From Automotive to Autonomous: Time-Triggered Operating Systems

Maria Spichkova, Milan Simic and Heinz Schmidt

Abstract This paper presents an approach for application of time-triggered paradigm to the domain of autonomous systems. Autonomous systems are intensively used in areas, or situations, which could be dangerous to humans or which are remote and hardly accessible. In the case when an autonomous system is safety critical and should react to the environmental changes running within a very limited time frame, we deal with the same kind of problems as automotive and avionic systems: timing properties and their analysis become a crucial part of the system development. To analyse timing properties and to show the fault-tolerance of the communication, a predictable timing of the system is needed.

1 Introduction

The trend in the automotive and avionics industry is to shift functionality from mechanics and electronics to software. Subsystems in automotive and avionics engineering express increasing level of autonomy as described in [12]. The goal of this changes is to make vehicles and aircrafts more reliable and fault-tolerant, focusing on the tasks that could be complicated for the human to solve quickly, precisely and safely. In avionics, fly-by-wire are used since many decades. These systems were developed to replace the manual (human) flight controls of an aircraft with an electronic interface. An example of a fly-by-wire system is Unmanned Aerial Vehicle (UAV) landing [22]. Many automotive manufacturers have presented drive-by-wire prototypes for premium cars. An example of these prototypes are Parking Assists Systems [9].

M. Spichkova (✉) · M. Simic · H. Schmidt
RMIT University, Melbourne, Australia
e-mail: maria.spichkova@rmit.edu.au

M. Simic
e-mail: milan.simic@rmit.edu.au

H. Schmidt
e-mail: heinz.schmidt@rmit.edu.au

G. De Pietro et al. (eds.), *Intelligent Interactive Multimedia Systems and Services 2016*, Smart Innovation, Systems and Technologies 55,
DOI 10.1007/978-3-319-39345-2_30

To analyse timing properties of a distributed system and to show the fault-tolerance of its behaviour, we require a predictable timing of the system. This can be solved using the time-triggered paradigm (TTP). In avionics TTP was applied successfully to distributed systems. This idea has been later propagated to the automotive domain.

In a time-triggered system (TTS), all actions are executed at predefined points in time. This ensures a deterministic timing behaviour of the system: task execution times and their order, as well as message transmission times are deterministic. Having a deterministic timing behaviour, we can predict and to prove formally the timing properties of the system with a reasonable effort.

In [36], we introduced a formal framework for modelling and analysis of autonomous systems (AS) and their compositions, especially focusing on the adaptivity modelling aspects and reasoning about adaptive behaviour. In our current work we extend this framework with the core features of a framework for the verification of properties for automotive TTS, presented in [20, 21]. In this paper, we suggest to apply the well-developed ideas of time-triggered systems within the domain of autonomous robotic systems, as timing properties and their verification are crucial for safety-critical systems.

Outline: The paper is organised as follows. Section 2 provides a brief overview of the related work. In Sect. 3, we discuss the application of TTP within automotive domain, time-triggered operating system OSEKtime as well as our previous research in this area. Section 4 introduces our TTP Model for AS, which is the core contribution of the paper. Section 5 introduces our model for OSEKtime. Section 6 present the core directions of our future work. In Sect. 7 we summarise the paper and propose directions for future research.

2 Related Work

The adaptation and context-awareness can have many forms: navigation applications to guide users to a given destination, robot motion [10], keyless entry systems [15], driver assistance applications [27], adaptive cruise control systems [14], etc. Next step from driver assistance and similar applications is complete takeover of the vehicle control. Reliable, real time robot (vehicle) operating system is extremely important for the safe and comfortable ride. A concise survey of concepts, architectural frameworks, and design methodologies that are relevant in the context of self-adapting and self-optimizing systems is presented in [2].

