An Approach for Structuring Heterogeneous Automotive Software Systems by use of Multicore Architectures

A.Knirsch[1,2], J.Wietzke[1], R.Moore[1] and P.S.Dowland[2]

[1]Faculty of Computer Science, University of Applied Sciences Darmstadt, Germany
[2]Centre for Security, Communications and Network Research,
University of Plymouth, United Kingdom
{a.knirsch, j.wietzke, r.moore}@fbi.h-da.de, pdowland@plymouth.ac.uk

Abstract

The significance of software within modern cars increases continuously. A manufacturer's competitive advantage relies more and more on compelling functionalities provided to the passengers. This raises the pressure on software architects of electronic automotive components. The successful integration of different software becomes a major challenge. This paper presents an approach to give support on that issue by the use of multicore hardware architectures. By reflecting the internal structure of the software in the underlying hardware, the scheduling of tasks performed in parallel can be determined based on the internal architecture rather than on performance aspects only.

Keywords

Multicore, Integration, Automotive, Parallel Development, Thread Affinity

1. Introduction

Modern cars are equipped with infotainment systems providing various services to the passengers. When observing current developments within the domain of such In-Car Multimedia (ICM) systems, an increasing amount of integrated functionalities can be noticed. The legacy radio within the dashboard has already merged with navigational and telephone devices. It provides capabilities to render video and audio media. To download add-on information, it is permanently connected to cellular networks and is able to synchronise content with mobile devices. This evolution affects the complexity of such highly integrated software systems, which causes a demand for software frameworks supporting the application developers. Such support has effects on the development duration and the quality of the product (Wietzke and Tran, 2005).

The use of a well-tested and approved framework facilitates an effective internal structuring of the software system. But a framework is only of limited use for a concurrent software development process, as long as it fails to provide support for the integration of different software. In this context 'different' is related to internal structure, priority models, vendor, or resource utilisation. An objective of this research is the development of a comprehensive software framework to improve the integration of software for embedded systems in the context of ICM devices.

1.1. Peculiarities of software development within automotive domains

The automotive sector has to consider several constraints which are relevant for building software systems (Pretschner et al., 2007). This includes the heterogeneous nature of the software, reaching from entertainment to safety and time critical control functions. Those are distributed across several connected Electronic Control Units (ECU) with interdependencies between them. The increasing amount of software within modern vehicles exposes the characteristics of complex IT systems. To keep the number of ECUs at a manageable size and in order to meet requirements regarding power consumption, space, weight and of course cost, more and more functionalities have to be combined and integrated. Additionally the amount of variants and different configurations has to be reflected by the software architecture. The resulting artefact can be compared with very large scale integration (VLSI) on the software level.

With 15 years and longer the lifetime exceeds those found with software systems in other domains, whereas the capabilities for maintenance are limited. Further the software engineering inherited the division of labour from the mechanical engineering of the automotive domain: each major component is designed and delivered by a different company. This causes increased efforts and risks for the integration of the targeted systems. The development process has to be coordinated independently of geographical borders, linguistic barriers, and specific domain knowledge while rendering a highly integrated system. This problem is going to be multiplied with respect to the expectations propagated by Volkmar Denner (CEO Robert Bosch GmbH) during his keynote speech at the Automotive Electronics 2010 in Ludwigsburg, telling that many non-automotive applications are going to be deployed into vehicles. Such statements are based on his perception that consumer electronics with its fast design cycles and short product lifetimes, is influencing the in-car environment, which will affect the entire design chain (Hammerschmidt, 2010).

