directed channels*. Streams in FOCUS are functions mapping the indexes in their domains to their messages. For any set of messages M, M^ω denotes the set of all streams, M^∞ and M^* denote the sets of all infinite and all finite streams respectively. $M^{\underline{\omega}}$ denotes the set of all timed streams, $M^{\underline{\infty}}$ and $M^{\underline{*}}$ denote the sets of all infinite and all finite timed streams respectively. A *timed stream* is represented by a sequence of messages and *time ticks* (represented by $\surd$), the messages are also listed in their order of transmission. The ticks model a discrete notion of time.

An empty stream is represented in FOCUS by $\langle\rangle$. $\langle x\rangle$ denotes one-element stream and $\#x$ denotes the length of the stream x. The FOCUS predicate $\mathsf{disjunct}(s_1, \ldots, s_n)$ is true, if all streams $s_1, \ldots, s_n$ are disjunct, i.e. in every time unit only one of these streams has any messages to transfer. To simplify the specification of the real-time systems we introduce the

FOCUS operator $\mathsf{ti}(s, n)$ that yields the list of messages that are in the timed stream s between the ticks $n-1$ and n (at the nth time unit). We also define the FOCUS operator $\mathsf{maxmsg}_n(s)$, which holds for a timed stream s, if this stream contains at every time unit at most n messages. $\mathsf{dom}.x$ denotes $[1..\#x]$ and $\mathsf{rng}.x$ denotes $\{x.j \mid j \in \mathsf{dom}.x\}$.

1.2. *FlexRay*

FlexRay is a time triggered communication protocol, developed by the FlexRay Consortium[6]. It's primary application domain are distributed real-time systems in vehicles. Today, most of these systems use a Controller Area Network (CAN)[17] as means of communication. The advantages of FlexRay over CAN are: higher bandwidth, integrated functionality for clock synchronisation, deterministic real-time message transmission and fault tolerance.

1.3. *OSEKtime FTCom*

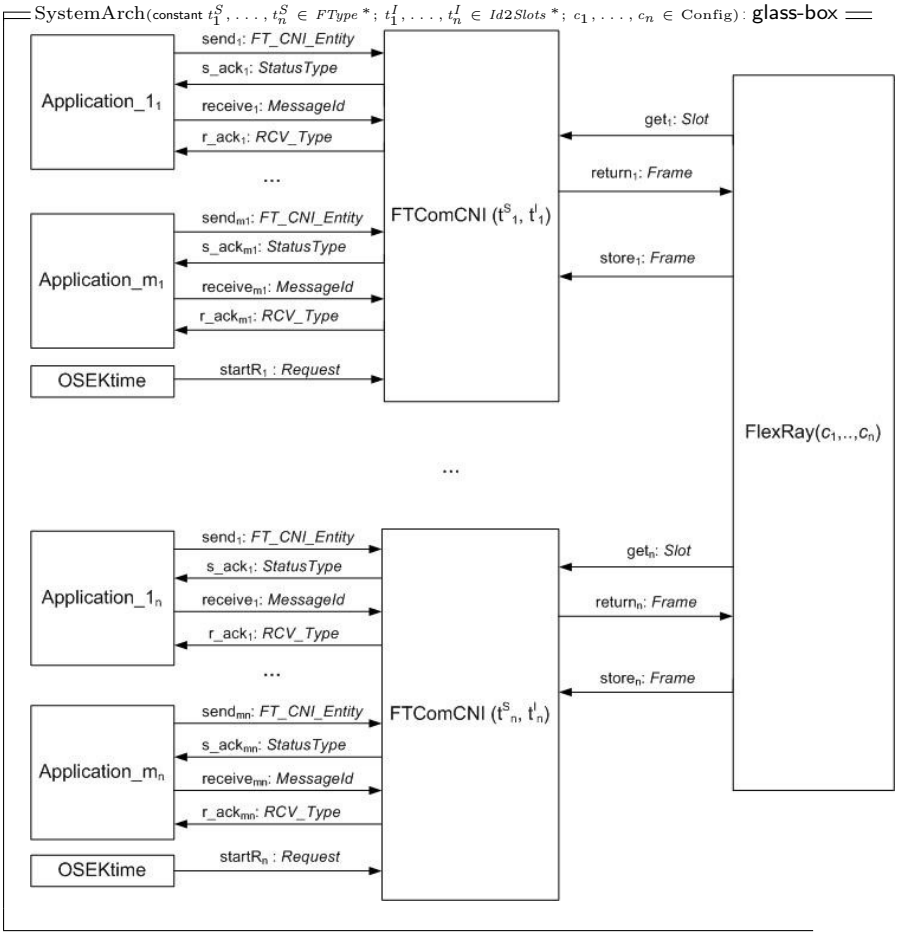
OSEKtime[15] OS is an OSEK/VDX[13] open operating system standard of the European automotive industry. The OSEKtime OS is a time-triggered OS that supports static cyclic scheduling based on the computation of the WCETs (worst case execution times) of tasks. WCETs are needed for scheduleability analysis and can be estimated from a compiled C program and the processor the program runs on[1].