Rushby [26] introduced an approach to derive a time-triggered implementation from a fault-tolerant algorithm specified as a functional program. Nolte et al. [23] presented an overview of wireless automotive communication technologies. The goal of Nolte et al. was to identify the strong candidates for future in-vehicle and inter-vehicle automotive applications. There are also a number approaches on analysis of worst case execution time (WCET) and the corresponding properties of the software components. For example, Fredriksson et al. [16] presented a contract-

based technique to achieve reuse of WCETs in conjunction with reuse of software components. A number of frameworks and methodologies for the formal analysis of time-triggered automotive systems were presented in [5–7, 17, 34]. The core features of these frameworks and methodologies can be applied within the field of AS, to increase reliability and to allow formal verification of the functional properties.

Both in the case of automotive and in the case of autonomous systems, we to take into account *cyber-physical* nature of these systems. Many approaches on *cyber-physical systems* do not include an abstract logical level of the system modelling, missing the advantages of the abstract representation. In our previous work [33], we introduced a platform-independent architectural design in the early stages of CPS development. The results of our ongoing work on simulation, validation and visualization of CPSs in industrial automation [3, 4] provide basis for the analysis and simulation of autonomous systems, as a special kind of CPSs.

3 TTP: OSEKtime Operating System

An operating system OSEKtime was developed by the European Automotive Consortium OSEK/VDX in accordance to the time-triggered paradigm. OSEK[1] is a standards body, founded by German automotive company consortium, in 1993. The consortium included many industrial partners (such as BMW, Robert Bosch GmbH, DaimlerChrysler, Opel, Siemens, and Volkswagen Group) as well as the University of Karlsruhe. The French automotive manufacturers Renault and PSA Peugeot Citroen had a similar consortium, VDX.[2] In 1994, a new consortium OSEK/VDX[3] was created, based on OSEK and VDX.

Thus, OSEKtime OS is time-triggered and supports static cyclic scheduling based on the computation of the WCETs of its tasks. The verification of an OSEKtime OS is being performed in the Verisoft-XT project [38]. WCET of the tasks can be estimated from a compiled program and the processor the program runs on, using the corresponding tool, e.g. an *aiT* analyser [1]. aiT statically compute tight bounds for WCETs of tasks in real-time systems, directly analyses binary executables, and is independent of the compiler and source code language.

As per OSEKtime specification [25], the core properties of this operating system are

- predictability,
- modular concept as a basis for certification,
- dependability,
- compatibility to the OSEK/VDX.

[1] German: Offene Systeme und deren Schnittstellen fr die Elektronik in Kraftfahrzeugen; English: Open Systems and their Interfaces for the Electronics in Motor Vehicles.

[2] Vehicle Distributed eXecutive.

[3] http://www.osek-vdx.org.

Thus, the system has a priori known behaviour, even under defined peak load and fault conditions. The dependability property is insured by a fault-tolerant communication layer FTCom (Fault-Tolerant Communication, cf. [24]) is This layer was introduced to insure interprocess communication within the OS and make task distribution transparent. It allows reliable operation through fault detection and fault tolerance.

In our previous research [20, 21], we presented a formal framework for the verification of application properties for time triggered systems, based on FlexRay and FTCom [19, 28]. FlexRay is a time-triggered communication protocol, developed by the FlexRay Consortium.[4] Primary application domain of the FlexRay protocol is distributed real-time systems in vehicles. A comparison of an established protocol TTP/C and FlexRay was presented by Kopetz in [18]. Kopetz came to the conclusion that FlexRay and TTP/C were designed against the same set of automotive requirements, but that there is a difference in goals: "*The inherent conflict between flexibility and safety is tilted towards flexibility in FlexRay and safety in TTP/C.*".

A distributed automotive system is built from a number of nodes. The difference between the nodes are the configuration data of each node and the applications running on them. On each node, there are three layers:

(1) Micro controller and FlexRay controller. The network cable connects the FlexRay controllers of all nodes.
(2) OSEKtime OS and OSEK FTCom.
(3) A number of applications, implementing the desired behaviour of the automotive system.

When we adopt these ideas to the domain of autonomous systems, a node becomes a single autonomous robot, and the connection between the robots has to be wireless. This implies that the FlexRay communication protocol might be not applicable for autonomous systems even after a number of modifications: FlexRay provides high-speed, deterministic and fault-tolerant communication, but it was designed with the focus on in-vehicle networking, to support x-by-wire applications. Thus, only the architecture of the layers (2) and (3) can be adapted for AS.

