Shared SW development in multi-core automotive context

Lothar Michel, Torsten Flaemig, Denis Claraz, Ralph Mader

Shared SW development in multi-core automotive context

Lothar Michel[1], Torsten Flaemig[2], Denis Claraz[3], Ralph Mader[4]

1 : Audi AG, 84045 Ingolstadt - Germany
2 : Volkswagen AG, Brieffach 39200, D-38037 Braunschweig - Germany
3: Continental Automotive France SAS, 1, av. Paul Ourliac, BP 1149, Toulouse - France
4: Continental Automotive Germany AG, Siemensstr.12, Regensburg - Germany

Abstract

We present a methodology for the common development of combustion engine control Software between TIER-1 supplier and OEM. The classical approach of shared development used in single core projects has to be adapted to the new challenges of integration and protection, in the multi-core context. New integration and protection constraints are specified at design time, which are considered at integration and protection time. A common integration step is defined, where interfaces and constraints at the border are agreed. After that, each part can be modified and protected independently, enabling parallel developments by the partners.

1. Introduction

In the automotive domain, the body controller and chassis systems markets are driven by the integration of new innovative features, resulting in an increase of ECUs in the car, e.g. in an AUDI A8 with up to 80 ECUs. In this context, the multi-core technology is seen as an opportunity to slow-down the inflation of ECUs in the car, by enabling the integration of loosely coupled functions in one same ECU, as a kind of fusion process.

On the other side, the combustion engine market is driven by an increase of engine throughput, a reduction of consumption (CO2), and a reduction of emissions. This results in more complex systems with tighter real time constraints, and finally in SW sizes above 1.5 million of lines of code. Such increase of computation power can only be achieved by the use of multi-core platforms (Fig. 1). The challenge is then in this case to distribute a highly cohesive system on different cores, as a kind of fission process.

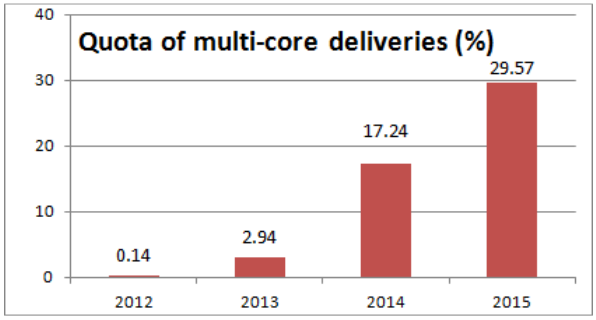

Fig. 1: Quota of deliveries based on multi-core CPU at VW/AUDI

In [1], it has been previously described how this challenge can be handled. We want now to focus on the common work between Continental Automotive, as TIER-1 (supplier), and Volkswagen Automotive Group, as OEM.
In an engine management system, a part of the functions is provided by the OEM, and another part is provided by the TIER-1, resulting in a complex integration of highly coupled SW modules and runnables. In addition, the reduction of time-to-market requires parallel development of these parts, on TIER-1 side and OEM side.

This paper describes the process developed between Continental and VAG to support an integrated shared development in a multi-core legacy (non AUTOSAR) SW, and is based on real project experience. The paper is organized as follows:

In a first chapter, we describe the context of engine control SW. We particularly elaborate on the high coupling of underlying modules, and the hard real time requirements of functions. The iterative development process and the need for parallel development between the parties are also explained.
In a second chapter, the general integration challenge is described. We introduce the concepts developed and in use internally, as well as across partners. We explain that an important step in the integration is the elaboration of a precise and exhaustive cartography of the SW.
In a third chapter, the data protection topic is addressed. We show the importance of this topic, in regard to the high coupling / data flow characteristic to engine systems. The basic mechanisms are developed, from the specification of protection requirements, until the implementation. We finally provide a comparison of 2 basic methods of intervention.
The fourth chapter describes the context of shared development and the different use cases. The need of defining a common architecture, a common integration frame, and the necessary adaptations of the integration and protection processes are explained.
Finally, in a last chapter, we provide the state of the art on these topics, as known from us. We draw a comparison of the standards AMALTHEA, AUTOSAR, and ASAM-MDX, which are the state of the art in the automotive domain. In particular, we point some weaknesses related to the shared development, and to the multi-core aspects, requiring evolutions of the standard.

2. Multi-core challenges at engine systems

Technical context

The importance of integration and data protection in engine systems context is due to the high coupling of combustion control functions, a unique situation in

automotive domain. Most of these functions control the same highly dynamic phenomena: the complete thermodynamic process from beginning of air intake till the end of exhaust pipe. The different sensors and actuators interact physically with each other, and therefore the corresponding SW control algorithms permanently exchange information signals.

A rough measurement of this coupling can be based on the number of exchanged signals (SW connectors in AUTOSAR language) (Fig. 2). In most of the cases, the exchange concerns 2 modules. This means one to one coupling (high coupling). In the other side of the spectra, some signals (e.g. engine rotation speed, air temperature…) are needed in many control laws, and therefore exchanged all over the SW. This means 1:n coupling, with n greater than 100 (i.e. 10% of the complete application).

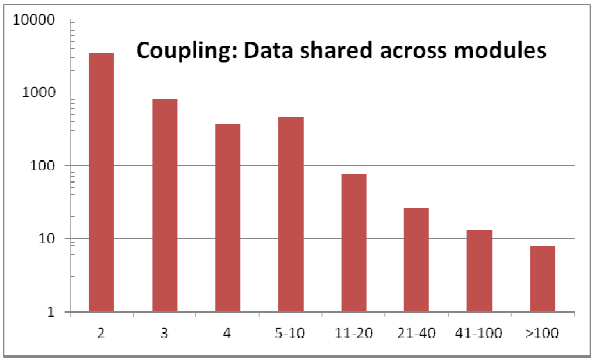

Fig. 2 : Number of modules sharing data

Finally, these signals are data implemented as simple scalars, complex structures, or arrays. For performance reasons, global variables are used.
But this module-to-module coupling gives only an overview of the static facet of the SW. These SW modules are based on several c-functions (executable entities in AUTOSAR language) executed at different rates: for one module, several executable entities might be necessary. Only in 10% of the cases, a single module ends-up in one single executable.

Therefore, ahead to the data flow between modules, a data flow between executables can be measured (Fig. 3), which gives a first idea of race conditions we have to tackle with.

