Systematic and Methodical Analysis, Validation and Parallelization of Embedded Automotive Software for Multiple-IEU Platforms

Dissertation

for the degree of
Doctor of Natural Sciences (Dr. rer. nat.)

Julian Kienberger

University of Augsburg
Department of Computer Science
Software Methodologies for Distributed Systems

1st Examiner:	**Prof. Dr. Bernhard Bauer** Department of Computer Science, University of Augsburg, Germany
2nd Examiner:	**Prof. Dr. Theo Ungerer** Department of Computer Science, University of Augsburg, Germany
Advisor:	**Stefan Kuntz** Continental Automotive GmbH, Regensburg, Germany
Defense:	**Thursday, 2nd May, 2019**

Abstract

In the past decades, the automotive industry has been seeking to include more and more features in its vehicles while simultaneously attempting to reduce the number of “Electronic Control Units” (ECUs) that execute the corresponding embedded software. As technical and economic limitations prevent furthermore rising the single-core computing power, the migration of ECU software to target platforms featuring multiple “Independent Execution Units” (IEUs), e.g., processor cores, is enforced and the necessary paradigm shift towards multi-core technology is in full swing.

In order to eventually exploit the extra processing power, there is much additional effort needed for coping with the tremendously increased complexity of such systems. This is largely due to the elaborate parallelization process consisting of partitioning, mapping and scheduling software parts as tasks on different IEUs. The associated, rapidly growing number of possible solutions results in a combinatorial explosion and thus spans a vast search space. Mastering this challenge requires both the prevention of impending data inconsistencies caused by race conditions and the avoidance of additionally required synchronization effort originating from the distributed execution of an application across different IEUs.

Therefore, there is a strong need for innovative concepts, methods and a continuous tool chain to, on the one hand, support the migration of legacy software by efficiently parallelizing it (i.e. first partition and then distribute the obtained parts to different IEUs) and, on the other hand, ease the creation of embedded multiple-IEU applications from scratch. In addition, it is crucial to support the development process of software being compliant with the leading standard “AUTomotive Open System ARchitecture” (AUTOSAR).

This PhD thesis starts with a description of the circumstances and factors leading to this situation before stating correlating challenges and deriving concrete objectives to eventually cope with them. In addition, a demonstrative running example is introduced that bases on a car’s wheel speed measurement function and is intended to illustrate ideas and methods throughout the thesis. Afterwards, basic knowledge, applied concepts and techniques are covered in order to provide a sound basis for the later presented methodology. Prior to the main chapter, relevant existing research approaches are introduced, categorized and evaluated with regard to their significance for the suggested methodology and conducted case studies.

The pivotal notion of the presented approach is to utilize a tool-aided data dependency analysis performed on AUTOSAR system descriptions (referred to as “models”) to eventually determine advantageous partitions as well as initial task-to-IEUs mappings. Therefor, the analysis’ results are used for a verification and validation process that modifies the model to achieve “multiple-IEU robustness”. This robustness is universally valid – irrespective of a specific partitioning and mapping solution.

Based on this and following the overarching methodology’s concepts, concrete partitioning strategies and algorithms are applied in order to efficiently identify advantageous disjoint subsets even within highly complex models. The succeeding mapping step then distributes the obtained model parts to available IEUs in an expedient manner.

Afterwards, the determined solutions serve as input for the simulation within a timing tool for embedded multi-core systems. Here, this initial solution is evaluated with respect to scheduling quality (e.g. fulfillment of task deadlines) and metrics like cross-IEU communication rate, communication latencies, memory consumption or load distribution. Lastly, a subsequent optimization process enhances the initial solution, iteratively generates improved variants and enables a comparative assessment.

In order to demonstrate the realization as well as to prove practicability and benefit, three case studies are presented that illustrate the feasibility and usage of the methodology by means of real-world examples coming directly from the industry. The added value is shown with the respective evaluations, e.g., via reduced effort and increased solution quality achieved with sticking to the introduced concepts and methods.

In conclusion, the proceeding is summarized and originally formulated objectives are confronted with the reached achievements. Finally, an outlook on currently prevalent trends within the automotive industry is given just before the concluding depiction of impending tasks as well as promising concepts for further development.

Zusammenfassung

In den vergangenen Jahrzehnten hat die Automobilindustrie enormen Aufwand betrieben um Fahrzeuge mit immer umfassenderer Funktionalität auszustatten. Parallel dazu wurde versucht, die Anzahl der verbauten Steuergeräte (“Electronic Control Units” – ECUs), auf denen die dazugehörige eingebettete Software ausgeführt wird, zu verringern. Da sowohl technische als auch wirtschaftliche Beschränkungen eine weitere Steigerung der Single-Core-Rechenleistung verhindern, wird die Migration der ECU-Software auf Zielplattformen mit mehreren unabhängigen Ausführungseinheiten (“Independent Execution Units” – IEUs) forciert. Der dafür notwendige Paradigmenwechsel hin zu Multi-Core-Techniken befindet sich bereits in vollem Gange.

Damit die hinzukommende Rechenleistung letztendlich genutzt werden kann, ist ein erheblicher Mehraufwand nötig um der enorm gestiegenen Komplexität solcher Systeme beizukommen. Dieser Mehraufwand ergibt sich in erster Linie durch den anspruchsvollen Parallelisierungsprozess, der aus der Partitionierung, dem Mapping und dem Scheduling von Softwareteilen als Tasks auf unterschiedlichen IEUs besteht. Die damit einhergehende, sprunghaft wachsende Anzahl an Lösungsmöglichkeiten führt zu einer kombinatorischen Explosion und umfasst daher einen gewaltigen Suchraum. Die Bewältigung dieser Herausforderung erfordert zum einen die Verhinderung von drohenden Daten-Inkonsistenzen, die von Race Conditions verursacht werden, sowie zum anderen die Vermeidung des zusätzlich erforderlichen Synchronisationsaufwands, der bei der über verschiedene IEUs verteilten Applikationsausführung entsteht.

Folglich besteht ein großer Bedarf an innovativen Konzepten, Methoden sowie einer durchgängigen Werkzeugkette, um einerseits die Migration von Bestandssoftware mittels effizienter Parallelisierung zu unterstützen (d. h. partitionieren und dann erhaltene Softwareteile an unterschiedliche IEUs zuweisen) und um andererseits die Neuerstellung eingebetteter Multi-IEU-Applikationen zu erleichtern. Zudem ist es von zentraler Bedeutung, den Entwicklungsprozess von Software, die zum maßgeblichen “AUTomotive Open System ARchitecture”-Standard (AUTOSAR) konform ist, zu unterstützen.

Diese Dissertation beginnt mit der Beschreibung von Umständen und Faktoren, die zu dieser Situation geführt haben. Im Anschluss werden damit

einhergehende Herausforderungen erläutert sowie Ziele zu deren Bewältigung abgeleitet. Des Weiteren wird ein durchgängiges Beispiel eingeführt, das auf der Messung der Raddrehgeschwindigkeit eines Autos basiert und mittels dessen die Ideen und Methoden der Dissertation veranschaulicht werden. Nachfolgend werden Grundlagen sowie angewandte Konzepte und Techniken behandelt um eine solide Basis für die später dargelegte Methodik zu schaffen. Außerdem werden im Vorfeld des Hauptkapitels dafür relevante, bereits bestehende Forschungsansätze beschrieben, kategorisiert und hinsichtlich ihres Stellenwerts für die vorgeschlagene Methodik und die durchgeführten Fallstudien bewertet.

Der zentrale Gedanke des vorgestellten Ansatzes ist die Nutzung einer werkzeuggestützten Datenabhängigkeitsanalyse auf AUTOSAR-Systembeschreibungen (hier als "Modelle" bezeichnet) um günstige Partitionen und (initiale) Task-zu-IEUs-Mappings zu ermitteln. Dafür werden die Resultate der Analyse für einen Verifikations- und Validierungsprozess verwendet, der das Modell modifiziert um "Multi-IEU-Robustheit" zu erreichen. Diese Robustheit ist universell gültig – unabhängig von einer konkreten Partitionierungs- und Mapping-Lösung.

Darauf aufbauend und den Konzepten der übergeordneten Methodik folgend, werden konkrete Partitionierungsstrategien und -algorithmen angewendet um selbst in hochkomplexen Modellen vorteilhafte, disjunkte Teilmengen auf effiziente Weise ausfindig zu machen. Der nachfolgende Mapping-Schritt verteilt die erhaltenen Modellteile dann zielführend auf verfügbare IEUs.

Anschließend dienen die so ermittelten Lösungen als Eingabe für die Simulation in einem Timing-Werkzeug für eingebettete Multi-Core-Systeme. Hierbei wird die initiale Lösung hinsichtlich der Scheduling-Qualität (z. B. Erfüllung von Task-Deadlines) und anhand von Metriken wie der IEU-übergreifenden Kommunikationsrate, Kommunikationslatenzen, Speicherverbrauch oder Lastverteilung bewertet. Schließlich bereitet ein darauf folgender Optimierungsprozess die initiale Lösung auf, erstellt iterativ verbesserte Varianten und ermöglicht eine vergleichende Bewertung.

Um die Umsetzung zu demonstrieren sowie die Praxistauglichkeit und den Nutzen nachzuweisen, werden drei Fallstudien vorgestellt, die die Durchführbarkeit und die Verwendung der Methodik anhand realer Beispiele aus der Industrie illustrieren. Mit den jeweiligen Evaluationen wird der Mehrwert gezeigt, der mit der Einhaltung der beschriebenen Konzepte und Methoden erzielt wird und der sich z. B. im verringerten Aufwand und der

gesteigerten Lösungsqualität widerspiegelt.

Zum Schluss wird das Vorgehen zusammengefasst und die anfangs formulierten Ziele werden den erreichten Ergebnissen gegenübergestellt. Der abschließende Ausblick beschreibt die aktuell in der Autoindustrie tonangebenden Trends und skizziert sowohl anstehende Arbeitsfelder als auch vielversprechende Konzepte für die Weiterentwicklung.

Acknowledgements

First and foremost, I would like to express my deep gratitude to Prof. Dr. Bernhard Bauer for giving me the opportunity to join his team and to graduate under his supervision at the SMDS professorship. His kind support, versed guidance and especially his affable manner are forming outstandingly pleasant working conditions and thus provide an ideal basis.

I am also heavily indebted to Stefan Kuntz and Continental Automotive AG. I want to sincerely thank Stefan for taking time for fruitful discussions and providing me with valuable insights in industrial practice as well as his long-term experiences. Continental Automotive AG has been an important sponsor of my dissertation, thus I am very grateful to the management for fostering me in the course of the company's doctoral candidate program.

Furthermore, I would like to cordially thank the Timing-Architects Embedded Systems GmbH (TA) – represented by their CEO and co-founder Dr. Michael Deubzer – for giving me the opportunity to use their software within the "TA Research Partner Program". This has been a great help for me – especially for conducting the case studies and evaluation. Concerning the latter, I would like to additionally thank Stefan Schmidhuber for his skilled feedback and support in his role as fellow PhD student as well as TA employee.

Last but not least, special thanks go to all my colleagues of the SMDS professorship for their companionship and for providing a uniquely cheerful working atmosphere that I enjoyed very much.

Contents

1 Introduction

This chapter initially defines the context and states relevant aspects that form the motivation and working basis for this thesis (Section 1.1). Afterwards, central challenges arising from this situation are depicted (Section 1.2). Based on this, the concrete objectives of this thesis are derived (Section 1.3). Finally, already published publications that include parts of this thesis are presented (Section 1.4).

1.1 Motivation and Context

The demand for increasing performance is an inevitable trend that affects the whole spectrum of computing and applies in particular for embedded systems.

The automotive domain cannot evade this course as it incessantly attempts to enhance driving dynamics, extend infotainment features, raise traveling comfort and improve the security and safety characteristics of their vehicles. Therefore, challenges like parallelization and multi-core platforms have to be addressed in order to enable, e.g., highly automated driving and car-to-x communication that pose new challenges for automotive software systems concerning aspects like dependability or cloud interaction [Für16].

1.1.1 Complexity Rise

However, adding further functionality like "Adaptive Cruise Control" or a "Lane Assist" generally increases complexity as well as required processing performance [DHM^{+}10,SZ10]. In addition, the functions themselves are enlarging while their inner structure is becoming progressively intricate. One of the reasons is that car domains do evermore correlate, which is – among others – due to growing requirements in terms of environmentalism and safety, which leads to, e.g., the entanglement of "the active collision mitigation, adaptive cruise control, and electronic stability control functions" that now "share I/O devices such as the brake, throttle, and many others" [YPS13].

Subsequently, collateral effort is incurred due to required surveillance, planning and resource management measures [YPS13]. A further cause is the ongoing complexity transfer "from the electrical/electronic architecture to the hardware and software architecture of the ECUs" [MNBSL12] which eventually leads to a "software-defined car".

This results in a need for additional ECUs, although there is a prevalent endeavor to save space and reduce weight by decreasing their number [Ebe16, MNBSL12]. The latter can be achieved by replacing ECUs with distinctly less (but more powerful) "domain controllers" [Mac15, Gra15].

Facing the expectation that ten times as much processing power as currently available will be needed in only ten years (cf. [Mad15]), the existing circumstances drive the automotive companies' pursuit of finding a possibility for boosting the available computing performance in order to stay competitive.

1.1.2 The Beginning Multi-Core Era

The rising demand has exceeded the capabilities of single-core technology whose processing power is almost completely exhausted and does not significantly increase anymore [GS12, KMKB14, Sut05]. This can be ascribed to the problem that further raising the clock speed (the "frequency") as "standard way to increase the performance of a processor" [SDM$^+$14] is now unreasonable from an economical and technical point of view: It inevitably leads to a disproportionate growth of processor power consumption (the so-called "Power Wall"), to an enormous rise of corresponding heat dissipation efforts (requiring expensive active cooling in some cases) and therefore to significantly higher costs when trying to further reduce the transistors' size [Sut05, Ebe16, MNBSL12]. Moreover, the addressed active cooling involves additional effort to realize and its possible failure poses another risk that would have to be taken into consideration especially for safety-critical applications like an "Engine Management System" (EMS) [SDM$^+$14].

According to the current state of research, embedded architectures featuring multiple cores (or as already broached and more generally speaking – IEUs[1]) are the most promising and maybe only solution having the potential to meet existing and upcoming requirements in terms of providing processing performance as well as minimizing size, weight and power consumption [PKQ$^+$14, MNBSL12, SDM$^+$14]. Or to put it more generally: "In the era of stalling CPU clock speeds, exploiting parallelism is probably the most important way to accelerate computer programs from a hardware perspective" [BMS$^+$16].

Facing escalating complexity, it is thus hardly surprising that both multi-core architectures and parallelization – which compulsory comes along with it – are becoming increasingly important [Wir11, MPS07].

In the area of desktop computing, the transition towards them started about ten years ago (and was quite successful), whereas the automotive sector was

[1]Though "multi-core" is the prevailing term when referring to such architectures, it is just one specific solution that is frequently used to vicariously represent the whole idea of parallel computing [BSER11]. As more correct and abstract term, "IEU" encompasses processing units that are independent of each other within the system's scope (like a core, a processor or an ECU).

rather recently forced to start migrating its ECU software in order to pave the way for further technical advancement, because – based on experience – automotive microcontrollers follow "common" information technology systems with a delay of roughly five to eight years due to the typical development cycle times of cars [Gra15]. Currently, multi-core processors are already employed in the automotive domain (cf. [PKQ$^+$14]), but this is at the moment rather the exception than the rule.

1.1.3 Multi-Core Potential and Use Cases

Potentially gained computing power coming along with this progress can be utilized in different ways. There are in particular some safety-related applications:

- "Separation" and "Segregation" are concepts that are lacking commonly accepted definitions clearly distinguishing them. Both are designated for distinctly distributing certain functionalities of an application to different IEUs in order to prevent them from interfering with each other. Generally speaking, the higher the level of parallelism is the more segregation is enabled [MNBSL12].
- "Diverse Redundancy" is employed to increase a system's reliability by performing additional calculations to raise a result's correctness probability via using different calculation methods, e.g., for determining the exact acceleration value from a gas pedal [Stefan Kuntz, personal communication, 2013].
- "Isolation" enables slivering an especially safety-critical software part[2] to run on its own IEU. Another practice is that a "core can be dedicated to a specialized usage" [MNBSL12].

These utilizations are especially relevant when ensuring to meet the requirements of "ISO 26262" – an international "standard for functional safety of electrical/electronic systems in road vehicles" [Sin11].

However, taken as a whole, safety does not automatically benefit from multi-core computing. It is rather appropriate to state, that there is basically an equilibrium of risks and chances when parallelized and distributed processing is introduced. As there is per se no guaranteed execution time

[2]A "program/software part" can be a whole process, a thread, a method or even a task that consist of only one instruction.

with, e.g., multiple threads, the inherent indeterminate "Worst Case Execution Times" (WCETs) may render multi-core approaches unsuitable for, e.g., an "Anti-lock Braking System" (ABS). In this context, the crucial point is to fully employ the multi-core potential as countermeasure clearly outweighing possible drawbacks, e.g., by focusing on proper verification and data validation (V&V) methods as well as tools that support the development of parallel software.

Besides safety aspects, the expected core benefit of multiple-IEU platforms is to save space and reduce weight by performing the same work on less ECUs than before. Beyond that, there is also a need to raise a vehicle's overall computing power.

According to the well-known laws of Amdahl and Gustafson (cf. [Gus88]), the theoretical speedup of a now parallelized and distributed application can be tremendous depending on the number of employed IEUs (cf. [Ebe16]):

- "Amdahl's Law", formulated in 1967, delineates the theoretical speedup for a task with fixed problem size (a constant workload) chiefly pertaining to applications with a considerable amount of non-parallelizable code. It describes the (limited) possible reduction of execution time when migrating to multiple IEUs.
- "Gustafson's Law", originating from 1988, quantifies the theoretical speedup for a task within a fixed time frame, being especially applicable for applications with a considerable amount of parallelizable code. It determines the (unlimited) augmentation of the problem size with the aid of multiple execution units.

A fundamental difference between them is that for Gustafson's Law the parallel portion grows together with the problem size. Thus, the non-parallelizable amount becomes smaller when the number of execution units is increased. In a nutshell, it boils down to different focuses: shortening the latency (Amdahl) versus increasing the throughput (Gustafson).

However, in the context of (automotive) real-time applications, simply shortening the execution time does not increase software quality and "might impact correct system behavior" [Här16]. The mere rise of computing power does not solve problems on its own as their exponentially growing complexity is not manageable by brute force methods [Rad17]. Therefore, Amdahl's Law is not very helpful for real-time purposes [Här17]. Consequently, speeding up an application's processing time is – in the majority of cases – no more than a minor goal.

As opposed to this, additional performance is rather used to cope with high-performance applications, e.g., “Advanced Driver Assistance Systems” (ADAS) including sophisticated real-time image processing or a modern EMS for electric or hybrid powertrains [MNBSL12].

In the context of these examples, both fundamental purposes of “multi-core ECUs” are applicable: uniting previously distributed applications to run on a single ECU (“application level parallelism”) or the parallelization of the hitherto sequential processing of one (rather complex) application (“function level parallelism”) [PKQ$^+$14].

In the former case, it seems natural to put already strongly connected software parts from different applications on one common IEU. This concept – also referred to as “pooling” – benefits from the fact that the communication between cores of one processor (mostly realized with shared memory) is considerably faster than the communication between ECUs via fieldbuses [NBN09, NNB10, SKS10].

The latter case (parallelization of one application) is significantly different, because it focuses on the inner structure of an application. Here, the once strictly consecutive execution order of its parts can be altered causing additional effort to ensure that its original behavior is preserved [Cor13]. A broad rule is, that as long as software parts are not explicitly sequential, they can by implication be parallelized.

1.1.4 Migration Issues

Current automotive software (operating systems as well as applications) was usually neither designed for being executed in parallel on functional level (as mainly addressed in this thesis) nor on application level. By contrast, it is most often based on a sequential execution model [NBN09].

Faced with multiple IEUs, different clock rates as well as significant changes in the memory and communication architecture, the timing behavior of applications turned out to be completely different on multiple-IEU platforms [GHKF11].

As “constraint satisfaction is a crucial aspect [...] in the design of embedded systems” [ABG$^+$13], preserving a software’s original behavior can be enforced by imposing timing constraints that have to be met. Unfortunately, adding constraints also exacerbates the important allocation step (distributing software parts to IEUs) and thus further increases complexity [ABG$^+$13]. A proper migration is therefore a very challenging task [Mad15, Für15, GLI15].

Overcoming these difficulties involves a "disruptive paradigm shift due to the introduction of true concurrency" [SBR10] as aspects like cross-core communication, synchronization overheads, shared resources, significance of memory location and the complex scheduling of software parts come into play when processing is distributed and sequential data consistency[3] has to be guaranteed [Mac15, Sch15a, Sch15b]. Therefore, the central "multi-core challenge" can be outlined as finding an expedient distribution of the software parts, i.e., one that meets its timing requirements and comes with minimal additional synchronization and communication overhead [Cor13].

In order to achieve the latter without producing unnecessary interference (i.e. overhead) among IEUs, it is crucial to appropriately determine the software's fragments in the first place ("partitioning") and to skillfully distribute them on the IEUs afterwards ("mapping"). These activities constitute enormous additional effort and pose the central downside of the migration process despite the fact that the succeeding creation of an appropriate execution plan ("scheduling") is certainly not a simple task either.

That is because it is in general intricate to coordinate multiple IEUs at executing parts of a common application: Due to dependencies between separately processed but still intertwined software fragments, problems like race conditions, dead locks, non-determinism and insufficient load balancing (seeking equal workloads for each IEU) can emerge [PD13, Pat10, May16, Ebe16, Cor13].

The occurring "multi-core overhead" can thus be ascribed to the additional effort "to synchronize the control flow, to exchange data and to protect shared data" [GHKF11] as well as to compensate for mistakes within the implementation of the corresponding mechanisms, such as (cf. [Här16]):

[3]We define "data consistency" as both "functional coherence" (processed data has uniform age) and "stability" (data is steady during processing), cf. [MFCM16].

- ordering failures, e.g., not sticking to a specific execution order that is necessary for correct processing (violation of precedence relation or mutual exclusion resulting in race conditions),
- synchronization failures, e.g., through faulty usage of synchronization caused by nested locking (which again leads to livelocks or deadlocks) and
- interleaving failures, e.g., induced by unwanted side effects like mode or shared data inconsistencies (e.g. due to cache inconsistencies [Ebe16]).

An additional form of synchronization overhead is the so-called "Inter-Core Communication" (ICC). It deals with coordinating accesses between IEUs that are mainly attributable to priority-based core interruption and do result in often unnecessary long wait times [Fuk18]. In many cases, ICC appears unexpectedly, e.g., as communication among now distributed components, and thus poses the "key to exploit multi-core performance" [Neu16].

The aspects described in this subsection cause a complexity rise that is directly correlated with the count of software parts to be handled, because both numbers of possibilities grow exponentially: the one for determining the software fragments to be distributed in the first place (partitioning) as well as the ways for their succeeding allocation on IEUs (mapping).

These circumstances result in a vast search space and, by association, also in a hardly comprehensible design (solution) space, which renders an exhaustive "design space exploration" infeasible. Consequently – and due to shortage of time – solutions are often found with the help of heuristics. As there is no guarantee that such a proceeding leads to a proper solution, adverse approaches can even issue into a deterioration of latency or throughput ("negative speedup", cf. [Ebe16]), especially when the design space's density, i.e., the rate of beneficial solutions, is rather low.

1.1.5 Conditions in the Automotive Sector

The latter is generally considered to be the case for embedded real-time applications like the combustion EMS employed in the case studies of this work (cf. Sections 5.2 and 5.3).

Basically, such embedded automotive software falls into the class of hard real-time systems [SDM$^+$14]. This means primarily that the correctness of a calculation does not only depend on the absence of errors but also on complying with the specified time span in which the result has to be provided [RTS16].

Thus, missing a deadline within a hard real-time system – like electronic steering or braking systems – equals to total system failure and may be accompanied by the violation of safety goals. As opposed to this, not adhering to deadlines in so-called "firm" or "soft" real-time systems is tolerable as it may just degrade their quality or performance, but will probably not go against safety requirements, e.g., for vehicle light systems [RTS16].

Automotive applications are subject to specific limitations like strongly reduced supply of energy, low clock rates (i.e. little computing power) as well as sparse internal memory and disk space. Therefore, the corresponding functional and non-functional requirements, such as dependability, safety and energy efficiency, do "go far beyond those of general purpose computing" [DLZV14]. As they must nevertheless be met, the according applications are – in contrast to typical desktop computing – constructed as highly specialized software [Cor13].

Hence, the often intricate structure of automotive software itself poses a considerable development challenge even when multi-core aspects are not regarded [SVZ15]. The complexity is mainly induced by "the fact that increasingly sophisticated functionalities [...] are causing complex interactions of highly integrated functions, provided by a combination of mechanical, electric/electronic, and especially software parts which implement those functions" [EZV$^+$17]. This is particularly true for larger systems such as an EMS which is "composed of multiple functional components that are tightly coupled via numerous communication dependencies and intensive data sharing" [HDK$^+$17].

In this context, a main driver is the strong trend towards evermore supportive, prescient and autonomously acting systems that inevitably leads to this highly coupled functionality. Taking this fact into account, both currently developed as well as upcoming automotive architectures are increasingly structured in a centralized way across established vehicle domains, which complicates them even further (cf. "domain controllers" in Subsection 1.1.1) [Sch18].

Now that "multi-core ECUs will drastically change the electrical/electronic architectures" [MNBSL12], e.g., by pursuing typical multi-core software targets like scalability (to support product variants) or flexibility (to enable easy extensibility as well as efficient load balancing), the automotive companies are suddenly compelled to rethink in general [May16, MNBSL12].

However, after more than 15 years of solely employing single-core hardware, the software development in this sector has become confined to "single-core specific thinking" [Ebe16]. The crux is that ECU software is hardly ever re-engineered from scratch but rather constantly refined because of lacking money and time induced by competitive pressure. The only exceptions are developing unprecedented functions or advanced research projects in a company (where the tool development never catches up) [Stefan Kuntz, personal communication, 2015].

Therefore, existing ECU legacy software has to be migrated. Unfortunately, automotive software is mostly not very suited for being executed in parallel by multiple IEUs, because its structure "does not necessarily represent a decomposition into parallel or independent components" [KPQ$^+$16].

Quite the opposite is the case when facing a plethora of functions and subfunctions that communicate intensively, like it is the case in a typical combustion EMS (cf. Section 5) including up to 8000 "Runnable Entities" (REs), which are the atomic executable and schedulable units of the "AUTomotive Open System ARchitecture"[4] (AUTOSAR). Here, it is obvious that multi-core approaches massively increase the internal complexity of ECUs and finding a favorable partition within such highly interconnected software is usually costly [Deu15, Für15].

1.1.6 Arising Needs and Consequences

Like already indicated in Subsection 1.1.4, a paradigm shift was initiated in the course of introducing multi-core technology in the area of mobility domains [DLZV14].

There is broad consensus among experts that innovative concepts for coping with emerging issues are required when facing the challenges associated with this change (cf. [SWL$^+$09, Eiß12, Arb11]), for example (following [SMD$^+$10, Für15, Mac15, Mad15, SVZ15]):

[4]AUTOSAR standardizes "an open software architecture for automotive electronic control units (ECUs)" [AUT14a], cf. `http://www.autosar.org`.

- finding suitable leverage points and heuristics for the process of migrating to parallelized versions of existing application software and
- supporting this process with tools that automatize as much work as possible and illustrate detected problems to the engineer in a meaningful way.

The need for automated migration and visualization becomes clear when casting a glance at, for example, Continental's former workflow: "Continental has hitherto tackled this task using design studies linking the function components and the technical components using an external spreadsheet tool" [EZV+17]. Being confronted with a "growing amount of functionality" and "an increasingly complex design space" [SVZ15], solving this challenge with a manual approach seems to be – by all indications – grossly inefficient. Therefore, Continental endeavors "to enable a highly automated tool-assisted complex system design process which captures all system design decisions explicitly" [EZV+17].

Apart from these new requirements emerging in the course of the "multicore era", a well-known aspect remains crucial: The heavy focus on models and their active inclusion (i.e. not only as supplement) in the whole process of software development and deployment is indispensable to keep an overview [Deu15, Flä15]. This becomes clear when taking a closer look at the work process in the automotive sector (cf. Subsection 6.3.4), which is characterized by multiple iterations V-Model development, several releases with strongly differing maturity and quality as well as being to a high degree based on the division of labor (even across different companies) [MFCM16]. The latter is not unusual as supporting model-driven development to enable "cross-enterprise" collaboration has already been an important research topic for several decades [RBM06, BMR04, BRM05].

For example, there are usually several hundred engineers working on an EMS since its multitude of interconnections with other ECUs makes it exceedingly complex [Stefan Kuntz, personal communication, 2016]. Efficient collaborative working on such a system is hardly possible without proper organizational structures and suitable work items like models.

Under these circumstances, there is apparently – and for all working steps – an urgent need for "complex tools [...] which should interpret the information given from each partner in adequate and standardized exchange formats" [MFCM16]. Ideally, such tools also provide and handle models on different levels of abstraction, e.g., separate models for architecture, design, implementation and operation [Kun17]. The main advantage is that a model-based approach "enables a variety of front-loading methods" [EZV$^+$17] allowing, e.g., models of computation, validation of deployments (mapping of software parts to IEUs) or load balancing. Eventually, this approach is worthwhile with regard to "costs savings due to early integration of analysis and synthesis techniques enabling early identification of design errors" [EZV$^+$17].

To sum up, a model-based workflow facilitates solving problems rather early (i.e. beginning on abstract levels, cf. [Ebe16]), systematically ("by design") and more targeted than hitherto in a unspecific manner and preferably via standard solutions. Correspondingly, the authors of [Här16] identified changing trends in the automotive sector's "development mindset":

- Concurrency: distribution on different cores now obviously supersedes preemption
- Memory: task-specific memory is used rather than global variables
- Consistency: consistency needs are specified instead of using explicit lock mechanisms
- Real-time: assured by checking the system requirements as opposed to exhaustive testing

Facing complexity becoming rampant, model-based approaches massively gained significance and have eventually "become a de facto standard in recent years" [EZV$^+$17].

1.1.7 Running Example

Concerns and problem statements are illustrated throughout this thesis by constantly modifying and refining a model example that describes dual usage of speed data delivered by several wheel speed sensors. Its basic structure is shown in Figure 1.1.

The "Wheel Speed System" (WSS) example consists of seven functional blocks, each acting as either sensor, controller or actor. The four sensors in the middle each write a variable ("speed-x") representing the speed of the respective wheel ("outgoing arrows"). The post-positioned bracketed "i" indicates different sensor instances. The variables are read ("incoming arrows") by the controller of the ABS in order to calculate if intervening is necessary to keep the vehicle controllable (i.e. to maintain tractive contact). They are also consumed by the speedometer's controller to determine the value to be displayed, which is passed on to the succeeding actor (the speedometer needle).

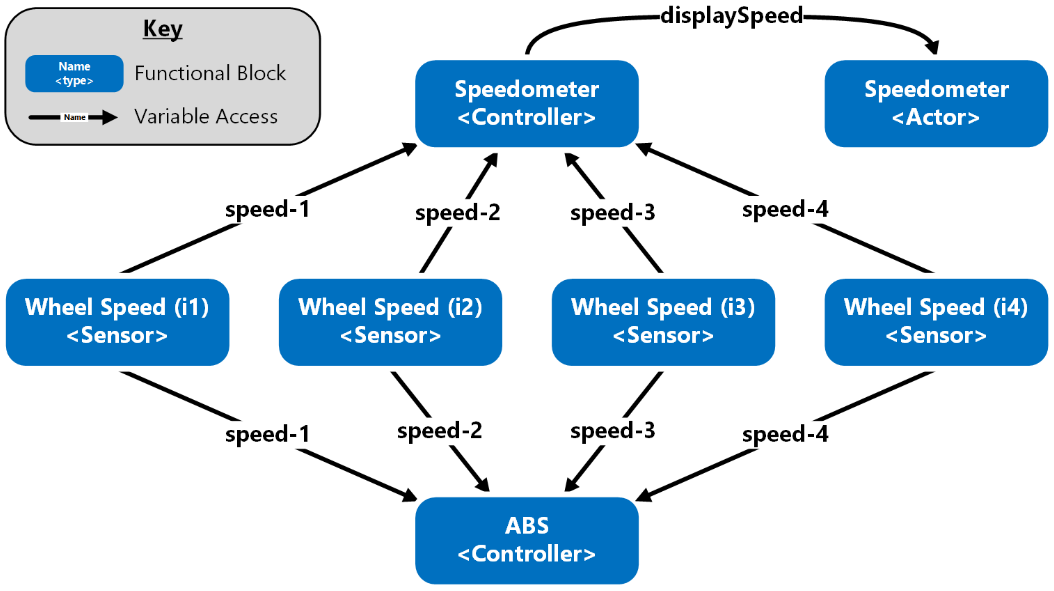


Figure 1.1: Basic structure of the "Wheel Speed System" example

1.2 Problems and Challenges

This section describes typical problems and challenges emerging when software engineers try to include concurrency aspects in embedded real-time systems. Basically, three core challenges are addressed in this thesis:

The first one is to efficiently analyze a given complex model in a methodological way by determining timing-relevant elements, finding existing data and control flow dependencies among them as well as illustrating determined connections in a clear manner (cf. Subsection 1.2.1).

The second one deals with working out a concept for a systematic interpretation of obtained information by evaluating a model's integrity (and thus its suitability for a consistent parallelization process), identifying all possible problems with regard to parallel execution as well as effectively supporting their resolution (cf. Subsection 1.2.2).

The third challenge is about finding (scalable) strategies to efficiently split up complex models into preferably independent parts, to skillfully determine appropriate granularities (sizes of its parts) as well as to distribute them on multiple IEUs while minimizing synchronization overhead (cf. Subsection 1.2.3).

1.2.1 Identification of Vital Elements as Analysis Basis

In software engineering, there is a broad consensus that finding weaknesses early is desirable when building, altering or migrating systems. Therefore, it seems reasonable to perform preparatory actions prior to actually trying to parallelize an application.

In the context of embedded real-time systems, it is especially important to firstly check a model's integrity as a whole (roughly corresponding to its syntax) in order to ensure having a sound basis for later working steps.

Afterwards, it is crucial to have a methodology clearly depicting which model elements are relevant in terms of logical correctness and timing, enabling a later reasoning. Thus, it is inevitable to determine a minimum set of information that has to be contained (if necessary, apportioned according to specific levels).

Subsequently, it is possible to distinctly analyze the parsed model elements and to discover existing dependencies between them, already aiming at the later revelation of potential weaknesses, so that developers are able to properly counteract as early as possible.