1.2. Multicore architectures

Multicore (MC) architectures have been common in the high performance computing (HPC) sector for decades. In the recent past they have emerged and proved applicability also in server and desktop market segments, to solve the need for more computational power while improving energy efficiency. This is mainly driven by the fact that the increase of clock speeds to improve performance reached a physical barrier due to current limits in transistor technologies. This is also valid for the domain of embedded systems, where special purpose cores support the main processing unit to form a heterogeneous system-on-chip (SoC) MC architecture. But also homogeneous MC architectures are already available for different instruction set architectures (Levy and Conte, 2009). Those provide a number of advantages, some of the most prominent of which are outlined in (Smit et al., 2008) as follows:

Scalability is supported, as the architecture itself does not grow in complexity with future technologies. Only the number of provided computational cores increases, depending on the density of the integrated circuits and the size of the silicon. The computational power of MC CPUs scales linearly with the number of integrated cores, although the exploitation will suffer slightly due to necessary overhead.

Energy efficiency can be obtained by switching off unused cores to reduce the static power consumption. Also the clock speed might be dynamically adapted to current needs for computation tasks which do not have to fulfil hard real-time constraints for determinism. Energy efficiency increases with reduced clock speeds, resulting in a lower thermal footprint.

Independency of computational tasks is realised by space division on MC architectures in contrast to the time division manner of multitasked software systems executing on single-core systems. That means that MC systems support a parallel processing whereas single-core systems have to perform jobs concurrently (‘as if they were parallel’). MC systems still have to compete for shared resources. Functional dependencies are realised by using an inter-core communication network for routing information between the cores, also referred to as network-on-chip (NoC).

As well as the benefits of this new stage of parallelism in embedded systems, there are also some drawbacks. To utilise the facilities of multiple cores, the applications to be applied have to address the issues of software executed in parallel (Cantril and Bonwick, 2008). Eventually MC CPUs were basically introduced to avoid the physical problem of increasing clock-speeds to enhance computational power and not as a new feature to provide more parallelism, which the software developers have to cope with. But nevertheless MC architectures do provide an opportunity to reflect a parallel software design in hardware, as presented below.

The following section focuses on hardware architectures utilising multiple homogeneous processing cores, sharing a common accessible memory region connected by a NoC that supports multiple concurrent communications utilised by an Portable Operating System Interface for Unix (POSIX) conform operating system (OS), which does support symmetric multiprocessing (SMP).

This paper is structured as follows:

- Section 2 develops the need for a new approach for a software framework supporting the integration of heterogeneous software. The example of an ICM system is used for illustrative reasons.

- Section 3 defines a set of architectural drivers, which have impact on the internal structure of a heterogeneous software system. Further a basic approach is presented that utilises the qualities of MC processors. Within that context, the focus is set on the support for integration of software systems in order to ease the creation of a homogeneous whole.

- Section 4 summarises and provides an outlook for further research.

2. The need for a new software integration approach

Within the automotive domain the Original Equipment Manufacturers (OEM) obtain almost all of the electronic components, including the software, from suppliers (tier-one OEMs). These have to meet the requirements and design specifications with respect to the interfaces defined by the OEMs. This implies that theoretically all the components forming the distributed system within a certain car are compatible to the predefined rules and therefore are also compatible with the Electronic Control Units (ECU) they depend on. Although all parties agree to a common set of specifications, the increasing amount of functionalities incorporated into such distributed systems causes a lack of predictability regarding the compatibility. The maintainability as well as the capabilities to identify the root causes for unexpected behaviour decrease (Sangiovanni-Vincentelli and Di Natale, 2007). Although the reasons for those problems could be denominated as organisational or communication problems during the development process, a solution on that level is not foreseeable due to incongruent interests of the involved parties. A supplier usually is only able to provide a competitive offer as long as the efforts necessary can be reused for multiple clients. Therefore the agreed specifications are good and necessary but seldom sufficient for a predictable integration. The supplier has not necessarily much interest in easing the integration efforts of the OEM if his own return of investment will suffer. That approach leads to the situation as depicted by Sangiovanni-Vincentelli and Di Natale:

> *"The integration of subsystems is done routinely, albeit in a heuristic and ad hoc way. The resulting lack of an overall understanding of the subsystems' interplay, and the difficulties encountered in integrating very complex parts, make systems integration a very challenging job."*
> (Sangiovanni-Vincentelli and Di Natale, 2007, p. 42)