4 TTP Model for Autonomous Systems

The modelling language that we use in our approach is FOCUSST [32], which is an extension of the FOCUS language used in [20, 21]. It allows us to create concise but easily understandable specifications and is appropriate for application of the specification and proof methodology presented in [29, 37]. The FOCUSST language was inspired by FOCUS [8], a framework for formal specification and development of interactive systems. In both languages, specifications are based on the notion of

[4]Core members of the consortium are Freescale Semiconductor, Robert Bosch GmbH, NXP Semiconductors, BMW, Volkswagen, Daimler, and General Motors.

streams and channels (a channel is in effect a name for a stream). The FOCUSST specification layout also differs from the original one: it is based on human factor analysis within formal methods [30, 31, 35].

A system in our model consists of N autonomous robots, communicating with each other. On the software level, a robot $Robot^i$ ($1 \leq i \leq N$) has Mi applications $AppRobot^i_1, \ldots, AppRobot^i_{Mi}$ that are running under OSEKtime OS, cf. Fig. 1. The *FTComCNI* component of each robot consists of two subcomponents: the *FTCom* component and a *CNIbuffer* (buffer of the Communication Network Interface), cf. Fig. 2. *CNIbuffer* is used to store messages that have to be sent to other robots via the communication protocol. The local communication among the applications $AppRobot^i_1, \ldots, AppRobot^i_{Mi}$ is conducted directly via *FTCom*.