As a final step, working up the processed information and represent them in a meaningful way is a demanding, but most certainly greatly beneficial task. If done skillfully, it effectively supports gaining an overview as delineated in [GS12], where the goal is to collect all dependencies that blend into an "overall picture" facilitating to draw conclusions about an appropriate system architecture.

1.2.2 Correctness and Data Consistency

In order to harness the information obtained by the preceding analysis (which are assumed to be detailed enough), it is necessary to interpret collected data in a well-founded manner.

A prerequisite for the following migration of a system is its integrity concerning the presence of mandatory model elements as well as their correctness. An according verification approach is therefore needed to check the soundness of specified elements as well as the relations among them with respect to an appropriate (standard) specification, e.g., AUTOSAR.

Subsequently, an application needs to be split up in parts that can be distributed and executed in parallel. As sequential programs were never intended to be processed by more than one IEU simultaneously, additional effort is required to ensure maintaining the program's behavior as well as the results' validity.

This is usually achieved by synchronization, which is defined as "restoring of data communication of sequential programs" [Moy13], which boils down to establishing and keeping multiple copies of a data set consistent. In other words, synchronization manages competitive accesses on shared resources like data structures, variables or peripheral devices and aims at ensuring data consistency (respectively validity), which was previously described as data stability and data coherence [GS12].

Unfortunately, synchronization on code level is demanding, costly, hardly avoidable (e.g. by architectural design patterns) and can easily bring along "side effects" like priority inversion, bad performance and further code complexity, e.g., by enforcing sequential access on shared data via "Mutual Exclusion" ("Mutex") mechanisms which can again cause deadlocks

[Moy13, GS12].

Pursuing this approach to ensure determinism compels to detect and resolve all conflicts that can lead to violations, which is very complicated to achieve (cf. [GS12]) as it requires a deep understanding of the system's functionality and thus cannot be done automatically (except for some trivial cases).

Hence, finding a feasible (and automatable) way to preserve an application's original behavior is necessary. In the context of model-based analyses, it is important to determine a concrete proceeding to use its results for drawing conclusions on potential data consistency conflicts with respect to any execution order (e.g. overlapping or completely parallel).

Utilizing timing constraints like those specified by AUTOSAR (cf. [AUT14d]) seems promising, as adding, modifying or removing them (according to a particular rule set) can iteratively lead to a model that distinctly excludes all unwanted behavior despite how it is split up and finally distributed among available IEUs.

In order to cope with huge (complex) models too, the mentioned rule set (or "solution strategy") must be sound and precise to enable automated processing.

Without drawing on "active" synchronization, such an approach would be independent from concrete code fragments and would reduce the synchronization complexity considerably. In addition, the succeeding partitioning can be done without re-thinking about synchronization and data consistency.

1.2.3 Partitioning, Mapping and Granularity

After conducting the verification and data validation, the application should be properly prepared and ready for its partitioning.

As opposed to this, an inevitable precondition for speeding up a program's execution via parallel computation are program parts whose processing can either be allocated to different IEUs or whose order can be changed, both without corrupting the overall result. In other words, the application must not consist completely out of atomic parts that do each consecutively depend on their particular direct predecessor. Assuming this as given condition, it is basically possible to split the program up (partitioning) and to distribute its parts to multiple IEUs (mapping).

In a nutshell, partitioning aims at creating disjoint subsets according to a specific goal. In case of parallelization, this goal is usually independence (cf. [CHS12]), i.e., preferring partitions with a low degree of coupling between its parts.

Although this does not appear complicated, partitioning is considered to be the "single most difficult part of moving to multi-core paradigm" [Moy13] and "one of the fundamental algorithmic operations" [BMS+16] as it involves identifying available concurrency (via analysis) as well as breaking (via partitioning) and repairing (via validation) dependencies. Being confronted with evermore larger and intricate applications, partitioning "becomes more and more important, multifaceted, and challenging" [BMS+16].

Consequently, there is a need for an efficient partitioning concept especially on complex models as they represent an enormous challenge when trying to skillfully split up while causing a minimum of additional synchronization (cf. Subsection 1.1.4). This might include configurable strategies for a scalable partitioning search being applicable to all kinds of models and which reliably yields exploitable results. The same holds true for mapping: determining the actual distribution of found parts to IEUs has to be done in accordance with a substantiated and yet to be developed method.

In addition, figuring out and justifying on which level to conduct these two activities, e.g., code level or task level, is significant for the whole parallelization process as it heavily influences the arising computational effort, the size of the search space to cover as well as the expressiveness of obtained results.

These results might – according to a specific model's properties – include varying part sizes (granularities) as well as nested parts. This is due to the insight that a fixed granularity size is usually adverse as it most commonly cannot properly reflect the according model's structure in terms of, e.g., coupling or cohesion.

Subsequently, offering solutions with differing granularities to choose from affects the mapping process as the number of possible solutions further rises. Therefore, it seems sensible to adapt and employ already established distribution algorithms and to assess their suitability for this "bin packing problem" when determining solutions aiming at, e.g., an adequate load balancing.

In order to being able to examine the added value of found solutions, there has to be an agreement on how to store and transfer determined partitions and mappings, so that a third party simulation/optimization tool can easily import the information enabling an evaluation of the results. In this context, using already existing standards is desirable.

Finally, there is a need for finding expressive assessment criteria to determine the suitability of a partition and mapping in general (i.e. for a yet unknown target platform) or with regard to a specific hardware configuration, because "there is no ideal way to split up a program" [Moy13] and optimization is always done according to a certain target platform (cf. [GS12]).

1.3 Objectives

Taking the presented problems and challenges as basis enables the derivation of concrete research objectives focusing on supporting the migration of complex automotive software to multiple-IEU platforms:

1. Identification of Vital Elements as Analysis Basis: Apply data dependency analyses on sufficiently detailed (low-level) models, find structural problems (model validity), identify relevant elements and enable a proper visualization/filtering system in order to facilitate gaining an overview of highly complex systems (cf. Subsection 1.2.1).
2. Correctness and Data Consistency: Hint at structural shortcomings of models, identify potential data consistency conflicts, offer concrete solutions, support automated conflict solving and thus prepare a system for being distributed and executed in parallel by multiple IEUs (cf. Subsection 1.2.2).
3. Partitioning, Mapping and Granularity: Find a fast and scalable method to identify an advantageous partition, appropriate granularities (with respect to flexibility) as well as an expedient mapping procedure for huge models, so that the synchronization overhead is minimized (cf. Subsection 1.2.3).

These specific objectives contribute to the overall goal to provide a suitable methodology that effectively supports the preferably automated parallelization and deployment of real-time automotive applications by:

- stating a distinct proceeding,
- assisting in finding and treating crucial system parts,
- illustrating how migration can be supported by proper visualization,
- providing expedient support for the partitioning and mapping,
- following scalability as guiding principle and
- putting emphasis on preserving the original system behavior.

The overarching approach and the objectives are depicted in Figure 1.2.

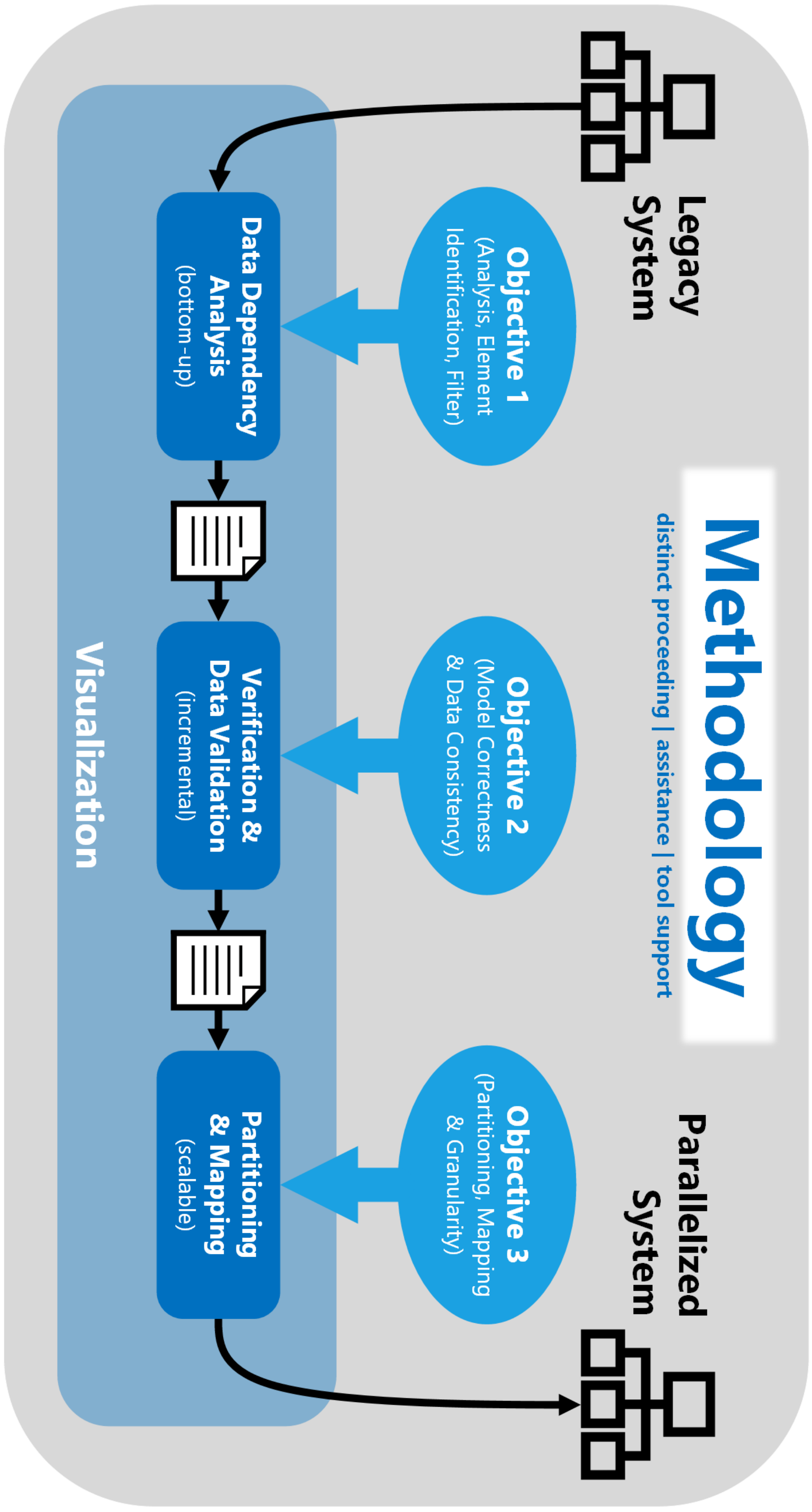

Methodology
distinct proceeding | assistance | tool support
Legacy System
Parallelized System
Objective 1
(Analysis, Element Identification, Filter)
Objective 2
(Model Correctness & Data Consistency)
Objective 3
(Partitioning, Mapping & Granularity)
Data Dependency Analysis
(bottom-up)
Verification & Data Validation
(incremental)
Partitioning & Mapping
(scalable)
Visualization

Figure 1.2: Overview of the approach and objectives

1.4 Publications

Some parts of the approach and results of this thesis have already been published in other disseminations.

The papers and reports (cf. [KMKB14, KK15, KSKB16, KSS$^+$17]) have been written in collaboration with colleagues at the professorship *Software Methodologies for Distributed Systems* (SMDS) of Prof. Dr. Bernhard Bauer at the University of Augsburg, employees of *Continental Automotive GmbH* and *Timing-Architects Embedded Systems GmbH* (meanwhile part of *Vector Informatik GmbH*).

The contributions to the ARAMiS II outcome documents ("project deliverables", cf. [ARA19a, ARA19b]) contain adjusted and refined contents mainly derived from the publications mentioned above.

The following listing shows the corresponding publications of the author together with an outline of his contribution.

1. J. Kienberger, P. Minnerup, S. Kuntz, and B. Bauer. Analysis and Validation of AUTOSAR Models. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 274–281, IEEE, 2014.
   The author of this thesis considerably contributed to the concepts, application conditions and scenarios as well as the case study. He planned and implemented the region search approach based upon previous research at the SMDS professorship. He acted as corresponding author and presented the work at the *MODELSWARD Conference*.

2. J. Kienberger and S. Kuntz. Systematic and Methodical Data-Dependency Analysis for Multiple-IEU Platforms. *ARAMiS Final Report – Continental Automotive GmbH*, pages 41–77, 2015.
   The author of this thesis planned the scope, defined the approach and parts of the challenges as well as the outlook in coordination with Stefan Kuntz. He outlined and realized the motivation, the conceptual basis and the prototypical implementation on his own.

3. J. Kienberger, C. Saad, S. Kuntz, and B. Bauer. Efficient Parallelization of Complex Automotive Systems. *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM)*, pages 40–49. ACM, 2016.

The author of this thesis distinctly extended and enhanced the approach of "1." (cf. [KMKB14]) and ideas of "2." (cf. [KK15]) in terms of coping with highly complex models. Therefor, he worked together with Stefan Kuntz for further developing the conceptual basis and synchronized with Christian Saad in order to properly integrate the improvements into the software platform that was under construction back then.
The author of this thesis acted as corresponding author and presented the work at *PMAM Workshop* held in conjunction with the *PPoPP Conference*.

4. J. Kienberger, S. Schmidhuber, C. Saad, S. Kuntz, and B. Bauer. Parallelizing Highly Complex Engine Management Systems. *Concurrency and Computation: Practice and Experience – Special Issue*, 29(15), 2017.
   Together with the co-authors – especially Stefan Schmidhuber –, the author of this thesis led the refinement of the approach of "3." (cf. [KSKB16]) and extended both the scheduling and optimization parts which served as basis for the afterwards conducted broadened in-depth case study.
   The author of this thesis acted as corresponding author for this extensive journal paper.

5. ARAMiS II Research Project Consortium. E3.2 Design Space Exploration – Achievement (outcome document, version 3). German Aerospace Center (DLR – project executing organisation). 2019.
   Based on his previous research, the author of this thesis adapted and integrated his approaches and findings into the sub-project's scope and goals considering model analysis, verification and data validation. The contributions appear as clearly separated sections that the author of this thesis is solely accountable for.

6. ARAMiS II Research Project Consortium. E3.3 Partitioning of Software Components – Achievement (outcome document, version 3). German Aerospace Center (DLR – project executing organisation). 2019.
   Based on his previous research, the author of this thesis adapted and integrated his approaches and findings into the sub-project's scope and goals considering partitioning, mapping and deployment. The contributions appear as clearly separated sections that the author of this thesis is solely accountable for.

Ideas, concepts, approaches, case studies and conclusions that have already been published in one of these publications are not additionally cited throughout this thesis.

2 Basics

This chapter discusses fundamental ideas, conditions and practices forming the basis for the later built approach and methodology (Section 2.1). Afterwards, typical difficulties arising in the course of the parallelization process – like broached in Subsections 1.1.4 and 1.1.5 – are specified and elaborated (Section 2.2).

2.1 Concepts and Techniques

In order to understand the approach and chain of thought, it is essential to provide an understanding of employed concepts and techniques.

2.1.1 AUTOSAR

A first step to manage the complexity rise (like addressed in Chapter 1) is to agree on a common working basis, which was undoubtedly advantageous even when multi-core approaches were still in their infancy. Nevertheless, it was clear that including more and more features in vehicles raises both size and "spread" of embedded software. Therefore, several leading manufacturers, suppliers and vendors of the automotive sector started a collaboration in 2003 and developed AUTOSAR.

AUTOSAR specifies a uniform software architecture and defines interfaces for communication as well as configuration formats which facilitate the exchange of ECU software, assure its possible reuse and make it scalable [AUT13, RN16]. In addition, it provides a standardized development methodology that facilitates the collaboration between various partners (e.g. companies) as well as a real-time operating system [Gra16, SBR10].

All this is guided by the principles of "Model Driven Engineering" as AUTOSAR's development process "is based on the use of models defined through meta-models and provides a development process based on progressive refinement of models" [WMM$^+$13].

In summary, AUTOSAR represents an "effective way to manage growing system-level complexity, keep costs affordable and protect for future innovation" [AUT14a] by simplifying the exchange of application software [Gra16].

Sticking to AUTOSAR has become virtually mandatory for both OEMs[5] and suppliers, because it has enormously gained in significance since its introduction. As no other commonly accepted alternative is available (nor is in prospect), this state and trend are unlikely to change.

[5]As the term "Original Equipment Manufacturer" (OEM) is ambiguous, its specific meaning depends on the context. In the automotive sector, it usually refers to an end-product producer/seller (like Audi or BMW) who integrates subsystems from other companies (suppliers like Continental or Denso) in its own products.

In conclusion, it can be stated that AUTOSAR provides a suitable basis to evaluate the added value of the approach presented in Chapter 4. Considering that, the succeeding paragraphs give an overview by outlining essential notions and aspects, namely AUTOSAR's structure (2.1.1.1), synthesis process (2.1.1.2), multi-core capabilities (2.1.1.3) and correlation with a high-level description language pre-dating it (2.1.1.4).

2.1.1.1 Software Stack

The layered architecture of AUTOSAR is shown in Figure 2.1. It can be roughly divided into three layers: the "Basic Software" (BSW) in the lower half (upon the actual hardware), the "Runtime Environment" (RTE) above and "Application Software" (ASW) on top.

The BSW consists of an "operating system" (OS) and services like input/output handling, memory management and hardware abstractions [AK13]. The "Complex Device Drivers" column on the right side is dedicated to unmediated hardware access, e.g., direct access on BSW functions or even communication without employing the RTE [SKS10].

The RTE above is the ECU-specific implementation of the "conceptual communication mechanism" [LLP+09] referred to as "Virtual Functional Bus" (VFB). Therefore, the RTE is the realization of the interfaces (and thus the communication) between "Software Components" (SW-Cs) on different IEUs and is responsible for, e.g., signal buffering or RE triggering [SKS10, Gra16, AK13]. In AUTOSAR, communication takes place via two basic approaches (according to [PKQ+14, LLP+09]):

- In the "Client/Server" pattern, a defined set of server operations can be called by one or multiple clients, i.e., provided services are invoked from other REs.
- For the "Sender/Receiver" pattern, a defined set of data elements can be sent and received via the VFB using global shared memory.

Communication is realized via buffering mechanisms, i.e., by copying the data in task-local buffers before processing them [MFCM16]. This approach is intended to preclude data races right from the outset ("by design").

As a direct result of the layered architecture, the actual applications (the SW-Cs on the ASW layer) are independent from the underlying infrastructure (the specific hardware) [LLP+09, AK13].

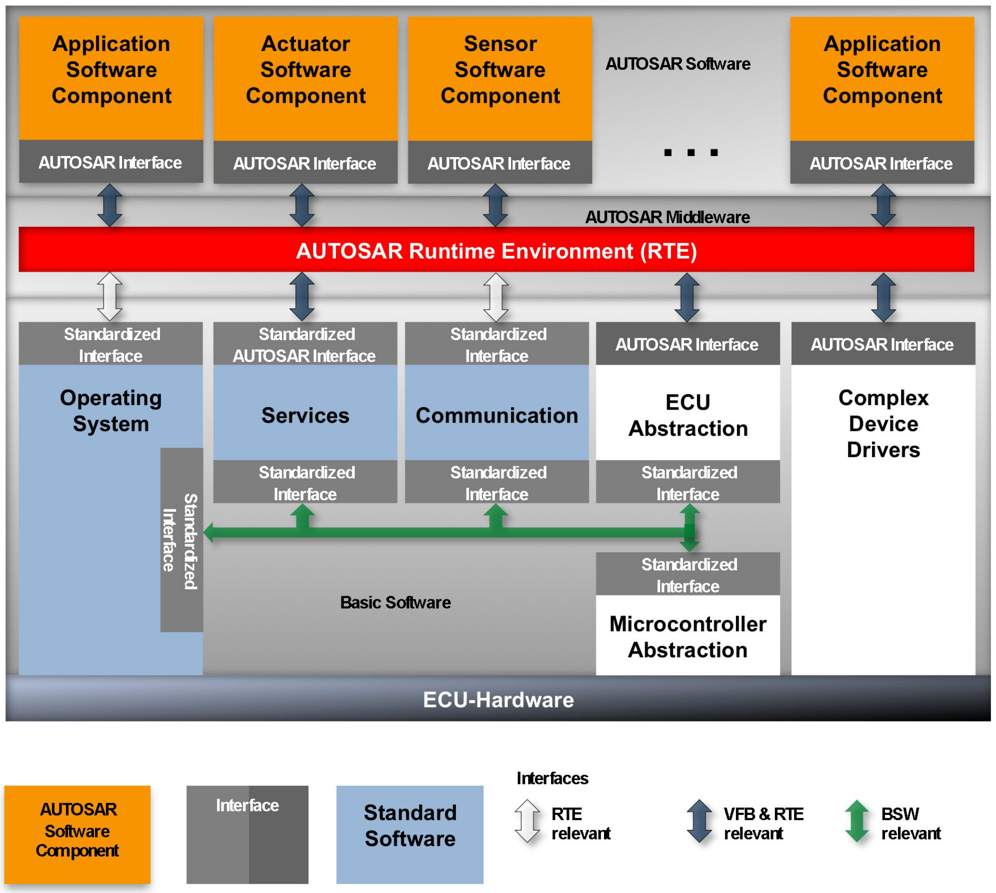


Figure 2.1: The layered architecture of AUTOSAR (from [AUT14e])

2.1.1.2 Synthesis Process

As already indicated, AUTOSAR "defines a synthesis process of software components and their connections in a set of fixed-priority OS tasks distributed over a network of ECUs" [WMM+13]. In other words, the functional architecture is transitioned into a (synthesized) real-time architecture. This process mainly contains four basic activities (cf. [WMM+13, PKQ+14]):

1. Allocation comprises the mapping of SW-Cs to ECUs, including contained REs, ports and accessed variables (signals).
2. Partitioning is the disjoint assignment of REs (as they are the atomic schedulable units) together with the signals in OS tasks.

3. Scheduling is about creating an execution plan for these tasks as well as setting static priorities for them.
4. Ordering determines a sequence of REs in tasks taking heed of imposed constraints.

In the course of this process, there are – according to [AUT14d, WMM+13, PKQ+14] – some typically applied practices, namely:

- the specification of "end-to-end" timing constraints between ports (already) on the highest abstraction level,
- the refinement of these constraints considering chains of REs (time span from stimulus to response),
- the calculation of emerging response times of REs chains to check if imposed constraints are met and
- the re-configuration of constraints if they are violated.

Based on experience, these steps are – to a considerable extent – carried out manually which makes the whole approach rather cumbersome and time-consuming. Therefore, there is again great potential for optimization.

2.1.1.3 Multi-Core Capabilities

Roughly nine years ago, the AUTOSAR consortium released version 4.0 of the standard [SKS10]. Since then, AUTOSAR supports distributed execution via partitions for the purposes of multi-core and safety (e.g. isolating safety-critical parts) [Sie16, Gra16]. That means in effect that the mapping of SW-Cs to different IEUs is enabled, which offers more freedom compared with distributing only SW-Cs to ECUs as before [ZG11, SKS10].

Although it is possible to assign REs from several SW-Cs to different tasks, AUTOSAR does – however – not allow to map these tasks to different IEUs and thus permits to distribute an SW-C across IEUs [BKL16]. This condition can be ascribed to complying with the guideline that safety imperatively requires decoupling (cf. [Sie16]), so that a "mixture" of different applications on a common IEU is not desirable.

Unfortunately, treating SW-Cs as atomic scheduling units limits both the integration flexibility as well as optimization capabilities. The former leads to an unnecessary often re-design of applications (e.g. due to reduced load balancing possibilities) while the latter complicates implementing "mixed architectures" [Sie16]. Furthermore, only the ASW can be distributed while the BSW must be run on one specific IEU in its entirety, which could turn out to be a significant "bottleneck" [SKS10].

Facing these limitations, many companies have developed own solutions to circumvent potential obstacles, e.g., in "PowerSAR", which is Continental's "powertrain implementation of AUTOSAR standards" [Con16], SW-C distribution across cores is enabled [Sie16].

In terms of scheduling, AUTOSAR pursues a static mapping approach, meaning that there is one scheduler per core (or OS instance) [Neu16, SBR10]. This corresponds to the partitioned scheduling pattern as opposed to global scheduling where "tasks are scheduled by a single scheduler based on their priorities and each task can be executed on any core" [NBN09], which means that task migration is enabled.

Cross-core interrupts and inter-core locking are possible, thus AUTOSAR uses the "Priority Ceiling Protocol" (PCP) in order to prevent priority inversion and deadlocks [Neu16, SBR10]. Since PCP is not working with partitioned scheduling, AUTOSAR's OS provides spinlocks for multi-core synchronization [SBR10]. Unfortunately, spinlocks can – without further measures taken – nevertheless result in priority inversion [Cor13]. In addition, it is possible to implement a multiprocessor variant of PCP in AUTOSAR [Neu16].

In the future, several improvements and extensions with regard to the multi-core capabilities of AUTOSAR are planned in order to meet market needs. This mainly addresses the following aspects (cf. [Asm17]):

- the portability of SW-Cs and their distribution (and synchronization) across cores,
- increasing the "depth" of the specification,
- achieving more independence on the microcontroller architecture,
- reducing implementation assumptions in the specification,
- support of the "Logical Execution Time" concept (LET – cf. Subsection 6.3.6),
- enhancing the distribution of BSW,
- improvements for inter-core communication and
- multi-core exchange formats.

2.1.1.4 Relation to EAST-ADL

As explained, AUTOSAR "provides means to describe software architecture architectures [sic]" [QCLT11], namely hardware entities and topology, SW-Cs with REs as well as their mapping to tasks and IEUs [MAE13b].

However, AUTOSAR is not intended to cover all abstraction levels of a typical development process. Rather, it is distinctly embedded in a methodology called "Electronic Architecture and Software Tools – Architecture Description Language" (EAST-ADL), that was designed to complement AUTOSAR with higher levels of abstraction as well as with additional concepts [Sei09, MAE13b, MAE13a, QCLT11].

Being confronted with increasing system complexity and criticality that eventually endanger quality and safety, EAST-ADL's motives are similar to those of AUTOSAR: Effectively supporting the development of automotive systems by collaboration on tools, methods and exchange models [MAE13b, MAE13a]. This is also reflected by the fact that both have the same meta-model, which consequently means that adding capability to EAST-ADL corresponds to doing the same to AUTOSAR [MAE13b].

Initially developed in the course of the "EAST-EEA"[6] research project, EAST-ADL draws on established modeling approaches like "SysML"[7] to provide a common basis for model-based embedded systems development [CFJ$^+$07, QCLT11, MAE13a].

As illustrated in Figure 2.2, EAST-ADL is structured in horizontal layers as well as vertical cross-cutting concerns [MAE13a]. The layers are organized from the most abstract one on top to the implementation level at the bottom (cf. [MAE13b, CCG$^+$07, CFJ$^+$07, Sei09]):

- **Vehicle Level**: The features of the vehicle are represented by the "TechnicalFeatureModel" involving use case diagrams, activity diagrams and dynamic aspects, e.g., time and frequency.
- **Analysis Level**: Here, the "FunctionalAnalysisArchitecture" contains abstract functions and defines context by means of system boundaries together with an environmental model. It delineates logical parts of the system like dependencies of functional components via inter-

[6] "Electronic Architecture and Software Tools – Embedded Electronic Architecture" project: `https://itea3.org/project/east-eea.html`

[7] "Systems Modeling Language": `https://sysml.org/`

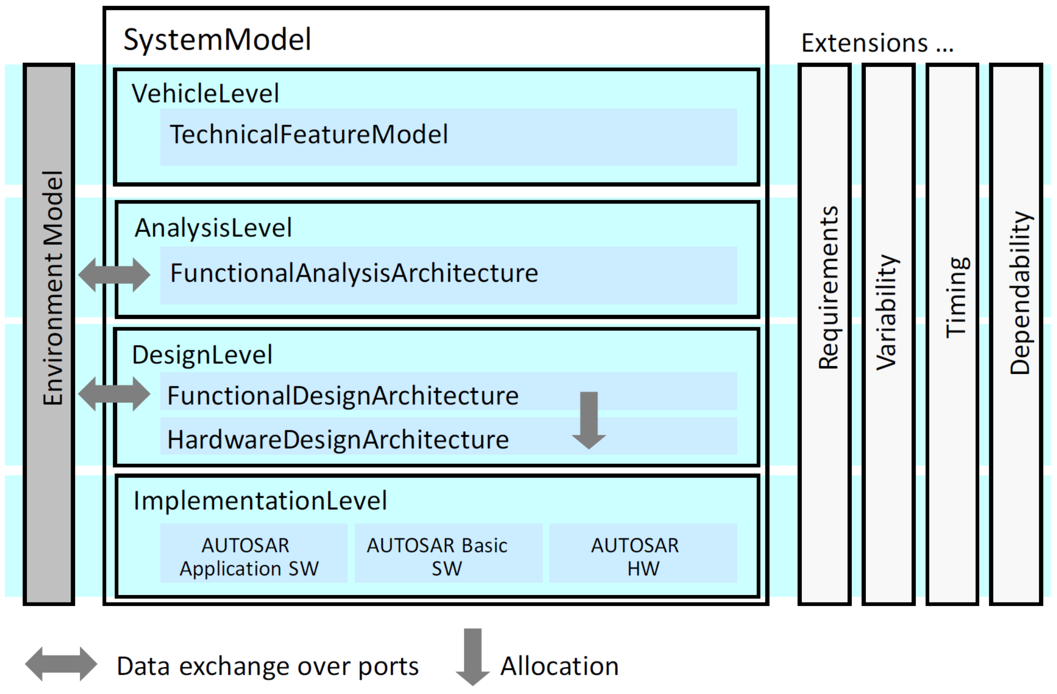


Figure 2.2: Abstraction levels of EAST-ADL (from [BLH+13])

faces/ports, data models and business objects as well as the communication between memory and bus.

- **Design Level**: As further refinement, this level's "FunctionalDesign-Architecture" comprises interaction with environment and end-to-end functional definitions – thus first implementation-oriented aspects are brought in. Subsequently, the "HardwareDesignArchitecture" represents the hardware topology with an allocation of concrete functions to hardware elements.

- **Implementation Level**: Taking heed of the constraints resulting from decisions made on higher levels, the instantiated implementation is represented by a specific software architecture, e.g., AUTOSAR.

In addition, the depicted environmental model and the extensions are vertically arranged in order to represent aspects like requirements, timing, dependability, traceability and – in order to enable feature modeling and product line management – variability [CCG+07, CFJ+07, MAE13a].

2.1.2 Artop

As already implied, there is a strong demand for tools that both harness the advantages of AUTOSAR and support the complexity handling when migrating to or constructing new architectures that feature multiple IEUs [SMD$^+$10, GNN$^+$06]. As AUTOSAR "only" provides a common basis for a variety of tools but lacks a ready-to-use implementation, the "AUTOSAR Tool Platform" (Artop) was created [Art12]. Artop serves as Eclipse infrastructure and virtually acts as "persistence layer" enabling common base functionality like easy access on AUTOSAR models that adhere to specific meta-model versions. Therefore, Artop facilitates the construction of AUTOSAR tools. Its major components include features like meta-model implementations, model comparison, model validation, explorers, editors and a code generation infrastructure [Art12].

2.1.3 Data Dependency Analysis on Models

Like further elaborated in Section 5.1, Artop is utilized to build a tool that performs a data dependency analysis on AUTOSAR models.

Dependency analysis is often considered to be a decisive step within the automotive development process [Fuk18]. This common opinion primarily arises from the goal to help mastering the vast complexity. Therefore, it is very useful to examine which program parts interact, what data is transferred and how often communication takes place.

Gathering such information is crucial in order to determine unintentional effects like corrupted data validity when shifting to a multiple-IEU environment. Moreover, it enables to clearly visualize a system's data dependencies by means of a directed graph with nodes representing functional blocks, compositions including disjoint node sets and edges indicating pairwise node dependencies. The latter are either "variable accesses" (VAs) between two nodes or timing statements that affect both of them.

The crucial point is how to interpret and evaluate this information. Making basic statements about the degree of interconnection between certain nodes is pretty easy, whereas it is much more challenging to find out whether a certain dependency might compromise the system's functionality when the execution order of the functional blocks in the migrated (multiple-IEUs) system differs from the original single "Execution Unit" (EU) one. This is usually the case when functional blocks are processed in parallel instead of well-matched and rigidly consecutive like on a single-EU system. As a consequence, conflicts like data not being available in time or data being read inconsistently can occur. Of course, consistency conflicts like this can arise in single-EU systems too, but multiple-IEUs systems are more prone to "evoke" them because here it is – due to concurrency – significantly harder to maintain consistency.

2.1.4 Conflicts and Backward Dependencies

A typical conflicting scenario would be the following situation: Data produced (i.e. initialized or written) by a functional block "fb-1" is consumed (i.e. read) by "fb-2", which uses it to produce new data. This data is then consumed by "fb-3" which runs on a different IEU and is actually executed earlier or at the same time as "fb-1". This results in a conflict due to inconsistency, because "fb-3" is forced to use "old" data (i.e. data from a previous computing cycle[8]) for its processing.

A data access like the one of "fb-3" on "fb-2" is categorized as "backward dependency", because it is virtually directed against the major program flow. A further problem would be "fb-1" reading data produced by "fb-3", because this would create a cycle in the dependency graph (cf. Chapter 4 for details). Figure 2.3 illustrates the described situation for the three functional blocks mapped on two cores.

Thus, the absence of any backward dependencies means that each producer of a certain data element is consistently executed before any consumer (within one computing cycle). In such a case, no potential multiple-IEU-induced problems emerge as long as the single-EU system itself was not defective right from the start.

2.1.5 Handling Upcoming Multiplicities

Unfortunately, it is very unrealistic to assume that no backward dependencies arise when a system is migrated to be carried out on multiple IEUs. However, a setup like the one outlined in the aforementioned example does

[8]"Computing cycle" is defined as the time elapsed between two events that involve periodically activated tasks (sets of functional blocks) being guided by the slowest (least triggered) task occurring.

not necessarily lead to serious problems – like wrong calculations or even a whole system failure – and can yet be unproblematic at all. Nevertheless, it is inevitable to take care of any potentially unintentional behavior within a system. The trouble here is the already addressed complexity rise caused by the exponentially growing number of possibilities to distribute tasks on IEUs, which leads – together with scheduling – to a tremendous count of possible execution sequences including many adverse, i.e., conflict-ridden, ones.

In order to ensure that a system will work properly irrespective of a certain task-to-IEU mapping, it is required to distinctly exclude all unwanted behavior ("adverse paths") and therefore cover every contingency. If it is not intended, possible or promising enough to change the system structure itself, a way to achieve "multiple-IEU robustness" is enriching the model with timing constraints or modifying pre-existing ones. They affect either sets of functional blocks or specific dependencies (VAs).