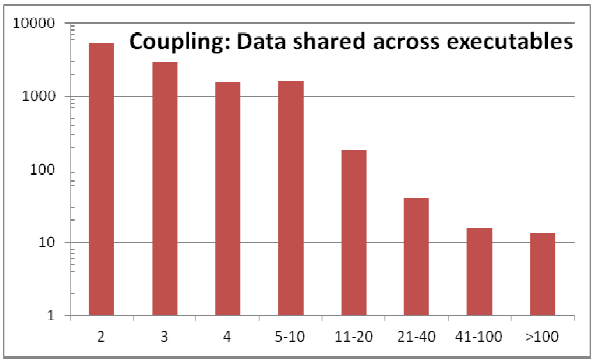

Fig. 3 : Number of executables sharing data

Such estimation gives a similar picture than on component level, but with a higher level of flow: While 70% of the data are encapsulated inside a module, only 30% of them are encapsulated inside one executable.

Finally, the full picture on the data is as follows:
- 1/3 are local to one executable
- 1/3 are exchanged between executables, but local to a module (“inter-runnable variables”)
- 1/3 are exchanged between executables of different modules (“sender-receiver”)

As these executables (8.000) are scheduled from more than 60 operating system tasks (ranging from 1ms to 1 s, mixing timing and angular frequencies, distributed on different cores), a significant part of the data flow is subject to race conditions in our multi-core environment.

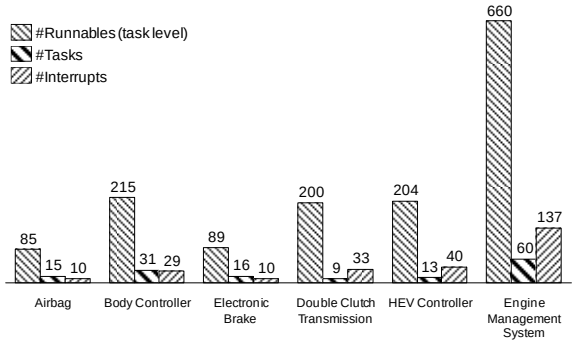

Fig. 4 : Number of dynamic artifacts for integration in different domains

Of course, on the static aspect, the mentioned coupling is reduced using SW composition, to allow a platform/reuse approach. But this has no effect on the race conditions.

Challenges

Therefore, the introduction of multi-core in engine systems is a challenging task.

On the scheduling and integration side, new constraints are adding complexity: The number of integration containers (tasks) increases, the relation between them is more complex (parallelization, chaining...), new types of constraints show up (affinities...), as well as new distribution strategies (all SW on one core, time-dependant SW on one core, safety-related SW on one core...), and the different partners (OEM, TIER-1, 3rd party...) may have different views/constraints on how to utilize the different cores.
On the protection side, a SW running previously in a protected single core cooperative environment (a.k.a. fixed priority with deferred preemption scheduling – FPDS) has now to support parallel execution. In particular, to achieve a maximum flexibility of the SW distribution, which is motivated by the high variability of project configurations, the module designs have to be independent of any core consideration. For instance, 2 runnables of the same module might run on 2 different cores – or not – depending on the project reusing the module. The same module must even still be reusable on the single core projects still under development.

Finally, due to the tight economic constraints, a complete rework of the existing SW is not affordable. Therefore the methodology has to cover the legacy SW not designed for multi-core.

3. Integration process

General

The topic of integration in this context covers the integration of one or more runnables of a software component or composition into one or more tasks of the complete software system, performed by an integrator. In this phase the correct position of the runnable in the sequence shall be determined and the necessary protection of data against concurrent access shall be generated, to ensure the stability or coherency of the data used by the runnable. It is further on assumed that the provider of the runnables and the integrator are time-wise and location-wise separated from each other as a consequence of worldwide software development of large scale software project.

Dynamic requirements for runnables:

For the correct real time behavior of the software functions in the target application, it is necessary to describe unambiguously the dynamic requirements for the integration of the runnables into existing tasks ("dynamic integration"). This is true for runnables of the supplier in context of a platform development and especially for runnables of the OEM, when looking on integration use cases in different applications of the same or even of different suppliers.
Due to this different use cases it is important to describe rather the requirements than a given solution. Describing the solution might be sufficient for one project but would manifest many design constraints to other projects, especially when looking across multiple suppliers using different architectures.

The minimum requirement on dynamic integration is the description of the point in time, when the runnable shall be executed. In the Continental PowerSAR architecture the available points in time ("rates") are standardized in a Reference Architecture and are called SystemEvents. They are characterized by different attributes like required minimum and maximum period, guaranteed deadline, and more. For each Runnable to be integrated, a so-called RunnableEvent is created, which specifies the integration constraint of this runnable by referring to an existing SystemEvent. This RunnableEvent should not be confused with the AUTOSAR RteEvent, which defines more an integration solution than a requirement.
Still missing are requirements concerning the relation between runnables using the same SystemEvent. For the description of requirements for a sequence one could either describe a relation to one or more runnables by name, the so called "Execution Order Constraint" (EOC) or use a requirement concerning the allowed age of a data, which is consumed, named "Data Age Constraint" (DAC) (Fig. 5).

The type of constraint to be used depends on the area of responsibility for a part of the software. Within a software composition where the responsibility is at a single person or small group of developers, EOC might be quite efficient, as giving the order of executables of the own composition combines a lot of refined requirements concerning the data flow and allows an easier description of the sequence requirements. On the other hand most of those execution orders are needed because of the data flow of some "important" data, which contribute to the dynamic behavior of a complete event chain. With EOC this information might get lost and in case of repartitioning or renaming of the software: the EOC might become invalid.

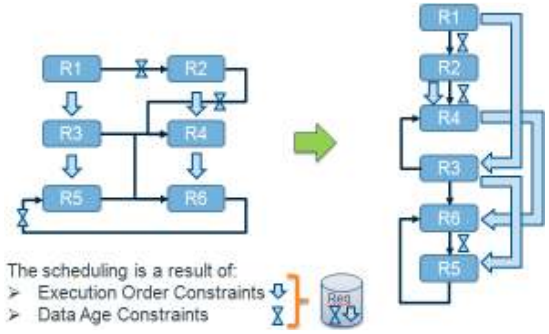

Fig. 5 : Solving Sequence of runnables using Constraints

When using a DAC, one is working on the interface of the runnables and does not depend on runnable names. In a model where only data which are consumed by a runnable are described with a DAC the requirement doesn't become invalid in case of a repartition of the runnables. If the data use is changed, this attribute has to be considered as well. This is especially helpful in a shared development context as the interfaces are the subject of discussion when defining the interaction of OEM and supplier software e.g. in Sequencing Workshops (See Fig. 6).

Fig. 6 : Effect of position in the sequence on the data flow between supplier and OEM runnables

Due to the complex coupling and data flow inherent to engine systems, an exhaustive analysis and resolution of data precedence problem is not possible: the problem has not always one solution. Therefore, the principle of the DAC approach is to identify, among all possible flows, the few ones which have a real impact on the system. For instance, low dynamic information, like the air temperature might have one – or more – recurrences of delay without big impact on most of the functionalities. At the opposite, having the wrong value for a cylinder index can be dramatic for the injection controller.