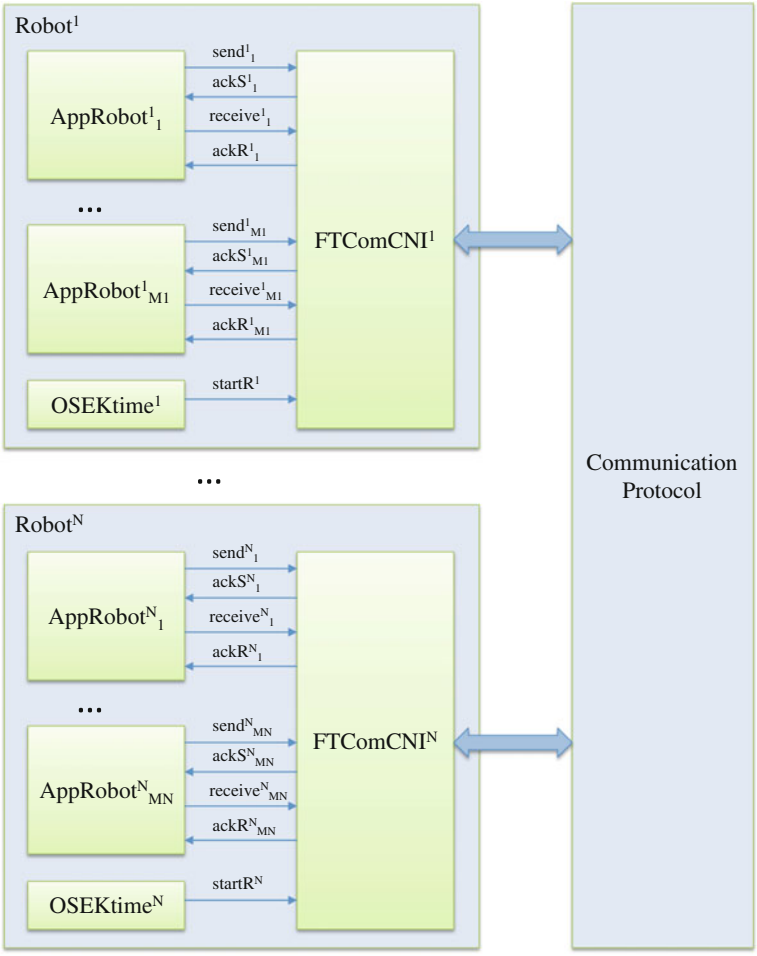

Fig. 1 TTP model for autonomous systems

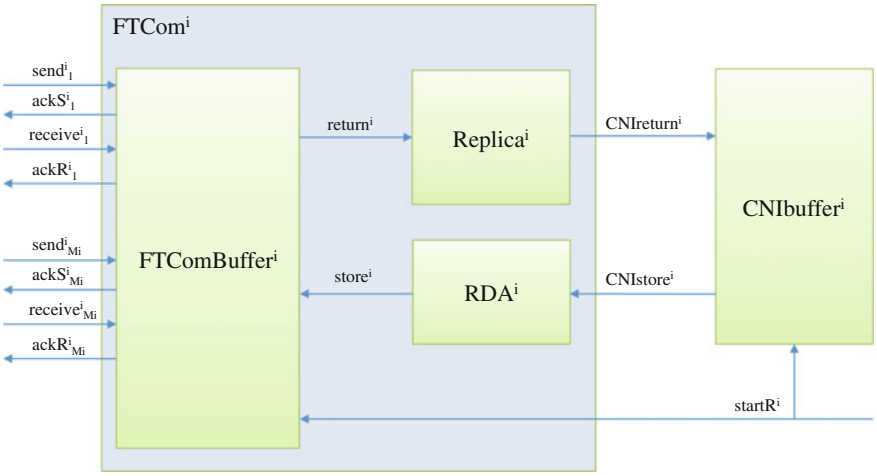

Fig. 2 Model of the *FTComCNI* component

In our model, the *FTCom* component consists of three subcomponents: *FTComBuffer*, *Replica* and *RDA* (Replica Determinate Agreement, cf. [24] for more details). The *Replica* and *RDA* components are optional. These components are used to ensure fault-tolerant communication between the robots: one application message is packed into several frames using the replication tables, such that every application message will be sent k times ($k > 1$). The *RDA* component of the corresponding receiver-robot, will unpack the frames the RDA-tables and compute the value of the message from the arrived replicas. The *Replica* and *RDA* components are called by the OSEKtime dispatcher every communication round. In our model, this is represented by sending request messages via the channel *startR*.

To ensure correctness of the replication and RDA tables, we specify two predicates, which will be used as verification constraints for the tables: For the replication table we have to check that

- the table is nonempty,
- every slot identifier occurs in the table at most once,
- a new frame can be build only from the messages produced by this robot.

For the RDA table we have to check that

- list of message identifiers must be nonempty,
- every message identifier occurs in the table at most once and does not belong to the identifiers of the messages produced by this robot.
- a new frame can be build only from the messages produced on this node.

5 Model of OSEKtime OS

We model OSEKtime as a tuple

$$OSEKtime = \langle Tasks,\ DT,\ State,\ SynchrP \rangle$$

where *Tasks* denotes the set of application tasks with the corresponding sets *State* of their internal states, *DT* is a dispatcher table, and *SynchrP* is a set of synchronisation parameters. An OSEKtime task can be in one of three states (*running*, *preempted*, *suspensed*), cf. also Fig. 3:

$$State = \{running,\ preempted,\ suspensed\}$$
$$SynchrP = \{synchr,\ asynchrH,\ asynchrS\}$$

OSEKtime allows two start-up techniques, which depend on synchronisation methods and availability of global system time [25]:

- Synchronous start-up (we model it as the parameter *synchr*). In this case, the tasks are not executed until a global time is available. For automotive systems based on the FlexRay protocol, the synchronous start-up has to be preferred, as FlexRay properties rely on a precise synchronisation. In the case of autonomous systems, we have to apply one of the options of asynchronous start-up.
- Asynchronous start-up: Tasks are executed according to the local time without waiting for the synchronisation to the global time.
  - In the case of a *hard synchronisation* (we model it as the parameter *asynchrH*), the synchronisation of the local time to the global time has to be performed at the end of a dispatcher round by delaying the start of the next dispatcher round.
  - In the case of a *smooth synchronisation* (we model it as the parameter *asynchrS*), the synchronisation of the local time to the global time is done during several dispatcher rounds by limiting the delay of the start of the next dispatcher round according to pre-defined configuration.