Such constraints can, e.g., dictate a rigid execution order between two or more functional blocks, which is suitable if they are logically (and seman-

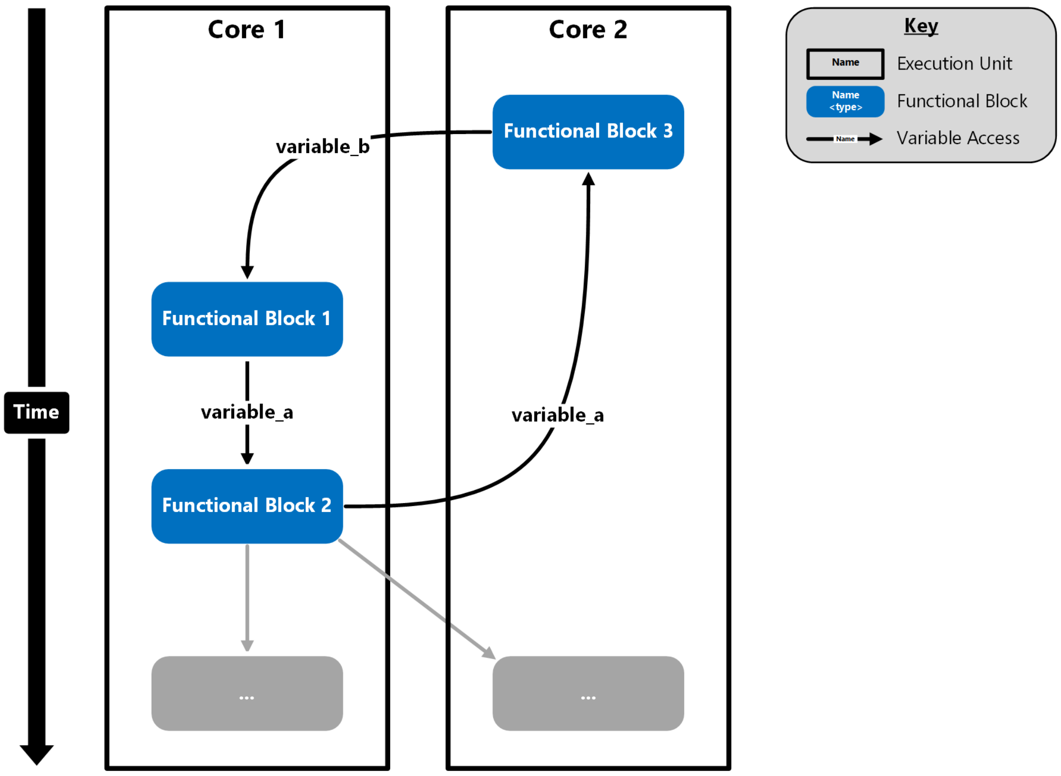


Figure 2.3: Possible conflicting scenario for a multiple-IEU system

tically) connected like it is the case in a classical "sensor-controller-actor-system", where sensors transfer measurement data to a controller that computes the appropriate action carried out by succeeding actors (e.g. brakes in an ABS). Another possibility to solve potential conflicts via constraint imposition is to mark a backward dependency as unproblematic, meaning to allow that certain accessed data originates from a previous computing cycle. This is in particular legit if the reading functional block does not require current data to work correctly, e.g., a speedometer – like in the WSS example – that cannot (and is not intended to) react within milliseconds due to the speedometer needle's inertia (cf. Chapter 4 for details).

After this process of eradicating all potential multiple-IEU problems, a model is ready to be split up safely into functional blocks that can be mapped on different IEUs. This is usually done according to a specific goal, e.g., only having little coupling between the functional blocks and therefore rather few communication, or assuring certain safety requirements, like with distinctly separating highly critical functional blocks. Finding an "optimal" partition is ranked as NP-hard problem and as the number of possibilities to distribute the functional blocks to IEUs grows quickly, this task is one of the biggest challenges for multiple-IEU systems [Sar87].

2.1.6 Starting Conditions

A final aspect to help overviewing multiple-IEU data dependency analyses is to take different initial conditions into consideration. Here, a model's degree of abstraction is one key factor, because it has essential influence on the expressiveness of the results. The analysis described here is carried out on the most fine-granular level for application software in AUTOSAR, namely on its smallest executable units – the REs[9].

As opposed to this, analyses on a more coarse-grained level, e.g., inspecting SW-Cs and neglecting the REs they encapsulate, would not reveal as much information but could provide a useful overview especially when dealing with huge models.

Another basic point is the "degree of predefinition" within the analyzed model, which denotes how many constraints are imposed from the start.

[9]As ASW's atomic units, REs are distinguish from "Basic Software Module Entities" that are their counterpart in BSW (providing infrastructural functionalities of an ECU) [AUT14b].

Such constraints can already define an execution order. Depending on this conditions, an analysis can have very different goals:

- Constraint-free models: trying to specify a proper execution order on the basis of the obtained dependency graph
- Constrained models: trying to specify a proper execution order with respect to the imposed constraints and check the constraints' validity
- Constrained models with a given execution order: find cycle problems within the corresponding data dependency graph

As described in Chapter 4, the approach is designed to cope with each of these conditions.

2.1.7 Refined Example

Figure 2.4 shows an enriched WSS according to AUTOSAR semantics. The seven functional blocks each represent an RE that belongs to a specific SW-C. Data accesses within an SW-C are indicated via dashed lines, whereas solid lines depict read/write operations that cross SW-C borders. The latter are represented in a simplified form (without AUTOSAR's "Ports" and "Connectors"). Furthermore, the gray boxes show how often the RE is triggered. As no constraints are imposed yet, every execution order – obtained by permuting the REs arbitrarily – is feasible (cf. calculation examples in Subsections 2.2.6 and 4.3.2).

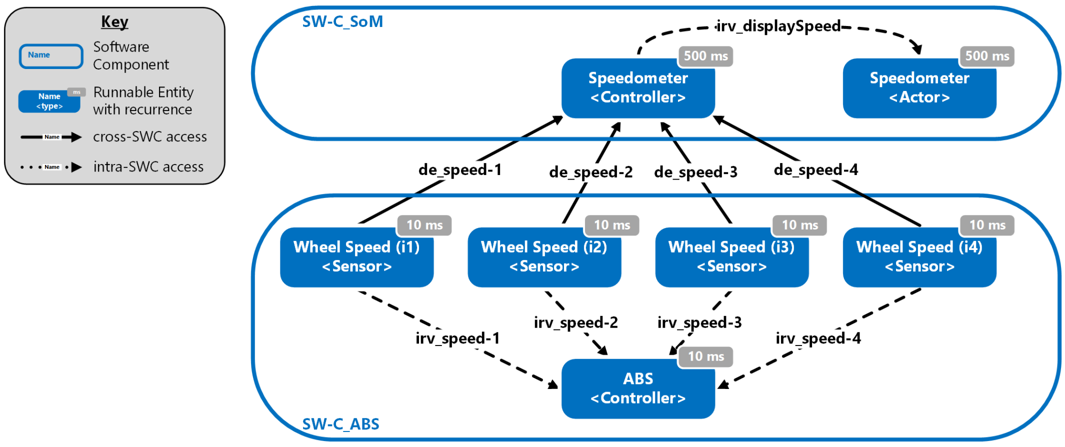


Figure 2.4: The Wheel Speed System with AUTOSAR semantics

2.2 Aggravating Factors

Generally speaking, there are sundry factors that can exacerbate gainful and expressive dataflow analyses (and succeeding partition searches) on AUTOSAR models. A start is made by explaining those that were broached in the previous sections.

2.2.1 From Single Chains to Multiple Paths

At the beginning, it can be quite difficult to determine differences between the behavior of a single-EU "source system" and a migrated one featuring multiple IEUs. In the former case, the execution order of the functional blocks was usually carefully designed and tested according to detailed knowledge and acquired experiences about execution frequencies and computing cycles.

For the latter case, it is necessary to take numerous contingencies into consideration that were per se excluded in the single-EU system with its one rigid execution chain. By enabling the capability of processing functional blocks actually parallel (and not "quasi-parallel"), a multiple-IEU system opens up many additional possibilities.

2.2.2 Preservation of Freedom

Unfortunately, many of these new "paths" are not expedient. Changing the original processing order can lead to a functional block reading invalid data because every data element has to be produced before it can be consumed. However, a fixed order cannot be maintained when migrating to multiple IEUs because there is a need for more freedom concerning the execution order of the functional blocks when they are processed distributed. Thoughtlessly sticking to the rigid order of a single-EU system would result in either using only one IEU or would cause needless latencies by forcing the IEUs to incessantly wait for each other.

Therefore, one challenge is to find a golden mean between ensuring that a multiple-IEU system will work correctly by imposing constraints and simultaneously leaving sufficient freedom to enable parallelization benefits. For a software engineer, it is a tightrope walk to preserve flexibility while prevent the system from entering problematic states that can cause, e.g., race conditions, data inconsistencies or dead locks. Experiences show that starting with imposing only a minimal set of constraints seems suitable concerning classic migration goals like load balancing, because many rigidly dictated sequences are prone to complicate the search for an expedient partition.

2.2.3 Complexity Rise Aspects

As indicated earlier, the complexity rise is probably the biggest challenge within the whole field of dealing with multiple IEUs. Assuming that available software and models are ready for parallelization (which can cause much extra effort), complications are mainly arising from two factors:

- It is difficult to find a partition (a set of disjoint executable units) that splits up the software into parts coping with several demands like appropriate chunk size, preferably few cut dependencies or aiming at a fixed number of fragments (irrespective of the specific software's structure).
- The number of possibilities to distribute the tasks on IEUs grows exponentially according to their count and as a consequence thereof, the number of execution sequences escalates too (cf. Subsection 2.2.6).

2.2.4 Optimal Partitions and Strategies

As already referred to in Subsection 2.1.5, it is an NP-hard problem to find an optimal partition in regard to certain characteristics. By and large, it is appropriate to start from the premise that cutting as little edges as possible is desired because every "broken" dependency causes some kind of synchronization effort to preserve the results' correctness [Moy13]. A common approach to do so is known as "Sparsest Cut Problem". It describes the difficulty to bipartition a graph's nodes aiming at minimizing the ratio "number of cut edges / number of nodes of smaller partition" [CKK$^+$06].

An established technique to proceed more target-oriented is providing the edges with weights to differentiate between their relevance for the whole system. The weight values can, e.g., characterize the criticality of data, its sheer amount when being transferred or an edge's usage frequency, which can be derived from the associated nodes' recurrence (also called "period" or "triggering frequency"). The "damage" caused by cutting off edges is then determined by adding up the edges' weights. The aforementioned sparsest cut formula can be adjusted to satisfy this by replacing the numerator with a damage sum.

2.2.5 Adjusted Partitioning

Sparsest cuts are often applied when the obtained chunks are intended to be (roughly) equal-sized, because this proceeding corresponds to "recursive bisection", i.e., a divide-and-conquer pattern used to repeatedly split a graph into two partitions, which leads to a 2^x number of partitions. This proceeding neglects that the graph's specific structure may not be suitable, e.g., software components in the automotive sector have structures that differ strongly according to the purpose of their ECU (according to [Fuk18] and [Achim Demelt, personal communication, 2014]):

- Passenger compartment software often has many discrete states for, e.g., the light ("on/off") or the number of the selected gear. This is a typical example for open-loop control, which only uses the current state of the system (and therefore no feedback about the controlled process) as input for its computation.
- Driver assistance systems like "Adaptive Cruise Control" or road sign detection predominantly use (rather nested) data structures and are realized with closed-loop control. The latter makes software in general more complex and therefore harder to partition.
- Engine Management Systems mostly process floating values for, e.g., injection pressure and temperature. They can be regarded as combination of closed-loop control and open-loop state systems.

Basically, there is a fundamental difference between control applications – characterized by rather small loops, absent data parallelism and strict real-time constraints – and typical high performance applications [KBL17].

A simple procedure like recursive bisection will not be able to adapt to such specific structures, while other approaches are more flexible: For instance, "Multi-Level Partitioning" (MLP) reduces huge graphs to smaller ones that represent their global structure (cf. [ACU08, MPS07]):

1. Initially, a graph is "coarsened", i.e., reduced via contracting edges while preserving its basic structure. Edge contraction is done by selecting a certain edge (e.g. according to aforementioned weights), removing it and merging the two nodes/vertices that were previously connected by them.
2. This process is repeated until a certain threshold, e.g., a fixed number of nodes is reached.
3. Now, the obtained shrunk graph is manageable and can be easily partitioned.
4. Afterwards, obtained results are incrementally mapped back by assigning "high level" vertices to partitions that match their "lower level" representatives.
5. The process is complete once the partitioning was applied to the (huge) original graph.

MLP with its gradual problem approximation is considered "the most successful heuristic for partitioning large graphs" [BMS+16], especially as global approaches have turned out to be too slow and simply yielding insufficient results when being applied to complex graphs [MPS07].

2.2.6 Numerousness of Possible Distributions

After finding a partition, the obtained sets of functional blocks (e.g. RE sets) have to be distributed. To illustrate the complexity associated with this, the running example that consists of seven functional blocks is used. Here, each block corresponds to a task intended to be separately distributed on multiple IEUs. Thus, having seven tasks and two IEUs leads to 128 (2^7) distribution possibilities. Accordingly, the number of options runs up to 2187 (3^7) for three IEUs and 16384 (4^7) for four IEUs. This is quite a lot in regard to the small size of the model.

As already mid-sized models involve trillions of partitions (roughly $1.1 * 10^{12}$ for 20 tasks and 4 IEUs), it is obvious that engineers cannot look at every alternative for a complex real-world ECU software like shown in Figure 2.5 from [Hob12] where Continental's Engine Management System "EMS2" is illustrated as clustered graph with about 8000 nodes (REs) and a strongly reduced set of edges ("maximum weighted spanning tree") indicating the dependencies.

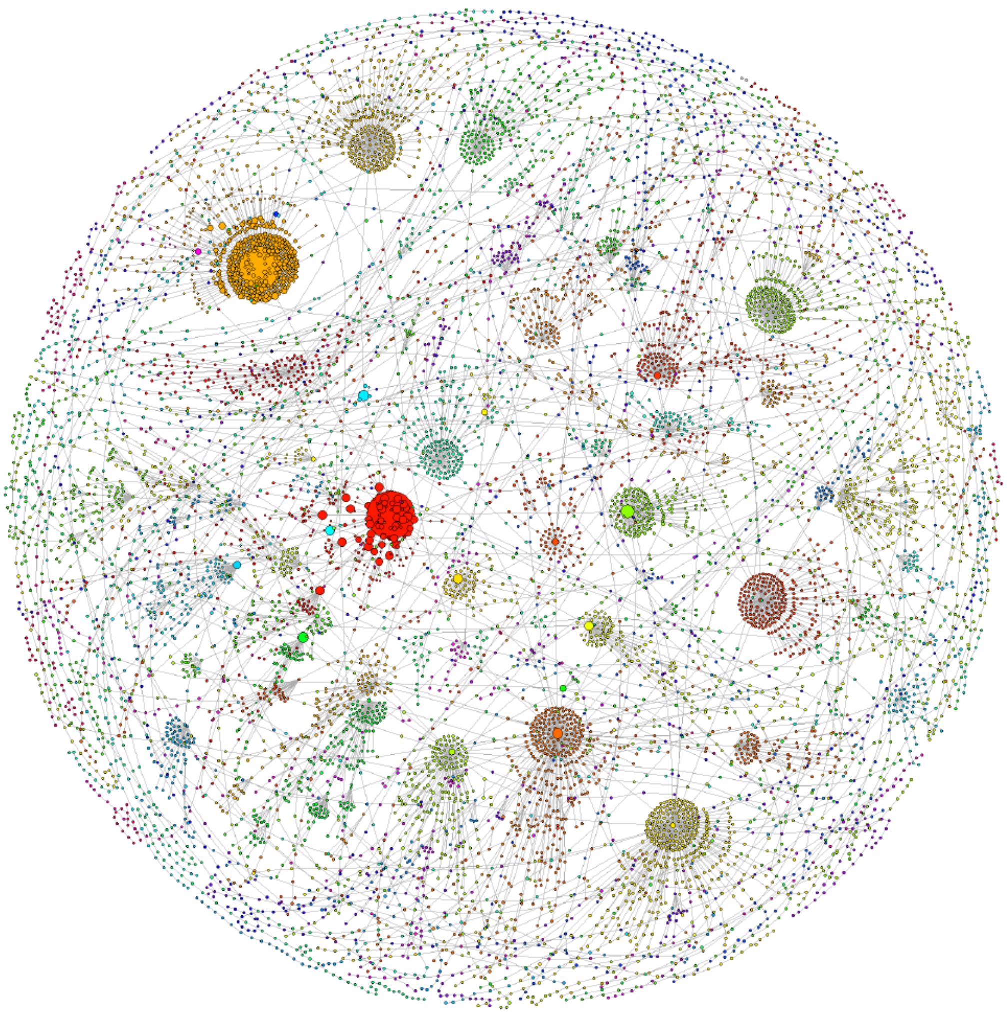


Figure 2.5: Continental's "EMS2" as clustered graph (from [Hob12])

2.2.7 Task Embedding and Scheduling

Up to now, "tasks" merely appear as containers for functional blocks. In fact, embedding in tasks is more than just treating them as "atomic scheduling unit", because each task has itself a certain recurrence that does not necessarily satisfy those of the blocks it includes. This can – in theory – change the whole system's behavior and might therefore further complicate the analysis by corrupting its results. An assumption is, that a scheduler will not act in a malevolent or unduly dumb way by unnecessarily breaking up partition suggestions (that preferably "cluster" REs with uniform recurrences) or providing tasks with recurrences that contradict those of the REs they represent.

3 Related Work

Within the past decades, various approaches and strategies for a proper and efficient parallelization process have been developed. This process can be divided into the optional (preparatory) actions “analysis” and “verification/validation” as well as the obligatory main tasks “partitioning” and “mapping”. Most of the introduced approaches focus on a specific subset of these activities.

This chapter categorizes and outlines existing research approaches in the range of parallelization of embedded multiple-IEU systems and shows differences in comparison with the methodology (Chapter 4) and case studies (Chapter 5) from this thesis. In conclusion, an overview and comparison are given at the end of the chapter (Section 3.4).

3.1 Partitioning Frameworks

Graph partitioning has been a research topic within the field of graph theory for many decades. This is mainly due to its frequent occurrence as underlying difficulty or subproblem in very different areas such as "circuit placement, matrix factorization, load balancing, and community detection" [KHKM11]. Its simplest representation – the graph bisection (cf. Subsection 2.2.5) – can be defined as "the task of dividing the vertices of a graph into two equally sized subsets such that the number of edges connecting vertices from both sets is minimal" [MPS07].

At a first glance, this does not sound exceedingly complex. However, graph bisection embodies a computationally demanding combinatorial optimization problem ranked as NP-hard [KHKM11, MPS07]. As a consequence, plenty of sophisticated approaches and solutions have been developed that provide support for this intricate subtask within the parallelization.

In many cases, elaborate research resulted in the generation of frameworks, e.g., "Karlsruhe High Quality Partitioning"[10] (KaHIP) [SS13]. KaHIP's core algorithm is based on a "multilevel graph partitioning approach, where the graph is recursively contracted to create smaller graphs which should reflect the same basic structure as the input graph" [SS15] (cf. MLP introduced in Subsection 2.2.5). This coarsening process is continued until the graph's size and complexity are manageable, so that it can be partitioned with reasonable effort. Afterwards, the graph is incrementally rebuilt and its partitioning is iteratively refined locally [SS15]. This proceeding draws on an idea that was published in 1995 by Hendrickson and Leland [HL95].

Similar to other frameworks, e.g., "Scotch"[11] that offers a variety of sequential and parallel partitioning techniques, KaHIP does provide an efficient solution for the classical "graph partitioning problem", but lacks – due to its generic character – necessary adaptations to the special requirements of embedded real-time systems. The same holds true for well-known and widely used MLP algorithmic realizations like "Chaco"[12] and "ParMetis"[13] as parallelized implementation within the "METIS" partitioning software family [BMS$^+$16].

[10] http://algo2.iti.kit.edu/kahip/

[11] https://www.labri.fr/perso/pelegrin/scotch/

[12] https://www3.cs.stonybrook.edu/~algorith/implement/chaco/implement.shtml

[13] http://glaros.dtc.umn.edu/gkhome/metis/parmetis/overview

3.2 Limited Approaches

There are a couple of approaches that are partly relevant and applicable within the context of this thesis. Considering the methodology described in Chapter 4, they each can be distinguished from it by a – sometimes deliberately – limited scope, by strongly simplifying assumptions or a coarse-grained abstraction level as well as by insufficient scalability. The latter is often indicated by small-scale case studies or experiments on purely fictional models.

3.2.1 Optimization of AUTOSAR Synthesis Process

In [WMM$^+$13], the authors introduce an approach for optimizing the synthesis process of AUTOSAR (cf. 2.1.1.2) and compliant architectures – namely the allocation of SW-Cs to ECUs and the succeeding assignment of REs to schedulable tasks. Both is carried out with respect to response times and resource consumption. Optimization is achieved by employing "Mixed Integer Linear Programming" (MILP – a wide-spread mathematical optimization method) and "Genetic Algorithm" (GA) techniques. The goal is to provide a "holistic approach that considers allocation, partitioning, scheduling and ordering together" [WMM$^+$13].

The major shortcoming is that the approach is hardly applicable on complex systems as MILP algorithms do not scale well and the used GA implementation was not yet parallelized. Consequently, the experiments were conducted on a rather small-sized model including merely nine nodes.

3.2.2 Allocation with OCL

The authors of [PH15] point out that specifying an allocation of software parts to hardware is a particular intricate task, because it is necessary to take heed of many dependencies (e.g. concerning topology or timing) as well as constraints (e.g. arising from memory or routing needs) all at once. Referring to ILP solutions like in the previous Subsection 3.2.1, they thus state that "encoding the allocation problem as a linear program is a complex and error-prone task".

Therefore, a model-driven allocation approach is presented that is based on the formal "Object Constraint Language" (OCL), which is a part of the

"Unified Modeling Language" (UML). It features the newly created "Allocation Specification Language" (ASL) which is intended to support both a facilitated specification of sound allocation constraints (for software and hardware models) as well as an automated synthesis of valid allocations.

In a nutshell, the suggested workflow begins with the allocation specification via ASL, proceeds with the derivation of linear equations, their resolution by a solver and finally the inference (and back-transformation) of an allocation model with a specific mapping of software parts to hardware elements (IEUs).

The approach is described as being particularly applicable and useful for defining constraints in domains where "constraint satisfaction is a crucial aspect that needs to be satisfied in order to the system being accepted" – which is certainly the case in the automotive section. Moreover, it enables engineers to depict and solve such problems by means of ILP despite having no further knowledge about it.

However, the authors do not make any statements about the runtime or scalability of the employed solver as well as the model-to-model transformations that have to be carried out. As the presented case study, a "Brake-by-Wire" system modeled with "MechatronicUML"[14] merely includes 13 atomic executable (and schedulable) entities, it remains unclear if the approach can handle complex models in an efficient way.

3.2.3 Mapping Rule Set

Already in 2009, Long et al. described general rules to minimize intra-ECU communication via skillfully mapping REs [LLP+09]. The optimization goals are to reduce task switching as well as to avoid data inconsistencies and unnecessary latencies. By considering possible communication behaviors of REs, six rules for a proper mapping are derived, e.g., "Map sender & receiver to the same task.", "Map multiple receivers to the same task." or "Map multiple senders to the same task." [LLP+09]. These simple rules provide a guideline for a basic mapping proceeding.

However, they are barely applicable in terms of consistency and adequacy when handling sophisticated systems with heavily communicating REs, which is, e.g., characteristic for powertrain applications. Accordingly, the performed case study shows how the rules can be applied to a small vehicle

[14]`http://www.mechatronicuml.org/en/index.html`

control application comprising only five SW-Cs with seven REs in total.

3.2.4 Mapping Optimization Issues

In [ZG11], the authors deal with optimization issues in the course of mapping AUTOSAR's SW-Cs to distributed hardware platforms featuring multiple ECUs. For this purpose, a preparatory analysis is conducted which determines connectivity between SW-Cs. The process itself comprises the GA-supported mapping of SW-Cs to ECUs, the succeeding placement of REs into tasks (with respect to calculated blocking times when the "Priority Ceiling Protocol" is applied) as well as the assignment of data consistency mechanisms to shared data elements. In this way, schedulability, minimized bus load, reduced memory consumption and data consistency are ensured.

As opposed to the approach of this thesis, this method does not challenge the assignment of REs to SW-Cs but deploys them as a whole. The associated case study shows its application on a very small model: six SW-Cs with one RE each and two ECUs. Thus it remains unclear if the process' implementation can also cope with highly intertwined models.

3.2.5 Partition and Sequencing Heuristics

Finally, another interesting approach was presented by Monot et al. [MNBSL12]. Here, all interdependent REs are grouped into a common task which can afterwards be scheduled independently from other tasks. Subsequently, a separate scheduling is performed by a specific sequencing algorithm for each core of a homogeneous target platform (i.e. featuring equally equipped IEUs). The main goal of this proceeding is achieving proper load balancing. As proof of concept, experiments were successfully conducted on complex models, namely on a "body gateway ECU with about 200 runnables [i.e. REs]" from Peugeot [MNBSL12].

However, a basic requirement for this approach to operate is that there exist only little dependencies among REs, so that enough independent tasks can be determined and distributed. This assumption is too optimistic and is definitely not met for the majority of embedded automotive systems, especially those in the powertrain domain (cf. case study in Section 5.3).

3.3 Comprehensive Approaches

Compared with the methods outlined in Sections 3.1 and 3.2, the following approaches exhibit a broader applicability, a higher degree of maturity or cover more common aspects with respect to the approach of this thesis.

3.3.1 Hierarchical Task Graphs and Cost Models

The dissertation of Daniel Cordes directly addresses automating the parallelization of embedded multi-core systems [Cor13]. Therefor, the fundamental idea of splitting the intricate parallelization problem into efficiently solvable subproblems is applied.

For this purpose, existing research findings on "Hierarchical Task Graphs" (HTG) have been continued and further evolved. HTGs follow the source code's structure and are used as "central intermediate representation" which embody different (separately processed) levels of hierarchy that mainly include control flow and dataflow together with corresponding performance metrics. Hence, the emerging complexity is tackled via this "divide-and-conquer fashion" to shrink the vast solution space.

Depending on the type of target platform (homogeneous or heterogeneous) as well as the number of optimization goals, "Integer Linear Programming" (ILP – the underlying technique of the previously broached MILP) and GAs are employed as parallelization techniques.

In further consequence, the obtained outcome is utilized to create "High-Level Cost Models" (HLCM) reflecting task information like execution times and communication costs. They are used to properly estimate the "benefit of different parallel solution candidates" which again supports, e.g., proper load balancing. Among these, the best solution is used to annotate the source code and to eventually implement parallelism.

Embedded within the encompassing parallelization framework, the approach aims at automatically conducting partitioning and mapping on embedded applications for heterogeneous platforms. It is suitable for – on the one hand – single-objective parallelization realized via ILP as well as – on the other hand – for multi-objective parallelization based on GAs, while they are employed to extract both task-level and pipeline parallelism.

The main difference compared with the methodology presented in Chapter 4 is that the analysis is carried out on sequential C code instead of model-based. Furthermore, the framework uses a profile-driven instead of a static dependency analysis (like in Section 4.2) due to considerably reduced development effort. The approach is illustrated in Figure 3.1.

3.3.2 AMALTHEA and Critical Path

In [HKI15], the authors deal with partitioning and mapping for embedded multi-core systems in the context of the "AMALTHEA Tool Platform", which is an "open source tool platform for engineering embedded multi- and many-core software systems" [Mac15]. In the meantime, "AMALTHEA" has become an official Eclipse project and was renamed to "APP4MC"[15].

AMALTHEA's basis are "Weighted Directed Acyclic Graphs" (WDAG), which are created with the help of a dependency analysis. This is followed by a cycle elimination process that takes heed of possibly existing ordering constraints, so that expected timing, safety and behavior characteristics are retained [The16]. The actual partitioning is performed following "Earliest Deadline First" (EDF), "Earliest Start Scheduling" (ESS) or "Critical Path" (CP) strategies.

According to several experiments, the CP partitioning approach seems to be the most promising heuristic: Here, an application model is partitioned according to its crucial sequence – the Critical Path – which "is defined by runnables and dependencies, forming a path from an entry node to an exit node, of which the sum of computation and communication costs is the maximum" [HKI15]. The Critical Path acts as invariable core that is not distributed in order to avoid additional communication or data exchange [HKI15]. Based on that, it is determined where – i.e. at which scheduling position – the remaining "runnables" (equals to AUTOSAR's REs) can be properly placed without violating constraints or causing overhead [APP18, HKI15].

Eventually, this proceeding concentrates on reducing overall response times [The16]. As opposed to the ESS algorithm, CP partitioning is not committed to produce a result with a preconfigured number of partitions [APP18]. Therefore, ESS seems to be expedient if the target hardware is known in advance.

[15] `https://www.eclipse.org/app4mc/`

After the partitioning step, the mapping is done based on established ILP methods and yields optimal solutions with respect to specific goals.

The approach is illustrated in experiments on several models and for different target platforms. Among them is also a complex model "derived from a real automotive engine control system" [HKI15], which consists of over 1200 runnables and is partitioned into 54 tasks in total. For this model, the best parallelization results in terms of execution time and reached degree of parallelism were achieved with CP partitioning.

This proceeding bears analogy to the approach of this thesis as a preceding dependency analysis serves as basis for properly preparing the model for parallelization. However, invariably imposing only sequencing constraints to "solve" cycles rather appears as "steamroller approach". In addition, the method confines itself to preserve the legacy systems original behavior (data validation) and does – apparently – not conduct a verification concerning the model integrity. It remains unclear if the approach scales well as concrete data on the case study models (like the degree of interconnection) is lacking and there is also no information given on the algorithms' runtime. Another difference is that the method in Chapter 4 does not draw on strongly simplifying heuristics like CP partitioning.

3.3.3 RunPar and Timed Implicit Communication

The authors of [PKQ$^+$14, KPQ$^+$16, KQBS15, BKL16] follow the principle of retaining a single-core application's original RE-to-task mapping in order to avoid assumed high effort for re-validating functional correctness.

With regard to the widespread division of migration methods into "reconfiguring" (changing mappings) and "preserving" (keeping mappings) parallelization approaches (e.g. in [KBL17]) they argue that the former are "ideal for simple applications without strict dataflow constraints, e.g. body control" [BKL16] as opposed to the latter which are more suitable for complex models.

Accordingly, a comparison between the benefits of RE-level versus task-level parallelism claims that the former is more suitable for small task sets and few IEUs while the latter is preferable for many tasks and IEUs [BKL16].

On RE-level, they first introduced "RunPar", which is an allocation algorithm that assigns single REs as tasks to IEUs while "keeping the sequential execution of tasks" [PKQ$^+$14]. Thus, RunPar prevents parallel execution of REs assigned to different tasks and ensures the same functional behavior as before migration. RunPar is intended to reduce the "Worst Case Execution Times" (WCETs) of tasks instead of an application's overall response time, so it draws on strategies like to firstly allocate the "longest chain of dependent runnables" [PKQ$^+$14].

Ongoing research led to a complementary task interleaving concept enabling the efficient utilization of occurring idle intervals (cf. [KPQ$^+$16]). This is reached by "combining consecutive [sic] executed tasks into one Supertask" – a concept based on "Logical Execution Time" (LET – cf. Subsections 6.3.5 and 6.3.6) [BKL16, KPQ$^+$16].

On task-level, "Timed Implicit Communication" (TIC) is presented, which is "a communication mechanism that transforms the data flow of the original task set" [KQBS15], so that the parallel execution of communicating tasks is enabled by decoupling them. It maintains the original dataflow within the migrated multi-core version of the applications though causing additional costs by having to instrument the tasks' communication with timestamps.

The applicability is illustrated in [PKQ$^+$14] by means of reaching a WCET speedup of approximately 1.4 for an EMS (12 tasks, 1000 REs) on a homogeneous target platform. In addition, TIC is evaluated with a diesel EMS attaining WCET speedups of 2.7 (on 4 cores) and 4.5 (on 8 cores) [KQBS15].

As contrasted with the approach from this thesis, the authors assume that an initial RE-to-task mapping is, firstly, always given and, secondly, should not be altered due to enormous impending re-validation effort [PKQ$^+$14, KPQ$^+$16, KQBS15]. As a consequence, RE-level parallelism is assumed as only being suitable for simple models (small task sets and few IEUs). Both conflicts with the approach presented in Chapter 4, where RE-level parallelism is consequently exploited irrespective of a model's complexity and predefined (single-EU) mappings are always challenged.