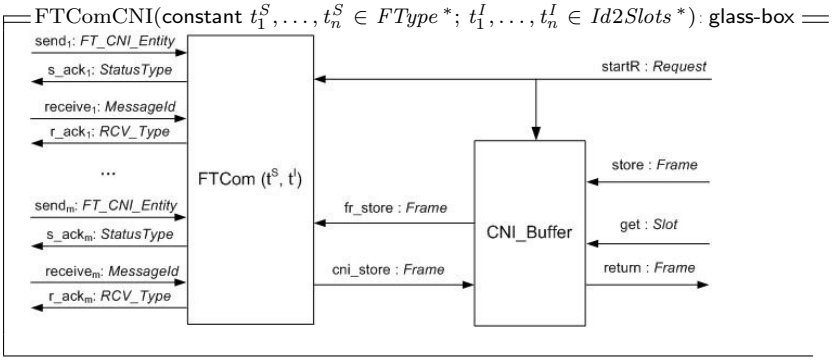
FTCom[14] (Fault-Tolerant Communication) is an fault-tolerant communication layer for OSEKtime that provides a number of primitives for interprocess communication and makes task distribution transparent.

2. Formal Specification

2.1. *Fault-Tolerant Embedded System*

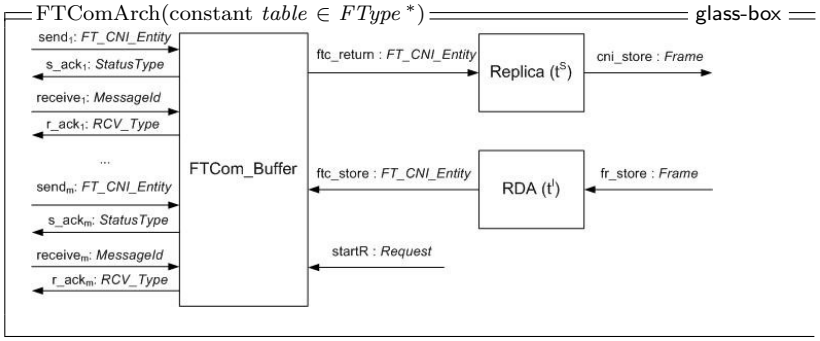
The architecture of the overall system is represented as a FOCUS specification *SystemArch*. The system consists of a number of nodes that are connected by a FlexRay bus. On each node runs the OSEKtime OS and a number of applications, and on each node there is a component *FTComCNI* that consists of two subcomponents: the *FTCom* itself and a *CNI_Buffer* (Communication Network Interface). In the CNI buffer all the messages that must be sent via FlexRay are stored, whereas the local communication on the node is done directly via FTCom.


SystemArch(constant $t_1^S, \ldots, t_n^S \in FType^*$; $t_1^I, \ldots, t_n^I \in Id2Slots^*$; $c_1, \ldots, c_n \in$ Config) : glass-box
Application_1$_1$
send$_1$: FT_CNI_Entity
s_ack$_1$: StatusType
receive$_1$: MessageId
r_ack$_1$: RCV_Type
Application_m$_1$
send$_{m1}$: FT_CNI_Entity
s_ack$_{m1}$: StatusType
receive$_{m1}$: MessageId
r_ack$_{m1}$: RCV_Type
OSEKtime
startR$_1$: Request
FTComCNI (t^S_1, t^I_1)
get$_1$: Slot
return$_1$: Frame
store$_1$: Frame
FlexRay(c_1,..,c_n)
Application_1$_n$
send$_1$: FT_CNI_Entity
s_ack$_1$: StatusType
receive$_1$: MessageId
r_ack$_1$: RCV_Type
Application_m$_n$
send$_{mn}$: FT_CNI_Entity
s_ack$_{mn}$: StatusType
receive$_{mn}$: MessageId
r_ack$_{mn}$: RCV_Type
OSEKtime
startR$_n$: Request
FTComCNI (t^S_n, t^I_n)
get$_n$: Slot
return$_n$: Frame
store$_n$: Frame