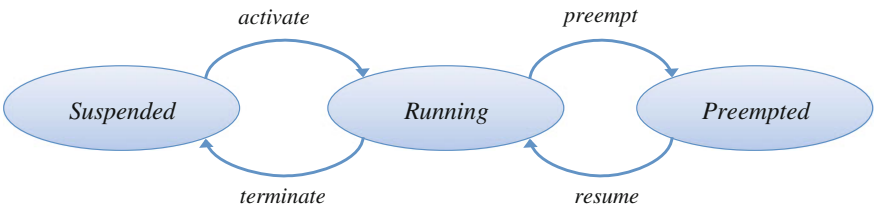

Fig. 3 Time-triggered model of OSEKtime tasks

Table 1 Example of an OSEKtime dispatcher table

DTentryID	Task	start	WCET
1	*Task*1	0	3
2	*Task*2	3	4
3	*Task*1	7	3
4	*Task*3	12	8
5	*Task*1	20	3
6	*Task*4	25	7
7	*Task*1	32	3

The specifications for set of tasks and the dispatcher table have to be introduced for each particular application. In general, they can be specified as follows:

$$DT = \langle N,\ STasks,\ start,\ wtime \rangle$$

where N is the length of the dispatcher table (number of items in it);

STasks is a subset of *Tasks*, which contains only the tasks to be scheduled, i.e., all the tasks from *Tasks* except auxiliary system tasks (such as *ttIdleTask*, *ErrorHook*, etc., cf. below);
$start = \{1 \dots N\}, STasks \to \mathbb{N}$ defines the mapping of tasks to their start times within the dispatcher round;
$wcet = \{1 \dots N\}, STasks \to \mathbb{N}$ defines the mapping of tasks to their worst-case execution times (WCETs).

Each task from *STasks* can also appear in *DT* as a number of subtasks. OSEKtime has a static scheduling, which allows specification based on WCETs of the subtasks.

Example An example of a dispatcher table $STasks = \{Task1, Task2, Task3, Task4\}$, $N = 7$, $T_{CYCLE} = 35$ is presented in Table 1. It is easy to see that while executing this dispatcher table, *ttIdleTask* will start at least twice in the middle of each round:

- between the second execution of *Task*1 (which has to be finished by 10th time unit) and *Task*3 (which has to be started at 12th time unit), and
- between the third execution of *Task*1 (which has to be finished by 23rd time unit) and *Task*4 (which has to be started at 25th time unit).

OSEKtime has a number of auxiliary system tasks that are not registered in a dispatching round (these tasks belong to *Tasks*, but do not belong to *STasks*). Examples of these tasks are

- *ttStartOS* routine starts the operating system,
- *ttIdleTask* acts as the idle task of the OSEKtime OS. It is the first task started by the OSEKtime dispatcher, and is always running if there is no other task ready. As this task is not registered in *DT*, it is not periodically restarted and does not