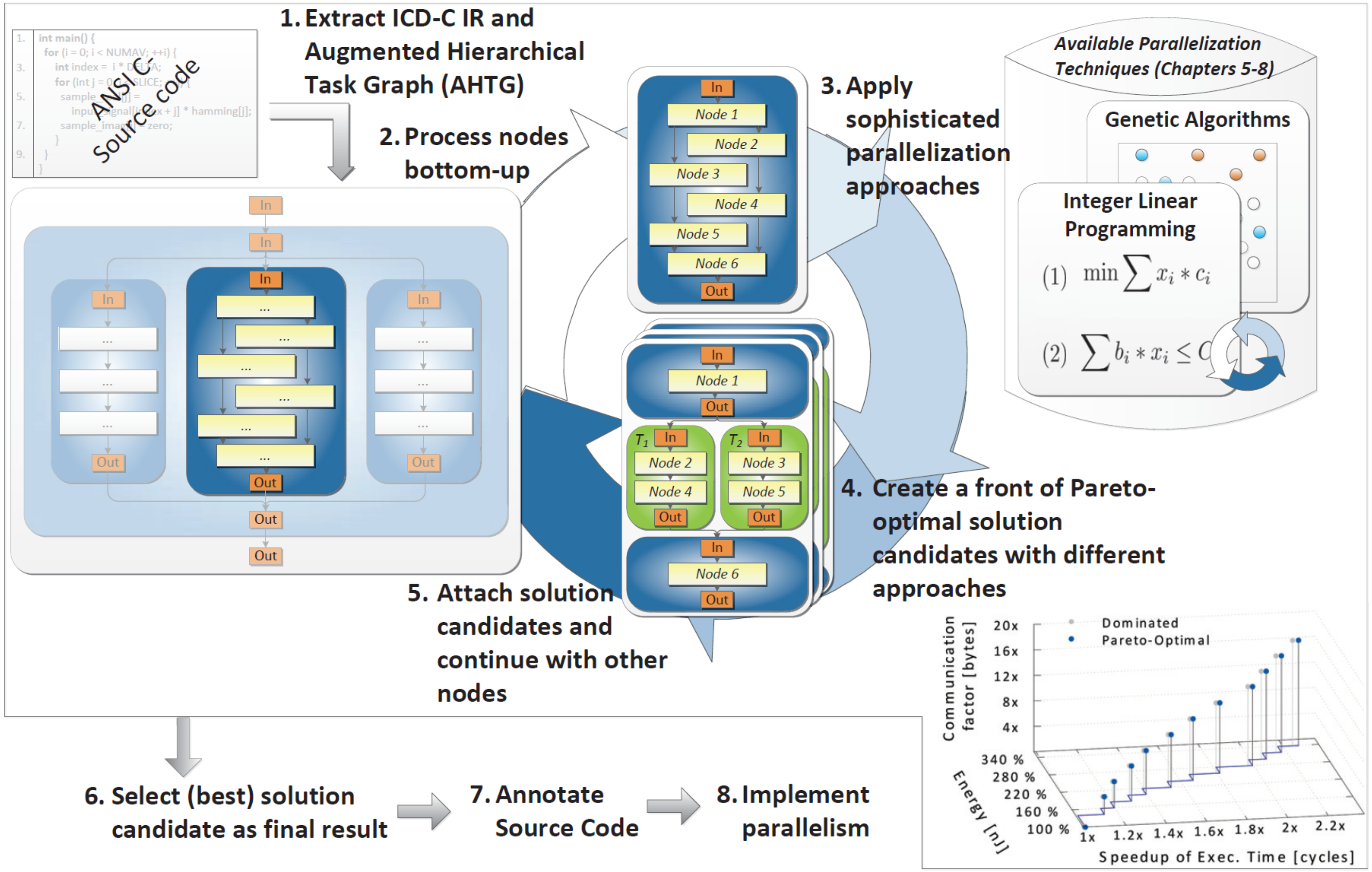

ANSI C-Source code
1. Extract ICD-C IR and Augmented Hierarchical Task Graph (AHTG)
2. Process nodes bottom-up
3. Apply sophisticated parallelization approaches
Available Parallelization Techniques (Chapters 5-8)
Genetic Algorithms
Integer Linear Programming
(1) $\min \sum x_i * c_i$
(2) $\sum b_i * x_i \leq C$
4. Create a front of Pareto-optimal solution candidates with different approaches
5. Attach solution candidates and continue with other nodes
6. Select (best) solution candidate as final result
7. Annotate Source Code
8. Implement parallelism
In
Out
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
T_1
T_2
Dominated
Pareto-Optimal
Communication factor [bytes]
20x
16x
12x
8x
4x
Energy [nJ]
340 %
280 %
220 %
160 %
100 %
1x
1.2x
1.4x
1.6x
1.8x
2x
2.2x
Speedup of Exec. Time [cycles]

Figure 3.1: Overview of the HTG parallelization approach (from [Cor13])

3.4 Overview and Comparison

The following Table 3.1 summarizes the comprehensive approaches from Section 3.3 and compares them to the one presented in Chapter 4.

The approaches are arranged vertically (columns), whereas the horizontal rows delineate their compared properties:

- Working Basis: On which level of abstraction does the approach work?
- Primary Scope: Which type of input data is preferably processed?
- Dependency Analysis: Is the input analyzed in order to support the parallelization process?
- Verification: Does the approach provide methods for checking the integrity of the processed input (model or code)?
- Data Validation: Does the approach provide methods for checking and validating the data consistency (coherence and stability) of the processed input?
- Partitioning
  - Principle(s): Which methods and principles are employed for the partitioning process?
  - Basic Strategy: Is the original execution order and/or task assignment preserved or does the approach create a new one without restrictions?
  - Level (Granularity): Which is the atomic, schedulable unit that the approach processes and creates a partitioning and mapping solution for?
- Mapping: Does the approach provide generic and/or target-specific task allocation?
- Tool Support: Is the approach implemented/realized as comprehensive tool suite?
- Scalability: Is the approach capable of processing large and/or complex input with an appropriate amount of time and resources?

Table 3.1: Overview and comparison of comprehensive parallelization approaches

	Hierarchical Task Graphs & HLCM	**AMALTHEA & Critical Path**	**RunPar & Timed Implicit Comm.**	**Approach of this PhD Thesis**
Working Basis	code	models	models	models
Primary Scope	ANSI C	AMALTHEA	AUTOSAR	AUTOSAR
Dependency Analysis	✓ yes (profile)	✓ yes (static)	✗ no	✓ yes (static)
Verification	✗ no	✓ yes	✗ no	✓ yes
Data Validation	✗ no	✓ yes	✗ no	✓ yes
Partitioning				
Principle(s)	ILP & GA	CP & ESS/EDF	TIC & LET	MLP derivative
Basic Strategy	indeterminate[a]	reconfiguring	preserving	reconfiguring
Level (Granularity)	task & pipeline	function (REs)	task[b]	function (REs)
Mapping	✓ yes, generic & target-specific	✓ yes, generic & target-specific	✓ yes, generic & target-specific	✓ yes, generic & target-specific
Tool Support	✗ unclear	✓ full	✗ unclear	✓ full
Scalability	✗ limited[c]	✓ high	✓ presumably high[d]	✓ high

[a]Considers only "from scratch" migration scenarios as no task assignment is initially given.
[b]Only simple models are processed at function level.
[c]Due to NP-hardness, the scalability of ILP approaches is distinctly limited [WMM+13, DK08, McD07].
[d]As long as CP is employed.

4 Approach

This chapter constitutes the core part of this thesis as it elaborates the parallelization approach and the corresponding methodology.

Firstly, underlying principles of the approach are depicted and an outline of the working steps is given (Section 4.1). Section 4.2 then describes the conducted data dependency analysis and the associated verification and validation process by reference to AUTOSAR. On the basis of a now achieved multiple-IEU robustness, Section 4.3 expounds how to efficiently perform a partitioning and mapping on complex embedded automotive software. Finally, Section 4.4 shows relevant tasks to properly simulate determined solutions, to evaluate obtained results and to carry out an expedient optimization.

4.1 Principles & General Approach

Most existing parallelization approaches are geared towards high-performance applications or classical desktop computing and are therefore hardly applicable for embedded devices [Cor13]. As opposed to this, the approach of this thesis is intended to particularly address resource-restricted embedded systems.

Having this in mind, considering the basics imparted in Chapter 2 and coping with the challenges depicted in Section 1.2, it is now possible to describe general thoughts as well as specific proceeding steps derived from them in order to satisfy the central need initially addressed in Chapter 1: specifying a process to migrate from existing ECU software to a version that is capable of handling multiple IEUs.

As a result, the core task is to "supply" the software with a consistent timing model, i.e., providing it with a minimum of timing information and constraints to ensure the preservation of its original functionality.

4.1.1 Principles

In order to achieve this, the approach builds on two principles – namely to proceed incrementally and bottom-up. Details are given in the following.

4.1.1.1 Acting Step-by-Step

Like already broached, a variety of possibilities (the design space) emerges when multiple IEUs are taken into consideration. As it is well-founded that "the development process of today's embedded systems is characterized by a sequence of refinement steps" [EZV$^+$17], it rather suggests itself to advance incrementally instead of trying to care about all alternatives at once.

This means that – after having identified critical spots – the process of eradicating "weaknesses" is carried out in a consecutive way by adding new or modifying existing timing constraints, whereas the order of the single steps should be preferably commutable. The latter implies that the result set (remaining proper sequences) is not affected by the specific succession of manipulation steps.

As it cannot be enforced to obtain analysis results without intersecting "conflict parts", there will always be a risk of not choosing an optimal path, i.e., a certain chance to unintentionally exclude promising solutions from the set of possible execution sequences. In order to reduce this risk, it is expedient to have recourse to suitable conflict resolution experiences ("best practices") and to tools that facilitate going through different solution procedures. However, experiences and tools only serve as basis and are far away from being able to ensure that an acceptable (or even optimal) solution is found for every setting.

By and large, this incremental approach is clearly more feasible than trying to find a proper solution off the cuff, because the latter almost inevitably leads to "stupidly" calculating all possibilities which mostly lacks foresight and systematic procedure. Furthermore, checking out every possible option usually takes more time and requires disproportionate computing power.

As already covered in Subsection 2.2.2, the actual decisive criterion is to stop when all detected conflicts are solved. Afterwards, a search within the remaining alternatives should be carried out before additionally (and maybe both needlessly and adversely) constraining the system. This boils down to gradually approximating the ideal case-related trade-off point (the "golden mean"), which follows established iterative improvement heuristics employed by many sophisticated solvers, e.g., in the field of graph partitioning [BMS+16].

4.1.1.2 Tackling From the Bottom

Assuming that time and money are not main restrictive factors, the ideal case would being able to build multiple-IEU software from scratch with using existing solutions only as reference. Unfortunately, this is hardly ever the case and ECU software is – being confronted with competitive pressure – rather constantly refined and extended than newly created. Exceptions are only the development of unprecedented functions or advanced research projects in a company (where the tool development never catches up) [Stefan Kuntz, personal communication, 2015].

Therefore, a top-down development process that supports abstraction from low-level data, skillful further decomposition as well as "early" feedback on system design is hardly applicable here.

As opposed to this, the approach starts from the other side by analyzing existing software on the most fine-grained level and by delivering feedback on given structures. The results can be re-used on "adjacent" levels to gain a more abstract view on the system. For example, knowing about the data-flow between REs in AUTOSAR allows to re-interpret and apply the results on the SW-Cs that the REs are assigned to.

This leads to a generalized and – especially for very huge models (cf. Figure 2.5) – more overseeable view on the system, because communication within SW-Cs is hidden. If the SW-Cs are not intended to be split up and being run on different IEUs, such an abstraction can also help to quickly find an appropriate partitioning. On the contrary, only few bulky SW-Cs may be inappropriate for being mapped on an IEU. Working and distributing on the most fine-grained level seems sensible to enable maximum flexibility and efficiency [Här16, Ste16].

Figure 4.1 shows the WSS from an SW-C view. Looking at it from a more coarse-grained perspective facilitates to gain an overview fast. Here, "Runnable Entity Instances"[16] (REIs) and the data accesses between them via "Inter-Runnable Variables" (IRVs) are hidden and only communication taking place between SW-Cs (via ports) is shown.

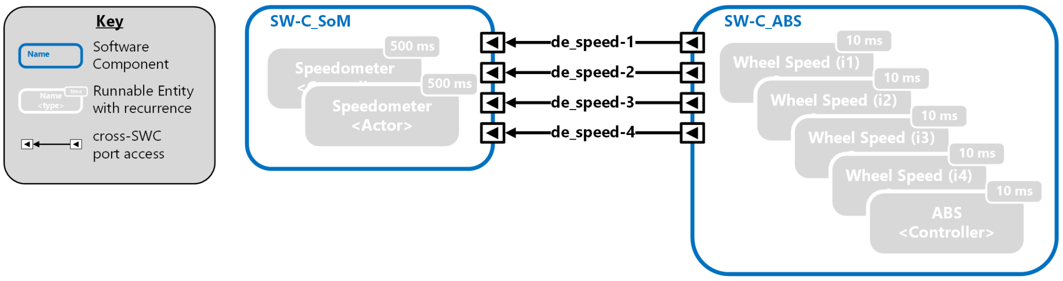


Figure 4.1: The WSS from SW-C perspective (without IRVs)

Generally speaking, the process of abstracting from a given to a higher (i.e. more coarse-grained) level of abstraction can iteratively be carried on as desired. The only restriction is the initial analysis' range that can vary strongly from analyzing the data dependencies within, e.g., one ECU's SW-C to those in a self-contained system spanning several ECUs. In the end, it is possible to give feedback on the whole system structure by interpreting "base level" information.

[16]This is not an official AUTOSAR element. The existence of REIs arises implicitly from the component structure where the same RE may appear in different contexts.

4.1.2 Overview

In order to mitigate the complexity of parallelized systems, the endeavor is to effectively support the goal-oriented migration of legacy ECU software to a suitable multi-core architecture.

In the course of this, the working basis is not code (but models) and the intention is not enabling the parallel execution of several "atomic applications". Instead, the focus lies on the model-based parallelization of a single application addressing "function-level parallelism" (also known as "task parallelism") – like previously covered in Subsection 1.1.3 (level of parallelism) and argued for in Subsection 1.1.6 (models).

The proposed migration process is depicted in Figure 4.2.

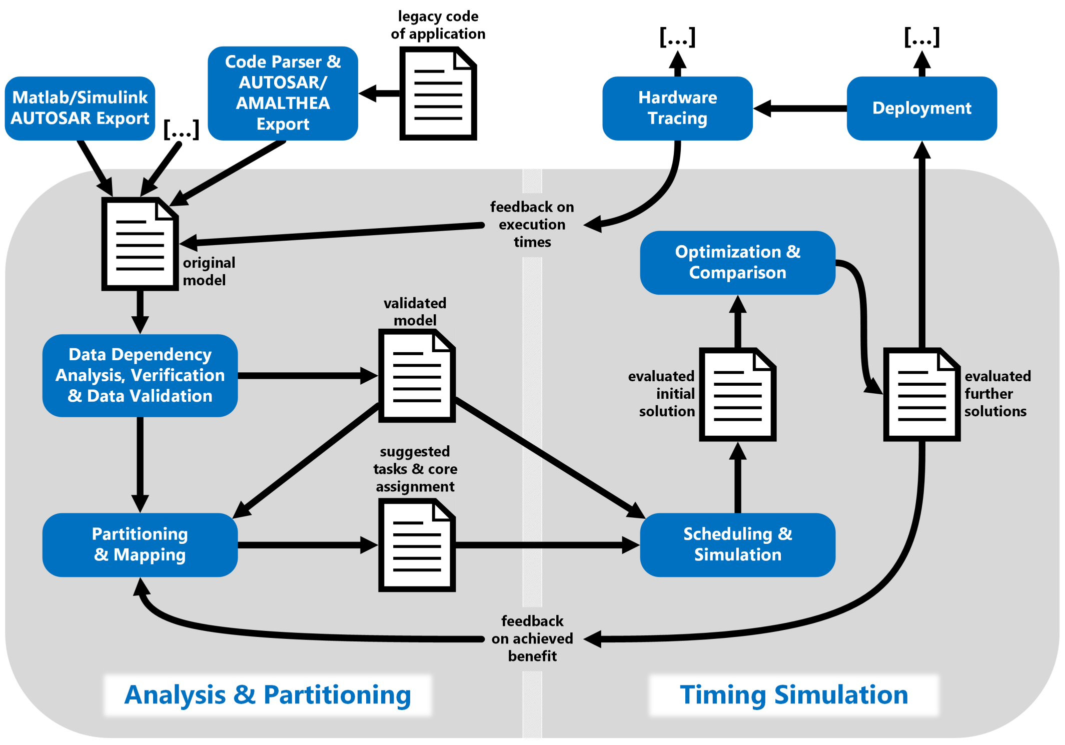

legacy code of application
[...]
[...]
Code Parser & AUTOSAR/ AMALTHEA Export
Matlab/Simulink AUTOSAR Export
[...]
Hardware Tracing
Deployment
feedback on execution times
original model
Optimization & Comparison
validated model
Data Dependency Analysis, Verification & Data Validation
evaluated initial solution
evaluated further solutions
suggested tasks & core assignment
Partitioning & Mapping
Scheduling & Simulation
feedback on achieved benefit
Analysis & Partitioning
Timing Simulation

Figure 4.2: Overview of the migration process

The gray box indicates the scope of work described in this thesis. Both activities on its left side are supported by the tool "AutoAnalyze" (cf. Section 5.1), which is implemented as plug-in based on the "Eclipse Modeling Framework" and the "Model Analysis Framework" within the "AUTOSAR

Tool Platform"[17] (Artop) [Ecl09,Saa09,SB13]. The two activities on the right side of the gray box can be carried out within several third-party tools that provide simulation and optimization features for embedded multi-core systems. The "TA Tool Suite" (TATS) is used for the case studies in Sections 5.2 and 5.3 [Tim15].

In general, there is indispensable information for each specific working step which is provided in Table 4.1. For each subsequent step, only the additionally required (mandatory or optional) information is listed. The last column of the table has been adapted from the work of Sailer et al. [SSH$^+$16] and indicates whether the data exchange formats AUTOSAR, AMALTHEA or ASAM MDX[18] can be used to store and exchange the respective information.

[17] Artop facilitates the construction of AUTOSAR tools by serving as Eclipse infrastructure and virtually acting as "persistence layer" that enables common base functionality like easy access on AUTOSAR models that adhere to specific meta-model versions [Art12].

[18] ASAM MDX stands for "Association for Standardisation of Automation and Measuring Systems – Meta Data Exchange Format" and "defines a description format for the software architecture and data definition of the software of an automotive ECU" [Ass12].

Table 4.1: Lineup of required information for the specific working steps

	Mandatory	**Optional**	**Exchange format**
Data Dependency Analysis, Verification & Data Validation	• shared data definitions • REs including data accesses • recurrences of REs	• SW-Cs and compositions • timing constraints	• AUTOSAR • AMALTHEA • ASAM MDX
Partitioning & Mapping	• basic hardware model (i.e. number of cores)	• basic operating system model (e.g. scheduling algorithm)	• AUTOSAR • AMALTHEA
Scheduling & Simulation	• definitions of tasks and Interrupt Service Routines (ISRs) with RE call sequence • basic OS model • OS configuration (e.g. task and ISR priorities and preemptability) • timing requirements (i.e. task deadlines)	• precise hardware and OS model • precise RE runtimes (i.e. runtime distribution) • precise task/ISR activation pattern (e.g. jitter and sporadic activations) • precise task call graphs (i.e. branches)	• AUTOSAR (only mandatory information) • AMALTHEA
Optimization & Comparison		• Affinity Constraints (e.g. RE pairing/separation)	• AMALTHEA

4.1.2.1 Data Dependency Analysis, Verification & Data Validation

The basis of the approach is a preceding data dependency analysis run on AUTOSAR models, e.g., like proposed in [SBR10]. It is intended to detect, visualize and solve potential conflicts related to a software's distributed execution on multiple IEUs:

1. The analysis identifies a model's structural elements such as the aforementioned REs, their variable accesses, their recurrence (also called "triggering frequency" or "period"), their specific execution time, the SW-Cs containing the REs as well as already imposed timing constraints.
2. The gathered information is assessed, the integrity and soundness of specified elements are detected (verification) and potential conflicts regarding data consistency are determined.
3. Integrity and soundness issues are handled by hinting at missing or incorrectly specified elements.
4. Inconsistencies are addressed by an incremental (stepwise) application or modification of timing constraints on the lowest level (i.e. on REs) for the purpose of achieving multiple-IEU robustness in terms of data consistency (data validation).

This semi-automatic process provides the software with a consistent timing model to ensure the preservation of its original sequential (single-EU) behavior on a multi-core platform. Hereby, data consistency can be guaranteed regardless of how the software parts are eventually distributed.

Typical synchronization mechanisms and problems which commonly come along when trying to achieve similar results on code level are rendered redundant by the multiple-IEU robustness achieved with this model-level approach. Accordingly, the author of [Moy13] emphasizes the associated huge overhead by stating that "fine-grained synchronization [...] can be 1000 times more costly than the cost of the operation". This is due to a tremendous effort necessary for handling specific challenges like (cf. [GS12, Moy13]):

- avoiding deadlocks and livelocks, e.g., by detecting cyclic dependencies via code instrumentation,
- avoiding non-determinism, e.g., by preventing calculation results to depend on time sequences,

- avoiding data races, e.g., by controlling simultaneous access on shared variables,
- providing fairness, e.g., by realizing mutual exclusion mechanisms with queues or
- ensuring scalability, e.g., by eliminating synchronization operations and keeping critical sections fine-granular.

Taken as a whole, an elaborate conflict detection (both statically and dynamically) as well as achieving and keeping an efficient program flow constitute considerable additional expense.

4.1.2.2 Partitioning & Mapping

Partitioning and mapping are the core tasks of parallelization. The former is – like explained in Subsection 1.2.3 – considered to be the key challenge of migration to multi-core platforms. However, this does not mean that the latter is simple by implication.

Mapping is a similarly demanding task as finding an appropriate allocation of software to hardware is guided (and restricted) by often adverse circumstances. For embedded devices, limited resources, hard deadlines, heterogeneous target platforms and difficult to predict communication behavior of the underlying network (due to concurrent access on shared resources) do considerably complicate the software parts' distribution [KWF15].

Considering the workflow involved by these two tasks, it is apparent that "facilitating these processes among an automatic, scalable and modular basis becomes an important issue" [HKI15]. This statement is well-founded as there is a general consensus that conducting partitioning and mapping for vehicles systems by hand has become infeasible, because "it is inefficient and error-prone to perform the mapping manually in a trial-and-error manner" [ZG11]. One reason is that manual methods often draw on simple partitioning strategies such as searching for entirely independent tasks or trying to group all dependent (or all uniformly triggered) REs into one task, which is not expedient for real-time systems, especially for a strongly interconnected application like an EMS [PKQ+14, WMM+13, NNB10].

As a consequence, this step's fundamental idea is to search on AUTOSAR's most-fine grained level (REs) for regions (sets of REs) with a relatively low coupling and to group them into tasks[19] in order to create a suitable partition as well as the subsequent task-to-core mapping. The immense search space can be remarkably reduced by providing a beneficial starting point for the simulation and optimization that are carried out to evaluate the initial solution and to search for further ones. This approach is based on techniques introduced in [JPP94, OO84, Tip95], which were further developed in [GRLB09].

Since its initial proposition (in [KMKB14]), the partitioning algorithm was extended to cope with highly complex models:

- Search Tolerance: A configurable tolerance is used to loosen the rather strict criteria for low-coupled regions, so that RE sets, which violate the demands to a certain tolerable extent, are not discarded.
- Dependency Weights: The relevance of the connection between two REs is calculated in order to determine concrete pair-wise weights. The computed values do in general depend on the number and type of variable accesses between two REs, their recurrences as well as the number and type of timing constraints imposed on them.
- Splitting Strategies: Based on experience, analyzing a whole model often unnecessarily increases the search effort while simultaneously shrinking the result set. This effect can be attributed to the high degree of interconnection in large models which exacerbates partitioning attempts [KBL17]. Thus, different splitting strategies are provided that follow best practices and proven approaches to handle such problems efficiently (like in [BMS$^+$16]), e.g., dividing the model into parts with uniform recurrences before running searches on each of them.
- Relevance Partitioning: As these parts often remain highly complex, the search gradually ignores pair-wise dependencies below a growing threshold until a sufficient number of regions is found. This partitioning style roughly corresponds to the Multi-Level Partitioning approach of first "coarsening", then clustering and afterwards restoring a graph [ACU08].
- Automated Search: The applied search parameters – namely the mentioned tolerance, the partitioning rate target ("coverage") as well as according step sizes (the "search granularity") – can be automatically determined. In this case, the algorithm dynamically adjusts the search

[19]Forming tasks is inevitable as the REs usually significantly outnumber the tasks allowed by the OS [MNBSL12].

parameters to obtain advantageous results according to the currently analyzed model part's properties.

The succeeding mapping algorithm is highly adjustable to meet the specific hardware platform's requirements (e.g. given by its number of IEUs). As demonstrated in the case studies (cf. Section 5.2 and Section 5.3), achieving a certain goal is possible by modifying the preferences when arranging the tasks and mapping them to a specific number of IEUs, e.g., the preferred granularity of the regions, a concrete load balancing strategy or enforcing certain regions to be assigned to the same IEU (via "pairing constraints"). Furthermore, the mapping can be tailored to meet the features of heterogeneous target platforms by independently defining a variety of properties for involved IEUs and associated local or global memories.

However, the quality of a calculated partition and an according mapping can hardly be objectively assessed. This is because optima are typically either not recognizable as such or are simply unknown and goals for parallelization efforts are often contradictory. For example, pooling strongly-connected software parts (low synchronization overhead), distinctly separating functionalities that should not interfere (support safety) and evenly spreading computational cost (proper load balancing) are usually not simultaneously achievable. Thus, obtained solutions mostly constitute a trade-off between a few personally prioritized goals.

Like previously mentioned, a pooling-oriented partition together with a mapping that enables satisfactory load balancing is sought. Of course, there are numerous other factors worth taking into consideration, e.g., if the target platform's processor is "homogeneous" concerning equally equipped IEUs and how many of them are available.

In the end, this step's purpose is to provide an advantageous starting point ("initial solution") which is effectively supporting the following simulation and optimization in order to find appropriate further solutions in an adequate amount of time.

4.1.2.3 Scheduling & Simulation

The subsequent step employs a third-party simulation tool in order to evaluate the usefulness of the provided initial solution using various timing and performance metrics, e.g., task response times. Regardless of what product is employed, some preparations have to be made before such a simulation can deliver expressive results:

1. Importing the validated AUTOSAR model together with the calculated partition and mapping from the previous working step (initial solution).
2. Setting up the underlying hardware model, e.g., a generic processor model or a detailed automotive micro-controller simulation model.
3. Setting up the underlying OS model, e.g., how the cores are managed by the operation system (global/local scheduling, online/offline scheduling and the scheduling algorithm).
4. Adding basic timing requirements (e.g. task deadlines) to later allow a general classification into valid and invalid solutions.

Running a timing simulation on highly complex models can take a lot of time, but eventually yields informative findings about a solution's general validity, occurring latencies, overhead caused by necessary synchronization or a basic statement about a software's overall degree of potential parallelization (indicating possible speed-up).

Optimizations for practical models typically require the simulation of several tens of thousands alternative solutions. As a consequence, the required time for optimization is mainly dictated by the simulation performance.

Generally speaking, it can be stated that a targeted preceding partitioning and mapping facilitates the optimization step considerably which is particularly important when being confronted with cross-core communication as a substantial new resource bottleneck [Mac15]. In addition, the obtained results are expedient for both static as well as dynamic scheduling approaches. The latter are becoming increasingly important in the automotive sector (cf. Subsection 6.3.6).

4.1.2.4 Optimization & Comparison

Having gained a first glimpse on the initial solution's quality enables a comparison with potential alternatives. The latter's creation is facilitated by taking the initial solution as basis to derive modified versions of it. In general, there are plenty possibilities to do so either manually or with tooling support. In many cases, the above-mentioned third-party simulation tools also provide optimization features.

The aforementioned initial solution serves as starting point for further improvements by the optimization step. Alternative solutions are hereby created by systematically modifying the initial solution in an iterative process. In order to steer the optimization process, each alternative solution is simulated to evaluate the improvement compared to the initial solution with respect to timing and performance criteria. This optimization process can be carried out either manually or with tooling support.

A focus on "weak spots" discovered by means of interpreting the simulation results can act as beneficial leverage point for model changes, e.g., splitting up "chunky" tasks that hamper proper load balancing or enforcing two heavily communicating tasks to be mapped to the same core.

Having found better solutions in terms of certain characteristics – like reduced cross-core communication or a shorter overall cycle time – allows to "close" the round-trip engineering circle by providing customized data on advantageous model-specific search parameters for the "Partitioning & Mapping" step. This iterative proceeding seems sensible in order to effectively broaden the search span by means of a fresh "solution seed".

4.2 Data Dependency Analysis, Verification & Data Validation

In the following, the first step of the migration process (cf. Figure 4.2) is described in detail.

The underlying methods are constituted of static "Verification and Validation" (V&V) techniques. As opposed to a dynamic proceeding, neither the execution of the model nor conducting a simulation is needed to obtain expedient data for the succeeding steps [The15]. This approach has proven to be advantageous with regard to efficiency especially when being confronted with increasingly huge and complex models. In general, static V&V techniques collect information on a model's syntax, structure as well as its data and control flow which subsequently enables the checking of its integrity, soundness and consistency [The15].

4.2.1 General Approach

Like previously stated, the central task is to determine a timing model, so that an embedded application's original behavior is retained. In this context, conflicts can occur when functional blocks – originating from legacy (single-EU) software – are processed in parallel instead of well-matched and rigidly consecutive like before. This poses a threat to data consistency, which was previously defined as stability (steady signals/values over a certain period of time) being paralleled by coherency (signals/values with uniform data age).

Possible conflicts are, e.g., data not being available in time or data being read inconsistently. Of course, conflicts like this can occur on single-EU platforms too, but multi-core systems are more prone to "evoke" them because here it is – due to concurrency – significantly harder to maintain consistency.

Therefore, it is inevitable to address any potentially unintentional behavior within a system. The challenge consists of the already mentioned complexity rise caused by the exponentially growing number of possibilities to distribute tasks on cores, which leads – together with scheduling – to a tremendous amount of possible execution sequences including many adverse (i.e. conflicting) ones.

To assure that software will work properly irrespective of a certain task-to-core mapping, it is required to distinctly exclude all unintended behavior, which is accomplished by adding, modifying or removing timing constraints in the model.

4.2.2 Principles & Steps

As implied in Section 4.1, the approach as a whole is based on the following principles:

- Bottom-up: Analysis and validation are both conducted on the most detailed (lowest) level, which corresponds to REs in AUTOSAR. A top-down approach would be barely applicable in this case as ECU software is – due to competitive pressure – rather continuously revised and augmented than created "from scratch". Additionally, more general views on a system can nevertheless be built by deriving from collected low-level data.
- Incremental: Validation is run stepwise instead of all at once in order to prevent the creation of "fresh" conflicts by overlapping constraint scopes and to avoid indiscriminately calculating all possibilities at the very start which would require disproportionate computing power (and much more time). This corresponds to the step-by-step proceeding of a typical development chain, where receiving feedback after each step is more beneficial than mindlessly going through all possibilities. As this process simplifies the recognition of ways leading to proper solutions, it is presumably the most appropriate and sensible way to overcome wide-ranging solution spaces.
- Minimal: For the purpose of not needlessly restricting the degree of freedom for subsequent steps (and thus not to unintentionally exclude promising solutions), only a minimum set of constraints is imposed.

Complying with this concept, a static data dependency analysis is performed directly on AUTOSAR models. Its basic steps are:

1. Parsing the model and analyzing existing connections (data dependencies) between the Runnable Entity Instances included in the SW-Cs and their potential execution sequences based on already given "Execution Order Constraints" (EOCs).
2. Drawing attention to either missing elements (or mandatory properties of them) as well as those that do not conform to a specification,

e.g., AUTOSAR (verification).

3. Classifying detected connections and filtering out potentially conflicting variable accesses among them.
4. Imposing or modifying timing constraints to reduce all possible execution sequences to a set that timely supplies every REI with its required input data (data validation).

4.2.3 Data Dependency Analysis

First of all, parsing the AUTOSAR model provides the information basis needed for further steps: AUTOSAR's SW-Cs are the main structural elements as their "Internal Behavior" comprises the contained REs, the communication taking place within the component (between several REs) and between different SW-Cs of one ECU. Each RE may be instantiated multiple times (e.g. four wheel speed sensors of a car), so every REI has its own data dependencies, which each arise from the interaction between at least two REIs. AUTOSAR differentiates seven kinds of variable accesses used for "local" access as well as for communication that crosses SW-C borders. All of them are taken into account:

- writing ("outgoing") VAs:
  - `dataSendPoint`
  - `dataWriteAccess`
  - `writtenLocalVariable`
- reading ("incoming") VAs:
  - `dataReceivePointByValue`
  - `dataReceivePointByArgument`
  - `dataReadAccess`
  - `readLocalVariable`

The last-mentioned VAs (the "local" ones) indicate "intra SW-C access", whereas the others denote communication over SW-C "ports" (SW-Cs' connection points). The former is represented by IRVs that may be accessed by all REIs within one common SW-C. The latter is realized by specifying ports according interfaces and "Assembly/Delegation Connectors" that actually hook up a pair of ports establishing either a asynchronous "sender-receiver" communication or a synchronous "client-server" one.

4.2.4 Timing Constraints

Besides those structural elements, AUTOSAR's timing constraints are identified: Currently, two out of the seven existing constraints are deployed: EOCs and "Age Constraints" (ACs) [AUT14d, Flä15]. The former are "used to specify the order of execution of ExecutableEntities" [AUT14d] (i.e. specify a fixed order for multiple REs) and the latter "to specify a minimum and maximum age that is tolerated when a variable data prototype is received" [AUT14d] (i.e. determine the tolerated data age of a read variable).

EOCs are applied to predetermine a rigid execution order between two or more REs ("explicit synchronization", cf. [Sch15a]). This is appropriate when they are logically (and semantically) linked like in a classical "sensor-controller-actor system" where sensors transmit measurement data to a controller which determines a suitable action that is afterwards carried out by associated actors, e.g., brakes in an "Anti-lock Braking System" (ABS).

By contrast, ACs resolve potential inconsistencies via tagging a possibly conflicting dependency as unproblematic by allowing that certain accessed data comes from a previous "computing cycle" ("implicit synchronization", cf. [Sch15a]). This is feasible if the reading RE does not imperatively need current data to work properly, e.g., a speedometer that is not able (and is not intended) to react within milliseconds because of the speedometer needle's inertia and the preservation of its readability by the driver.

As opposed to the imposition of timing constraints, synchronization mechanisms do – for example triggered by reaching a barrier or synchronization point – actively align data that is accessed by various program parts. Here, the synchronization definition "restoring of data communication of sequential programs" [Sch15a] fits ideally as it expresses the need for additional mechanisms to ensure data coherency. A drawback of this approach is that mechanisms require detecting where synchronization is actually needed in the first place [Sch15a]. Furthermore, such specific (fine-grained) synchronization is considered to cause much additional effort [Moy13].

This strongly differs from the concept of ACs that show where and to what extent coarse-grained synchronization is feasible. With the approach presented in this thesis, active synchronization is made superfluous by design.

4.2.5 Verification & Data Validation

Before using the data to filter out semantic weaknesses (timing issues), a verification step is carried out in order to ensure that the analyzed model is syntactically complete and sound, i.e., it is build correctly (integrity) and provides all information required with respect to the specification. Here, it is crucial to highlight problems like erroneously referenced REs that prevent a well-founded interpretation of a model's timing behavior. In addition, it is very helpful to hint at ambiguities, e.g., more than one triggering frequency for an RE (a so-called "multi-rate network", cf. Subsection 6.4.4), that are allowed but can still exacerbate the parallelization endeavor when staying concealed.

Having verified the model's integrity now allows to take the gathered information as a whole into consideration and to map them on a directed graph illustrating the data dependencies by means of nodes that represent the REs and edges standing for the variable accesses semantically connecting them.

4.2.5.1 Dependency Classification

A second step harnesses the gathered information, so that the graph can be used to derive sets of node neighborhood, i.e., successor and predecessor relations between REIs, for the access on a specific variable. These sets are useful to make statements about possible execution sequences and to accordingly classify the dependencies in order to find possible inconsistencies, which are represented by every contingency of unintentionally consuming data before producing it in the scope of one computing cycle.

As embedded software in the automotive domain (and ECU software in general) is characterized by running periodic tasks, the given REss' recurrences together with the data on imposed timing constraints can be used to set the execution of the according REIs into a temporal perspective.

This is done by determining whether the dependencies act as "Forward Dependency" (FD) or "Backward Dependency" (BD) with regards to the REIs' execution order. FDs are marked by the fact that in one computing cycle, a specific variable is always first written and afterwards read. As opposed to this, BDs are all dependencies remaining if FDs are not taken into consideration, concerning those data accesses on one variable that are not explicitly ordered, so it can be first read and afterwards written in the scope of one computing cycle.