Based on the experience gained through an inter-institutional co-operation with a tier-one OEM of ICM devices, it is assumed it is not possible to adequately specify a system in advance with currently available design techniques. For example it can be questioned how to express the load behaviour of a software (sub-)system on different task priorities. This is particularly necessary by subsystems which actively and autonomously change their task's prioritisations to circumvent the need for explicit synchronisation by use of semaphores and mutual exclusions. The same is valid for ones which adapt with dynamic prioritisation changes to current system utilisation by use of latency measurements between tasks' change from 'ready' to 'running' state. Integration of such software can't be performed routinely, which applies even more if it is provided as binary.

One single supplier historically provides one ECU. It is contained in a separate box that is connected to sensors, actuators, and at least one field bus system of the in-car communication network. That box reacts to the input given by sensors and messages received through the in-car communication network. With increasing variability the complexity increases due to the also increasing number of combinations of valid system setups. This also applies for user-event driven systems which communicate with ECUs. A modern ICM system provides a good example for a software system relying on interdependencies with internal and external control units. Such units form logical sub-domains, supporting the abstraction of the overall complexity as outlined in figure 1. An integration of the domains implies the shared use of system resources.

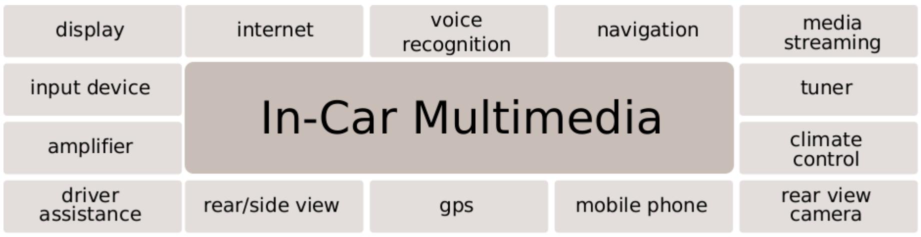

Figure 1: Exemplary domains of an ICM system

The vertical separation of the overall system functionality into ECUs to be integrated by the OEM is derived from the automotive mechanical engineering production process. If the content of a system is driven by the complexity and extent of the architectural drivers, it is an appropriate solution to parallelise the development process. By dividing a system into distinct parts, the resulting pieces can be handled independently as long as the interfaces of the arising modules are sufficiently defined. That strategy implies the coordination of a concurrent development process with a large number of parties and interests involved (Pretschner et al., 2007).

The tier-one OEMs adopt that ‘approved’ policy and alter from producer to integrators, as the ‘car manufacturers’ changed to ‘car integrators’ within the last decades. They also separate their targeted systems into subsystems, to be developed concurrently by independent software engineering teams within, as well as outside their organisation. In some cases the integrator is even able to make use of already existing components, originally targeted for a different purpose. In result, the development process follows the system’s functionalities’ breakdown structure. That way the process scales with the system complexity, but inherits the complexity derived by the dependencies between independent development teams of interdependent software components.

In contrast to the network of different (loosely coupled) ECUs, which basically share one (or more) common communication media, the software subsystems of a single ECU have to share all the resources provided by the underlying hardware. This includes computational power, memory, and access to external interfaces like the in-car communication network. The use of these has to be coordinated. For such highly integrated systems this can be realised by a priority based scheduling, supported through an underlying OS. By use of given priorities assigned to each task, the scheduler decides which one is allowed to compute and implicitly can have access to available hardware resources. A widely accepted scheduling algorithm is ‘round robin’ (POSIX: SCHED_RR) which grants each scheduled task up to a fixed time quantum before it is replaced (pre-emptive scheduling) by the next queued task that holds the highest priority. While the situation is more complex for SMP and MC systems, the decision on what to compute next is still essentially based on priorities.