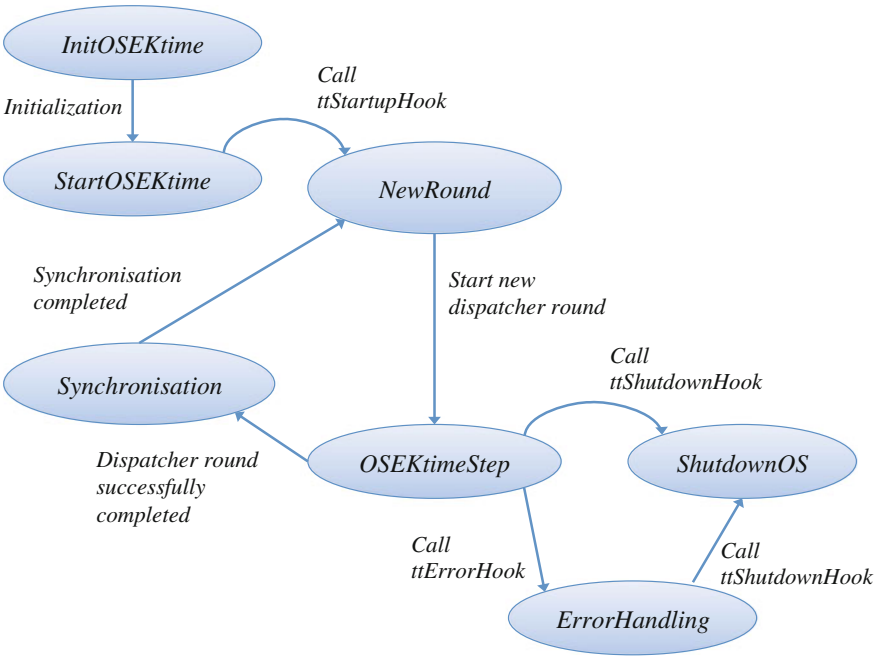

Fig. 4 OSEKtime as a state machine

have any deadline. *ttIdleTask* has the lowest priority and it can be interrupted by all interrupts handled by OSEKtime.

- *ErrorHook* task is responsible for error handling for deadline violations;
- *ttShutdown* task is responsible for shutdown of the system.

OSEKtime as an abstract state machine is presented in Fig. 4.

FTCom layer has the following core functions: *ttSyncTimes*, *ttSendMessage* and *ttReceiveMessage*. The *ttSyncTimes* function specifies the routine to synchronise the local time with the global time in the system. This function will be called by OSEKtime OS in the state *Synchronisation*. The functions *ttSendMessage* and *ttReceiveMessage* are called by tasks. *ttSendMessage* stores a message in the FT-CNI buffer, which is used to communication between nodes (in the case of a time-triggered automotive system, the content of the FT-CNI buffer has to be sent via FlexRay). *ttReceiveMessage* checks the FT-CNI buffer for the requested by an application message (i.e., the message with the requested id) and stores this message to the local buffer.

6 Future Work

The next step in our research is to implement results of the investigation presented here to our experimental electrical vehicle, RMIT University Autonomous System. Operating systems view is given in the Fig. 5a. At the highest level, RMIT University Autonomous Vehicle is controlled by a customised installation of Microsoft Windows Server. Majority of the higher level system processes are run on a dedicated Laptop PC as a computer hardware base for the whole system. Localisation, mapping and motion planning processes are running in MATLAB environment. The only process running outside of MATLAB is a Rapidly exploring Random Tree path planning generation, implemented with Python 2.7 as already explained and presented in [13].

All data acquisition activities are handled by an embedded C controller, running on an Arduino micro controller. Full proportional-integral-derivative (PID) control of actuators was implemented for vehicle velocity and steering angle control. Detailed presentation of the control process is given in [11]. RMIT Autonomous Vehicle during path following testing, in the real time, is presented in Fig. 5b. All this real time processing will be improved by aligning system with the TTP Model for Autonomous Systems, as shown earlier in Fig. 1. We expect that application of the TTP will increase reliability of the localisation, path planning and path following features, as it would allow us verify the corresponding timing properties in a formal way, also using semi-automated theorem provers. This would allow to use the RMIT University Autonomous System for safety-critical tasks, especially focussing on the interaction between humans and the AS.

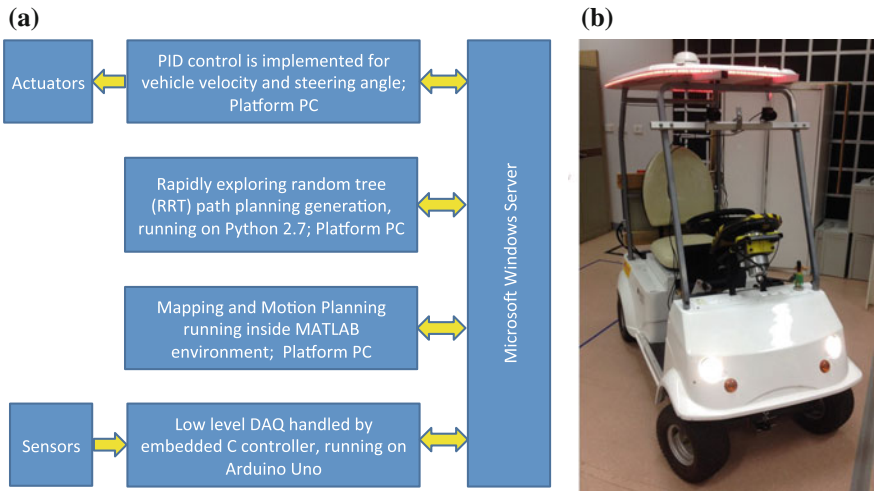

Fig. 5 RMIT autonomous vehicle. **a** Model to be aligned with TTP. **b** Path following testing