Today this information is exchanged via non formal requirement specification in a textual way due to the fact that MDX V1.2 [2] doesn't offer means for a formal description and most OEMs didn't migrate up to now to AUTOSAR, where the timing extensions would offer a possibility to describe a RTE Event and in addition runnable sequence needs or a DAC.

In addition to these two types of constraints, EOC and DAC, another concept can be used: the phase concept [3]. It consists in partitioning the time domain according to standard features in automatism. As example, logically, the runnables dealing with acquisition and diagnoses need to be executed before the runnables dealing with output commands. Therefore, a phase is associated to a given runnable, according to its inner functionality, and therefore is valid independently of the integration environment. This leads to a fully reusable and independent integration constraint. This method has been introduced at Conti since 3 years now, and used internally. Nevertheless, its extension on shared development with external partners is more difficult.

SW Cartography

For a correct integration and protection of the SW, it is necessary to know the whole structure of the code and the data accesses for read and write performed throughout the whole call graph of the application. No grey area shall be left over, as it can have severe consequences. As precise the cartography will be, as precise the protection and integration will be.
Verifying a DAC is only possible if the full data and control flow of the relevant sequence is known. The same applies to EOC, where the involved runnables are not mandatorily those directly integrated in the sequence.
Depending on the type of the SW which has to be integrated, this cartography can be established based on C-code, MDX description, or ARXML SW-components, a mixture of these use cases needs to be supported.

4. Protection process

Structuring the problem

In terms of data protection against race conditions, two types of problems are taking special importance in multi-core context: data stability and data coherency.

Stability: As soon as runnables can execute in parallel, and exchange data, it becomes very probable, that a data is modified while it is used on another core. If the data is scalar, and atomic, each individual read access cannot be corrupted by a write access on another core. The read access point will get the newest or the old value. But, if there are several read accesses, or if the same value of the data is expected across 2 successive runnables, then the stability of the data might be corrupted. In some cases, this might be acceptable, because for instance, the same data is used in different decoupled parts of the algorithm, and/or because the data has a low dynamics and only a small change of its value is possible. But in other cases, like for instance for booleans or state machines, the impact of a change during an algorithm can be severe.

Coherency: The modification of an elaborated data (structure, array, a set of atomic data…) on one core while it is accessed on the other core can have severe consequences, too. For instance, 2 exclusive information (a flag and its complement…) need to be written and read in a coherent manner. The reading of the data set might be "interrupted" by the writing of these data on this other core. Or on the opposite, the writing of the data-set might be "interrupted" by the reading on the other core. Even both cases can happen. Here again, this does not concern all variables of the SW, but only sub-sets. For instance, in one given algorithm, it is often the case that variables from different rates are used, and therefore cannot be coherent, by essence.

Consistency needs

Considering the huge data flow across runnables and tasks, as depicted in chapter 2, our approach is to be selective on the data and runnables to be protected. Ahead of limiting the HW resources consumption, this limits the protection cases that may have negative functional impacts.

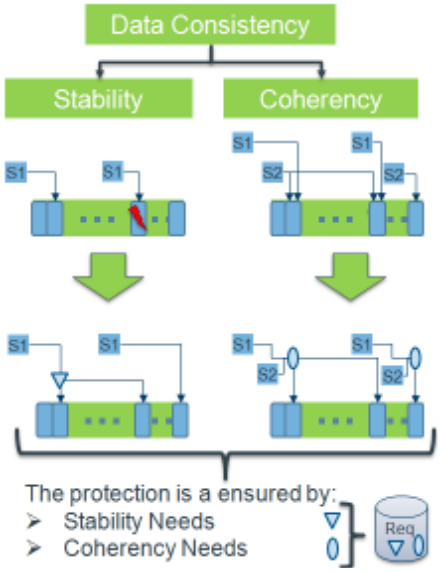

Fig. 7 : Ensuring data protection using Stability and Coherency Needs

Therefore, functional consistency needs are specified at design phase by function experts: stability of certain data accessed by distinct but coupled executables, and coherency for sub-sets of coupled data in certain executable. The function experts are responsible to specify the stability and coherency needs where required, and only there. In this way, the learning and development effort related to multi-core is minimized, and the function experts can concentrate on their core competence: physics and control laws. They have to concentrate on the "what" (to protect) and not on the "how" (to solve the protection).

It has to be noted that the initial AUTOSAR approach ("implicit communication"), where protection was applied everywhere, has been modified in AUTOSAR 4.1.1[4] (and in ASAM-MDX 1.3 [5]), with the introduction of

Data Stability and Data Coherency Needs. But this represents a significant evolution in the RTE (explicit and implicit) communication paradigms.

At integration and protection step, the requirements for data consistency are analyzed and checked against the project architecture (task configuration) and cartography (data flow). In case a consistency is required, a race condition is requested, and the SW is ready to be protected, then the protection of the relevant data is established in the relevant executables.

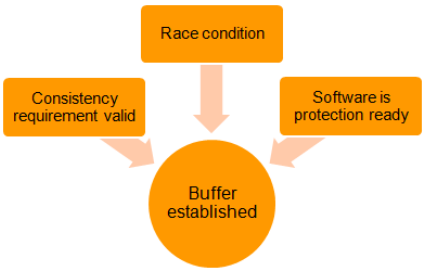

Fig. 8 : Data consistency buffer evaluation

To cover the simple cases where consistency is systematically required, and to reduce the effort of needs specification, complementary automatic strategies are used. For instance, coherency of non atomic (64 bits) data and stability of multiple accesses of same data inside an executable do not need to be specified.

Protection by buffering

The protection of data against race condition is mostly (but not uniquely) ensured by a buffering mechanism. It consists in copying the data in a task-local buffer, and using the buffer instead of the original variable in the algorithm. Fill and flush routines are inserted in the program flow (task bodies) in order to copy the variable into the buffer, and vice versa.
In the AUTOSAR implicit communication, the copies are done at beginning/end of task, resulting in long "buffering segments", and high resource consumption. In classical approaches, the copy is done at beginning/end inside each runnable, which is resource consuming too, but in addition does not ensure consistency across runnables.
In our case, the copies are done along the task to avoid these drawbacks. For instance, the filling of the data into the buffer is done at the latest possible position in the task ("as late as possible"), in order to benefit from the latest available value in the global system.

Access Modification in executable

For modifying the data access to a buffer access in a legacy SW, 2 basic techniques are available:

The Data Reference Modification (DRM) [6] consists in changing the address of the original data by the address of a buffer in the binary ELF file. Here the integration of object code sets some limits in terms of protection. For instance, in case of multi rate executables the function design has to be modified as the DRM process allows data protection in only one context (variable address substituted in binary by buffer address). In this particular case of multi-rate executables, the resulting re-design might have other drawbacks like e.g. code duplication.