Due to the concurrent development process, the system's components are developed in a simplified environment not including all interdependencies of the targeted system. But the tight coupling of the targeted system affects the behaviour and stability of implementations relying on concurrent execution on shared resources. A notable amount of ECU failures can be put down to insufficient software qualities causing not recoverable failing states (Broy, 2006). An enumeration of causes for errors related to parallel execution would contain for example dead- and live-locks due to insufficient thread safety and conflicting prioritisation. These can probably only be observed after the integration of all components, because they simply don't show up during an isolated development and testing (without shared resources).

Whereas the other causes for errors named above can be reduced to erroneous code, a conflicting prioritisation is more likely the result of inadequate communication during the implementation or insufficient specification during the design phase. Also the process of transferring the design to the implementation level, while utilising a task-based prioritisation could introduce deficiencies. On the design level certain priorities could be assigned to given use-cases and activities, which derived system components have to follow (e.g. a phone component might have to follow different priorities than a radio tuner component, especially when receiving a phone call). Additionally, components might use an internal priority scheme, due to the fact that internal activities also vary on importance – probably also related to the current state of the overall system. Ideally, healthy cooperation and/or management would ensure that a commonly agreed upon scheme eases the integration process. But, driven by issues of cost-efficiency, not only special purpose software but also OTS (off-the-shelf) and legacy parts have to be integrated. Those do not have to follow a commonly agreed scheme, due to a completely independent development (Pellizzoni and Caccamo, 2007).

Under the assumption that the complexity of automotive software systems is increasing continuously, which has an impact on the number of subsystems, the predictability of necessary integration efforts gains importance. The following section depicts how the integration process of highly interdependent software systems can be stabilised to improve predictability by use of MC hardware architectures.

3. Defining self-contained execution domains

When beginning to structure a system with predefined functionality where the architect possesses all degrees of freedom (no predefined architectural restrictions, OTS software or legacy code to consider), it is suitable to decompose the functionality into fine-grained separate tasks using software engineering techniques to transfer abstract functionality to implementation level. In this context a task is defined as the smallest schedulable execution item, like a thread in QNX Neutrino RTOS (Real-Time OS) or GNU/Linux OS. Based on certain constraints they can be partitioned into groups of tasks. Those can be agglomerated into execution domains, which can be mapped to the available computational cores as illustrated in figure 2. The constraints can be derived from the attributes of the individual tasks in association with their interdependencies. An appropriate way is to differentiate the tasks and the related implementation (if already available) based on their internal architecture, their shared memory regions, their vendor (especially for OTS parts), and their lifecycle or ability to be reused. Considering those factors will lead to a generally coarser grained segmentation in comparison to a design with all degrees of freedom.

The internal dependencies can be rated by use of design structure matrices (DSM). Those reflect the amount, frequency and direction of information exchanged between tasks (Sangal et al., 2005). Such dynamic information most certainly will change during system operation. Therefore an adequate system profiling is advisable to achieve expressive and effective results.

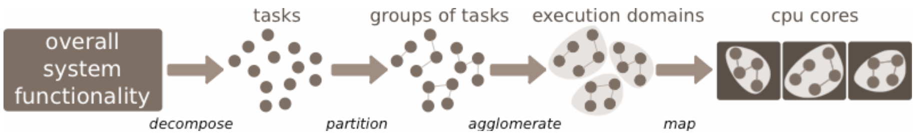

Figure 2: Decomposition and mapping on the hardware level

In the real world the decomposition and development commonly does not look like the process depicted in figure 2. To manage the complexity of the overall system, functionalities can be outlined using use-cases. Accompanied with further high-level design information the implementation can be separated into distinct chunks to be provided by suppliers. The task of the integrator is to unite the resulting chunks and arrange those onto the available hardware as depicted in figure 3. Architectural design concepts like use-cases, components and modules, for proper abstraction and modelling a system, help to cope with complexities. But the resulting design artefacts still have to be transferred to the level of implementation utilising the application programming interface (API) of the OS and programming languages.