7 Conclusions

This paper presents an approach for application of time-triggered paradigm (TTP) to the domain of autonomous systems. TTP has been applied successfully to distributed systems in avionics and automotive domain. In this paper, we apply the well-developed ideas of time-triggered systems within the domain of autonomous robotic systems. The core feature of TTP is that all actions within the system have to be executed at predefined points in time. This ensures a deterministic timing behaviour of the system, which makes the system behaviour more predictable and provides many benefits for the analysis of the system properties.

We introduce a TTP model of an autonomous robotic system as well as the corresponding models of OSEKtime operating system and its Fault-Tolerant Communication layer. We also briefly discuss our future work on the application of the presented ideas to the experimental electrical vehicle, RMIT University Autonomous System.

References

1. aiT WCET Analyzer: Worst-Case Execution Time Analyzers. http://www.absint.com
2. Bauer, V., Broy, M., Irlbeck, M., Leuxner, C., Spichkova, M., Dahlweid, M., Santen, T.: Survey of modeling and engineering aspects of self-adapting and self-optimizing systems. Technical Report TUM-I130307, TU München (2013)
3. Blech, J.O., Spichkova, M., Peake, I., Schmidt, H.: Cyber-virtual systems: Simulation, validation and visualization. In: 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2014) (2014)
4. Blech, J.O., Spichkova, M., Peake, I., Schmidt, H.: Visualization, simulation and validation for cyber-virtual systems. In: Evaluation of Novel Approaches to Software Engineering, pp. 140–154. Springer International Publishing (2015)
5. Botaschanjan, J., Broy, M., Gruler, A., Harhurin, A., Knapp, S., Kof, L., Paul, W., Spichkova, M.: On the correctness of upper layers of automotive systems. Formal Aspects Comput. **20**(6), 637–662 (2008)
6. Botaschanjan, J., Gruler, A., Harhurin, A., Kof, L., Spichkova, M., Trachtenherz, D.: Towards modularized verification of distributed time-triggered systems. In: FM 2006: Formal Methods, pp. 163–178. Springer (2006)
7. Botaschanjan, J., Kof, L., Kühnel, C., Spichkova, M.: Towards verified automotive software. SIGSOFT Softw. Eng. Notes **30**(4), 1–6 (2005)
8. Broy, M., Stølen, K.: Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement. Springer (2001)
9. Elbanhawi, M., Simic, M.: Examining the use of B-splines in parking assist systems. Appl. Mech. Mater. **490491** (2014)
10. Elbanhawi, M., Simic, M.: Sampling-based robot motion planning: a review. IEEE Access, **30**(99) (2014)
11. Elbanhawi, M., Simic, M., Jazar, R.: Improved manoeuvring of autonomous passenger vehicles: Simulations and field results. J. Vib. Control (2015)
12. Elbanhawi, M., Simic, M., Jazar, R.: In the passenger seat: investigating ride comfort measures in autonomous cars. IEEE Intell. Transp. Syst. Mag. **7**(3), 4–17 (2015)
13. Elbanhawi, M., Simic, M., Jazar, R.: Randomized bidirectional b-spline parameterization motion planning. IEEE Trans. Intell. Transp. Syst. **17**(2), 406–419 (2016)