FTComCNI(constant $t_1^S, \ldots, t_n^S \in FType^*$; $t_1^I, \ldots, t_n^I \in Id2Slots^*$) : glass-box
send$_1$: FT_CNI_Entity
s_ack$_1$: StatusType
receive$_1$: MessageId
r_ack$_1$: RCV_Type
send$_m$: FT_CNI_Entity
s_ack$_m$: StatusType
receive$_m$: MessageId
r_ack$_m$: RCV_Type
FTCom (t^S, t^I)
startR : Request
fr_store : Frame
cni_store : Frame
CNI_Buffer
store : Frame
get : Slot
return : Frame

2.2. *FTCom*

The component *FTCom* consists of three subcomponents: *FTCom_Buffer*, *Replica* and *RDA* (Replica Determinate Agreement[14]). The FTCom buffer is used for the local communication – between applications that are deployed on the same node, so that local messages are not sent via FlexRay. The *Replica* and *RDA* components are needed for the fault-tolerant communication with other nodes of the system.



The type *FT_CNI_Entity* represents the type of application messages. It consists of a message identifier of type *MessageId* and an application data type *DataType*. The data types *RCV_Type* and *Status_Type* represent the result types of the standard FTCom functions[14] *ttReceiveMessage* and *ttSendMessage* that are used by the applications to access the *FTCom-buffer*.

The Replica component assumes the replication task: one application message is packed into several FlexRay frames using the replication-tables t^S – an application message will be transported during several FlexRay slots of each communication round. A replication-table is specified as list of type FType, which is defined below.

$$\textsf{type}\ FType \quad = \quad ft(slot \in Slot,\ msl \in MessageId\,*)$$

The RDA component assume the RDA task: frames are unpacked using the RDA-tables t^I. A RDA-table is specified as list of type *MessageId*. From these replicated messages the current one is build using some RDA algorithm, e.g. average, majority vote, "pick any" (see also Sect. 3). The

Replica and RDA tasks are called by the OSEKtime dispatcher every communication round. In the FOCUS specification this represented by using request messages of type *Request* on the channel *startR*.

We define[11] several properties for the correct Replica- and RDA-tables[a]. For the overall system the same properties must also hold for the unions of the corresponding tables of the overall system, namely gS is the union of all t^S tables and gI of all t^I tables. Moreover, the tables gS and gI must be "inverse" in sense of the predicate *InverseSI*[11]. In such a way we can formally show (using a theorem prover Isabelle/HOL[12,21]) for the concrete tables that they are correct according to these properties.

InverseSI

$gS \in FType^*$; $gI \in MessageId^*$

$\forall\, i \in \mathsf{rng}.gI : \exists\, s \in \mathsf{rng}.gS : i \in \mathsf{rng}.msl(s)$
$\forall\, ft(s,\ id\_list) \in \mathsf{rng}.gS : \forall\, i \in \mathsf{rng}.id\_list : i \in \mathsf{rng}.gI$
$\forall\, i, j \in \mathsf{dom}.gS : i \neq j \Rightarrow sl(gS.i) \neq sl(gS.j)$

2.3. *FlexRay*

FlexRay contains a set of complex algorithms to provide the communication services. From the view of the software layers above FlexRay only a few of these properties become visible. The most important ones are static cyclic communication schedules and system-wide synchronous clocks. These provide a suitable platform for distributed control algorithms as used e.g. in drive-by-wire applications. The formalization described here is based on the "Protocol Specification 2.0"[8].