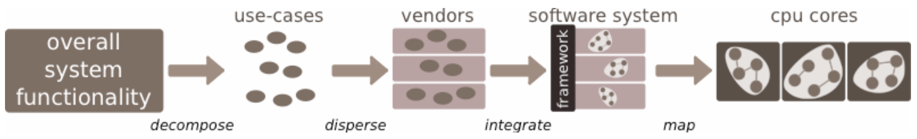

Figure 3: Decomposition and mapping on the system level

The performance of an integrated software system is highly dependent on efficient messaging and memory access. This is especially applicable for cost-efficient solutions which do not allow an extensive communication overhead introduced by an abstraction layer to support a loose coupling. Therefore inter-process communication (IPC) based on shared memory is still a method of choice in efficient environments opposed to more loosely coupled techniques. MC hardware architectures support this by coherent cache hierarchies to reduce memory access latencies by decreasing cache misses. That means the throughput can be improved when communication partners use a shared cache instead of the main memory. Information stored within shared caches can be accessed about 10 (or more) times faster than memory connected via system bus. An optimised arrangement of tasks can improve overall system performance (Tam, 2007). Therefore a sensible clustering of tasks on the CPU cores can be justified also for economical reasons when looking for time and space efficient solutions in terms of usage of computational and memory resources.

With SMP a single OS abstracts the hardware resources and employs a single scheduler to disperse computational resources based on predefined priorities. For a MC system this implies that tasks ready for computation are scheduled to available computational cores considering an equal dispersion of load while reflecting given priorities. Neither the constraints outlined above for grouping and bundling of tasks, nor the communication flows between the tasks influence this distribution. In best case a scheduler keeps a task on a particular core as long as possible to reduce cache bouncing effects. Cache bouncing increases cache invalidations which cause read and write misses and therefore rising latencies during memory access (Tam, 2007).

With the objective to ease the integration of heterogeneous software by utilising MC architectures, a software framework reflecting the depicted peculiarities improves usability. Such a system should be able to mitigate risks caused by unforeseeable integration expenses and thus help to predict the success of development projects. For the successful isolation of execution domains a set of essential capabilities has to be provided by an underlying framework in association with the utilised OS.

- Tasks and groups of them can be bound to certain predefined CPU cores.
- That binding is inherited to dynamically created sub-tasks by default.
- The scheduler supports the parallel execution of tasks, which make use of different priorities on different computational cores.
- The static mapping of tasks to cores appears transparent to the application developers and is managed only by the system integrator.

The capabilities given above constitute the minimum to implement the proposed approach. Further features can provide more flexibility basically on the integration level as well as during runtime. For example a logical hierarchy of agglomerates provides capabilities to dynamically merge and split groups of tasks to adapt to current computational needs for efficient resource utilisation.

The enabling technology for the approach is thread affinity (Love, 2003; Nagarajan and Nicola, 2009). The computational cores are addressable by use of bit-masks. With this feature the system scheduler is ordered to schedule a particular task only for computational cores matching the defined bit-mask. This introduces the dimension of space, beside priority and which task was queued first in the waiting line. Thus an integrator can define for any given agglomerate which core (or group of cores) the contained tasks are allowed to be computed on. He is enabled to define an execution domain in form of a dedicated core by adapting the scheduler's allocations depending on implemented use-cases, independent of the software's internals. This applies also for agglomerates, which aren't available in source code.

This approach was incorporated into an embedded software framework, developed at the ICM-lab of the Faculty of Computer Science of the h_da - University of Applied

Sciences Darmstadt (OpenICM, 2010). First evaluations served as proof for the applicability and validity: The API for the application developer did not need to be changed, whereas the integrator is now empowered to cluster the components on certain CPU cores with minimal efforts (only one additional argument for indicating the targeted core is necessary for the subsystems start routine). The framework enables a system architect even to reuse components not targeted for parallel computation. Further it provides the capability to optimise cache performance for appropriate partitioned systems, due to the achievable reduction of unnecessary cache invalidations. The implementation scales very well with increasing complexity, simulated by an increasing number of different components.