14. Feilkas, M., Fleischmann, A., Hölzl, F., Pfaller, C., Scheidemann, K., Spichkova, M., Trachtenherz, D.: A top-down methodology for the development of automotive software. Technical Report TUM-I0902, TU München (2009)
15. Feilkas, M., Hölzl, F., Pfaller, C., Rittmann, S., Schätz, B., Schwitzer, W., Sitou, W., Spichkova, M., Trachtenherz, D.: A refined top-down methodology for the development of automotive software systems—the KeylessEntry system case study. Technical Report TUM-I1103, TU München (2011)
16. Fredriksson, J., Nolte, T., Nolin, M., Schmidt, H.: Contract-based reusableworst-case execution time estimate. In: Embedded and Real-Time Computing Systems and Applications, pp. 39–46. IEEE (2007)
17. Hölzl, F., Spichkova, M., Trachtenherz, D.: Autofocus tool chain. Technical Report TUM-I1021, TU München (2010)
18. Kopetz, H.: A comparison of TTP/C and FlexRay. Technical Report, TU Wien (2001)
19. Kühnel, C., Spichkova, M.: FlexRay und FTCom: Formale Spezifikation in FOCUS. Technical Report TUM-I0601, TU München (2006)
20. Kühnel, C., Spichkova, M.: Upcoming automotive standards for fault-tolerant communication: FlexRay and OSEKtime FTCom. In: EFTS 2006 International Workshop on Engineering of Fault Tolerant Systems (2006)
21. Kühnel, C., Spichkova, M.: Fault-tolerant communication for distributed embedded systems. In: Software Engineering of Fault Tolerance Systems, vol. 19, p. 175. World Scientific Publishing (2007)
22. Lu, K., Li, Q., Cheng, N.: An autonomous carrier landing system design and simulation for unmanned aerial vehicle. In: Guidance, Navigation and Control Conference (CGNCC), IEEE Chinese, pp. 1352–1356 (2014)
23. Nolte, T., Hansson, H., Bello, L.L.: Wireless automotive communications. In: Proceedings of the 4th International Workshop on Real-Time Networks (RTN?05), pp. 35–38 (2005)
24. OSEK/VDX: Fault-Tolerant Communication. Specification 1.0. http://portal.osek-vdx.org (2001)
25. OSEK/VDX: Time-Triggered Operating System. Specification 1.0. http://portal.osek-vdx.org (2001)
26. Rushby, J.: Systematic formal verification for fault-tolerant time-triggered algorithms. In: Dependable Computing for Critical Applications, vol. 11. IEEE (1997)
27. Simic, M.: Vehicle and public safety through driver assistance applications. In: Proceedings of the 2nd International Conference Sustainable Automotive Technologies (ICSAT 2010), vol. 490491, pp. 281–288 (2010)
28. Spichkova, M.: FlexRay: verification of the FOCUS specification in Isabelle/HOL. A case study. Technical Report TUM-I0602, TU München (2006)
29. Spichkova, M.: Specification and seamless verification of embedded real-time systems: FOCUS on Isabelle. Ph.D. thesis, TU München (2007)
30. Spichkova, M.: Human factors of formal methods. In: In IADIS Interfaces and Human Computer Interaction 2012. IHCI 2012 (2012)
31. Spichkova, M.: Design of formal languages and interfaces: “formal” does not mean “unreadable”. In: Blashki, K., Isaias, P. (eds.) Emerging Research and Trends in Interactivity and the Human-Computer Interface. IGI Global (2014)
32. Spichkova, M., Blech, J.O., Herrmann, P., Schmidt, H.: Modeling spatial aspects of safety-critical systems with FocusST. In: 11th Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2014 (2014)
33. Spichkova, M., Campetelli, A.: Towards system development methodologies: from software to cyber-physical domain. In: First International Workshop on Formal Techniques for Safety-Critical Systems (FTSCS’12) (2012)
34. Spichkova, M., Hölzl, F., Trachtenherz, D.: Verified system development with the AutoFocus tool chain. In: 2nd Workshop on Formal Methods in the Development of Software, pp. 17–24. EPTCS (2012)

35. Spichkova, M., Liu, H., Laali, M., Schmidt, H.: Human factors in software reliability engineering. In: Workshop on Applications of Human Error Research to Improve Software Engineering. WAHESE'15 (2015)
36. Spichkova, M., Simic, M.: Towards formal modelling of autonomous systems. In: Intelligent Interactive Multimedia Systems and Services: 2015, KES-IIMSS, pp. 279–288. Springer (2015)
37. Spichkova, M., Zhu, X., Mou, D.: Do we really need to write documentation for a system? In: Model-Driven Engineering and Software Development (2013)
38. Verisoft XT Project. http://www.verisoftxt.de