	DAM (source code modification)	DRM (binary code modification)
Need of source migration	Yes: Accessers (GET/SET) have to be added to the code. But, a migration can be done by a tool; and code generators can be adapted, in the MBD case. Finally, a similar migration is required for AUTOSAR introduction.	No. But, redesign of the functions might be required in special cases.
Source code exchange	Yes. But some TIER-1 and OEMs are used to work with obfuscated code. In some cases, the OEM functions are even coded by the TIER-1. Finally, AUTOSAR process might need source code exchange, in context of engine systems.	No: object code is the standard for IPP, in legacy context.
Verification and validation	Early: Modification of accessers can easily be checked at compile time.	Late: Need to compile and link before having the modification.
Compiler chain independence	Yes: Accessers modified on source code level.	No: Same configuration of the compiler chain across all partners. Furthermore, the chain has to support DRM technique.
Openness to complex cases	Yes: Step by step, new use cases are supported, which need a more complex redirection of the Accesser, than a simple address modification (e.g. multi-rate cases). Furthermore, the addressing mode to the buffer can be different than the one to the original data, to gain performance.	No: Only address modification can be done, limiting the possible intervention.
Coherency cartography vs. intervention	Yes: the cartography of the SW and the intervention (accesser modification) are based on the same model: The SW-code. This guaranties the global coherency.	No: Unless the cartography is based on the obj. code, which is late in the process, there might be a gap between the cartography and the real implementation in the binary leading to severe mismatches on the protection.

Table 1: Comparison of Data Reference Modification and Data Accesser Modification

The Data Accesser Modification (DAM) consists in modifying the source code. Standardized APIs (data "accessers") are used in the code, and can be redirected - or not - to the buffer, using an include file. This technique, which is similar to AUTOSAR, gives more flexibility and optimization potential. For instance, it allows runtime context dependant accessers. On the other hand, the different development parties like to protect their IP and are reluctant to share source code, which is necessary for the DAM process.
In Table 1, we provide a non-exhaustive list of pros and cons of each technique.

5. Shared development

Development process

One additional dimension of the SW development for engine systems is the increasing integration, and therefore mixture of SW-components coming from different sources, and often provided with different formats (Fig. 9). It becomes a classical use case, for instance, to integrate SW functions from the OEM in the TIER-1 ECU, as well as components from 3rd party. In extreme cases, the TIER-1 has to integrate SW modules from different OEMs (engine co-developments), or even from own competitors. The amount of external SW may be high ("box-business" model, where the TIER-1 only provides the ECU plus the BSW), or on the opposite, null ("full turn-key" programs). In between these two extreme cases, the SW-part from the OEM to integrate might be provided as models, C-code, or object-code. It might comply with AUTOSAR standard, or simply be legacy SW.

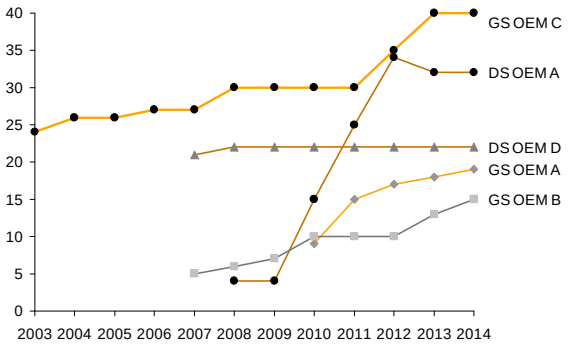

Fig. 9 : Percentage of OEM part in total program code

Of course, this "OEM plug-in" (monolithic or not) has to be integrated in the existing task system, as easily as the rest of the SW. Integration constraints are therefore discussed between the parties. Constraints on the needed SystemEvents are defined, as well as sequencing constraints or event chains. The core distribution is finally derived from all those requirements. The used protection methodology has to give a maximum of flexibility in order to meet all use cases. The OEM plug-in has to be protected against race conditions, like the TIER-1 SW, using the same principles, but with DRM instead of DAM due to IPP reasons.

In total, a complete project lasts in average 24 to 36 months, during which several synchronizations are done between TIER-1 and OEM: releases of the OEM plug-in to the TIER-1; and releases of the complete integrated SW from the TIER-1 to the OEM. Similarly to the TIER-1 functions, the OEM plug-in integrated in the ECU follows a development cycle, and is updated several times during the project life-time. To shorten the development loops, the OEM needs to be able, "at home", to further develop, re-integrate, and validate his functions. Therefore he must be able to build again the system. This kind of process was a market standard in single core engine systems, where the TIER-1 SW is delivered as object code, and the OEM plug-in is modified, re-integrated, and re-compiled on OEM side without intervention of the supplier. Thanks to cooperative scheduling, a policy extensively used at Continental, there was no need of special mean to ensure data consistency.

Now, as the OEM wants to distribute his SW over different cores, and as the protection of the SW requires a particular analysis and treatment, a segregation of the SW is done, between OEM and Supplier. The TIER-1 part is frozen, protected, and compiled at TIER-1 side, with a first version of the OEM part. Library files are released to the OEM, with a build environment. Starting from this point, the OEM part can be modified, and re-protected at OEM side. This means that independent buffering strategies are applied on the different parts.

At the end, the shared development process can be seen as an alternative or a complement to Rapid Prototyping, enabling short development loops between TIER-1 and OEM.

Common architecture

In this code sharing context, a common understanding of the basic system behaviour is essential to be reached. Right in the beginning, a definition of the features provided by the Operating System has to be negotiated and agreed: a common dynamic architecture. This common architecture has to enable an efficient protection, an easy integration, and it has to support simulation and validation of the scheduling.

First (and already known from single core systems) a set of common SystemEvents, as defined in chapter 3, has to be defined. It might be a subset of the complete set needed by the TIER-1, plus some extensions. Then, these SystemEvents are implemented as tasks, which have a priority and pre-emption behaviour, and a core allocation. In an engine management system typically we have a mixture of time based system events (from 1ms to 1000ms) and angle based system events (crankshaft and camshaft synchronous). Typically, different system events with a same angular period, but different phasing relative to the Top Dead Centre might be required.
But, in addition to periodic system events, sporadic initialization events, such as ECU start-up, ignition key transitions or failure memory clearing have to be specified. Here mainly, the precise position of the event, and its system meaning have to be clear to all parties.

The principle of initialization events is inspired from the object orientation concepts of constructors and destructors, and allows having a coherent system initialization across the complete SW. This concept, used in single core projects, has been enhanced in the scope of multi-core context, as it is important that all cores "toggle" get initialized synchronously and coherently.