4.2.5.2 Detecting Conflicts

Generally speaking, no problems occur as long as the producer of data is executed before any consumer. Every BD is considered as potential conflict, because they offer the possibility to later distribute the tasks (containing the REs) to the available IEUs in a way that does not prevent reading a specific variable before writing it, e.g. by parallel executing both accesses on a variable. This can lead to the processing of "old" values, i.e., data from previous computing cycles. In such a case, a constraint preventing this is needed.

Furthermore, it is checked whether existing timing constraints are correctly applied (valid). EOCs should only be set for REs that have the same recurrence, because when dealing with ECU software, static scheduling is still prevalent. Therefore, a once found execution order stays the same. In contrast, EOCs are rather inappropriate for REs with diverging recurrences, because the execution order of them within a computing cycle can change, e.g., when a "consuming RE" is triggered more often than a "producing one". Here, an EOC demanding the producing RE to be executed first (within a computing cycle) can cause extra latency if the consuming RE is triggered earlier than the producing one. In such cases, ACs are the more adequate means.

Besides simply missing constraints, typical fault cases are, e.g., EOCs imposed on REs with divergent recurrences, EOCs that directly or indirectly contradict each other by forming a cycle (the easiest case is "(A before B) && (B before A)") or "insufficient" ACs that allow a smaller data age than effectively arising.

It is important to know that EOCs do heavily restrict the "degree of freedom" for mapping the REs (grouped as tasks) to IEUs. Thus, the potential for parallel execution is greatly reduced when a model is highly order-constrained. Parallelism can even be entirely prevented if the EOCs' combination lead to a single-chain execution order. Therefore, as little EOCs as feasible are imposed, whereby they are preferably set in a local scope (e.g. being valid only within one SW-C). In the case of imposing constraints on dependencies that cross SW-C borders, ACs are an appropriate choice, because their (global) effect is less limiting and they do not per se reduce the number of possible execution orders (which is advantageous for the multiple-IEU use case).

The previously described basic dependency classification of "forward" and "backward" is now reused: Here, FD means that an EOC is imposed on the associated REIs which guarantees that the specific variable is written before read within one computing cycle. If the EOC itself is not cyclic or invalid, the according dependency is thus considered unproblematic. All remaining dependencies are classified as "backward" and represent possibly critical paths of the system's design.

As previously stated, all possible backward dependencies within one computing cycle are inspected and subdivided into "intentional" and "unintentional" ones. The former are characterized by the existence of appropriate ACs explicitly allowing the transferred data to be "outdated" (i.e. coming from a previous computing cycle). If no matching AC exists, the BD is probably unintentional. To sum up, (restricted) permissions for backward execution are represented by ACs whereas EOCs are used for their prevention.

4.2.5.3 Resolution of Conflicts

These reflections lead to the validation proceeding that takes care of every "unhandled" (i.e. backward) dependency. An indispensable prerequisite for it is that each RE is triggered according to one invariant recurrence though multiple ones are possible and even common in some cases (cf. Subsection 6.4.4).

- *Missing constraints* indicate that existing dependencies may form a cycle resulting in a risk to read deprecated data. An AC should be added if the recurrence of the involved REIs differs or if the corresponding REI does not necessarily need the most current data for its calculations, e.g., when it "only" monitors other REs (diagnosis).
  If recurrences do not differ, the choice between EOC and AC depends on a software engineer's intention. As an EOC affects all dependencies between two REs, it should rather be imposed if they are deliberately executed in succession (see example below).
- *Redundant constraints* can either be harmless, e.g., needlessly set ACs for REs with uniform recurrence, or they can lead to ambiguity if set contradictory, e.g., cyclic EOCs. In the latter case, the software engineer has to choose which one best matches the envisaged functionality and can then remove the other one.

- *Invalid constraints* denote either EOCs imposed on REIs with diverging recurrences or insufficient ACs. The obvious solution for the former is to replace the EOC with an AC if the dependency needs to be constrained. In the latter case, the allowed maximum data age of the AC has to be increased.

If there are multiple ways to resolve a potential conflict, it is not always obvious which alternative is preferable. In the case of a missing constraint where imposing both an EOC and/or AC is possible, there is no universally better way, so it is necessary to have further knowledge about the system in order to make a well justified decision.

For example, a typical sensor-controller-actor-system like an ABS reads the rotating speed from the wheel sensors before it processes if actions to adjust the wheelspin are necessary (like in the WSS). In this context, knowing the system's functionality and the importance of the deployed data's age helps to decide if it is sensible to enforce the REs being executed in succession or, e.g., to explicitly allow the use of "old" data for wheel speeds below a certain threshold.

However, ACs and EOCs are not mutually exclusive, but rather can be very expedient in combination (depending on the specific situation). This is in particular the case if a variable is used for different purposes, e.g., the value of a current wheel speed is frequently read by an ABS but only seldom by the speedometer.

4.2.5.4 Conclusion & Example

After eradicating all potential conflicts including those that influence parallelization behavior, it is possible to ensure a system's validity with regard to data age and the corresponding model. Having reached this state of multiple-IEU robustness means that the model is now ready to be split up safely into functional blocks. These blocks can be mapped on different IEUs and can be executed in every possible order without corrupting data consistency.

Figure 4.3 shows a data validated (and verified) WSS which is prepared for distribution on different IEUs. The green lines indicate EOCs that, e.g., enforce the wheel speed sensors to be triggered before the ABS controller. The purple circles with arrows stand for ACs imposed on specific variables.

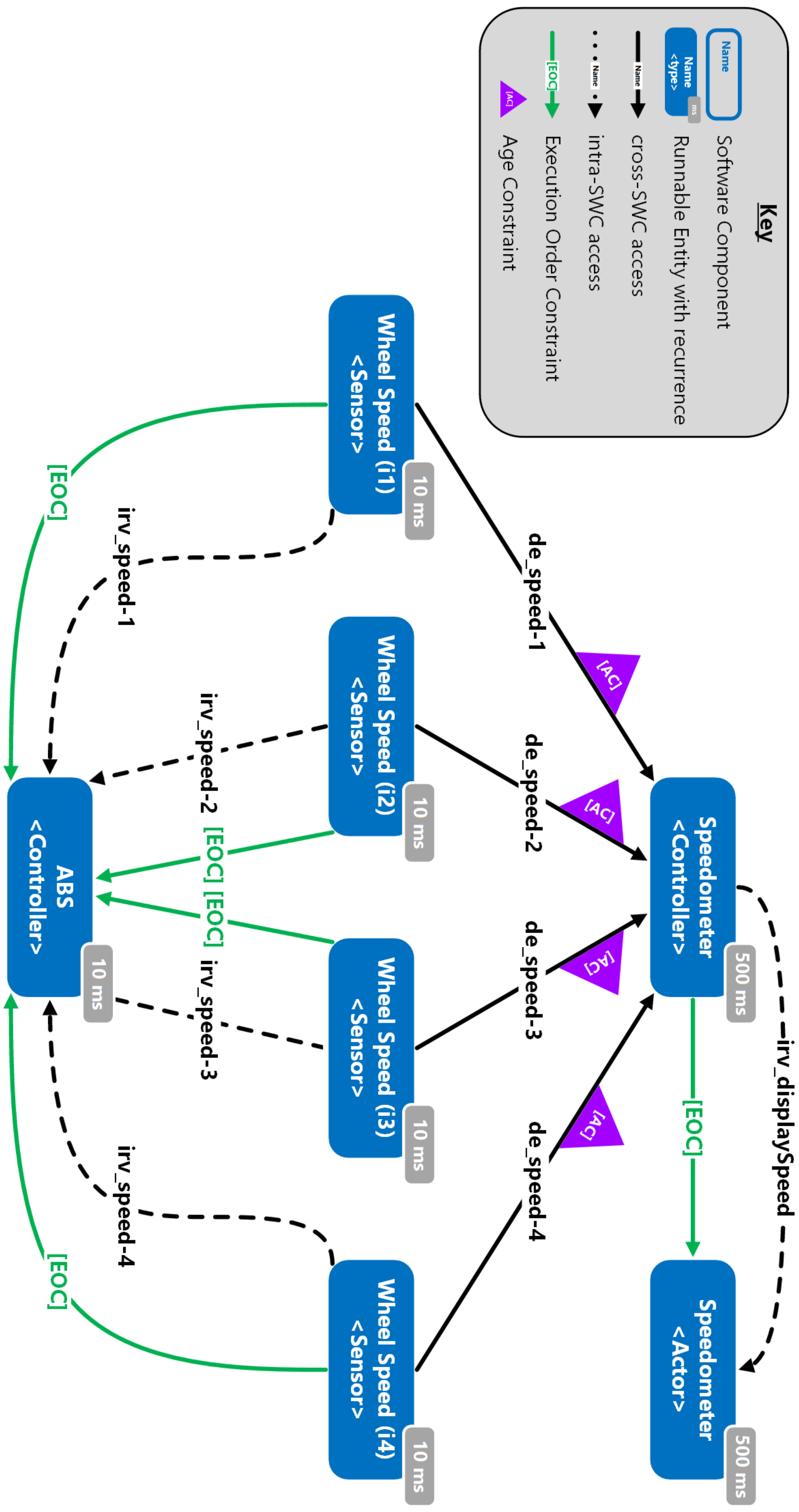


Figure 4.3: The validated Wheel Speed System

4.3 Partitioning & Mapping

Once the consistency threats have been identified and solved with the help of constraints, the next logical step is to figure out how the software can be split up and distributed in an expedient way: At first, “partitioning” breaks up a model into sets of REs according to a given objective, then the succeeding “mapping” means to determine concrete tasks within the obtained partition and to assign them to specific IEUs. Afterwards, their actual execution can be scheduled.

4.3.1 Conditions & Approach

As a matter of fact, there is no general-purpose approach to find a suitable partition or mapping and it is difficult to assess whether a specific solution will satisfy certain properties. Therefore, it is essential to thoroughly consider the desired aspects of the target system and its according objectives in advance. This is usually done with respect to definite goals like (cf. [Kun17, $\text{EZV}^{+}17$, Gra17]):

- reaching a preferably low coupling rate between the tasks and therefore rather little necessary synchronization as well as communication (“pooling”),
- ensuring the adherence to safety requirements like distinctly separating highly critical tasks,
- preserving the processing of logically related software parts on the same IEU, e.g., REs contributing to one common function,
- providing a certain minimum level of performance or throughput,
- supporting a system’s robustness, e.g., by means of redundancy,
- seeking equal core utilizations (load balancing),
- economizing energy expenditure,
- reducing memory consumption as well as
- minimizing required hardware resources in general.

Reaching several of these goals at once is very hard as there usually is no overall optimal solution, i.e., one that does not compromise.

Besides, there is a clear trend away from monolithic architectures and toward more fine-grained software parts (cf. [Kun17]) which renders partitioning an even greater challenge.

4.3.1.1 Escalating Possibilities

As there are numerous possibilities to fractionalize a model, finding an "optimal" partition according to specific goals is ranked as NP-hard problem [BJ92]. In addition, searching for an advantageous "task-to-core" mapping entails traversing an overwhelmingly huge solution space, because the number of mapping possibilities grows exponentially according to the amount of given tasks. Together, both activities represent one of the biggest challenges when trying to build an optimized multi-core system.

It is easy to show that this can quickly escalate: The running example – the Wheel Speed System – consists of seven REs. Assuming that each RE is supposed to be mapped separately on one of four IEUs available, there are 16384 (4^7) different ways to do so (cf. calculation in Subsection 2.2.6). After choosing one of these distribution solutions, there are again many possible execution sequences arising: there are already 5040 ("7!") different sequences for executing all REs successively on one IEU.

Taking a rather mid-sized example illustrates the enormous complexity rise: The "Brake-by-Wire" application from "TIMMO"[20] and "TIMMO-2-USE"[21] consists of clearly organized 18 REs [TIM09, TIM10].

For this model, the same calculation as for the WSS already results in about 387 million (3^{18}) different mappings for a target platform featuring three IEUs. Each mapping solution involves again a vast number of possible execution sequences as there are already over six quadrillion ("18!") sequences for executing all REs successively on only one IEU. And there are a lot (exponentially) more options in a multiple-IEU setting, because most REs can theoretically be processed in parallel (fully or partially overlapping).

Generally speaking, every random set of REs can be simultaneously exe-

[20]The project "TIMing MOdel" developed "a common, standardized infrastructure for the handling of timing information during the design of embedded real-time systems in the automotive industry" [TIM09].

[21]The project "TIMing MOdel - TOols, algorithms, languages, methodology, and USE cases" provides "tools, algorithms, languages, methodology, and use cases for dealing with timing requirements and properties for timing analyses during the development of distributed embedded automotive systems" [TIM10].

cuted as long as it is valid regarding the absence of two REs being interconnected by an EOC. The exact count of possibilities depends on the number of available IEUs (defining the maximum set size), the number of tasks encapsulating the REs and possibly given minimum requirements for load balancing (together with execution times).

4.3.1.2 Degree of Freedom

As opposed to this, it can easily be illustrated how initially "much" design freedom gets quickly decimated by constraints: As a finally found schedule is only valid if it sticks to the imposed constraints, five EOCs in the WSS example already lead to a strongly reduced number of 504 legit execution sequences (out of 5040 initially), which is shown in Figure 4.4.

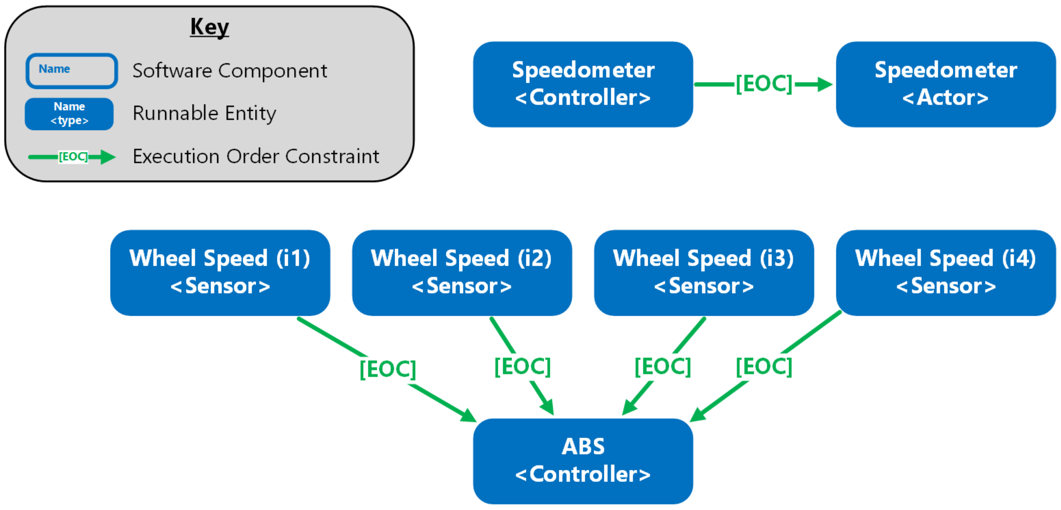


Figure 4.4: The simplified WSS for calculating possible sequences

In this view on the WSS, data accesses and recurrences are left out to concentrate on the number of possible execution sequences remaining after some EOCs are imposed. The five constraints are sufficient to exclude 4536 out of 5040 possibly successive sequences.

The calculation process is as follows: There are 24 ("4!") possible sequences for the sensors and the ABS controller, because the sensors' order can be permuted and the controller has to be executed after them. For each of these specific orders, there are 42 possibilities to insert the two remaining speedometer REs, where only half of them (21) are valid (fulfilling the EOC): $24 * 21 = 504$.

4.3.1.3 General Approach and Added Value

Considering this, the goal is to reduce the number of possibilities with an appropriate sense of proportion by first providing a beneficial initial partition and secondly – based on this starting point – an advantageous initial mapping, which increases the efficiency of the following scheduling, simulation and optimization.

As the partition is created with respect to imposed constraints and existing dependencies, the necessary computational effort, e.g., done via permuting all possibilities, as well as reflections on the actual distribution and deployment are limited to a "corridor" of preferably promising solutions. Since it cannot be guaranteed that proper paths are not discarded, this process should be repeated in order to ensure a balance between searching deeply and broadly.

Without a given partition and if no further knowledge of the system is available, a simulation tool would be forced to draw on simple strategies to obtain initial tasks, e.g., preferably encapsulating REs with equal recurrences and therefore creating homogeneous and easily relocatable tasks. Such regions are particularly suitable for being executed on a common IEU, so that the duration of one "computational iteration" on this IEU is not needlessly delayed due to REs' diverging recurrences that are cumbersome to reconcile.

However, such a partition can be very adverse too, especially when load balancing is hampered by strongly differing task sizes or when – as it is almost always the case – cross-core communication is an issue and heavily connected REs are not assigned to the same core. According to gained experience, this holds particularly true for highly complex models like those used in the case study (cf. Section 5.3).

4.3.2 Partitioning

Partitioning support is given by advising which model elements are suitable for being grouped to tasks. The intention is to speed up the search for a convenient partition as it serves as basis for a subsequent mapping and scheduling. Avoiding an adverse partitioning is vital as it is able to spoil both performance and efficiency.

As stated in 4.1.2.2, "low coupling" (corresponds to "pooling") acts as standard partitioning objective. It is determined by assessing the dependencies (i.e. their edge weight, cf. 4.3.2.2) that cross region borders within a certain partition, i.e., data accesses that are "broken" by assigning the involved REs to different regions. Restoring these dependencies (preserving their function) requires additional synchronization effort, because at scheduling, the execution of the respective REs has to be coordinated according to their specific "Cross-Linking Degree" (CLD). The partitioning does not depend on the target system's processor properties, i.e., both architectures equipped with "homogeneous" (equal) as well as "heterogeneous" (differing) IEUs are addressed by this step.

This concept is realized by the "Single-Entry Region Analysis" (SERA) algorithm that searches for relatively isolated RE sets within the model. They are characterized by a common starting point (the entry node) and by having preferably few dependencies to outside nodes before a common end point (a merger node) "closes" the region.

In the beginning, the algorithm used to be not productive enough for highly complex models as its strict rules were not defined for heavily interconnected graphs. In order to make it applicable to all kinds of models, it was purposefully extended to meet the emerging requirements.

4.3.2.1 Search Tolerance

Being configurable according to the specific model's complexity, the algorithm accepts a certain number of "isolation violations" without immediately discarding the identified RE set. This is useful to perform a search that takes the "Average Node Degree" (AND – the number of dependencies per node) into consideration, making it possible to find "hot spots" even in dense graphs. Based on experience, it is – in most cases – relatively easy to detect a sensible upper limit for this tolerance, because found groups beyond this "turning point" are – often out of a sudden – bulky and evidently not significant anymore.

4.3.2.2 Dependency Weights

Treating the pair-wise connection between all nodes equally would be obviously not expedient when having to decide which one to "break" while trying to form RE sets.

Therefore, the first approach was to calculate the weights for the connection degree of every linked node/RE pair. The respective values could be determined by using the information usually available in AUTOSAR models: the REs' period and the number of dependencies (variable accesses) connecting them. The weight value rises according to decreasing periods (corresponds to higher triggering frequencies) and an increasing number of dependencies, resulting in the formula:

$$weight = (1/period1 + 1/period2) * dependencies \tag{4.1}$$

It was easily adaptable if further information (like the amount of transferred data of a specific variable access) is given and used to serve as enabler for the "relevance partitioning" principle (cf. 4.3.2.3).

However, focusing on the edge cuts is not enough as their number does not automatically correlate with the actual communication volume and relevance [MPS07, Hen98]. Thus, edge cuts are per se not a reliable indicator for additionally caused synchronization between software parts whose execution takes place distributed, e.g., on different IEUs [SKK00].

Therefore, the initial dependency weighting approach was refined in order to better reflect the implied coherence of pair-wise nodes. The approach follows the guiding idea of an "architecture-aware partitioning" pointedly taking heed of "cost of communicating data between a pair of processing elements" [BMS+16]. Correspondingly, it rests on the following principles:

- Type-Specific Weights: The dependencies between a node pair are now distinguished and specifically assessed according to their type, e.g., different single weight values in AUTOSAR for
  - valid Sender/Receiver (constrained),
  - invalid Sender/Receiver (missing AC),
  - invalid Sender/Receiver (missing AC or EOC) and
  - Client/Server dependencies.
- Execution Order Factor: Predefined (sequential) execution orders are now treated as separate formula factor, because their relevance is not intended to depend on the number of variable accesses occurring between the nodes that they address.
  For example, the relevance of EOCs in AUTOSAR decreased according to a rising number of VAs due to totaling up the values to one sum

when using the initial weighting formula. Now, their weights are fixed correspondent to their type, e.g., valid, cyclic or invalid EOCs. Setting high weight values for EOCs equals to imposing affinity or pairing constraints on tasks.

- Exceptions: Missing recurrences (the nodes' triggering frequencies), reflexive dependencies and other special cases are dealt with separately. For AUTOSAR, this means that
  - there is a base weight for every pair-wise dependency of two REs (e.g. if there are EOCs imposed but no VA is specified),
  - missing periods (recurrences) are compensated with a minimum base weight (e.g. a fraction of the smallest weight occurring) and
  - reflexive edges are weighted 0 as they do not represent a disjoint pair of REs.

These principles resulted in the following three-part formula:

$$weight = (1/period1 + 1/period2) * (1 + accessWeightSum) * eocFactor \tag{4.2}$$

Following empirical data, suitable parameters for AUTOSAR are the ensuing relative weight factors:

- The formula's "accessWeightSum" results from adding up data access weights (including base value 1):
  - 2 for Sender/Receiver dependencies that are valid (properly constrained),
  - 1 for Sender/Receiver dependencies that are possibly conflicting due to a missing AC (VA between REIs with divergent periods),
  - 3 for Sender/Receiver dependencies that are possibly conflicting due to a missing AC or EOC (VA between REIs with uniform periods) and
  - 1 for Client/Server dependencies.
- The formula's "eocFactor" complies with the status of the Execution Order Constraints:
  - 10 for valid EOCs as the REIs have to be executed rigidly consecutive and are therefore strong candidates for being grouped into

one common task,

- 5 for cyclic EOCs as the order constraints are per se correct, but there are expected to be less of them after the data validation process and
- 1 (identity element) for invalid EOCs as an increased weight is not justified due to unclear importance.

Elements like Age Constraints or "Parameter Accesses" (providing fixed initial variable values) are deliberately not associated with own weight factors as they do not represent dependencies between nodes. The former serve as given or missing "temporal dependency supplement", whereas the latter are lacking coherency semantics as there is no node pair associated with them.

4.3.2.3 Relevance Partitioning

In most cases, it is rather fruitless to pursue a simple partitioning approach like "Sparsest Cut" which repeatedly cuts a graph into two (roughly) equal-sized pieces [CKK^{+}06]. This is due to strongly differing model structures which are usually not suitable for being strictly divided into a number of 2^x parts.

Thus, a more sophisticated approach vaguely resting on Multi-Level Partitioning is used, which better adapts to specific model structures [ACU08]. MLP reduces a graph via "edge contraction" ("coarsening") in order to cluster and afterwards restore it (cf. Subsection 2.2.5 for details).

However, the SERA algorithm does not remove nodes/edges, but gradually increases the relevance threshold for dependencies considered by the search until the graph is manageable enough to find appropriate RE sets. This is done based on the dependency weights for the current subgraph (partial model) determined with the recently described method (cf. 4.3.2.2). They are used to create an aggregated view (a weight statistic) showing the frequency distribution of certain weight values as well as their share of all dependencies. If the partitioning search was not able to achieve the desired coverage (the target quota of REs assigned to tasks), the weight threshold for considered dependencies is raised. Taking heed of this, predecessor and successor lists are re-created deliberately ignoring dependencies below this new limit. Afterwards, the SERA search is re-run.

4.3.2.4 Splitting Strategy

As previously mentioned, it is basically advantageous to identify groups whose REs have a uniform recurrence. This can be achieved by different strategies:

- "Split, then analyze": Experience shows that building subgraphs which consist of uniformly triggered REs and then running partitioning searches on each of them has produced the most valuable results for highly complex models.
  The crucial point is that the number of connections to consider is remarkably reduced. Consequently, the remaining dependency weights gain expressiveness as they are calculated solely from relevant neighboring nodes. To the best of our knowledge, this strategy is the most efficient way to find homogeneous groups.
- "Analyze, then split according to periods": This strategy takes the graph as is and assumes that the search finds sufficient groups, which can afterwards be split according to the number of diverging RE periods occurring. This is rather suitable for small heterogeneous or for huge but loosely connected models as the graph's connectivity is inherently preserved.
  By contrast, applying the first strategy ("split, then analyze") to such models could result in "shattered" subgraphs which hardly possess exploitable cross-linking. However, the according risk of occurrence for complex – and therefore rather dense – models is negligibly low.
- "Do not split, discard mixed regions": Here, identified regions are discarded if they do contain REs with diverging periods. This can be useful for models with a small amount of different periods that are nevertheless relatively complex.
  This strategy is stricter than the previous one since all "mixed" groups are discarded instead of being split up into homogeneous ones.
- "Do not split, keep mixed regions": As pretty simple strategy, this approach is rather used as starting point to gain an insight into the possible partitioning degree of a model in general. Since it does not yield homogeneous REs, the industrial relevance of this strategy is limited.

4.3.2.5 Automated Search

Determining proper search settings is a crucial step when trying to find suitable partitioning and mapping solutions. For the approach of this thesis, there are multiple parameters that have to be set:

- Tolerance: the maximum number of allowed "Single-Entry Regions" (SERs) violations when trying to identify relatively isolated RE sets
- Coverage Target: the aimed at proportion of nodes/REs assigned to sets/groups
- Dependency Weights and EOC Factors: the relevance of data accesses and ordering constraints used when calculating the weight of a node pair's dependency
- Step Sizes for Relevance and Tolerance: parameters to adjust the thoroughness of the SERA partitioning search

Among them, the tolerance value is the most important parameter: While too low values often prevent useful coverage rates, high tolerances can unnecessarily lengthen the search and corrupt its results. In general, it is very hard to determine suitable parameters without having extensive knowledge about the model and the applied algorithm.

In order to automate this process, it is important to properly assess a model's complexity in the first place. Measuring it by computing the Average Node Degree is not expedient as it does not necessarily represent a graph's complexity. This is due to the fact that reflexive edges/dependencies as well as isolated nodes do distinctly distort the AND.

For the purpose of focusing on the relevant indicators for the interconnection rate of a model, the complexity is thus expressed by the number of connected node pairs together with the number of REIs that are not isolated.

Like previously mentioned (cf. Subsection 4.3.2), the Cross-Linking Degree reflects these properties:

$$CLD = connectedNodePairs/nonIsolatedREIs \tag{4.3}$$

It is therefore employed to support the automated SERA configuration.

For being able to effectively measure a partitioning solution's range and progress, it is moreover crucial to know the rate of nodes assigned to groups (SERs). In a previous version or the approach and its implementation, the coverage rate was calculated by putting assigned nodes in proportion with all nodes. As this does not exclude isolated nodes – like it was the issue with the AND figure – such a coverage rate cannot serve as significant indicator for the progress of a partitioning process.

For example: The simplified EMS model "AMALTHEA DemoCar" (cf. Subsection 3.3.2) consists of 43 REIs with three different periods:

- 9 REIs with period 5 ms,
- 33 REIs with period 10 ms and
- 1 REI with period 20 ms.

When the model is split up according to periods, all of the 5 ms REIs, 27 out of the 10 ms REIs and the single 20 ms REI are isolated. Thus, only 27 out of 43 REIs (about 63 %) are basically assignable if mixed regions shall be avoided.

Therefore, it is often very hard (and sometimes impossible) to reach high coverage targets because of scattered subgraphs (partial models). Consequently, searching until high coverage rates are achieved may easily lead to a reduced solution quality due to enforced high tolerance and relevance partitioning thresholds.

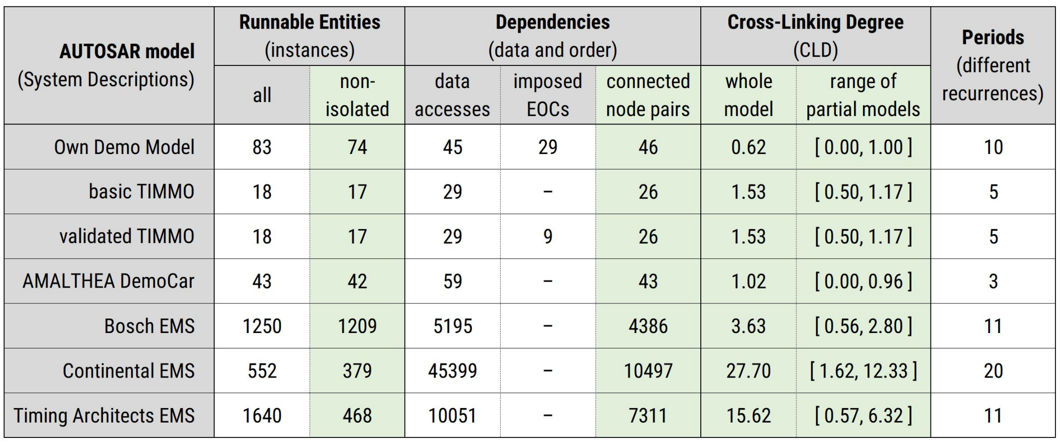
An effective countermeasure is to calculate the coverage rate for each partial model considering only actually assignable (i.e. non-isolated) instead of all REIs and add them up to obtain a "combined coverage" rate across all partial models. As a result, a globally set coverage target can be reasonably applied to every partial model. Moreover, a selected coverage target is now both realistically attainable as well as comparable across different model types and sizes.

In order to illustrate employed model properties, Figure 4.5 shows data collected from selected test models:

- The "Own Demo Model" has been developed and extended for several years in order to verify and test the parallelization approach and implementation. It contains all detectable types of nodes, dependencies and other elements.
- Both "TIMMO" models are variants of a braking system model created within the TIMMO project (cf. description and footnotes in 4.3.1.1).
- The "AMALTHEA DemoCar" is a freely available sample project de-

scribing a simplified EMS (cf. Subsection 3.3.2).

- "Bosch EMS" is a synthesized and realistic EMS model provided within the "FMTV Verification Challenge" initiated by Robert Bosch GmbH [WAT16, Qui16] (cf. Subsection 5.3.1).
- "Continental EMS" is a partial real-world EMS model.
- "Timing Architects EMS" is a synthesized EMS test model from the former "Timing-Architects Embedded Systems GmbH".



AUTOSAR model (System Descriptions)	**Runnable Entities** (instances)		**Dependencies** (data and order)			**Cross-Linking Degree** (CLD)		**Periods** (different recurrences)
	all	non-isolated	data accesses	imposed EOCs	connected node pairs	whole model	range of partial models	
Own Demo Model	83	74	45	29	46	0.62	[0.00, 1.00]	10
basic TIMMO	18	17	29	–	26	1.53	[0.50, 1.17]	5
validated TIMMO	18	17	29	9	26	1.53	[0.50, 1.17]	5
AMALTHEA DemoCar	43	42	59	–	43	1.02	[0.00, 0.96]	3
Bosch EMS	1250	1209	5195	–	4386	3.63	[0.56, 2.80]	11
Continental EMS	552	379	45399	–	10497	27.70	[1.62, 12.33]	20
Timing Architects EMS	1640	468	10051	–	7311	15.62	[0.57, 6.32]	11

Figure 4.5: Model properties used for the automated partitioning search

The values employed for automatically detecting appropriate search parameters are highlighted (green background). With an overall CLD of 27.7 as well as the highest number of dependencies and different periods, the Continental EMS is clearly the most complex model.

In this thesis, the three basic approaches to determine the important tolerance value for such models are:

- "Manual": Applying a fixed maximum tolerance value on all partial models following either experience or a trial-and-error principle.
- "Ranges": Here, the maximum tolerance is fixed as well. It is determined according to the peak CLD values of the involved partial models' coarsely taking the number of them as well as their differing complexity into consideration.
- "Adaptive Formula": In this approach, the tolerance value is dynamically adapted for each SERA on a partial model according to its respective CLD.

The remaining parameters are set as follows:

- Relevance and Tolerance Step Sizes: Starting with traversing each possible search setting, the step sizes are increased if a certain number of runs or runtime is reached. Experiments show that this practice's influence on the outcome quality is very low, especially if a sufficient number of runs was already conducted.
- Coverage Target: The automated search takes "80 %" as default coverage value as such a share of assigned nodes preserves a certain – commonly intended – degree of freedom, e.g., to effectively support a proper load balancing at the mapping step. In addition, it has turned out to be a reasonable trade-off between effort (runtime/runs) as well as achieved coverage. By contrast, trying to group as many nodes as possible by going for "100 %" yields the most detailed and comprehensive results.
- Dependency Weights and EOC Multiplication Factors: As the values listed in 4.3.2.2 were empirically chosen with care, sticking to them is suggested in order to gain useful results.

Considering the Adaptive Formula adjusting the tolerance dynamically, the automated parameter setting is independent from the selected splitting strategy because the number of (partial) models to be analyzed does not affect individual results (as opposed to the Ranges approach).

Although the splitting strategy is not an integrative part of SERA, it is worth considering to include its initial selection within the automated process. However, the strategy is deliberately not set according to, e.g., a global or interval (range) CLD value, because there is simply too little information to reasonably derive the most "expedient" strategy. Since it is the standard case to seek (temporally) homogeneous groups, "Split, then analyze" acts as well justified default strategy.

In order to enable a basic quantitative comparison of the three approaches listed above, a series of test runs was conducted on two EMS models: the Continental EMS and the Bosch EMS.

As expected, the manual "steamroller" approach achieves the highest overall coverage ratio for both models, as it continuously searches for regions with up to maximum tolerance even in smaller or loosely coupled partial models. Here, the Adaptive Formula yields slightly lower rates than the manual approach but by far surpasses the Ranges approach.

In terms of time consumption, the automated approaches do considerably outperform the manual one as they need significantly less runs and runtime with similar search depth settings. The same holds true regarding efficiency, i.e., the ratio of reached coverage and runtime, which becomes particularly noticeable with the Bosch model.

The quality of the manual solution is distinctly lower as it virtually ignores the partial models' specific structure and complexity. However, a solution's appropriateness for efficiently migrating software to a multiple-IEUs platform heavily depends on the adaptability of the applied search algorithm. Accordingly, the "ranges" approach delivers acceptable search parameters, but does not pay enough attention to strongly differing subgraph structures. As a consequence, the Adaptive Formula creates the most target-oriented and accurate results.

4.3.2.6 Conclusion & Example

Due to the dynamic adaption (e.g. automatically rising the tolerance and dependency threshold until a certain coverage rate is reached) as well as the automated search, the algorithm can effectively cope with models of any size and complexity. Although this does not mean that every application can be efficiently parallelized, it is always possible to identify a proper partition according to certain circumstances.