As a result of such a static configuration, the tasks are scheduled as defined by the integrator based on the interdependencies and predefined characteristics, rather than 'only' depending on an equal dispersion of workload. That means different software (e.g. different by means of vendor, change rate, internal structure, internal priorities, mission) is separated. A task with high priority does not displace a low priority task, as long as both are defined for different execution domains (which implicitly means they could make use of different priority schemes). This probably doesn't support a most optimal performance, but improves the deterministic behaviour and helps to reach a higher grade of stability. Erroneous behaviour is not necessarily propagated beyond the boundaries of one computational core (or one set of computational cores predefined with a bit-mask). Depending on the implementation it is even possible to handle failures of affected subcomponents (Aggarwal et al., 2007).

Even more positive effects can be expected in terms of timely behaviour. The scheduler does not have to merge concurrent tasks of different software for one computational unit. Therefore a component behaves very similarly in timing as to what is observed when it is executed exclusively on a single-core system, similar to a separate ECU, as systems were designed in the past (but without additional housings, power supplies, etc.). Imminent conflicts are effectively reduced, without the need for changing the components internals.

4. Related Work

Other related research on integration is performed by Vergata et. al., which use the availability of the virtualisation capabilities of current hardware architectures to separate different software components. This approach effectively creates execution domains by use of virtual machines, not depending on a fixed number of CPU cores (Vergata et. al., 2010). Increased overhead to establish and run those machines has to be accepted.

QNX Software Systems provide the capability to partition software by use of resource budgets with their Adaptive Partitioning Scheduler (APS) (Johnson et. al., 2006). This scheduler relies on application specific policies (round robin, FIFO, etc.) and priorities, overlaid by configurable budgets (guaranteed portions) of system resources. Although each of those capabilities can provide benefits as long as utilised separately, a combination of all three has the potential to interfere with each other. Related research is currently performed at the ICM-lab of the h_da.

A great deal of research for structuring and govern complex software systems has been done in the field of service-oriented architectures. Automotive software systems do not pose an exception (Krueger et. al., 2004), but with improved abstraction of the complexity the overhead during runtime increases.

The OSGi Alliance propagates an open dynamic component platform to assure interoperability of applications and services. The platform addresses the integration issue of software provided by different vendors with respect to reliable operation, shared resources and the ability to add functionality during runtime. With a sophisticated service model it follows a service-oriented approach, relying on a Java Virtual Machine (JVM) (Kriens, 2008).

The approaches described above give support on the integration of heterogeneous software subsystems. An examination of the research work reveals the lack of efficient exploitation of the underlying hardware architecture or neglect issues caused through conflicting scheduling schemes and task prioritisation.

5. Conclusions and outlook

The increasing complexities within the automotive domain are historically countered by a divide and conquer strategy. This multi-branched recursion targeting on breaking down a problem into manageable parts was adapted by the software development approaches implemented by suppliers, and their subcontractors. These have to integrate heterogeneous software parts into a reliable and efficient system based upon a single hardware platform. But the concurrent implementation of sub-parts involves a high risk of an unpredictable integration process. This paper has demonstrated that this approach to utilise MC hardware architectures to retain the structure of the software system by use of defined execution domains is an appropriate way to mitigate that risk. It is lightweight enough to be usable in practice and proved practicalness by adopting it to an existing embedded software framework.

Essentially, this paper has identified problems related to software integration on the intersection of evolving MC architectures and automotive software systems. A main focus is set on the question of how flexible and reliable software frameworks can support the integration process. However, the problems of concurrency remain for unique peripheral resources shared by multiple computational resources. Therefore additional research is needed to determine the correct allocation and best parallel use of shared resources. Further research is necessary on how a system can be prioritised efficiently on design level, including a consideration on how to transfer such information to the implementation level. Another important question is how an appropriate partitioning of systems based on a system profiling can be achieved.