Finally, with the step-wise deployment of AUTOSAR, it becomes important to fix also some basis on the use of the RTE, as there are different interpretation/use between partners can cause severe incompatibilities at integration time. Therefore, in the definition of this common architecture, the AUTOSAR configuration has also to be addressed, in particular if the OEM wants to integrate AUTOSAR SW and therefore fixes some implementation choices in its SW-Component descriptions.

For an OEM like VAG the challenge is to find a system setup which is similar in all TIER-1 ECUs, and which is proven to work in a multi-core context. For the above named SystemEvents, this is possible across all ECU suppliers. But the SystemEvents abstract implementation details like OS configuration of priority and pre-emption, core allocation, task chaining and handling of sporadic system transitions (synchronized or not). With multi-core, a new complexity is added to the system.
For a TIER-1 supplier like Continental, the challenge is to find a setup which fits to its generic functions, as far as generic functions are integrated in the project. In effect, in front of the TIER-1, there is a high variability of OEMs, with different visions of the architecture. The TIER-1 challenge is then to fill the gap between the different visions.

Common integration frame

With the ongoing change from single-core to multi-core systems the rising complexity of integration has a huge effect on the software development. It extends the high-level goals of the classic software development, like high reusability, reliability and correctness, IPP. Additional methods are needed, to achieve a closer examination of the sw-architecture. The given heterogeneous tools for integration and protection on the different TIER-1-side must be enabled through standardized general description of integration and protection needs on the OEM side. Additionally it is necessary to consider legacy software, because it is not divisible with further ado. Additional specification and in some cases refactoring is needed.

The main challenge for efficient integration on multi-core systems is an independent partitioning of the software that can run in parallel. But for that it is necessary to find and describe the dependencies and avoid conflicts, to protect and minimize inter-core data-access. Since two years, Volkswagen, Continental and other TIER-1 work together on a model-based approach which resolves the following main question of shared development for multi-core:

- What and how to specify and define the integration and protection needs?
- On which architecture level should the specification be done?
- What is additionally required in methodology and collaborative process?

This model-based approach is a continuous roundtrip from specification until verification [7]. As Fig. 10 illustrates, this roundtrip includes all needed steps in the shared context, considering the typical iterative engineering process in automotive industry, where the software is developed in several iterations of the V-Model and with this has several different releases with different maturity and quality. In a first step to generate the basic element for all considerations in shared context – the system model – a model consolidation combines system descriptions including hardware, operating system and TIER-1-software information given in the AMALTHEA format with the software description including the OEM-software information and requirements in the MDX format. This basis element combines the required information. It is the fundament for shared methods, like the already introduced sequencing workshops. It extends the collaborative software development with new information to be exchanged in new or enhanced formats.

With a consolidated system model, the software architecture can be visualised and graphically annotated with requirements, while analysing the data flow and signal paths. In this second step typically the TIER-1 is responsible to process the requirements and define the final system design. The system-model and defined requirements can be used with tool-support to find a pareto-optimal design, with efficient resource usage and requirement fulfilment. In the third step the updated system design is checked up against the requirements with simulation of the dynamic behaviour in an early design phase, before final integration to take into account, that the design is executable and fulfils all requirements for the given target hardware. Finally in a last step after integration and measurement, the system is verified with evaluation of the requirements taking real software or hardware traces, which includes all system events on required call tree level for referenced elements in the requirements.

In each step enhanced and complex tools are necessary which should interpret the information given from each partner in adequate and standardized exchange formats.

The specification of integration and protection needs, like coherency groups or data ages should be done on an abstract level, considering the design rules of the SW architecture. On SW composition level the requirement engineering is feasible for legacy software and considers the existing development process and given static architecture, because SW compositions already encapsulate functional dependencies given from requirement and architecture engineering. It reduces the costs in development process because complexity and efforts are reduced to specify the integration and protection needs. For that different architecture views have been created, each one fitting best to the use case of requirement engineering.

A static architecture view visualises the logic hierarchical grouping of the static software structure and their interfaces. On this first view it is recommended to define, analyse and check the more static requirements, but also signal grouping for coherency needs are possible.
A second view, the dynamic software architecture view shows more the design of tasks, runnable sequences and their data accesses and data flow. This view is typically used to discuss and analyse dynamic dependencies like execution order constraints, data ages and event chains on runnable level in sequencing workshops.
A new developed view combines both worlds, the static and dynamic architecture: It shows the runnable groups in a SW composition grouped for their period. If it is defined, that this SW-composition specific runnable groups are indivisible (what means the scheduler should not interrupt this group), then it is recommended to define intra execution order constraints for this groups and use data ages for the inter execution order.

Another aspect concerns the compatibility of the SW components to each other. For instance, in order to reach a similar level of parallelism, it would probably be useful to have a similar approach of developing SW components (e.g. rules, patterns) across the parties. Therefore, in a joint research project we look for patterns which are suitable to develop SW components utilizing a high degree of parallelism. These patterns shall be described in a kind of cook book which can be used by OEM developers as well as TIER-1 developers.

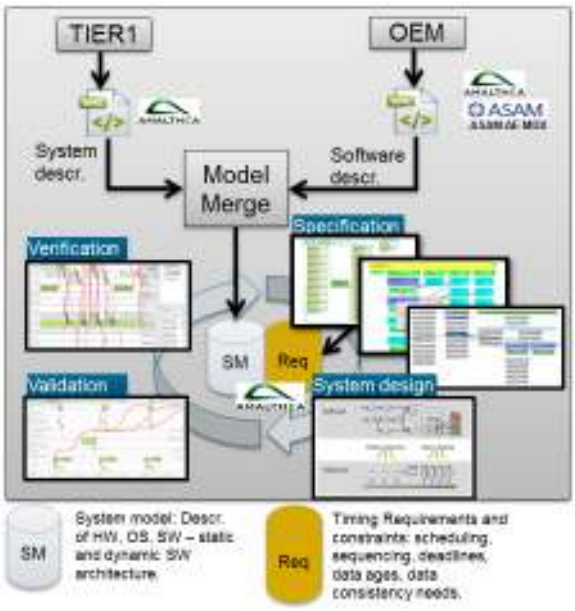

Fig. 10: Continuous Roundtrip for shared model-based system design in multi-core projects

Then there is the topic of IP protection: even in a close collaboration the different partners need a good level of IP protection. The exchange of source code is maybe not the best solution to reach this goal, unless new technologies like remote build or obfuscated code are used. Object code can be used as exchange format, but with the other drawbacks already mentioned. Due to the permanently increase of inter-penetration of different parties in the final SW, this topic is gaining importance. In addition the exchanged information of the system- and software-descriptions needs IP-protection. For this data it is necessary to obfuscate specified signals and runnable names for IP-parts of the software. But it is recommended to obfuscate only as much as really necessary but not more, otherwise needed information to analyze and specify dependencies and requirements for the interaction of the OEM- and TIER-1 software are lost and in effect the specification in this shared context isn't possible.