These insights can be applied on the WSS when attempting to maintain strongly-connected parts in the process of splitting up the model. An appropriate task assignment – guided by similar recurrences and preserving EOCs – is depicted in Figure 4.6. Here, the dependencies between the four wheel speed sensors and the ABS controller ("irv-speed-x") are brought into action 50 times more frequent than those caused by the speedometer controller's reading access on the sensors' output ("de-speed-x"). In addition, the former variables are accesses within one SW-C as opposed to the speedometer's consuming that crosses SW-C borders and needs to be done via ports. Taking the added constraints into consideration does also suggest to split up according to existing component allocation, because rigid sequences imposed by EOC are easier to adhere (schedule) to when their processing takes place on a common IEU.

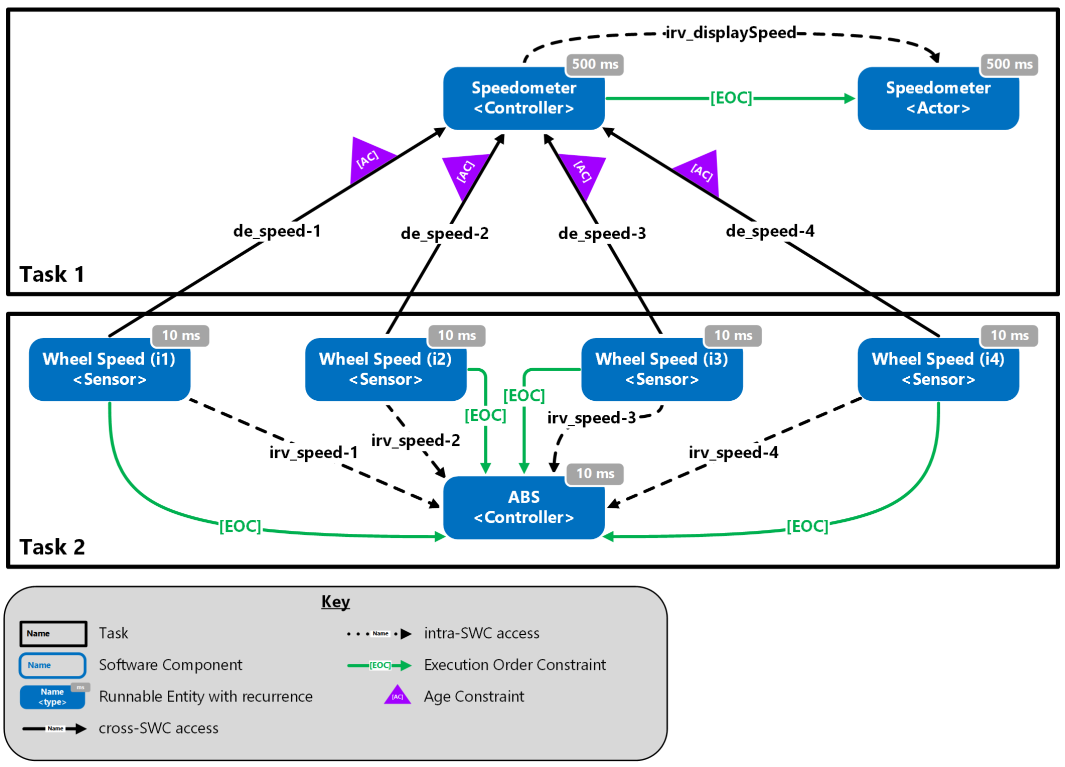


Figure 4.6: The REs of the WSS distributed into two tasks

4.3.3 Mapping

The size of the parts found by the partitioning, i.e., the granularities, plays a central role in parallel computing, because they can be adverse in particular cases and therefore even foil an efficient mapping of program parts to IEUs right from the start.

The partitioning algorithm determines preferably large RE sets, which can – hierarchically structured – contain smaller ones. This is done deliberately in order to maintain RE sets of every size and therefore to retain all granularities for a later mapping of tasks to IEUs. It is easy to see that the number of the IEUs (if known) acts as bottom limit for the count of parts the program should be split into [GS12].

The count for both, the mapping of tasks to IEUs and the possible execution sequences, strongly depends on the initial number of tasks. Seeking to prevent a too fine-grained partition (many small tasks) is a sensible trade-off because fine granularities involve extra effort due to a sharp increase in workload for distributing, coordinating and synchronizing the parts.

On the contrary, there are several reasons to split up more "fine-grained" than at least necessary:

- If there is only one rigid target device, the size of the parts should theoretically enable approximately equal execution times, so that the IEUs' workload can be balanced and idle times are avoided. However, this assumption is rather unrealistic.
- In case of various or even unknown target hardware, it is obvious that the most probable (or all possible) configurations should be effectively supported, which is unlikely if the number of available tasks is rather small and does therefore not provide the necessary flexibility.
- Furthermore, a granularity that turns out to be too fine-grained can be clustered again, whereas there is no quick solution if the program parts are bulky.

As opposed to these facts, a coarse-grained partition "can more easily result in an improvement" [Moy13]. This seems to be an appropriate first step since only a few big program chunks make partitioning easier and require fewer management effort. Thus, handling them is less error-prone than coordinating many tiny tasks. However, having only very few large tasks can make it difficult to distribute them skillfully on different cores without again causing overhead for additional synchronization, e.g., if being forced to map two intensively connected partitions on different cores or when trying to achieve even workloads for them (cf. itemization above).

In conclusion, there is no universally valid guideline for identifying the most convenient granularity as its suitability is affected by a variety of factors, e.g., the application's structure (cf. Subsection 2.2.5), the target platform's properties as well as the later applied optimization goals. In general, it is sensible to make the "basic decision", e.g., going for rather chunky or fine-grained granularity, as late as possible, i.e., when the partitioning showed which solutions are expedient anyway and when the mapping step is imminent.

This decision – as well as the whole mapping process – is crucially facilitated when the target hardware is known as fundamental information, e.g., number and performance of available cores, are decisive. Such circumstances lead to a rather target-oriented proceeding and thus restrict the search space for an adequate model of parallelism considerably. However, undetermined target hardware does provide a bigger degree of freedom for the system design. In the automotive sector, the usual case is – due to intended variability and variants – that applications have to run properly on a variety of possible hardware combinations and configurations.

Being aware of that, our (mapping) principles remain pooling and load balancing, because they preserve the possibility to later pursue common optimization goals like:

- minimizing the Inter-Core Communication,
- shortening the duration of certain event chains (e.g. the Critical Path),
- staying below predefined "load distances" (load balancing deviation),
- complying with maximum response times of certain REs and
- reducing hardware costs (e.g. by efficient workloads and small buffers).

Therefore, it is needed to sensibly choose a suitable size for each available RE set in order to find the most convenient mapping. This is due to the fact that a partition does usually not contain groups with uniform size. Consequently, following a rather coarse-grained approach should not lead to a brute force method like "streamlining" the partition by reducing large groups.

Eventually, the following aspects are taken into consideration when creating a mapping:

- Hardware of Target Platform (individually configured or via choosing a preset like in Subsection 5.2.4):
  - global memory
  - number of available cores (IEUs), their specific frequency and local memory (cache)
  - times for global, local and intra-task read/write operations (in cycles)
- Task Clustering:
  - individually chosen regions or desired preference of selected region size (task granularity) if they are nested
  - clustering strategies for "remaining" runnables: own task for each RE, create clusters w.r.t. to periods, aggregate single REs in default task etc.

- Task & Core Assignment:
  - memory management: default or according to LET, variable assignment (preferably local or global, according to specific memory size)
  - task assignment: strategy (e.g. bin packing or Round-robin), task assignment order (according to period, utilization), assignment scores (utilization, communication)
  - default settings: values for missing periods, execution times and variable sizes
- Requirements: create requirements for region and/or remaining tasks (e.g. critical response times)
- Constraints: create constraints for processes (pairing of tasks) and maximum core utilizations

In AUTOSAR, there are four types of RE execution times given within the according SW-C's "ResourceConsumption": estimated, analyzed, simulated and measured (sorted by descending precision). In case of availability, the most precise of the specified execution times is used and an RE's expected utilization is computed according to the formula

$$utilization = (reInstances * reExecutionTime) \;/\; rePeriod \tag{4.4}$$

These results are again employed to calculate the according workload of the comprising tasks in the course of their assignment to IEUs.

Finally, an exporter tool is employed which creates a CSV[22] file comprising these aspects. Additionally, it produces basic timing requirements (task deadlines according to given periods) and sets task priorities in order to support the succeeding simulation.

[22]"Comma-separated values" (CSV) denotes specifically structured text files designed to easily exchange information.

4.4 Simulation & Optimization

In order to determine the actual benefit when harnessing the information acquired by the previous two steps, it is necessary to create a final schedule and to simulate the software's execution on a multiple-IEU platform. This step (and the following optimization) can be carried out by use of various third-party tools that are designed for simulating (and optimizing) embedded real-time software, such as:

- "chronSIM" by Inchron GmbH is a "tool for design, visualization, quality testing and analysis of embedded systems" [INC15].
- "SymTA/S & TraceAnalyzer" by Symtavision GmbH are "tools for scheduling analysis, architecture optimization and timing verification for: ECUs and software integration, Embedded networks and communication, Distributed embedded systems (E/E)" [Sym15].
- "TA Tool Suite" by Timing-Architects Embedded Systems GmbH is a "discrete, event-based simulation tool" [SSH$^+$16] intended to be integrated in the "whole development process of multi-core systems" and is used for "designing, developing, and verifying embedded multi- and many-core-systems" [Tim15].

4.4.1 Prerequisites & Preparations

There are some basic steps that are necessary before being able to perform a simulation:

4.4.1.1 Model Import

As the data dependency analysis and the consistency validation are conducted directly on AUTOSAR system descriptions, it is crucial that the employed tool can handle such models by processing its structural elements (e.g. REs, SW-Cs or VAs) as well as included timing information like the REs' recurrence and timing constraints (ACs, EOCs).

In general, every piece of information that can be retrieved further refines the simulation, e.g., data on the REs' execution times remarkably improves a simulation's accuracy.

4.4.1.2 Include Partitioning and Mapping Data

This information can either be stored directly in the AUTOSAR models (like described in [KMKB14]) or it can be imported from an external file. Currently, the latter approach is used to import necessary data from a CSV file: the assignment of REs to tasks, the mapping of tasks to specific cores, the tasks' recurrence and their priorities.

4.4.1.3 Setting Basic Requirements

In addition to the aforementioned data on tasks and their core assignment, it is essential to add basic timing requirements for them, i.e., upper limits (deadlines) for the tasks' response time according to their respective periods. They enable the basic assessment of a simulated solution regarding its general validity.

4.4.1.4 Hardware Model Configuration

Providing the simulation with concrete details on the target hardware helps to enhance the significance of the results. This includes – among other things – the number of IEUs, their clock rate (frequency), their cache and available shared memory.

In addition, existing signals (i.e. the model's variables) have to be initially mapped to certain memory modules. In the case study in Section 5.3, the model of a real-world automotive microcontroller featuring three cores is used and the signals are assigned to core-local memories according to the made task-to-core mapping.

4.4.1.5 Selection of Scheduling Algorithm

Most simulation tools provide different algorithms for the static scheduling of the tasks, whereas dynamic scheduling is virtually never employed because making "online decisions" during runtime is – concerning real-time systems – inconvenient due to deteriorated predictability as well as impending overhead [NBN09,Cor13,MNBSL12].

Popular algorithms are "Rate-Monotonic Scheduling", "Deadline Monotonic Scheduling", "Earliest Deadline First" (EDF) or "Proportionate Fairness" [ABRW93, SSRB12, LL04]. There is no universally valid heuristic helping to select the optimal one according to a certain model. Thus, running tests with a couple of them seems to be expedient.

In addition, these algorithms can be applied "globally" or "partitioned" [NNB10, Mel15]. The former's idea is to conduct a scheduling for all IEUs at once, whereas the latter employs a scheduling algorithm for each IEU separately. Global scheduling usually results in higher utilization, lower average response times and better load balancing whereas partitioned scheduling is considered to be more efficient and existing single-core algorithms can easier be adapted and adopted [NNB10, Mel15]. Moreover, the latter is by design suitable for static scheduling like supported by AUTOSAR where task migration to other cores is not intended.

4.4.2 Simulation

In pursuance of evaluating the initial partitioning and mapping solution, a discrete-event simulation tool is used to conduct the evaluation regarding valid scheduling (i.e. fulfillment of task deadlines), specific reaction times of critical execution paths, communication overhead, memory consumption and core load distribution. Compared to analytical methods, simulation techniques only yield approximated timing metrics like task response times [Gri04]. However, analytical methods usually provide very pessimistic estimations resulting in, e.g., overestimated worst-case response times. On the contrary, simulation techniques allow more realistic typical case approximations. Additionally, the proposed overall process as presented in Subsection 4.1.2 incorporates hardware measurements of simulated solutions after deployment as a final timing verification step.

4.4.2.1 Discrete-Event Simulation

Discrete-event simulators utilize the fact that in-between two consecutive events, a system cannot change its states [WGG10]. Consequently, only the discrete points in time where state transitions happen are simulated.

All state transitions which occur during simulation together with the respective time stamps are recorded in a "Better Trace Format" (BTF) trace [Ecl16].

The simulator operates on the AUTOSAR-compliant and AMALTHEA-compliant timing model [SDM$^+$14], which consists of abstract descriptions of the application software, hardware, operating system, runtime environment and environment (i.e. external stimuli). As already broached in Subsection 4.1.1, the simulation needs certain required information.

For example, the operating system model must include a specification of the schedulers which manage the execution of tasks and "Interrupt Service Routines" (ISRs) on the respective cores. Moreover, the used scheduling algorithms and the scheduling-relevant properties of tasks and ISRs (like priorities) have to be provided.

While simulation is already possible with basic hardware model information like the number of cores together with their clock frequency and instructions per cycle, detailed vendor-specific processor models greatly improve simulation precision. When the exact memory topology and behavior descriptions including memory modules, caches, bus networks or crossbars are provided, memory access times, cross-core communication delays as well as contention effects can be considered in a simulation.

4.4.2.2 Timing and Performance Metrics

After the application of statistical estimators to the resulting event trace of a discrete-event simulation, various timing and performance metrics can be calculated. In the following, the most important metrics required for the optimization step are introduced:

- maximum Normalized Response Time (mNRT): The mNRT metric quantifies the relative worst-case response time which occurred in a simulation [SDM$^+$14]. "Relative" means that the response time of each task has been normalized with respect to its relative deadline. If all deadlines are met, the mNRT is smaller than "1" whereas greater values denote deadline violations during simulation.
- Inter-Core Communication (ICC) Rate: The ICC metric quantifies the amount of data in bits per time unit which is exchanged between the cores. It is an indicator for the expected cross-core communication overhead.
- CPU Load: The CPU Load metric quantifies the average load of a processor or individual core over the complete time span covered by the simulation.

- Maximum Load Distance: The Maximum Load Distance metric quantifies to what extent the overall load is equally distributed to the individual cores. It is the maximum of the absolute differences between the CPU Load values of each core and the per-core CPU Load value obtained by dividing the overall load by the number of cores.
- Buffer Size: The Buffer Size metric quantifies the additional required memory in bits needed to enforce data consistency by a buffering technique [MFCM16].
- Event Chain Duration: Event Chains – as defined in the AUTOSAR Timing Extensions [AUT14d] – connect arbitrary subsequent events like the activation of a task, the termination of an RE or write accesses to a specific variable. An Event Chain consists of at least a stimulus and response event but can also be further detailed by segments and strands. The Event Chain Duration metric quantifies the time span between a stimulus and response event of an Event Chain. Thus, the reaction time of critical processing paths in the system, e.g., across multiple REs of different tasks can be evaluated.

4.4.3 Optimization

After the simulation has returned key figures for the initial solution, further ones can be generated by using it as alterable seed. According to the strategy of the applied optimization software, the initial solution is modified to a certain extent and then re-assessed in order to learn if the changes are beneficial.

Some common "leverage points" for variation are, e.g., changing the core assignment of tasks and – accordingly – the signals' mapping to core-local memory, altering task priorities or splitting given (preferably chunky) tasks at several positions.

The outcome of the optimization is heavily influenced by the set of parameters which can be changed by the algorithm. The approach does not exclude any part of a solution from being altered. Thus, no limits are set for modifying a model, which allows a variety of different optimizer settings.

In the case studies (Section 5.2 and Section 5.3), the employed third-party optimizer uses a genetic algorithm to automatically create new "solution generations". Doing this manually by inspecting the simulation results and purposefully varying (now simpler to identify) critical parts is also expedient yet most likely more time-consuming.

The underlying problem of re-partitioning and re-mapping the software is equal to the bin-packing problem, which is known to be NP-hard [CJGJ96]. Consequently, an exhaustive search, i.e., evaluating every possible alternative solution, is not an option in practice. As Genetic Algorithms have been shown to be particularly suitable for such problems [Deb01], integrating a genetic optimization tool in the workflow seems promising.

4.4.3.1 Genetic Algorithms

These algorithms simulate natural selection and evolution in an iterative approach and operate on a set of alternative solutions [KCS06]. This set ("population") is modified within each iteration ("generation") of the algorithm. Every genetic algorithm consists of the following steps:

1. Create initial population: The initial population consists of randomly created solutions, e.g., using a uniform distribution.
2. Fitness assignment: A scalar fitness value is assigned to every solution of the population which is used to quantify the quality of a solution compared to another one.
3. Selection: Solutions are sorted by descending fitness values. The best ones according to fitness are kept while the remaining ones are discarded and removed from the population.
4. Evaluate stop criterion: The algorithm terminates when the stop criterion is fulfilled. This can either be the case after a predefined number of solutions have been created or after reaching a specific number of generations. Another possibility is to stop after a stagnation threshold has been reached, e.g., when the best solution did not improve for a certain amount of generations.
5. Perform variation: Mutation and crossover techniques are used in order to create new solutions. For the former, one or more properties of an existing solution are randomly modified in order to create a new solution. For the latter, the properties of two or more solutions are combined to create one or several new ones.

These steps only define the generic framework of Genetic Algorithms. The discrete-event simulator presented in 4.4.1.1 is used to evaluate every created solution and to provide the required metrics for fitness assignment. Regarding further implementation details of the used genetic algorithm, the reader is referred to the work of Schmidhuber et al. [SDM+14].

4.4.3.2 Optimization Parameters

In order to configure a specific optimization run, configuration parameters have to be provided for each of the aforementioned steps. They are as follows:

- Per-Solution Simulation Time: This is the time span covered in the simulation for each created solution during optimization.
- Configuration of the Fitness Function: Several timing and performance metrics as introduced in 4.4.1.2 are aggregated together into a scalar fitness value for each solution by using a modified euclidean norm [KCS06, SDM+14]. For each incorporated metric, a weight factor as well as a lower and upper limit for normalization has to be provided.
- Population Size: This parameter defines the number of solutions created in the initial population as well of the number of new solutions which are created during each iteration of the algorithm.
- Selection Size: The selection size is the amount of best solutions (according to fitness) which are taken over into the next iteration. Those selected solutions are also used to create new solutions by means of mutation and crossover.
- Stop Criterion: For the stop criterion, the minimum and maximum number of solutions and/or generations are specified. Moreover, the stagnation threshold is configured, i.e., the algorithm stops if the best solution according to fitness did not improve over a given amount of iterations. All these criteria are evaluated simultaneously, which means that all minimum requirements (e.g. minimum number of generations) and at least one maximum requirement (e.g. stagnation threshold) have to be fulfilled to result in the optimization's termination.

4.4.3.3 Design Modifications

Design modifications denote different categories of architecture changes which are applied to an existing solution in order to create new alternative solutions during variation. It is hereby possible to perform multiple design modifications at once.

For certain categories, design constraints which restrict the respective degree of freedom can be stated as well. If specified, such design constraints will be fulfilled by every single solution produced during optimization. One example for such constraints is the requirement to map certain tasks to different cores, e.g., due to safety requirements which demand spatial separation of the respective functionality.

Employed design modifications are:

- Process Mapping: Process mapping results in the re-mapping of tasks or ISRs to the different cores.
- Task Splitting: Tasks are split into two or more smaller tasks which are afterwards mapped to separate cores. The split tasks are triggered one after another to maintain the original RE execution order.
- Data Mapping: Data Mapping allows the optimizer to change the variable-to-memory mapping.
- Periodic Offset Assignment: This modification varies the offset of periodically activated tasks.

In the end, the crucial insight is if the optimization can – in regard to the mentioned key figures – deliver distinctly better solutions and how much additional effort (most notably time) is necessary. If the latter remains within acceptable limits, the optimization results can give valuable feedback for the "Partitioning & Mapping" step in the form of, e.g., narrowing down the "corridor" of expedient amounts of groups and their sizes or indicating which groups are recommended to be assigned to one common core from the very start ("pairing"). This feedback closes the development and refinement cycle outlined in Subsection 4.1.2 and Figure 4.2.

5 Case Studies & Evaluation

This chapter illustrates the realization and application of the methodology described in Chapter 4 on the basis of three case studies.

The first case study delineates the implementation of the analysis, validation and parallelization procedure as Eclipse-based tool referred to as "AutoAnalyze" (Section 5.1). A first version was created in 2013 and is undergoing continual development since then. It serves as proof of concept that displays the feasibility of the approach by applying it on AUTOSAR and illustrates how mastering the complexity can be effectively supported via proper visualization and filtering techniques.

The second case study shows the applicability and practicability of the approach as well as the added value achieved with it by means of real-world examples (Section 5.2).

The third case study presents an extended series of experiments using an advanced version of the SERA algorithm and pursuing goals originating directly from automotive companies (Section 5.3).

5.1 Realization

In order to efficiently demonstrate the feasibility of the ideas, the concept and specific proceeding were realized as plug-in for Artop.

The implementation uses the "Model Analysis Framework" (MAF) created by Christian Saad, which is based on the "Eclipse Modeling Framework" and provides "a core framework [...] allowing the implementation of dynamic model analysis" [Saa09, Ecl09]. Its main application (and this use case) is the execution of dataflow analyses as particularly described in [SB13] and in a wider context in [Saa15]. The arising tool executes data dependency analyses directly on AUTOSAR models, determines possibly conflicting dependencies, visualizes its results as graph, provides semi-automatic conflict resolution and writes back the modifications to the model. To support the later parallelization of the system, the tool additionally determines groups of Runnable Entities that seem suitable to run on a common IEU (partitioning) and accordingly suggests a customizable mapping solution.

5.1.1 Bringing the Principles to Fruition

The tool's basic goal is making the migration to multiple-IEU systems (or their "from scratch creation") in AUTOSAR manageable despite the imminent complexity rise. Following the principles "incremental" and "bottom-up", it thus provides feedback to developers about the validity of their design by determining ambiguities as well as inconsistencies. Furthermore, it assists in the subsequent process of preparing a model (and therefore the system) for parallelization by offering concrete solutions.

The feedback involves hints on structural deficiencies (verification), the number of potential conflicts and possible constraint modifications to validate the model regarding its multiple-IEU suitability (data validation). On the one hand, existing (unresolved) conflicts hint at the remaining amount of validation effort when preparing a system for its migration to a multiple-IEU platform. On the other hand, a huge number of required (or pre-imposed) Execution Order Constraints indicates a rather low "degree of freedom" when mapping the found groups as tasks to IEUs, because enforced successive execution hampers parallelism. Altogether it is possible to make a rough estimate of the general capability to parallelize a specific system.

5.1.2 Graphical Representation

These results are visualized using the "yFiles"[23] graphic library and its layout algorithms that are particularly suitable for huge graphs. Basically a model is displayed as (hierarchically structured) groups, included nodes as well as edges (dependencies) connecting them. The latter, i.e., data accesses or imposed constraints, are colored and counted according to their type and can be shown or hidden in groups. Various filtering options allow to quickly gain an overview and concentrate on vital parts even for models that include hundreds of REs:

- Indicator filters for timing constraints: valid or invalid (conflicting) ACs and EOCs
- Indicator filter for dependencies: valid or possibly conflicting (unconstrained) Sender/Receiver accesses, Client/Server dependencies, parameter accesses
- Indicator filter for structural problems: REs with either no or multiple periods, unconnected ports
- Period filter: REs grouped according to specific periods
- Service filter: components explicitly marked with a service flag
- Component filter: hierarchical list of containers (components) and corresponding REs

Figure 5.1 shows the "Analysis Filter" tab of AutoAnalyze.

As a consequence, it is, e.g., possible to display certain REs according to their triggering frequency, vicinity or their assignment to an encompassing SW-C, which distinctly facilitates to gain an overview and purposefully modify dense or widespread models. In addition, several model statistics, sortable lists with found potential conflicts, solutions and affected elements as well as logging information with selectable level of detail are available.

This visualization is used as basis for directly editing the model: Support for conflict resolution is provided by offering suitable actions for each potential conflict, e.g., by indicating that a constraint is missing and what type is applicable. Selected actions are immediately applied to the model (or cached and applied altogether when saving the model). When an AC is imposed, the tool automatically calculates an appropriate maximum data age for the

[23]cf. `http://www.yworks.com/en/products_yfiles_about.html`

specific variable. Moreover, aggregated actions (multiple modification triggered at once) enable the user to solve all conflicts of a certain type at once, which comes in useful especially for huge models.

Figure 5.2 shows a screenshot of "AutoAnalyze" displaying a validated part of the "Brake by Wire" example from the TIMMO project (cf. 4.3.1.1). The visualization shows the REs as nodes and the dependencies between them as colored edges:

- The big box shaded in gray depicts an SW-C containing several REs. The tailored spots on the top border denote ports for accesses to/from other SW-Cs.
- Green edges indicate unproblematic dependencies (due to according EOCs and ACs).
- Blue edges represent correctly imposed EOCs.
- Purple boxes show (sufficiently specified) ACs each imposed on a certain variable access.
- The RE/node boxes contain the following information (top down): recurrence, input VAs, output VAs, direct EOC successors and all EOC successors.

In addition, the plug-in shows various kinds of metrics at the bottom, a bird view on the model (bottom left) as well as applied filters or suggested partitions on the right side. The according visualization offers a highly adaptable view on the model, including – among other things – several node and line layout algorithms, zooming and collapse/expand functions.

5.1.3 Search for Partition & Mapping Solution

The result is a validated model whose REs are ready for being distributed to the available IEUs. Following the approach explained in Subsection 4.3.2, the tool supports this by searching for rather "isolated" regions that have only loose coupling with other parts of the system and are therefore promising candidates for being processed on the same IEU. They act as coarse-grained initial partition assigning as many nodes as possible to groups, which is important to support rummaging purposefully (and thus efficiently) within the enormous search space.

This is accomplished by the SERA algorithm, which is inspired by MAF's

"Single Entry, Single Exit" concept, whereas the basic idea originates from [OO84, Tip95] as well as [JPP94] and its further development as "Token Analysis" in [GRLB09].

Since found groups can overlap, being nested within each other and have therefore different sizes, the tool offers the possibility to separately show a partition with certain "granularity". The grouping suggestions were initially stored directly in the AUTOSAR model as so-called "SpecialDataGroups" within the "AdminData" section. Later, the tool was modified, so that the determined groups could be exported as additional CSV file. In both cases, the result serves as advantageous starting point for third-party simulation and optimization software.

Figure 5.3 shows a visualization of the partitioning 's result: After applying a "Split, then analyze" strategy (aiming for temporally homogeneous node sets) and running SERA with automated search settings (Adaptive Formula) on three partial models, all 27 basically assignable REs could be allocated to one of the colored groups.

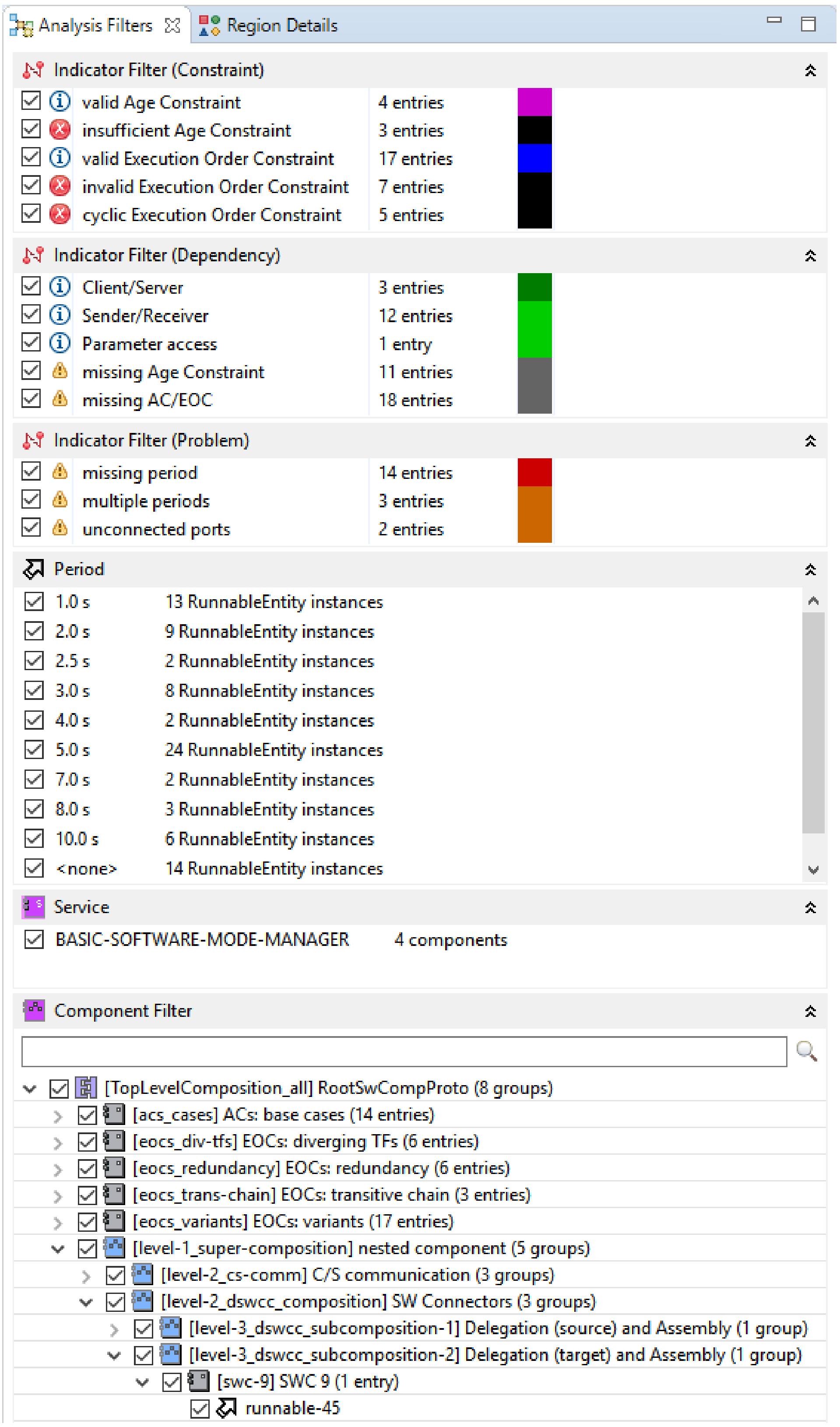
In order to realize the mapping proceeding as described in Subsection 4.3.3, specifying the according properties – i.e. the named allocation parameters and target platform characteristics – was integrated in AutoAnalyze which eventually creates a customized mapping. This mapping can be again stored in the above mentioned CSV file so that it can easily be read and further processed by third-party tools (cf. Sections 5.2 and 5.3). Figure 5.4 displays the properties of a customized mapping together with predicted core utilization rates for a multiple-IEU target platform.

5.1.4 Conclusion

As field test, the tool was employed on many different models and turned out to work properly even with highly complex ones like the "Continental EMS" – a partial real-world EMS that consists of 552 REs and includes 45399 data dependencies.

Altogether, the tool appreciably facilitates the verification and data validation of AUTOSAR models as well as the search for an advantageous mapping of the processing tasks to the available IEUs. This is achieved through supporting the stated goals via proper visualization, automatic processing and editing actions supported by a graphical user interface. Therefore, its purpose of serving as the methodology's proof of concept is fulfilled.

Subsequently, there are several possibilities to make use of the results, e.g., simulating the validated model and giving feedback on the partitioning/mapping suggestions or optimizing it according to the specific target platform for a subsequent deployment.