2.3.1. *Abstractions*

To reduce the complexity of the system several aspects of FlexRay have been abstracted in this formalization: (1) There is no clock synchronization or start-up phase since clocks are assumed to be synchronous. This corresponds very well with the *time-synchronous* notion of FOCUS [4]. (2) The model does not contain bus guardians. (3) Only the static segment

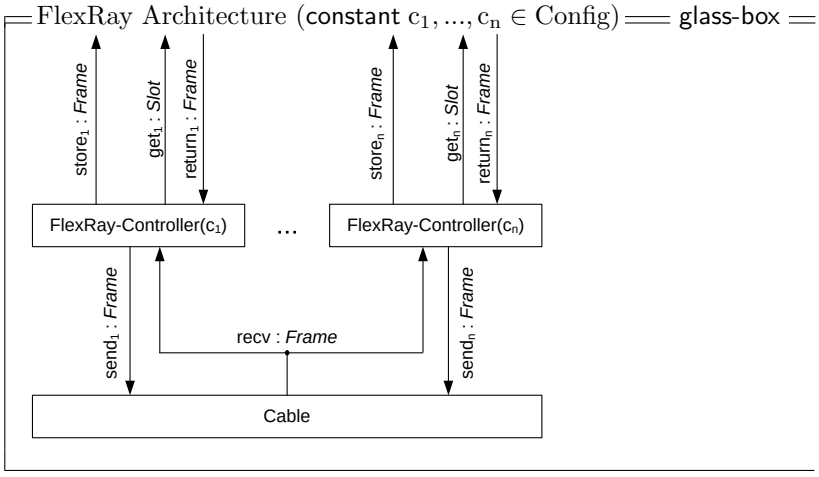
[a]Like "Every slot identifier can occur in the frame table t^S at most once", "the table is non-empty", etc.

has been included not the dynamic, as we are mainly interested in time-triggered systems. (4) The time-basis for the system is one slot i.e. one slot FlexRay corresponds to one tick in in the formalization. (5) The system contains only one FlexRay channel. Adding a second channel would mean simply doubling the FlexRay component with a different configuration and adding extra channels for the access to the *CNI_Buffer* component.

2.3.2. *FlexRay Architecture*

The component *FlexRay Architecture* is a refinement of the component *FlexRay* (see below) and consists of several *FlexRay- Controller* and a network *Cable*. The unconnected channels of each controller are to be connected to those of the *FTCom-CNI* components. Since FOCUS does not contain a concept for broadcast communication this is simulated in the component *Cable*: It forwards a received frame to all connected nodes.

The type *Slot* describes here one time slot during a FlexRay communication cycle and is equal to the type of natural numbers $\mathbb{N}$. A *Frame* that represents a FlexRay frame consists of a slot identifier *slot* and the payload *Payload*. The definition of the type *Payload* depends on the configuration of the FTCom (see Sect. 2.2 and Sect. 3). The FlexRay bus configuration *Config* contains the bus scheduling table *schedule* of the node and the length of the communication cycle *cyclelength*.



type *Frame* = $frm(slot \in Slot,\ payload \in Payload)$

type *Config* = $conf(schedule \in Slot^*,\ cycleLength \in \mathbb{N})$

The component *FlexRay* contains the assumptions and guarantees for the FlexRay network. In *IdenticCycleLength* it assumes that the length of the communication cycle is identical for all nodes, in *DisjointSchedules* that for each slot of a cycle there is at most one sending node. These are the basic requirements of a static cyclic time division multiplexing network.

FlexRay (constant $c_1, ..., c_n \in$ Config) — timed

in $return_1, ..., return_n : Frame$

out $store_1, ..., store_n : Frame;\ get_1, ..., get_n : Slot$

asm $\forall\, i \in [1..n] : \mathsf{maxmsg}_1(return_i)$

$DisjointSchedules(c_1, ..., c_n)$

$IdenticCycleLength(c_1, ..., c_n)$

gar $FrameTransmission(return_1, ..., return_n, store_1, ..., store_n, get_1, ..., get_n, c_1 ..., c_n)$

$\forall\, i \in [1..n] : \mathsf{maxmsg}_1(get_i) \wedge \mathsf{maxmsg}_1(store_i)$

The guarantee $maxmsg_1$ defines that over the channels get_i and $store_i$ at most one frame is transmitted in each tick. The former one is required by the *CNI_Buffer*, the latter by *Cable*. The guarantee *FrameTransmission* describes how frames are transmitted over a FlexRay-network.