Common standard formalism

In order to reach a smooth integration into the defined integration frame the SW description and the specified timing requirements have to be exchanged between the parties. This exchange has to be based on a machine readable standard format, as integration and simulation processes of such complex system are only possible with tool support. Further on the use of a standard helps to define a common understanding of system features and forces the usage of a common wording. As each standard has some space for interpretation, the harmonization of semantics and used tags is necessary.
In cooperation between OEM and TIER-1 the ASAM MDX file format is currently widely used for SW sharing in non-AUTOSAR context. In those projects MDX is used to deliver information necessary for integration purposes to the integrator of a SW component.

The MDX standard is well defined for data definition purposes, as well as for the exchange of SW features information. Also variant handling can be described via system constant definition and settings. Basic scheduling information can be transferred to the integrating party.
As described above, in projects using multi-core CPUs there is the need to exchange further information:

- Data flow information – this currently possible with the existing MDX standard V1.2, but the data access frequency (access multiplicity) has to be added for a more detailed view on the real SW.
- Data stability needs – not defined in V1.2
- Data coherency needs – not defined in V1.2
- Data Age Constraints – not defined in V1.2
- Scheduling requirements for runnables – already possible with V1.2
- Scheduling dependencies between SW components – not defined in V1.2

A new version of the MDX standard has been defined to address the missing topics: the V1.3 released since June 2015.
Data stability groups can be specified as well as data coherency groups using new tags in the SW collection area.
Data Age Constraints and access multiplicity in one executable can be specified on data access elements in SW services (runnables).
With these extensions a SW component provider is able to exchange the defined timing requirements and constraints of its SW components to the integrating party as discussed above.

As there is a lot of legacy code at VW/AUDI this way has been chosen to bring these SW components to the new multi-core world, limiting the effort of reengineering.

After defining this step in exchange format, the focus now changes to:

- implementing of MDX V1.3 in development tools
- gathering all the requirements and constraints to be transported via MDX V1.3
- training the teams to this new process

In the near future the new features of the MDX standard will be used in practice.

The contract = Interface freeze

Freezing of interfaces consists of mainly two steps. The first step happens at the end of interface and sequencing workshops when the interface is agreed between the involved parties and fixed in terms of mapping of content, names, ranges, resolution, DAC and EOC constraints.
The second step is performed, when the interface adaptation is implemented at the TIER1 and the OEM receives software for the parallel development. To manifest and freeze the contract, the software has to be prepared in a certain way.

Protection adaptation: Part Management
When integrating TIER-1 and OEM SW parts, all integration and protection Needs are collected. Each artifact in the project (data, runnable, module…) is allocated to one of the three parts: TIER-1, or OEM part.
This allows applying different buffering policies on the different parts: For instance, if the OEM has not a proper description of the protection Needs (stability, coherency), then an automatic strategy can be applied, which are not necessary on TIER-1 side.
Also, this partitioning allows to select between different formats for the input (C-code, ARXML, MDX), but also for the output (DAM, DRM).

Finally, the final goal of a clear partitioning of the SW is to minimize the interactions between the parts (or at least to concentrate them in an adaptation part) and to enable a partitioning of the protection process. For instance, dedicated buffers, dedicated copy routines, dedicated task sections can be defined and fixed for one part, while the other part is updated. It allows an independent build of the SW at OEM side: The TIER-1 SW is built (protected, compiled, validated) at TIER-1 side, while the OEM SW is re-built as many times as requested, at OEM side.

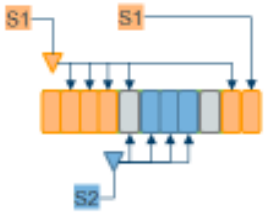

Fig. 11 : Stability Needs apply to different parts

In Fig. 12, we show the resulting buffering for a concrete example. The TIER-1 and OEM runnables are identified by their respective colour. Different buffers are used in the TIER-1 area, which are not reused in the OEM area. The OEM area can then be modified independently of the TIER-1 area, and re-built.

Parallel development / Parallel builds

Having a frozen interface allows starting a parallel development in various stages depending in the amount of changes and the timing requirements of the developed solution. Each partner (OEM, TIER-1) can modify its part without impacting the other one.
When doing only small changes, a parallel development using a tooling for internal bypassing (e.g. eHooks) is appropriate. The limitation of the internal bypass is mainly due to the tight internal resources (RAM, ROM) of the controller.
If the changes grow, or completely new functions are developed, the use of an external bypass system can be useful. With this, an existing function is cut out of the sequencing and replaced by a calculation in an external CPU. The communication to this external ECU is done at defined points before and after the existing function. So the new function gets the same timing environment than the existing one. The communication via separate data buffers to the external CPU ensures stability by default for the external calculated functionality. When using additional variables in the external calculation a coherency need might be not fulfilled.

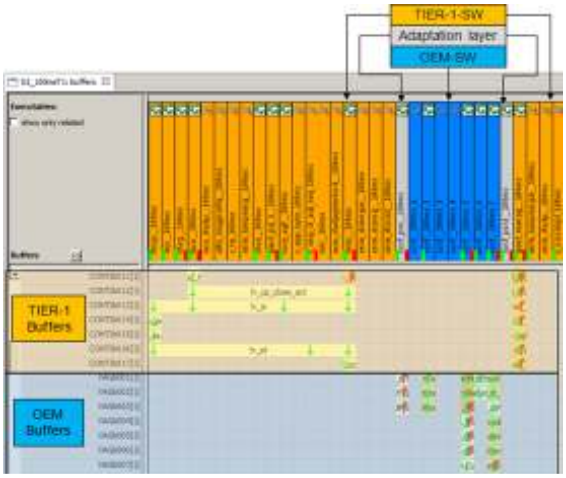

Fig. 12 : Partitionning of runnables and Buffers in shared context

In both cases of external and internal bypassing, if race conditions are modified, the buffering configuration is not anymore valid, due to the modified cartography. It might be that a change in a bypassed function generates a change of buffering in a non modified & stable function. For instance, changing a write access to a read access, or changing the multiplicity of a read access might have impacts on the buffering status elsewhere. Therefore, the type of modifications that can be applied on an algorithm w/o impact on the race condition is very limited, in multi-core context.
In addition, if the required change affects tight integration requirements (e.g. working on a 100µs task in an electrical engine controller), or connection to HW features (e.g. special ASICs), that cannot be fulfilled by rapid prototyping, external and internal bypassing are also no options.

In this case, the solution developed by Continental is the only alternative to the re-delivery of parts between OEM

and TIER-1 (with the consequences it has). As mentioned earlier, a high flexibility is provided to the OEM as long as the interface (= integration of the adaptation runnables) is unchanged. It is even possible for the OEM to change its own integration, and in particular investigate different core distributions of his SW.