Analysis Filters | Region Details

Indicator Filter (Constraint)

☑	valid Age Constraint	4 entries	🟪
☑	insufficient Age Constraint	3 entries	⬛
☑	valid Execution Order Constraint	17 entries	🟦
☑	invalid Execution Order Constraint	7 entries	⬛
☑	cyclic Execution Order Constraint	5 entries	⬛

Indicator Filter (Dependency)

☑	Client/Server	3 entries	🟩
☑	Sender/Receiver	12 entries	🟩
☑	Parameter access	1 entry	🟩
☑	missing Age Constraint	11 entries	▨
☑	missing AC/EOC	18 entries	▨

Indicator Filter (Problem)

☑	missing period	14 entries	🟥
☑	multiple periods	3 entries	🟧
☑	unconnected ports	2 entries	🟧

Period

☑	1.0 s	13 RunnableEntity instances
☑	2.0 s	9 RunnableEntity instances
☑	2.5 s	2 RunnableEntity instances
☑	3.0 s	8 RunnableEntity instances
☑	4.0 s	2 RunnableEntity instances
☑	5.0 s	24 RunnableEntity instances
☑	7.0 s	2 RunnableEntity instances
☑	8.0 s	3 RunnableEntity instances
☑	10.0 s	6 RunnableEntity instances
☑	<none>	14 RunnableEntity instances

Service

☑	BASIC-SOFTWARE-MODE-MANAGER	4 components

Component Filter

```
☑ [TopLevelComposition_all] RootSwCompProto (8 groups)
  ☑ [acs_cases] ACs: base cases (14 entries)
  ☑ [eocs_div-tfs] EOCs: diverging TFs (6 entries)
  ☑ [eocs_redundancy] EOCs: redundancy (6 entries)
  ☑ [eocs_trans-chain] EOCs: transitive chain (3 entries)
  ☑ [eocs_variants] EOCs: variants (17 entries)
  ☑ [level-1_super-composition] nested component (5 groups)
    ☑ [level-2_cs-comm] C/S communication (3 groups)
    ☑ [level-2_dswcc_composition] SW Connectors (3 groups)
      ☑ [level-3_dswcc_subcomposition-1] Delegation (source) and Assembly (1 group)
      ☑ [level-3_dswcc_subcomposition-2] Delegation (target) and Assembly (1 group)
        ☑ [swc-9] SWC 9 (1 entry)
          ☑ runnable-45
```

Figure 5.1: Filtering features of “AutoAnalyze”

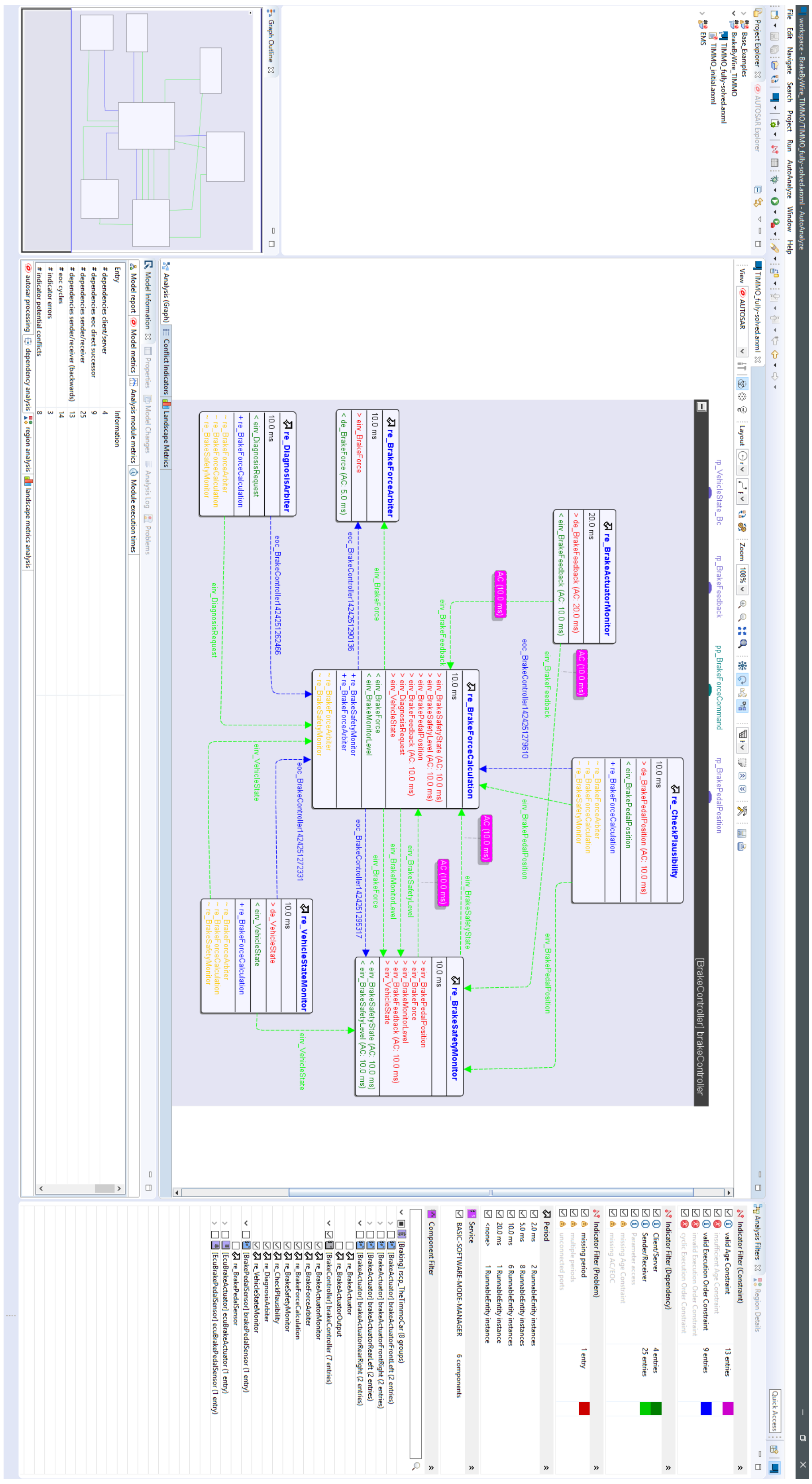


Figure 5.2: "AutoAnalyze" visualizing the "Brake by Wire" example

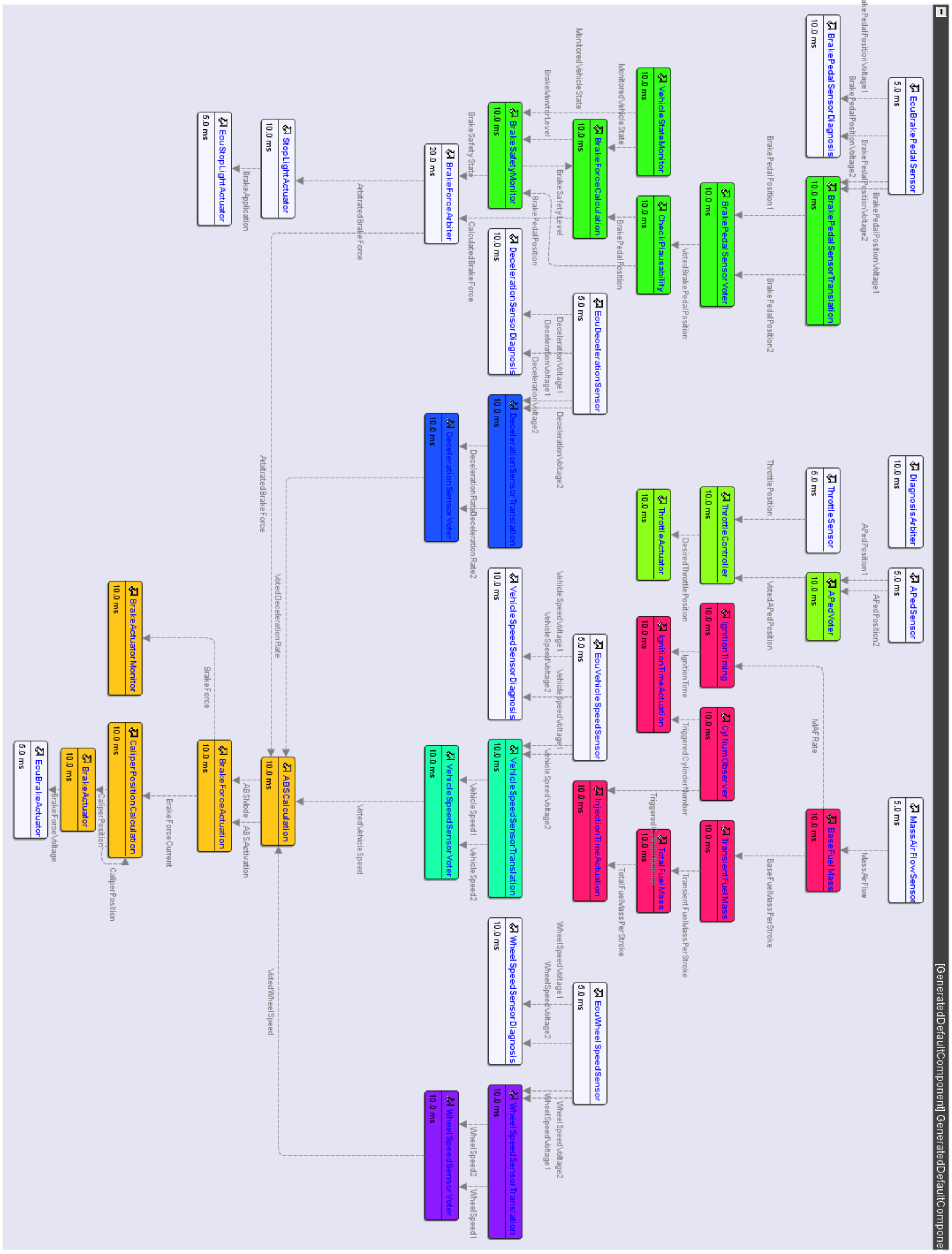
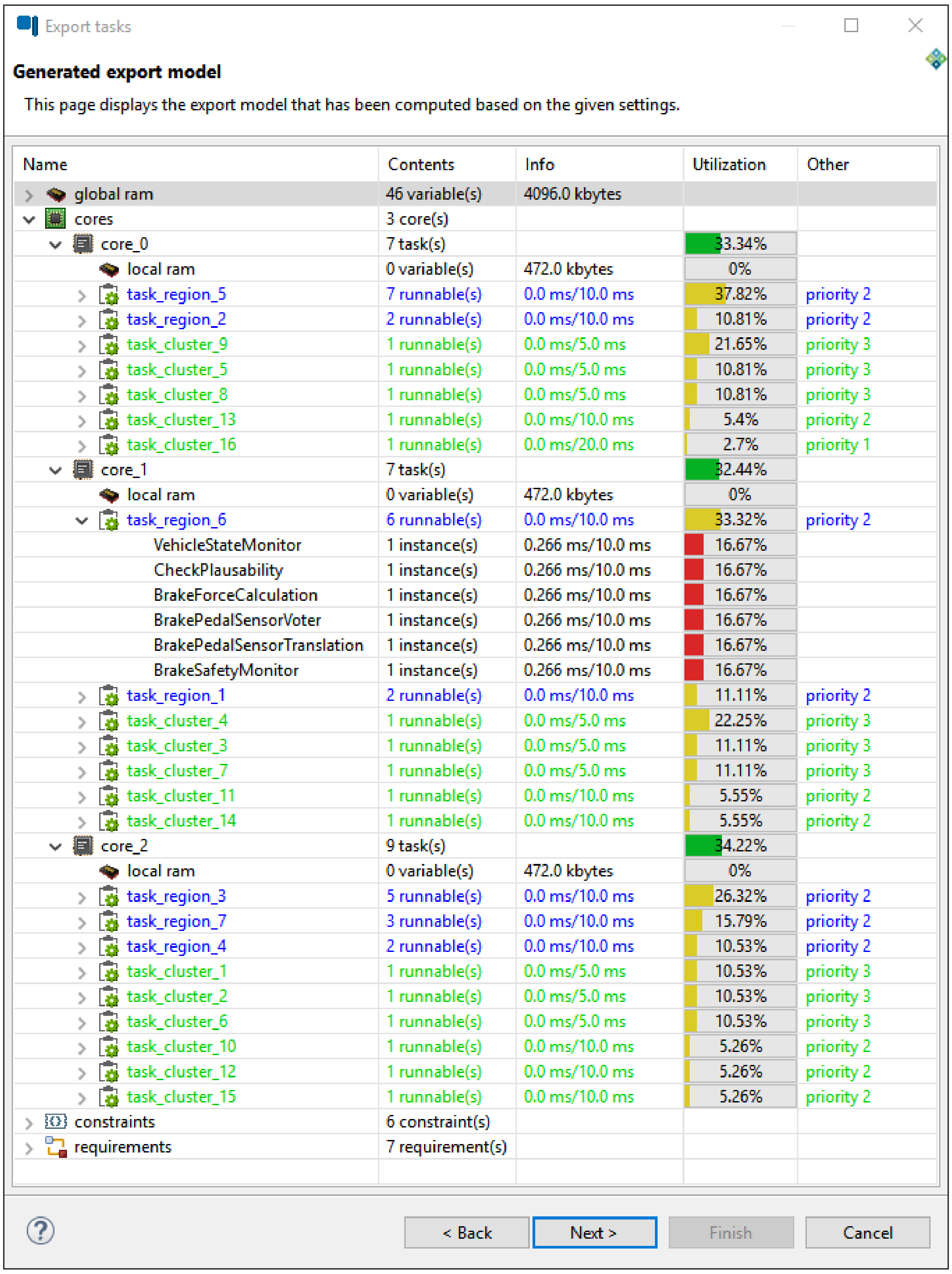

[GeneratedDefaultComponent] GeneratedDefaultCompone
EcuBrakePedalSensor 5.0 ms
BrakePedalSensorDiagnosis 10.0 ms
BrakePedalSensorTranslation 10.0 ms
BrakePedalSensorVoter 10.0 ms
VehicleStateMonitor 10.0 ms
CheckPlausability 10.0 ms
BrakeForceCalculation 10.0 ms
BrakeSafetyMonitor 10.0 ms
BrakeForceArbiter 20.0 ms
StopLightActuator 10.0 ms
EcuStopLightActuator 5.0 ms
EcuDecelerationSensor 5.0 ms
DecelerationSensorDiagnosis 10.0 ms
DecelerationSensorTranslation 10.0 ms
DecelerationSensorVoter 10.0 ms
DiagnosisArbiter 10.0 ms
ThrottleSensor 5.0 ms
ThrottleController 10.0 ms
ThrottleActuator 10.0 ms
APedSensor 5.0 ms
APedVoter 10.0 ms
IgnitionTiming 10.0 ms
IgnitionTimeActuation 10.0 ms
CylNumObserver 10.0 ms
EcuVehicleSpeedSensor 5.0 ms
VehicleSpeedSensorDiagnosis 10.0 ms
VehicleSpeedSensorTranslation 10.0 ms
VehicleSpeedSensorVoter 10.0 ms
InjectionTimeActuation 10.0 ms
TotalFuelMass 10.0 ms
TransientFuelMass 10.0 ms
BaseFuelMass 10.0 ms
MassAirFlowSensor 5.0 ms
EcuWheelSpeedSensor 5.0 ms
WheelSpeedSensorDiagnosis 10.0 ms
WheelSpeedSensorTranslation 10.0 ms
WheelSpeedSensorVoter 10.0 ms
ABSCalculation 10.0 ms
BrakeForceActuation 10.0 ms
BrakeActuatorMonitor 10.0 ms
CaliperPositionCalculation 10.0 ms
BrakeActuator 10.0 ms
EcuBrakeActuator 5.0 ms
BrakePedalPositionVoltage1
BrakePedalPositionVoltage2
BrakePedalPosition1
BrakePedalPosition2
VotedBrakePedalPosition
BrakePedalPosition
MonitoredVehicleState
BrakeMonitorLevel
BrakeSafetyLevel
BrakeSafetyState
CalculatedBrakeForce
ArbitratedBrakeForce
BrakeApplication
DecelerationVoltage1
DecelerationVoltage2
DecelerationRate1
DecelerationRate2
VotedDecelerationRate
ThrottlePosition
APedPosition1
APedPosition2
VotedAPedPosition
DesiredThrottlePosition
IgnitionTime
MAFRate
TriggeredCylinderNumber
MassAirFlow
BaseFuelMassPerStroke
TransientFuelMassPerStroke
TotalFuelMassPerStroke
VehicleSpeedVoltage1
VehicleSpeedVoltage2
VehicleSpeed1
VehicleSpeed2
VotedVehicleSpeed
WheelSpeedVoltage1
WheelSpeedVoltage2
WheelSpeed1
WheelSpeed2
VotedWheelSpeed
ABSMode
ABSActivation
BrakeForce
BrakeForceCurrent
CaliperPosition
BrakeForceVoltage

Figure 5.3: "AutoAnalyze" showing a suggested partition for the "AMALTHEA DemoCar"



Generated export model

This page displays the export model that has been computed based on the given settings.

Name	Contents	Info	Utilization	Other
global ram	46 variable(s)	4096.0 kbytes		
cores	3 core(s)			
core_0	7 task(s)		33.34%	
local ram	0 variable(s)	472.0 kbytes	0%	
task_region_5	7 runnable(s)	0.0 ms/10.0 ms	37.82%	priority 2
task_region_2	2 runnable(s)	0.0 ms/10.0 ms	10.81%	priority 2
task_cluster_9	1 runnable(s)	0.0 ms/5.0 ms	21.65%	priority 3
task_cluster_5	1 runnable(s)	0.0 ms/5.0 ms	10.81%	priority 3
task_cluster_8	1 runnable(s)	0.0 ms/5.0 ms	10.81%	priority 3
task_cluster_13	1 runnable(s)	0.0 ms/10.0 ms	5.4%	priority 2
task_cluster_16	1 runnable(s)	0.0 ms/20.0 ms	2.7%	priority 1
core_1	7 task(s)		32.44%	
local ram	0 variable(s)	472.0 kbytes	0%	
task_region_6	6 runnable(s)	0.0 ms/10.0 ms	33.32%	priority 2
VehicleStateMonitor	1 instance(s)	0.266 ms/10.0 ms	16.67%	
CheckPlausability	1 instance(s)	0.266 ms/10.0 ms	16.67%	
BrakeForceCalculation	1 instance(s)	0.266 ms/10.0 ms	16.67%	
BrakePedalSensorVoter	1 instance(s)	0.266 ms/10.0 ms	16.67%	
BrakePedalSensorTranslation	1 instance(s)	0.266 ms/10.0 ms	16.67%	
BrakeSafetyMonitor	1 instance(s)	0.266 ms/10.0 ms	16.67%	
task_region_1	2 runnable(s)	0.0 ms/10.0 ms	11.11%	priority 2
task_cluster_4	1 runnable(s)	0.0 ms/5.0 ms	22.25%	priority 3
task_cluster_3	1 runnable(s)	0.0 ms/5.0 ms	11.11%	priority 3
task_cluster_7	1 runnable(s)	0.0 ms/5.0 ms	11.11%	priority 3
task_cluster_11	1 runnable(s)	0.0 ms/10.0 ms	5.55%	priority 2
task_cluster_14	1 runnable(s)	0.0 ms/10.0 ms	5.55%	priority 2
core_2	9 task(s)		34.22%	
local ram	0 variable(s)	472.0 kbytes	0%	
task_region_3	5 runnable(s)	0.0 ms/10.0 ms	26.32%	priority 2
task_region_7	3 runnable(s)	0.0 ms/10.0 ms	15.79%	priority 2
task_region_4	2 runnable(s)	0.0 ms/10.0 ms	10.53%	priority 2
task_cluster_1	1 runnable(s)	0.0 ms/5.0 ms	10.53%	priority 3
task_cluster_2	1 runnable(s)	0.0 ms/5.0 ms	10.53%	priority 3
task_cluster_6	1 runnable(s)	0.0 ms/5.0 ms	10.53%	priority 3
task_cluster_10	1 runnable(s)	0.0 ms/10.0 ms	5.26%	priority 2
task_cluster_12	1 runnable(s)	0.0 ms/10.0 ms	5.26%	priority 2
task_cluster_15	1 runnable(s)	0.0 ms/10.0 ms	5.26%	priority 2
constraints	6 constraint(s)			
requirements	7 requirement(s)			

Figure 5.4: “AutoAnalyze” showing a mapping solution for the “AMALTHEA DemoCar” on a predefined target platform

5.2 Real World Examples

In order to illustrate the practicability and benefit of the approach, it is applied to two real-life AUTOSAR models: the mid-sized sample "DemoCar" originating from the "AMALTHEA Project" and a part of a huge real-world Engine Management System from Continental ("Continental EMS"). The former consists of one SW-C containing 43 REs with 3 different recurrences, 71 variables/signals and 59 VAs (dependencies). The latter comprises 178 SW-Cs including 552 REs with 20 different recurrences, 11460 variables/signals and 45399 VAs.

5.2.1 EMS Characteristics

Engine Management Systems do perfectly embody the complexity and difficulties when trying to parallelize embedded automotive software: Being unique among already highly complex control applications, Engine Management Systems do largely lack big loops and data parallelism but stand out due to their inhomogeneity, high coupling, complex dataflow, strict consistency needs, continuously intense data exchange and their numerous REs (often more than 600) [BKL16, Sie16, MFCM16, PKQ+14].

Consequently, Engine Management Systems are rather hard to parallelize and therefore serve as ideal test object. In addition – and when neglecting multimedia – "the engine systems domain is the first one in automotive requiring an introduction of multi-core processors due to a lack of computing power" [MFCM16].

5.2.2 Applied Metrics

The following key figures are selected to represent the solution's quality:

- General validity: A solution is valid if the simulation proves that all basic timing requirements of the tasks are fulfilled, i.e., all tasks are fully processed before they are triggered again.
- Average latency: If compared to the best known other solution, the maximum response time for a whole model (derived from the latencies of its tasks) is a meaningful value revealing needlessly caused overhead. In addition, it gives a hint on the model's overall poten-

tial degree of parallelization.

- Communication overhead: As different cores execute tasks that depend on each other, a certain rate of “cross-core communication” is virtually always inevitable. Like the latency, this rate can be assessed relative to other solutions through comparison.
- Average core load: This indicates how uniform the division of processing work is (proper load balancing).
- Length of computing cycle: By comparing the time needed for one computing cycle on a single-core system with its runtime within a specific multi-core setting, it is possible to make a general statement about the potential performance gain which can then be used to run further applications or to increase the workload for the now parallelized application (according to “Gustafson’s Law” [Gus88]).

This measured data serves as basis of comparison for solutions calculated by the succeeding optimization.

5.2.3 Analysis, Partitioning & Mapping

First, the data dependency analysis tool “AutoAnalyze” (cf. Section 5.1) analyzes the structure, VAs and timing properties of the models. The subsequent partitioning step employs the “Split, then analyze” splitting strategy and conducts a low-coupling search – SERA with increasing tolerance and relevance partitioning weights – on partial models.

For the DemoCar, it is relatively easy to quickly find a near-to-optimal result as it is structured in a straightforward way and does not include many data dependencies (cf. Figure 5.3). The Continental EMS is significantly more complex, thus only a combination of rising tolerance values and onward coarsening allows to find reasonable partitions. The tool usually needs less than a minute[24] for the analysis and partitioning search when being run on customary laptops or desktop computers.

Two partitions are created for both models: one with rather small (“fine-grained”) RE sets and one with large (“coarse-grained”) ones. In order to enable a comparison to searching without a preceding data dependency analysis and partitioning/mapping, a simple partition is also included for both

[24]That applies, e.g., to a six-year-old “Dell Precision M6700 Mobile Workstation” laptop.

models: It assigns all REs with uniform period to one task/group (simulating a situation where no further knowledge about a system is available).

The mapping is geared towards an embedded platform featuring three cores (details below). The initial distribution follows a "bin packing" approach that estimates the specific RE set's core utilization and assigns it accordingly as task on the least busy core aiming at a proper load balancing.

In addition, process requirements are created that define fundamental deadlines for each task considering their recurrence. That means that the included REs' uniform period determines the whole task's maximum response time. They are mandatory for the purpose of classifying later found solutions into "valid" (viable) or "invalid" (unsuitable).

5.2.4 Scheduling & Simulation

The earlier mentioned TA Tool Suite is used to simulate and optimize the models as it provides the required import functions for both AUTOSAR system descriptions and task assignments provided via CSV files.

In order to achieve results that are as realistic as possible, the model of an "Infineon AURIX TC27x"[25] – a widely used microcontroller – is employed as hardware platform. This microcontroller is "designed for ultimate reliability in harsh automotive environments" [Inf15a]. It features three processing cores that can be regarded as homogeneous [GLI15, Inf15b].

In TATS, a separate scheduler is assigned to each core. A selection of algorithms is available for the scheduling of the tasks on one core. For this case study, either "EDF" or "AUTOSAR" is chosen as strategy for all cores within one test run. The last necessary adjustment is the mapping of signals/variables to the memory. The signals are initially assigned to the local memory of the core reading them. If multiple cores are involved, the signals are distributed equally across them.

Now, the simulation can be carried out. An execution of the system is simulated that lasts one second and delivers extensive feedback as well as the selected key figures. The crucial information is whether the basic requirements are met, i.e., if the hardware is – considering the given tasks, mapping and schedule – capable of executing the software fast enough to keep

[25]The "AUtomotive Realtime Integrated NeXt Generation Architecture" is a microcontroller family for the automotive sector featuring three independent 32 bit "TriCore" CPUs [Inf15a].

up with the recurring tasks.

5.2.5 Optimization

Based on this, the succeeding optimization is conducted to compare the initial partition/mapping with further possible solutions calculated by a Genetic Algorithm employed in TATS. Appropriate basic settings were conscientiously selected for the optimization:

- Simulation time: 1000 milliseconds
- Optimization goals:
  - maximum Normalized Response Time (mNRT)
  - Inter-Core Communication (ICC) rate
- Exploration size:
  - initial population size: 32
  - variation count: 16
  - selection count: 16
- Stop criteria:
  - "stagnation for 5 iterations"
  - similar ranges of "generations"

The most important setting is the granted "degree of freedom" when altering the initial solution. It is represented by selecting which modifications are allowed during optimization. The following options offered by TATS are used [Tim15]:

- "Runnable Sequencing" (RS) to "change the order of runnables inside call sequences"
- "Process Allocation" (PA) to "change the scheduler where processes are allocated to"
- "Task Splitting with Enforced Migration" (TS-EM) to "enforce migration of tasks to other schedulers/cores"
- "Task Splitting with Inter Process Activation" (TS-IPA) to "split tasks into several subtasks"

- "Automatic Task Parallelization" (ATP) to "partition tasks into several subtasks which run in parallel on different cores"
- "Periodic Stimulus Offset Assignment" (PSOA) to "change the Offset of Periodic Stimuli"

Among these, TS-EM and TS-IPA as well as TS-IPA and ATP are each considered mutually exclusive because they interfere with each other. With respect to that and in order to cover a broad search span, different "strategy sets" are applied to find preferably advantageous solutions. The steady basic setting is to allow RS and PA as well as to choose maximum allowed splitting/migrating values for either TS-EM or TS-IPA. The concrete sets arising out of this are:

1. allow RS, PA, TS-EM and ATP with optimization goal mNRT
2. allow RS, PA, TS-EM and ATP with optimization goal ICC
3. allow RS, PA, TS-IPA and PSOA with optimization goal mNRT
4. allow RS, PA, TS-IPA and PSOA with optimization goal ICC

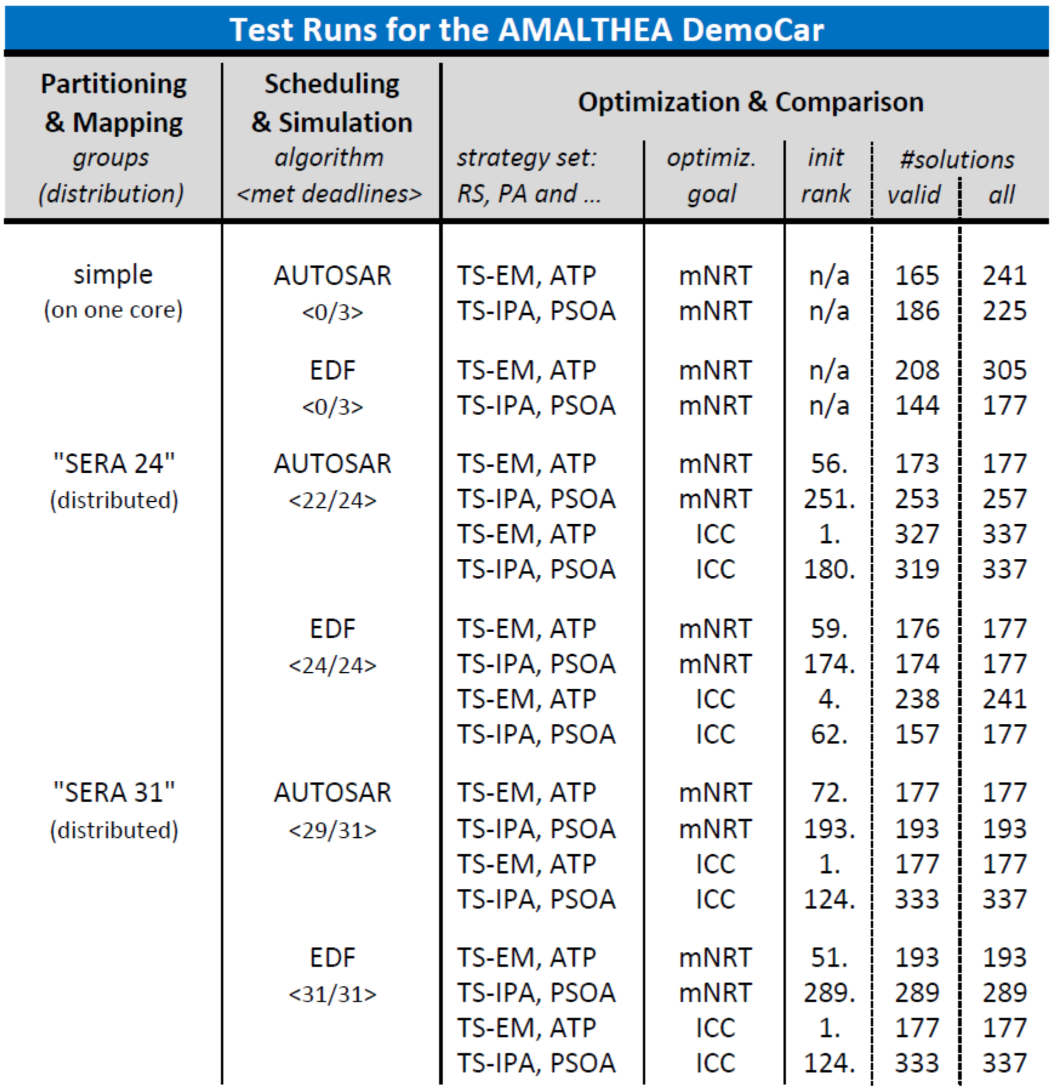
5.2.6 Results & Evaluation

The optimization of complex models like the Continental EMS can – due to the vast search space – take up to a whole day even with low exploration sizes and performed on a rather powerful laptop like the one employed: a Dell M6700 with Intel Core i7-3729QM, 8 GB RAM, SSD hard drive and Windows 10 64 Bit. The results include a variety of data and statistics. Among these the focus lies on the ones representing the key figures described in Subsection 4.4.2:

- The general validity is represented by entirely fulfilled "Process Requirements" (deadlines).
- The average latency is expressed through the maximum Normalized Response Time.
- The communication overhead is shown as Inter-Core Communication.
- The core load balance is indicated by the difference of the cores' individual "CPU Load (Utilization) average".

The results of the test run series are shown in Figure 5.5 ("DemoCar") and

Figure 5.6 (“Continental EMS”).



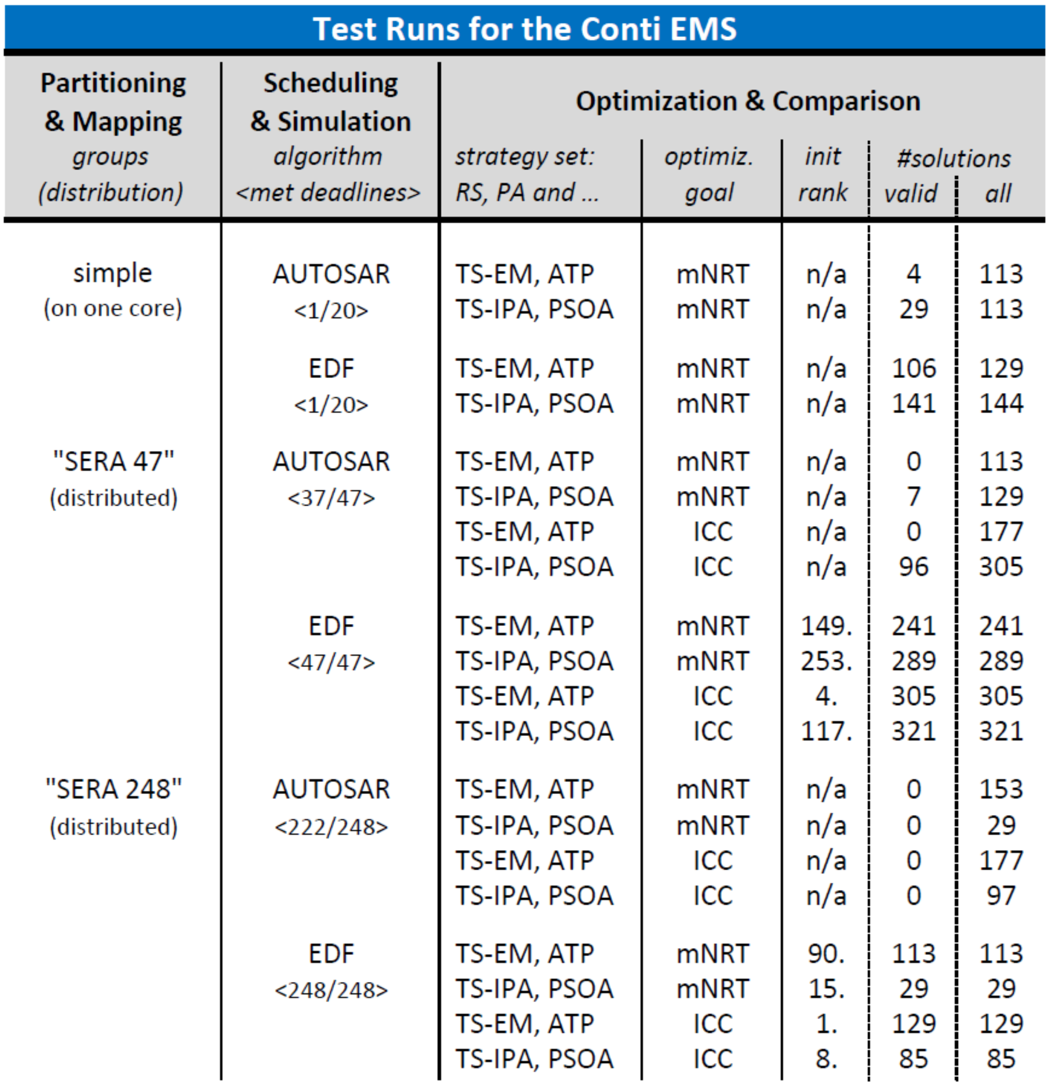
Test Runs for the AMALTHEA DemoCar

Partitioning & Mapping	**Scheduling & Simulation**	**Optimization & Comparison**				
groups (distribution)	*algorithm <met deadlines>*	*strategy set: RS, PA and ...*	*optimiz. goal*	*init rank*	*#solutions valid*	*all*
simple (on one core)	AUTOSAR <0/3>	TS-EM, ATP	mNRT	n/a	165	241
		TS-IPA, PSOA	mNRT	n/a	186	225
	EDF <0/3>	TS-EM, ATP	mNRT	n/a	208	305
		TS-IPA, PSOA	mNRT	n/a	144	177
"SERA 24" (distributed)	AUTOSAR <22/24>	TS-EM, ATP	mNRT	56.	173	177
		TS-IPA, PSOA	mNRT	251.	253	257
		TS-EM, ATP	ICC	1.	327	337
		TS-IPA, PSOA	ICC	180.	319	337
	EDF <24/24>	TS-EM, ATP	mNRT	59.	176	177
		TS-IPA, PSOA	mNRT	174.	174	177
		TS-EM, ATP	ICC	4.	238	241
		TS-IPA, PSOA	ICC	62.	157	177
"SERA 31" (distributed)	AUTOSAR <29/31>	TS-EM, ATP	mNRT	72.	177	177
		TS-IPA, PSOA	mNRT	193.	193	193
		TS-EM, ATP	ICC	1.	177	177
		TS-IPA, PSOA	ICC	124.	333	337
	EDF <31/31>	TS-EM, ATP	mNRT	51.	193	193
		TS-IPA, PSOA	mNRT	289.	289	289
		TS-EM, ATP	ICC	1.	177	177
		TS-IPA, PSOA	ICC	124.	333	337

Figure 5.5: Test run results for the “AMALTHEA DemoCar”

The included tables show how different combinations of specific partitions, mappings, scheduling algorithms for the simulation and optimization strategies are used to search for advantageous solutions. The rows can be interpreted as follows:

- Partitioning & Mapping:
  - “Simple” indicates a grouping without further knowledge as basis of comparison whereas “SERA X” denotes partitions and



Test Runs for the Conti EMS

Partitioning & Mapping	Scheduling & Simulation	Optimization & Comparison				
groups (distribution)	*algorithm <met deadlines>*	*strategy set: RS, PA and ...*	*optimiz. goal*	*init rank*	*#solutions valid*	*all*
simple (on one core)	AUTOSAR <1/20>	TS-EM, ATP	mNRT	n/a	4	113
		TS-IPA, PSOA	mNRT	n/a	29	113
	EDF <1/20>	TS-EM, ATP	mNRT	n/a	106	129
		TS-IPA, PSOA	mNRT	n/a	141	144
"SERA 47" (distributed)	AUTOSAR <37/47>	TS-EM, ATP	mNRT	n/a	0	113
		TS-IPA, PSOA	mNRT	n/a	7	129
		TS-EM, ATP	ICC	n/a	0	177
		TS-IPA, PSOA	ICC	n/a	96	305
	EDF <47/47>	TS-EM, ATP	mNRT	149.	241	241
		TS-IPA, PSOA	mNRT	253.	289	289
		TS-EM, ATP	ICC	4.	305	305
		TS-IPA, PSOA	ICC	117.	321	321
"SERA 248" (distributed)	AUTOSAR <222/248>	TS-EM, ATP	mNRT	n/a	0	153
		TS-IPA, PSOA	mNRT	n/a	0	29
		TS-EM, ATP	ICC	n/a	0	177
		TS-IPA, PSOA	ICC	n/a	0	97
	EDF <248/248>	TS-EM, ATP	mNRT	90.	113	113
		TS-IPA, PSOA	mNRT	15.	29	29
		TS-EM, ATP	ICC	1.	129	129
		TS-IPA, PSOA	ICC	8.	85	85

Figure 5.6: Test run results for the "Continental EMS"

mappings found by the approach including a number of "X" tasks (RE sets).