6. References

Aggarwal, N., Ranganathan, P., Jouppi, NP. and Smith, JE. (2007), *“Configurable Isolation: Building High Availability System with Commodity Multi-Core Processors”*, In: Proceedings of the 34th annual international symposium on Computer architecture, p. 470-481, ACM.

Broy, M. (2006), "*Challenges in automotive software engineering",* In: Proceedings of the 28th International Conference on Software Engineering, p. 33–42, ACM.

Cantril, B. and Bonwick, J. (2008), "*Real-World Concurrency*", In: ACM Queue, volume 6, issue 5, p. 16-25, ACM.

Hammerschmidt, C. (2010), "*Bosch sees massive challenges ahead for automotive electronics"*, Automotive Design Line.

Johnson, K., Clarke, J., Leroux, P. and Craig R. (2006) *"OS Partitioning for Embedded Systems"*, QNX Software Systems.

Kriens, P. (2008), *"How OSGi Changed My Life"*, In: ACM Queue, volume 6, issue 1, p. 44–51, ACM.

Krueger, IH., Nelson, EC. and Prasad, KV. (2004), *"Service-Based Software Development for Automotive Applications",* In: Convergence International Congress & Exposition On Transportation Electronics, SAE International.

Levy, M. and Conte, TM. (2009), "*Embedded Multicore Processors and Systems",* In: IEEE Micro, volume 29, issue 3, p. 7-9, IEEE.

Love, R. (2003), "*CPU Affinity"*. Linux Journal, volume 2003, issue 111, p. 8, Specialized Systems Consultants, Seattle, WA, USA.

Nagarajan, S. and Nicola, V. (2009), *"Processor Affinity or Bound Multiprocessing? Easing the Migration to Embedded Multicore Processing"*, QNX, Ottawa, Ontario, Canada.

OpenICM (2010), Webpage of the OpenICM Framework, h_da - University of Applied Sciences Darmstadt, Germany, http://openicm.fbi.h-da.de (last accessed 20-Aug-2010).

Pellizzoni, R. and Caccamo, M. (2007), "*Towards the Predictable Integration of Real-Time COTS Based Systems",* In: Proceedings of the 28th IEEE Real-Time Systems Symposium, p. 73-82, IEEE.

Pretschner, A., Broy, M., Kruger, IH. and Thomas S. (2007), "*Software engineering for automotive systems: A roadmap",* In: International Conference on Software Engineering 2007, p. 55-71, IEEE.

Sangal, N., Jordan, E., Sinha, V. and Jackson, D. (2005), *"Using dependency models to manage complex software architecture"*, In: Proceedings of the 20th annual conference on Object-oriented programming, systems, languages, and applications, p. 167-176, ACM.

Sangiovanni-Vincentelli, A. and Di Natale, M. (2007), *"Embedded System Design for Automotive Applications"*, In: Computer, volume 40, issue 10, p. 42–51, IEEE.

Smit, GJM., Kokkeler. ABJ., Wolkotte, PT. and van de Burgewal, MD. (2008) *"Multicore architectures and streaming applications"*, In: Proceedings of the 2008 International Workshop on System Level Interconnect Prediction, p. 35–42, ACM.

Tam, D., Azimi, R. and Stumm, M. (2007), *"Thread Clustering: Sharing-Aware Scheduling on SMP-CMP-SMT Multiprocessors"*, In: Proceedings of the 2nd European Conference on Computer Systems 2007, p. 47-58, ACM.

Vergata, S., Wietzke, J., Schütte, A., and Dowland, PS. (2010), *"System Design for Embedded Automotive Systems",* In: Proceedings of the Sixth Collaborative Research Symposium on Security, E-learning, Internet and Networking (SEIN 2010), Plymouth.

Wietzke, J. and Tran, MT. (2005), "*Automotive Embedded Systeme"*, Xpert.press, Springer.