FrameTransmission

$store_1, ..., store_n, return_1, ..., return_n \in Frame^{\underline{\omega}}$

$get_1, ..., get_n \in Slot^{\underline{\omega}}$

$c_1, ..., c_n \in Config$

$$\forall\, t \in \mathbb{N}, k \in [1..n] :$$

$$s \in schedule(c_k) : s = t \bmod cycleLength(c_k) \rightarrow$$

$$\mathsf{ti}(get_k, t) = \langle s \rangle \wedge$$

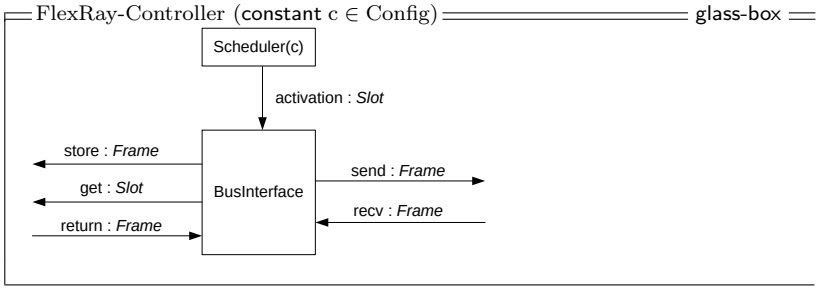
$$\forall\, j \in [1..n], j \neq k : \mathsf{ti}(store_j, t) = \mathsf{ti}(return_k, t)$$

This predicate specifies that if at time t the node k should be sending according to it's schedule, then it requests the frame which should be sent from the *CNI_Buffer* over the channels get_k and $return_k$. This frame is then send to the other nodes of the system. These receive the frame and store

it in their respective *CNI_Buffers* over the channel $store_j$. This guarantee is the theorem that has to be proved for an implementation of the FlexRay protocol.

2.3.3. *FlexRay Controller*

A *FlexRay-Controller* component consists of a *Scheduler* and a *BusInterface*. The *Scheduler* evaluates the schedule *schedule(c)*. This schedule specifies in which time slot this node should send a frame. Based on this information the *Scheduler* notifies the *BusInterface* in case a frame should be sent. The slot number specifies the time slot when the frame should be sent and also the type of frame, since this mapping is static. The *BusInterface* then fetches the corresponding frame from the *CNI_Buffer* and sends it on the channel *send*. If the node is not sending and a frame is received over the channel *recv*, this frame is forwarded to the *CNI_Buffer* over the channel *store*. For the sake of brevity, the specification details[11] of these components are omitted here.



3. Collaboration between FlexRay and FTCom

The formal specifications of FlexRay and FTCom allow us to argue about their properties in a precise, formal manner[19] and also infer the collaboration between their properties. In this section we discuss the the examples of the collaboration properties.

3.1. *Level of the Fault-Tolerance*

According to the FlexRay specification[8] a frame contains a 24 bit CRC (cyclic redundancy check) checksum to ensure the integrity of the frame transmission. The probability of undetected network errors[16] is less than

$6 \cdot 10^{-8}$, this means that at 10,000 messages per second and a bit error rate of 10^{-6}, this is means approximately $2 \cdot 10^{-6}$ undetected erroneous frames per hour. Using FTCom's replication mechanisms we can reduce this probability even further and increase the fault-tolerance.

If the FlexRay error rate is good enough for a certain system, we can use simpler configurations of the *RDA* component of the FTCom. Because of the high probability of the communication error detection by the FlexRay, we can assume that if such an error occurs, the wrong frame will be not saved in the CNI buffer. Thus, as the RDA algorithm for this case a "pick first appropriate" can be chosen to make the data processing faster.

The configuration of the replication table for the Replica-task, depends on reliability of physical connection.

3.2. *FlexRay Frames*

The type of payload in the FlexRay frame (see Sect. 2.3) depends on the configuration of FTCom. In the general case, we represent the payload part of the FlexRay frame as a list of application messages – the type *Payload* is then defined as a list over the type *FT_CNI_Entity*.

If the number of replicated messages in the system is smaller than the number of the FlexRay communication slots in a round, then it is possible to use the model "One_message_per_frame", in which the types *Payload* and *FT_CNI_Entity* are equal. In the simple case, when the bus connection is reliable enough to send an application message without replication, i.e. once every FlexRay round, the message identifier can be taken as the corresponding slot number[b] $Payload = DataType$.

3.3. *Schedule Dependences*

Combining the time-triggered OS and bus one can synchronize not only the communication, but also the computations in the systems. FlexRay provides OSEKtime OS with a globally synchronized clock. For this purpose, the length of the OSEKtime dispatcher round must be a multiple of the length of the FlexRay round (counted in FlexRay slots).