All the ways to evolve the functionality of the SW system will exist in future. Depending on the need for flexibility and the type of modification, the internal bypass will be a perfect solution for a rapid development. But due to its limitations other solution are needed. On the other hand, the parallel build gives full data protection and guarantees exact real time behaviour, but with the drawback of comparably long turnaround times due to build and flash times.

6. Related works / State of the art

For the discussed topics of shared development of embedded automotive software, state of the art are three quasi industry standards, depending on the use case and target platform in the given project context. The public promoted research project AMALTHEA [8] is of higher interest for software engineering of future multi- and many core software-systems. Parts of the pre-released results with a well-defined data-model from this project are already in use for software architecture specification and description. Currently in usage for exchange more present in the automotive embedded context are the working groups of ASAM MDX [5] and AUTOSAR [4][9]. Unified exchange and interoperability for software description are supported; all data models have equivalent core data.
All these three standards enable software architecture engineering and exchange of relevant information. Depending on the use case, they are more or less suitable. In Table 2, the coverage of the different description context for all discussed use cases is compared for this three standards, where "x" means full support, "(x)" means partly supported and "-" means currently not supported. In this comparison there where considered the latest versions with the highest coverage of information.

About MDX

The MDX description as an ASAM standard is used for single-core projects and with additions since version 1.3 [10]. Also, software description for multi-core projects are supported with the mayor basic multi-core features, like basic timing requirements, data consistency needs and scheduling requirements from the OEM point of view. Complete system description with hardware- and operating system features are not supported. For the use case to describe and exchange the integration needs of the OEM application software it is suitable.

About AUTOSAR

Advanced description and new concepts like Application partitioning and more static architecture support is given with AUTOSAR from version 4.2, which includes the integration- and protection needs and further timing- and architecture description. It is the most established format in the automotive industry, has the most support from architecture and analysis tools and on this reason highly recommended. Nevertheless, the support of consistency needs ("groups") on RTE side is still an open point.

About AMALTHEA

Finally the AMALTHEA format from version 1.1.1 [11][12], as the newest possibility in these cases, adds features and more support for the dynamic description of the software. It extends the architecture and timing requirements and gives possibilities to describe more technical design properties, for example of the target platform with its hardware or the operating system. For use cases like software simulation and partitioning of the software in multi-core context, this is recommendable [12]. This format is suitable for the exchange of complete system description typically generated from the TIER-1 side, which is responsible for the integration.

	ASAM-MDX (v1.3)	AUTOSAR (v4.2)	AMALTHEA (v1.1.1)
Software			
Runnable level			
Data access (i.e. interfaces)	x	x	x
Access occurrences	x	-	x
Runtime	x	x	-
Process level			
Activation (periodic, sporadic, single)	x	x	-
Call sequence of runnables	x	-	x
Hierarchical call sequences	x	-	x
Logic grouping of runnables	x	-	-
Signals			
Data description	(x)	x	x
Requirements			
Execution Order Constraint	x	x	x
Execution Order Constraint (hierarchical)	-	x	x
Data Stability Needs	x	x	x
Data Coherency Needs	x	x	x
Data Age Constraints (time based)	x	x	x
Data Age Constraints (cycle based)	x	x	x
Hardware			
Cores (frequency, instruction per cycle, topology)	x	x	-
Core features (lock-step, peripherals)	x	(x)	-
Memory topology (bus, crossbar, caches, access times)	(x)	(x)	-
Operating System			
Scheduling (algorithm, core resources)	x	(x)	-
Process configuration	x	x	-

Table 2: Comparison of Standards currently in use in Automotive domain

About the automotive domain

The engine systems domain is the first one in automotive requiring an introduction of multi-core processors due to a lack of computing power (with the exception of multimedia). This is the domain where the deployment of multi-core is most advanced, and which has the tightest constraints as mentioned in chapter 2.

About other industrial domains

To our knowledge, there is no other industrial domain where the development of embedded multi-core SW has similar constraints. In aeronautics, space and defense, the time to market, and target system price are not on the same magnitude, like security and safety requirements. As example, the following article shows the growth of automotive embedded SW with the aeronautic case [13]. In particular, in the aeronautic domain, the main focus is on the scheduling topic: due to safety issues, offline scheduling is widely used, which requires a safe estimation of the Worst Case Execution Times (highly impacted by new multi-core architectures). As example, in [14], the authors consider the access to shared resources only in the point of view of timing impact. In Automotive, domain, offline scheduling is not so often used, as the mostly used OS is AUTOSAR OS or OSEK OS. This is even more true in the engine systems domain, where half of the SW is executed at angular (i.e. time variable) rates. Concerning data protection, some studies are conducted, which concern the detection of race conditions, but in our case, we aim not only to identify them, but also to protect them automatically. Furthermore, the shared resource topic is addressed under the view point of impact on timing and WCET. Also, the integration topic is also very specific to engine systems context, as mentioned in the chapter 2 about coupling.

About ARAMIS and ARTEMIS ECSEL EMC^2

ARAMIS:
In [15][16], the authors address the integration topic, but on a vehicle level, and on a basis of distribution of functions across ECUs. Multi-core is seen as an opportunity to reduce the number of ECUs in the car but requires a good distribution of the functions on the cores [17]. In [18], the particularity of engine systems is recognized as it is qualified as a central ECU.
Several papers [19] address the topic of scheduling, of the verification of timing properties. But in general, individual components are considered, designed independently of any framework (reference architecture).
Other papers [20][21][22] address the topic of detecting race conditions, but once again, our purpose is not limited to detection.
ARTEMIS ECSEL EMC^2:
The project EMC^2 is dedicated to multi-core mixed criticality systems, dynamically reconfigurable. The objectives of this project have no relationship to our purpose, as the mix of OEM vs. TIER-1 Sw is not organized on a criticality basis (e.g. TOER-1 Sw low critical and OEM-SW high critical), but rather on a functional basis. Also, the critical/safety aspects are not part of this paper. Concerning the dynamic configuration, and/or reallocation of functions is not seen as a short term option, in regard to the tight coupling and real time requirements in the engine control area. One interesting paper [23] concerns the migration of legacy SW to multi-core platforms, but the approach is different than the one chosen on our side, as a redesign of the functions is requested, according to a pre-established core allocation (which is part of the design itself). In our case, our clear goal is an independence of the design from the integration, which can change from project to project.

7. Conclusion

The formal requirement engineering on dynamic aspects has a relevant impact on the development process and artifacts to be handled. New templates, guidelines and trainings have been set up to cope with these challenges. New design rules are necessary, which will facilitate the parallelization of the control algorithms, but at the same time have to minimize the re-design effort: on OEM side, like on TIER-1 side, the design of the functions have to be prepared for multi-core, but have to be independent of any core and memory distribution, a choice which is highly project specific. It is also not possible to fix core distribution for a function for the next 10 years. To reach this goal of flexibility, it is therefore essential that the function development focuses on the original requirement (protection and integration needs), rather than on any implementation (e.g. using of double buffering).