As mentioned in the Section 2.2, the Replica and RDA tasks must be called by the OSEKtime dispatcher every communication round to have current data both in the FTCom and CNI buffers. Generating the OSEKtime dispatcher tables, this property must be taken into account.

[b]This implies that the types *MessageId* and *Slot* ($\mathbb{N}$) are equal.

4. Conclusion and Future Work

The paper presented the formal specifications of FlexRay and FTCom in FOCUS and also inferred the correlation between their properties. These FOCUS specifications allow us to argue about the properties of FlexRay and FTCom in a precise, formal manner. Using the presented specification of FlexRay we also have verified[19] that this FlexRay specification conforms the FlexRay requirements.

An Overview of the verification of TTA (Time Triggered Archicecture) [10] was presented by J. Rushby[18]. Since a comparison of TTA and FlexRay[9] have shown several differences, the results of the verification of TTA can not be transfered directly to FlexRay and FTCom.

A verification of the clock synchronization algorithm of FlexRay is in progress at LORIA, based on their framework[5].

Another future work is the extension of the formal specification with a second FlexRay-channel and Bus Guardians[7], which would improve the fault-tolerance.

The work on the verification of the lower layers[2] of the specified system is in progress in the Verisoft project[20].

Acknowledgments

We would like to thank Manfred Broy for his valuable feedback on the FOCUS specifications.

References

1. AbsInt Angewandte Informatik GmbH. Worst-Case Execution Time Analyzers. `http://www.absint.com/profile.htm`.
2. S. Beyer, P. Böhm, M. Gerke, M. Hillebrand, T. In der Rieden, S. Knapp, D. Leinenbach, and W.J. Paul. Towards the formal verification of lower system layers in automotive systems. In *23nd IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 2005), 2-5 October 2005, San Jose, CA, USA, Proceedings*, pages 317–324. IEEE, 2005.
3. J. Botaschanjan, L. Kof, Ch. Kühnel, and M. Spichkova. Towards Verified Automotive Software. In *ICSE, SEAS Workshop*, St. Louis, Missouri, USA, May 21 2005.
4. M. Broy and K. Stølen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement.* 2001.
5. D. Barsotti and L. Prensa Nieto and A. Tiu. Verification of Clock Synchronization Algorithms: Experiments on a Combination of Deductive Tools. In *Electr. Notes Theor. Comput. Sci. 145*, 2006.
6. FlexRay Consortium. `http://www.flexray.com`.

7. FlexRay Consortium. *FlexRay Communication System – Bus Guardian Specification - Version 2.0*, 2004.
8. FlexRay Consortium. *FlexRay Communication System – Protocol Specification – Version 2.0*, 2004.
9. H. Kopetz. A Comparison of TTP/C and FlexRay. Technical report, Institut für Technische Informatik, Technische Universität Wien, 2001.
10. H. Kopetz and G. Bauer. The time-triggered architecture. In *Proceedings of the IEEE*. IEEE, 2003.
11. C. Kühnel and M. Spichkova. FlexRay und FTCom: Formale Spezifikation in Focus. Technical Report TUM-I0601, Technische Universität München, 2006.
12. T. Nipkow, L.C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
13. OSEK/VDX. `http://www.osek-vdx.org`.
14. OSEK/VDX. *Fault-Tolerant Communication – Specification 1.0*, 2001. `http://www.osek-vdx.org/mirror/ftcom10.pdf`.
15. OSEK/VDX. *Time-Triggered Operating System – Specification 1.0*, 2001. `http://www.osek-vdx.org/mirror/ttos10.pdf`.
16. M. Paulitsch et al. Coverage and the use of cyclic redundancy codes in ultra-dependable systems. In *2005 International Conference on Dependable Systems and Networks (DSN'05)*, pages 346–355, 2005.
17. Robert Bosch GmbH. *CAN Specification Version 2.0*, 1991.
18. J. Rushby. An overview of formal verification for the time-triggered architecture. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, volume 2469 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
19. M. Spichkova. FlexRay: Verification of the Focus Specification in Isabelle/HOL. A Case Study. Technical Report TUM-I0602, Technische Universität München, 2006.
20. Verisoft Project. `http://www.verisoft.de`.
21. M. Wenzel. *The Isabelle/Isar Reference Manual*. Technische Universität München, 2004.