The introduced model based approach needs long term establishment, but in prototype projects the first experiences confirm significant easement, better system understanding for each party in collaborative process and in conclusion a key enabler to reach the multi-core challenges for SW development. Step by step the process and tools are adapted.

Finally new technologies will arise, which will influence the design of the functions. For instance, dynamic scheduling / allocation to cores, different partitions in the ECU… Also, the increase of computation power linked to multi-core will certainly motivate higher integration of systems, going towards mixed domains ECUs. We can think of course about integration of Transmission Control Unit and Engine Control Unit, PowerTrain Controllers. But it is to be expected that functions out of the PowerTrain domain start to be integrated, leading to an even higher variability, and therefore needs for partial reprogramming, for instance.
At the end, it is doubtless, that the multi-core introduction is at the origin of a big evolution of architectures, and the presented shared development process will be a key enabler.

Works Cited

[1] D. Claraz, F. Grimal, T. Leydier, R. Mader and G. Wirrer, “Introducing Multi-Core at Automotive Engine Systems,” in *ERTS2-2014*, Toulouse, Feb. 2014.

[2] ASAM, *ASAM AE MDX Meta Data Exchange Format standard V1.2.0*, http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%5BshowUid%5D=2559&start=, Dec. 2012.

[3] D. Claraz, S. Kuntz, U. Margull, M. Niemetz and G. Wirrer, “Deterministic Execution Sequence in Component Based Multi-Contributor Powertrain Control Systems,” in *ERTS2-2012*, Toulouse, 2012.

[4] AUTOSAR, *AUTOSAR 4.1.1 Specification of RTE - AUTOSAR_SWS_RTE 3.3.0*, http://www.autosar.org/specifications/release-41/, 2013.

[5] ASAM, *ASAM AE MDX Meta Data Exchange Format standard V1.3.0*, http://www.asam.net/nc/home/standards/standard-detail.html?tx_rbwbmasamstandards_pi1%5BshowUid%5D=3244&start=, Jun. 2015.

[6] R. Bosch, "Method for preventing data inconsistency between accesses of different functions of an application to a global variable in a data processing installation". Germany Patent EP 1 738 257 B1, 30 03 2005.

[7] T. Flämig, H. Jelden, C. Kornmesser, A. Schulze, L. Michel, C. Ebert and B. Cool, *Software architecture methods for multi-core – Distributed development and validation of architecture in collaborative engineered multi-core systems*, EMCC Embedded Multicore Conference Munich 2105, Jun. 2015.

[8] A. Project, *AMALTHEA Project Deliverable: D 4.4 - report on model and tool exchange.*, https://itea3.org/project/workpackage/document/download/1675/09013-AMALTHEA-WP-4-D44-Reportonmodelandtoolexchange.pdf, Apr. 2014.

[9] A. Sailer, *Timing Simulation of Multi-Core Systems based on AUTOSAR Models*, http://www.timing-architects.com/fileadmin/user_upload/Log-In_Miscellaneous/Whitepaper_Timing_Simulation_AUTOSAR.pdf, Aug. 2014.

[10] ASAM, *Release Presentation ASAM AE MDX V1.3.0*, http://www.asam.net/index.php?eID=tx_nawsecuredl&u=0&file=fileadmin/documents/standards/AE/MDX/V1.3.0/ASAM-AE-MDX-V1_3_0_Release_Presentation.pdf&t=1446754174&hash=468d38849977145083, Jun. 2015.

[11] AMALTHEA, *AMALTHEA Project Deliverable: D 4.4 - report on model and tool exchange.*, https://itea3.org/project/workpackage/document/download/1675/09013-AMALTHEA-WP-4-D44-Reportonmodelandtoolexchange.pdf, Apr. 2014.

[12] H. Mackamul, *Innovation report: Building an open source, extendible development platform*, https://itea3.org/project/result/download/6729/AMALTHEA%20Innovation%20Report.pdf, Sept. 2014.

[13] R. Charette, "This Car Runs on Code," *IEEE Spectrum*, no. http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code, Feb. 2009.

[14] C. Cullmann, C. Ferdinand, G. Gebhard, D. Grund, C. Maiza, J. Reineke, B. Triquet and R. Wilhelm, "Predictability Considerations in the Design of Multi-Core Embedded Systems," in *ERTS-2010*, Toulouse, Feb. 2010.

[15] W. Schwitzer, R. Schneider, D. Reinhardt and G. Hofstetter, "Tackling the Complexity of Timing-relevant Deployment Decisions in Multicore-based Embedded Automotive Software Systems," in *SAE 2013 World Congress & Exhibition*, Apr. 2013.

[16] D. Juergens, D. Reinhardt, R. Schneider, G. Hofstetter, U. Dannebaum and A. Graf, "Implementing Mixed Criticality Software Integration On Multicore - A Cost Model And The Lessons Learned," in *SAE 2015 World Congress and Exhibition*, Apr. 2015.

[17] R. Schneider, A. Kohn and D. Juergens, "Software Parallelization in Automotive Multi-Core Systems," in *SAE 2015 World Congress and Exhibition*, Apr. 2015.

[18] D. Reinhardt and M. Kucera, "Domain Controlled Architecture – A new approach for Large Scale Software Integrated Automotive Systems," in *International Conference on Pervasive and Embedded Computing and Communication Systems 2013*, Feb. 2013.

[19] I. Stierand, P. Rainkemeier and P. Bhaduri, "Virtual Integration of Real-Time Systems Based on Resource Segregation Abstraction," in *International Conference on Formal Modeling and Analysis of Timed Systems, 2014*, Sept. 2014.

[20] D. Nowotka and J. Traub, "Formal Verification of Concurrent Embedded Software," in *International Embedded Systems Symposium, IESS 2013*, Jun. 2013.

[21] T. Ehlers, D. Nowotka and P. Sieweck, "Finding race conditions in real-time code by using formal software verification," in *12th International Conference on Formal Modeling and Analysis of Timed Systems, 2014*, Sept. 2014.

[22] N. Koutsopoulos, M. Northovery and T. Felden, "Advancing Data Race investigation and Classification through Visualization," in *3rd IEEE Working Conference on Software Visualization (VISSOFT 2015)*, Sept. 2015.

[23] G. Macher, A. Höller, E. Armengaud and C. Kreiner, "Automotive Embedded Software: Migration Challenges to Multi-Core Computing Platforms," in *IEEE INDIN 2015 - International Conference on Industrial Informatics*, Jul. 2015.