- "On one core" acts as initial mapping for the "simple" partitioning whereas "distributed" follows the approach of low coupling and load balancing.

- Scheduling & Simulation: "AUTOSAR" and "EDF" (cf. Subsection 3.3.2) are the employed scheduling algorithms and the ratio below represents the validity of the initial solution.
- Optimization & Comparison: Here, the settings and strategies of the

specific optimization run are stated together with the rank of the initially calculated solution among the total number of determined valid as well as all solutions.

The additional ICC optimization runs were only conducted if the initial solution was not assessed as "predominantly invalid", i.e., violating more than 50 % of the deadlines. Thus, they were left out for both models when the "simple" partitioning/mapping solution was evaluated. This also applies for the rank of the initial solution column "init rank" which is not available if it cannot be compared to valid ones. Within the latter, a rank is determined by TATS via comparing the solutions' specific "fitness" – a value reflecting the calculated goal achievement.

Regarding the "length of computing cycle" key figure from Subsection 5.2.2, it can be stated that the Infineon AURIX TC27x is not able to run the EMS on a single core without violating deadlines as the calculation of one computing cycle takes more time than available. By contrast, there is 66 % at free disposal after the parallelization process which distributes the calculations across three IEUs.

5.2.7 Conclusion

In order to sum up the outcome of the test runs, it can be stated that the approach – especially its focal point described here – contributes to ...

- ... avoiding most adverse starting points where many basic deadlines are violated. This is illustrated by comparing the simulation results of "simple" partitions/mappings to those calculated by the approach ("SERA X").
- ... quickly finding promising starting points for optimization when primarily aiming at low response times.
- ... quickly finding remarkably advantageous starting points for ICC-optimized solutions, where the EDF-scheduled initial solutions are all valid right from the start in this case study.
- ... generally challenge the existing software structure (such as the assignment of REs to SW-Cs) by analyzing, partitioning and mapping directly on RE level and independent from their given assignment to SW-Cs.

The time saved by providing a viable initial solution comes even more into effect when the exploration size is increased and optimization durations are – as a consequence of exponential growth – multiplied, e.g., when several days of optimization do not result in a – according to a certain goal – considerably better solution than the initial one that was created within minutes.

In this case study, the specific settings were chosen in order to compare preferably many and diverging partitioning/mapping solutions. Distinctly bigger exploration sizes for the optimization are possible with a significantly increased time exposure or by utilization of high performance computing resources.

5.3 In-Depth Optimizations and Evaluation

To demonstrate the added value of the refined methods, a previous case study (cf. [KSKB16]) is substantially expanded by applying the approach to two complex Engine Management Systems and by showing in-depth arising advantages compared to a parallelization process without preceding dependency analysis and initial partition/mapping suggestions.

The following hypothesis is stated: An optimization algorithm will yield significantly better results compared to a predefined initial solution in the same given time if a preceding dependency analysis and the resulting initial partitioning/mapping is used as a starting point.

For each of the two Engine Management Systems, the following experiment was conducted:

1. Definition of a "reference solution" (i.e. reasonable initial solution) in terms of partitioning, mapping and OS configuration.
2. Optimization I: Creating alternative solutions using "TA Optimizer" (part of TATS) [Tim15].
3. Optimization II: Using "AutoAnalyze" to provide the starting point for subsequently creating alternative solutions using "TA Optimizer".
4. Comparing the relative improvements to the reference solution yielded by Optimization I/II.

5.3.1 Setup

As mentioned before, the following two complex AUTOSAR models are used:

The first one is a part of a huge real-world EMS from Continental ("Conti EMS"), which consists of 178 SW-Cs including 552 REs with 20 different recurrences, 11460 variables/signals and 45399 data dependencies (each arising from a write and according read access on a specific variable).

The second one is "a full blown [sic] performance model of a modern EMS" ("Bosch EMS") [HZKL16], which is publicly available as AMALTHEA model for the "FMTV Verification Challenge" of the "WATERS" workshop [WAT16, Qui16]. It comprises 1250 REs with 11 different recurrences, 9983 variables/signals and 5195 data dependencies. It was converted to

AUTOSAR for being able to apply to the working steps.

Hereinafter, all made configurations are described that are necessary for carrying out the experiment steps mentioned before.

5.3.1.1 Reference Solution

The adjustments below have been made to the Conti EMS and the Bosch EMS models to create the aforementioned reference solutions:

- Conti EMS: Since the AUTOSAR description only contains the SW-Cs, REs and variables, the following initial partitioning, mapping and OS configuration were conducted. Moreover, the "Infineon AURIX TC27x" [Inf15a] simulation model was used as hardware description. The clock frequency of the three cores is set to 200 MHz.
  - Partitioning: One task per recurrence has been created which executes all REs belonging to it.
  - Mapping: The tasks have been mapped to the cores using a typical separation scheme for different recurrences.
  - OS configuration: Each core is managed by one AUTOSAR scheduler. Priorities have been assigned using the "rate monotonic scheme" [LW82] (i.e. the shorter the recurrence, the higher the priority) and each task was configured to be fully preemptive.
- Bosch EMS: The existing configuration is used as reference solution, as the Bosch EMS system already contains the complete partitioning and mapping information as well as the operating system configuration. However, the clock frequency of all four cores was increased from 200 MHz to 1 GHz to prevent scheduling errors as the analysis at experiment setup had shown that the system is basically not schedulable with 200 MHz.

5.3.1.2 Experiment Configuration

This case study consists of twelve experiments in total. Their configurations are itemized in Table 5.1. All experiments without an AutoAnalyze configuration indicate that TA Optimizer has been solely used to create alternative solutions. Some general configuration parameters for simulation and optimization are equal for all experiments: The per-solution simulation time is set to 5 seconds, the population size was set to create 32 solutions for the initial population and 16 new solutions for each subsequent iteration. The selection size was set to keep the 16 best solutions according to fitness and discard the remaining ones. For the stop criterion, a fixed value of 256 alternative solutions is used.

Table 5.1: Lineup of the different experiments performed within the scope of this case study

Experiment Identifier	AUTOSAR Model	Optimization Goal	AutoAnalyze Config.	Optimizer Config.
Exp-1	Conti EMS	Goal-1a	-	TAOPT-1
Exp-2	Conti EMS	Goal-1b	-	TAOPT-1
Exp-3	Conti EMS	Goal-1c	-	TAOPT-1
Exp-4	Conti EMS	Goal-1a	AA-1	TAOPT-2
Exp-5	Conti EMS	Goal-1b	AA-1	TAOPT-2
Exp-6	Conti EMS	Goal-1c	AA-1	TAOPT-2
Exp-7	Conti EMS	Goal-1a	AA-2	TAOPT-2
Exp-8	Conti EMS	Goal-1b	AA-2	TAOPT-2
Exp-9	Conti EMS	Goal-1c	AA-2	TAOPT-2
Exp-10	Bosch EMS	Goal-2	-	TAOPT-3
Exp-11	Bosch EMS	Goal-2	AA-1	TAOPT-4
Exp-12	Bosch EMS	Goal-2	AA-2	TAOPT-4

5.3.1.3 Optimization Goals

All of the four optimization goals are detailed in Table 5.2 and denote the minimization of one single criterion or the simultaneous minimization of multiple criteria, respectively. They are adopted from real-life projects as well as industrial cooperations and are therefore considered to have high practical relevance.

5.3.1.4 AutoAnalyze Configuration

The two different AutoAnalyze configurations – as mentioned in Table 5.1 – are stated in the following.

Table 5.2: Lineup of the different optimization goals stated in Table 5.1

Optimization Goal	Metric	Weight	Lower Limit	Upper Limit
Goal-1a	ICC	1	0	24.34 MBit/s
Goal-1b	ICC	1	0	24.34 MBit/s
	Maximum Load Distance	1	0	20 %
Goal-1c	mNRT	10	0	2
	Buffer Size	5	0	17.88 kB
	Maximum Load Distance	5	0	20 %
Goal-2	Event Chain Duration (EffectChain1)	1	0	94.60 ms
	Event Chain Duration (EffectChain2)	1	0	601.3 ms
	Event Chain Duration (EffectChain3)	1	0	12.50 ms

- AA-1:
  - partitioning: rather small groups, relatively high search tolerances (20 for Bosch EMS, 30 for Conti EMS)
  - mapping: bin packing, preferably equal distribution (totally equal for Bosch EMS, roughly equal for Conti EMS)
- AA-2:
  - partitioning: rather large groups, relatively high search tolerances (10/20 for Bosch EMS, 30 for Conti EMS)
  - mapping: bin packing, preferably equal distribution (totally equal for Bosch EMS, rather unbalanced for Conti EMS)

In order to effectively compare strongly differing solution granularities, diametrically opposed RE set sizes were chosen. As a consequence, the maximum search tolerances for the Bosch EMS as well as the load balancing quality of the Conti EMS were affected.

5.3.1.5 TA Optimizer Configuration

There are four different TA Optimizer configurations used in this case study. They are distinct from each other regarding the applied design modifications as introduced in 4.4.3.3.

- TAOPT-1: Process Mapping, Task Splitting and Periodic Offset Assignment
- TAOPT-2: Process Mapping and Periodic Offset Assignment
- TAOPT-3: Process Mapping, Task Splitting, Periodic Offset Assignment and Data Mapping
- TAOPT-4: Process Mapping, Periodic Offset Assignment and Data Mapping

The selected combinations have proven to yield particularly advantageous results and can be considered "best practices".

Task Splitting is only used in the experiments where AutoAnalyze was not employed to provide the partitioning for the initial solution. Further splitting the tasks of a fine-grained task set would lead to an unnecessary increase of the already vast search space.

5.3.2 Results

The results of the experiments described in the previous section are shown in Table 5.3. For each experiment, the fitness of the reference solution, the fitness of the best alternative solution and the relative fitness improvement of the best alternative solution are compared. Moreover, the added value (additional improvement by AutoAnalyze) is provided.

All experiments – with the exception of Exp-10 – yield a significant (around 50 % or greater) improvement compared to the reference solution. AutoAnalyze always resulted in an additional improvement compared to the respective experiment where TA Optimizer was solely used to create alternative solutions. The highest improvement by AutoAnalyze was achieved with experiments Exp-11 and Exp-12.

5.3.3 Evaluation

As stated in Subsection 2.2.6, full-scale optimizations usually consist of several ten thousand alternative solutions. However, the optimizations were configured to produce only 256 solutions. This setting represents a typical "potential exploration" in order to evaluate rather quickly to what extent an initial solution can be improved. This is due to the fact that simulations of complex systems like an EMS are quite costly in terms of runtime – especially when detailed simulation models for the hardware are used. Therefore, the goal is to save time and resources by first evaluating the improvement potential before starting a full-scale optimization.

Each experiment conducted in the case study took around 16 hours to complete on a computer with a "Intel Core i7-2930K" processor (6 cores, up to 12 simultaneous threads) with a clock frequency of 3.2 GHz and 16 GB RAM. The TA Optimizer was configured to use 5 out of the 6 cores to run up to 10 simulations in parallel. AutoAnalyze on the contrary only requires a few seconds to provide an appropriate initial partitioning and mapping on such a computer.

Since AutoAnalyze led to an additional improvement for every experiment, the initially stated hypothesis is fulfilled. In case of the experiments with the Bosch EMS system, the combination of AutoAnalyze and TA Optimizer could improve the reference solution more than twice as much within the same given time compared to the experiment where TA Optimizer was solely used.

In addition, the following particular observations were made:

- Comparing the experiments 4–6 as well as 7–9 on the Conti EMS: The biggest improvement could be achieved when ICC was exclusively employed as optimization goal (“Goal-1a”). This did not come unexpected as SERA deliberately focuses on low coupling and pooling, respectively (cf. correlation with other optimization goals in Subsection 4.3.3). It is notable that the coarse-grained partition (“Exp-7”) led to a distinctly better solution (higher fitness) than the fine-grained one (“Exp-4”). This outcome can mainly be ascribed to the fact that the design space is remarkably reduced if there are less software parts to distribute.
- Referring to the same set of experiments, they yielded significantly better results when aiming at the “Goal-1c” (double mNRT, Buffer Size and Load Distance) than at “Goal-1b” (evenly ICC and Load Distance). By all indications, SERA seems to be more effective for minimized ICC and response times than for balancing loads.
- The experiment series 10–12 illustrates how the duration of event chains, i.e., the time span from input to reaction, can be tremendously shortened while causing only minimal overhead (additional waiting times and synchronization effort).
- TATS delivered acceptable results despite rather adverse starting points when no preceding analysis, partitioning and mapping was conducted. Obviously, the GA was able compensate the missing knowledge about the model and the enforced drawing on simple heuristic approaches.

Table 5.3: Results of the performed experiments

Experiment	Fitness (reference)	Fitness (best alternative)	Fitness Improvement (best alternative) [%]	Additional Improvement by AutoAnalyze [%]
Exp-1	0.04000	0.007394	81.51	-
Exp-2	1.719	0.08156	95.25	-
Exp-3	43.14	3.944	90.85	-
Exp-4	0.04000	0.007394	93.38	11.86
Exp-5	1.719	0.08156	97.67	2.410
Exp-6	43.14	3.944	97.96	7.100
Exp-7	0.04000	0.007394	99.18	17.66
Exp-8	1.719	0.08156	98.10	2.850
Exp-9	43.14	3.944	97.70	6.840
Exp-10	0.06928	0.05366	22.54	-
Exp-11	0.06928	0.03097	55.29	32.75
Exp-12	0.06928	0.03784	45.38	22.83

6 Conclusion

This chapter finalizes the thesis by outlining its contents, describing its added value and delineating potential future development.

First, Section 6.1 gives a summary of the central aspects covered by the thesis. Subsequently, Section 6.2 determines to which extent the initially formulated objectives (cf. Subsection 1.3) have been achieved. Afterwards, Section 6.3 states vital factors and prevalent trends shaping the current development and nearby future of the automotive domain. Finally, Section 6.4 makes concrete proposals to purposefully extend the work presented in this thesis.

6.1 Summary

Due to the inevitable complexity associated with migrating single-EU legacy ECU software for a proper execution to multiple-IEU platforms, innovative methods and approaches are urgently needed.

With the objective of enabling an efficient parallelization of AUTOSAR application software on function level, a tool-assisted systematic approach and methodology is introduced that supports software engineers when analyzing, verifying, validating, partitioning and mapping AUTOSAR models.

This process narrows down the search space for the following working steps scheduling, simulation and optimization which are carried out to determine advantageous solutions in terms of low overall latency, minimal cross-core communication rates as well as proper load balancing.

The approach is designed to detect and solve potential consistency conflicts as well as structural model deficiencies right from the outset. Based on this, the parallelization of AUTOSAR models is fundamentally facilitated by automatically providing both concrete partitions and mappings which eventually results in considerably reduced search effort.

In order to verify the benefit of the approach, it is implemented and employed in three case studies. The first of them demonstrates the technical feasibility by transferring the methodology into a tool that effectively supports the parallelization via graphical visualization, provided editing features and integrating partitioning and mapping approaches (cf. Section 5.1). The second case study shows the practicability of the proceeding specified by the methodology as well as the determined added value when being employed on real-world examples (cf. Section 5.2). The third one involves refined algorithms and industry-related optimization goals applied on two complex Engine Management System models as extended series of experiments yielding results with augmented expressiveness (cf. Section 5.3).

The case studies show that a preceding data dependency analysis combined with a skillful partitioning and mapping (that builds on its outcome) is able to significantly enhance the solution quality while reducing the required time for finding it.

In conclusion, it can be stated that the automotive sector's demands are rapidly rising. Moreover, it is already evident that even many-core technology becomes progressively common, e.g., as seen in a growing number of cores with distributed memories or heterogeneous connectivity [Mac15].

Therefore, the approach of this thesis serves as valuable starting point for coping with the challenges that arise from this development.

6.2 Achievement of Objectives

The overarching goal of this thesis is to facilitate the deployment of a parallelized real-time automotive applications by effectively supporting the migration of single-EU software to multiple-IEU platforms while preserving its original behavior and ensuring correctness with respect to data consistency.

From this goal, three challenges were derived which again led to three concrete objectives (cf. Section 1.3) whose achievement is discussed in the following three subsections. Three contributions are realized following the created methodology and with the help of the tool ("AutoAnalyze") instantiating its defined proceeding.

6.2.1 Identification of Vital Elements as Analysis Basis

The first objective – an analysis yielding all necessary structural and timing information – is achieved by the data dependency analysis conducted on the most fine-grained level of detail relating to AUTOSAR system descriptions. Hereby, all timing-relevant elements are detected, which enables to determine dependencies as well as to eventually filter out potential data consistency conflicts among them.

A prerequisite for finding data consistency threats is fulfilled by hinting at structural problems in order to ensure the completeness and soundness of the analyzed model. The succeeding data consistency analysis can then follow the basic rule to find every not distinctly specified contingency which could eventually lead to unintentional timing behavior. The obtained analysis results serve as basis for the visualization within AutoAnalyze and are essential for further working steps.

6.2.2 Correctness and Data Consistency

The second objective – the verification and data validation – is achieved by two actions within the methodology's systematic proceeding: Firstly, hinting at structural deficiencies ensures a sound working basis. Secondly, multiple-IEU robustness is reached via incremental and/or pattern conflict solving which gradually eliminates every potential threat to data consistency. This process is carried out via the goal-oriented imposition and modification of timing constraints according to a clear rule set that is intended to validate the parallel execution while preserving as much "freedom" as possible concerning the subsequent partitioning and mapping. Again, this working step is tool-supported, thus enabling automated conflict solving as well as handling models of virtually any size and complexity.

6.2.3 Partitioning, Mapping and Granularity

The third objective was finding efficient solutions for the NP-hard partitioning problem and an according advantageous mapping. The former is achieved by providing a scalable and configurable approach for quickly finding beneficial partitions including suitable (and flexible) granularities of the contained parts. The latter is effectively enabled by the employed mapping procedure that takes heed of the number of available IEUs as well as the expected workloads when assigning the identified tasks to a target platform in an expedient manner.

As system structures heavily vary according to a software's specific purpose (and the domain it is originating from), both steps are designed focusing on adaptivity. There are, e.g., different splitting strategies that can be employed according to a certain model type in order to support the subsequent partitioning search.

In the end, the impending synchronization overhead caused by distributed execution is minimized by an efficient proceeding that avoids expensive "brute force methods" and can be adapted for application in different contexts (e.g. other models) with manageable effort.

6.3 Outlook

This section depicts clear directions of development within the class of embedded automotive software that do already have a distinct effect on the software engineering process and whose relevance is expected to intensify. Thus, research work within this domain should be carried out with respect to the following aspects.

6.3.1 Main Drivers

In [RN16], the authors name some "main drivers" that neither OEMs nor suppliers can evade.

One of the most much-noticed is "Autonomous Driving", which involves developing new hardware and software topologies including fail-operational systems, cross domain computing platforms as well as high-performance micro-controllers [Ebe16, YPS13, RN16]. This is necessary to provide a sound basis for, e.g., deep learning approaches that are gradually superseding "traditional" camera vision because of their superiority concerning detection rates [Lan16].

A correlating trend are so-called "Car-2-X" applications denoting the data exchange between a vehicle and external systems like other cars, traffic lights or cloud services. Here, typical use cases are software updates "over-the-air" (like already performed by Tesla[26]) or increasing safety by warning against traffic congestion or oncoming accident sites [Ebe16, Lan16].

These two general drivers induce the evolution of concrete technologies like employing "Ethernet" as central bus (due to its high bandwidth and efficient point-to-point communication) and developing flexible heterogeneous architectures as well as processors with increased performance ("multi-cores" and "many-cores") [RN16, Deu16].

In a nutshell, it is legitimate to state that "future cars will be architected as a coherent computer" [Lan16].

[26]cf. `https://www.tesla.com/en_GB/support/software-updates?redirect=no`

6.3.2 Heterogeneous Architectures

In order to lay the foundations for features like “Autonomous Driving” or “Car-2-X”, the need for more processing power has led to the solution of utilizing multiple IEUs. As the “multi-core introduction is at the origin of a big evolution of architectures” [MFCM16], advancements in hardware topologies are in full swing too. More specifically, “the trend towards heterogeneous multi-core architectures is inexorable” [Cor13] because of the different requirements that come along with the variety of running applications.

For example, it is sensible that highly safety-critical hard real-time applications are executed on a real-time IEU while soft real-time applications (e.g. infotainment services) can be run on a faster IEU that does not ensure to calculate the specific results within a fixed time span [RTS16]. Therefore, safe (hard real-time) processing is distinctly separated from high-performance processing [May16].

The whole concept behind this is also often referred to as “Asymmetric Multiprocessing” (AMP) in order to distinguish it from “Symmetric Multiprocessing” (SMP) where tasks or processes are assigned to identical multipurpose IEUs [RTS16].

6.3.3 Distributing and Merging Applications

In the course of using multiple IEUs, one central advantage is that several IEUs of one ECU can now execute applications that were once distributed over different single-EU ECUs (“integration”) as well as spreading the processing of an application across several IEUs due to safety or parallelization reasons (performance or migration) [YPS13].

The former “is becoming increasingly widespread in the automotive industry” [MNBSL12] as there is a clear trend towards more centralized architectures [Ebe16]. Consequently, the “old” paradigms (one function per ECU) are forced to be jettisoned in favor of new design patterns [YPS13].

The latter is especially important with respect to variability and continuing use of legacy software: In the automotive sector, the “global market with regionally very heterogeneous requirements leads to variety of variants” due to “country-specific rules on emissions, fuel consumption, admission and safety requirements” [IAS15]. In addition, many OEMs offer strongly customizable vehicles which further increases the number of variants.

Taking this situation into account requires that different variants of, e.g., powertrain applications can be properly executed on varying hardware platforms. Therefore, both hardware (cf. preceding Subsection 6.3.2) and software have to be flexible. In terms of software, this means that its proper function is not bound to a specific hardware setup which is facilitated by splitting it up in separately schedulable and executable tasks.

6.3.4 Model-Based Collaborative Development

One impact of the rising complexity and the increasing degree of interconnection is that the OEMs and suppliers have to work together more closely [Ebe16]. In complex systems, several partners contribute software parts in different formats to one common application, so that it is essential to find methods which enable an efficient collaborative development [MFCM16].

In order to master the complexity and to provide an abstracted view on the specific hardware and software components (as well as their timing behavior), model-based development approaches are on the advance (cf. [GHKF11]) and are about to replace the "traditional vehicle development process" [OLKY05].

According processes, e.g., like described in [SSH$^+$16] and [MFCM16], are advantageous regarding testing, evaluation and documentation, which can result in a reduction of development effort [OLKY05]. Such a proceeding is urgently needed when being confronted with significantly shorter software development cycles as well as an increased need for associated V&V activities [Gra17]. This is achieved by sticking to a common development methodology and by using models to ensure correct timing and data consistency as well as to obtain specific views on the system, e.g., a static architecture view or a dynamic dataflow view [MFCM16, Ebe16].

6.3.5 Adaptive AUTOSAR

Already in the year 2013, the authors of [AK13] stated that there is a need for a "dynamic component model that extends an AUTOSAR based control unit" enabling a (dynamic) plug-in mechanism for external applications and therefore opening the market for third-party developers.

As AUTOSAR is designed to particularly address real-time requirements and safety criticality, its applicability for, e.g., an infotainment application (no real-time, less safety, much computing power) is by implication limited [RN16].

Facing virtually inevitable trends like autonomous driving, "vehicle-to-x" (communication of the vehicle with its surroundings), "over-the-air" updates, electrification, dynamic deployment (relocation of functionality) or domain controllers, it becomes apparent that novel approaches are needed to handle the associated tasks which are often computationally intensive [OG18, Gra17].

As a consequence, the "AUTOSAR Adaptive Platform" was defined as different software platform instance that complements the "Classic Platform" [Für16]. Its goals are to support the flexible software development for centralized target platforms and the establishment of a standardized middleware for the intra-vehicle communication that bridges the gap between "deeply embedded" software and infotainment applications [OG18]. Table 6.1 contrasts crucial properties of the two platforms based on [Asm17, RN16, Für16, OG18, Gra17].

Table 6.1: Comparison of AUTOSAR's two software platform instances "Classic Platform" and "Adaptive Platform"

Classic Platform	**Adaptive Platform**
Runnable-based (REs)	thread-based
based on "OSEK"	based on "POSIX"
applications use same (shared) address space	each application has its own (virtual) address space
code execution directly from ROM	load application from persistent memory into RAM
optimized for signal-based communication, e.g., via CAN	service-oriented communication, e.g., via "SOME/IP"
fixed (static) task configuration	dynamic scheduling and run-time configuration
"deeply embedded" software	resource-intensive functionality

As described, the Adaptive Platform relies on POSIX[27] as opposed to the Classic Platform that is grounded on OSEK[28] specifications. Besides using service-oriented communication (SOME/IP[29]), the fundamental difference of the Adaptive Platform is its dynamic character (scheduling and deployment) as well as the "interaction with non-AUTOSAR systems" [Für16].

The main advantages coming along with this approach are (according to [Für16, RN16, OG18, Gra17]):

- Software development, e.g., for autonomous driving, is facilitated due to support of POSIX and service-oriented communication.
- The flexibility is increased as non-automotive standards can be reused, e.g., "libraries from the areas of high performance computing, embedded vision and machine learning" [OG18].
- The principle of service-orientation supports chain modeling, exchangeability and hierarchy due to the self-contained nature of its building blocks.
- As a consequence of employed external libraries and further decoupling software and hardware, application release cycles and included V&V activities can be expedited.

In conclusion, Figure 6.1 illustrates the Adaptive Platform's role as AUTOSAR's connector to infotainment systems in consideration of the three vital aspects real-time, safety and computing power.

6.3.6 Dynamic Integration, Allocation and Scheduling

Dynamic integration concepts do certainly have great potential. As stated in [OG18], enabling dynamic scheduling can result in a performance increase for applications. However, predicting the according temporal behavior is significantly harder than for static scheduling. Therefore, dynamic integration concepts are needed for handling the sequencing of program execution

[27]The "Portable Operating System Interface" is a "standardized programming interface between the application and the operating system" [OG18].

[28]"Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug" is a standards body which passed a series of specifications "for an embedded operating system, a communications stack, and a network management protocol for automotive embedded systems" [AUT18a].

[29]The "Scalable service-Oriented MiddlewarE over IP" is a communication protocol that connects AUTOSAR's Classic with the Adaptive Platform [AUT18b].

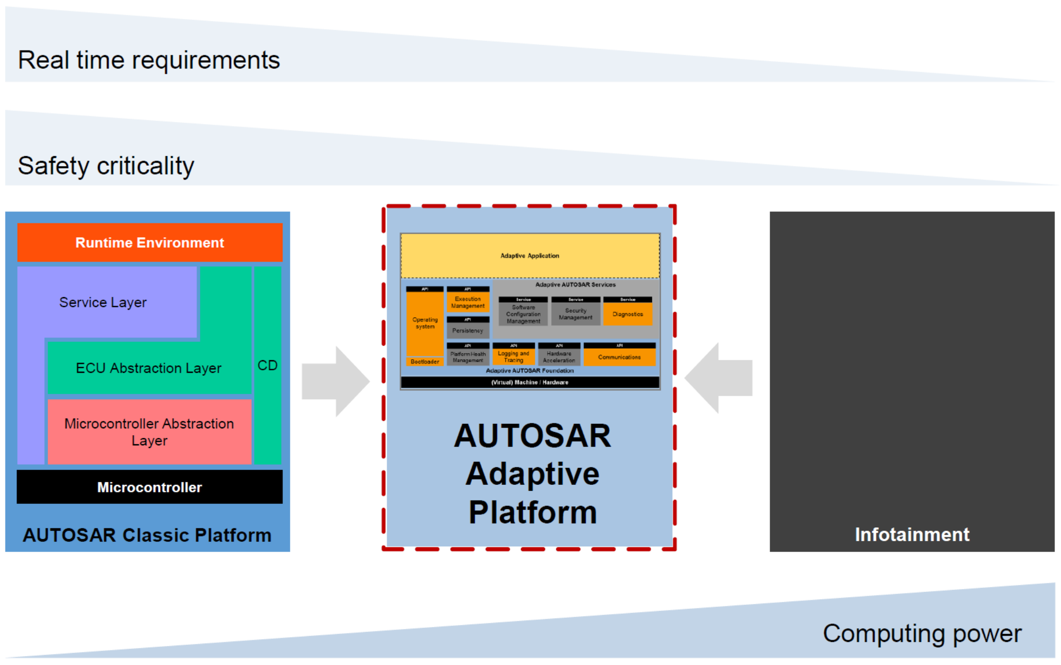


Figure 6.1: Categorization of AUTOSAR software platforms (from [RN16])

by specifying according data consistency and timing requirements [Deu16]. Here, the goal is to depict the "dynamic requirements for the integration of the runnables into existing tasks" [MFCM16] in an unambiguous manner, which inevitably results in a certain amount of overhead [SSH$^+$16].

Furthermore, there are some additional programming requirements that most legacy applications do not consider since the function design was not adjusted to independence of core and memory distribution [MFCM16, ZZZ$^+$12]. As a consequence, the desired high degree of flexibility can only be reached if the function development is geared to "original requirements" like protection and integration needs instead of specific implementation issues like specific buffering mechanisms [MFCM16]. The authors of [SSH$^+$16] point out that – if done properly – dynamic allocation will lead to high efficiency and overcoming volatile execution times.

6.4 Future Work

Following the approach and applying it via tool support is an important first step towards making the complexity of particularly huge models manageable. However, sticking to some basic principles is not a panacea, as the way of presenting data and offering editing functionality are just as important as gathering it in the first place.

6.4.1 Extending Existing Functionality

Therefore, one crucial point is to offer efficient filtering techniques that enable software engineers to quickly find and concentrate on decisive spots of the system design. This is what is pursued by implementing the filtering options mentioned in Section 5.1.

This process is ongoing as it is intended to, e.g., include a coarse-grained view that shows the SW-Cs, hides their respective REs and intra-SW-C communication, re-assesses the dependencies between SW-Cs concerning potential multiple-IEU conflicts and therefore offers similar analysis and editing functionality on a higher level of abstraction. In addition, this coarse-grained view could be modified to separately analyze SW-Cs with different purposes like "service", "diagnostics" or "regular functions", making it possible to distinctly exclude certain "SW-C types" from the analysis.

Another important factor is whether the model editing (and conflict solving) process enables fast and easy modifications. Gained experiences show that it is usually necessary to care about the vast majority of existing dependencies in order to ensure multiple-IEU robustness. For huge models, manually processing each single dependency is cumbersome or even not feasible as a system can easily contain several tens of thousands of edges (cf. Subsection 5.3.1). Hence, the possibility to apply multiple modifications at once is significant.

This is implemented prototypically by providing options to solve potential conflicts of the same kind all at once, e.g., multiple cases with missing ACs when EOCs are not applicable. It is planned to extend this by making solution strategies configurable. This can be achieved by following user-defined rule-sets that assign specific actions for different situations and certain RE sets or types, which could be realized with the help of, e.g., “Xtext”[30].

Such a rule-set equals an assignment of a conflict type to a specific solution, optionally only valid for certain parts of the model. An example for AUTOSAR is given in Table 6.2.

Table 6.2: Example rule-set for data validation in AUTOSAR

Conflict	**Solution**	**Scope**	**Priority**
invalid EOC	replace EOC by AC with appropriate time value	global	9 (highest)
missing EOC	add missing EOC for involved REs	global	8
cyclic EOCs	delete according to main flow direction	local (SW-C)	7
cyclic EOCs	randomly delete one	global	6
missing AC	add missing AC with minimum time value	global	5
insufficient AC	adjust time value (shorten or lengthen)	global	4
redundant EOCs	ignore EOC “twins”	local (SW-C)	3
redundant EOCs	randomly delete one EOC “twin”	global	2
redundant EOCs	delete EOCs without any data access between the according RE pair	global	1 (lowest)

In this context, it is important to take heed of the consequences of automated conflict resolution as several changes at once can cause additional problems. Therefore, according heuristics for multiple operations are indispensable. For example, only applying non-intersecting modifications and re-running the analysis before proceeding is recommended in order to make sure that the number of conflicts is strictly monotonically decreasing.

[30]Xtext is “a framework for development of programming languages and domain specific languages.”, cf. `http://www.eclipse.org/Xtext/`

6.4.2 Including Untapped Timing Constraints

There are still many open research questions that will be dealt with in the near future. One of the most promising ones is the issue of deployed classification criteria. At the moment, two out of seven timing constraints defined by AUTOSAR are used, namely Age Constraints and Execution Order Constraints. Therefore, five constraints could be additionally included to refine the analysis, verification and data validation of models (cf. [AUT14d] as well as examples in [Gli18]):

- "Event Triggering Constraint" (EvTrCs) are "used to specify the particular occurrences of a given timing description event", e.g., for monitoring jitter.
- "Offset Timing Constraints" (OTCs) are "used to specify an offset between the occurrences of two timing description events", e.g., for restricting the time offset between the occurrence of specific timing events.
- "Latency Timing Constraints" (LTCs) are "used to specify the amount of time that elapses between the occurrence of any two timing description events", e.g., for preventing data loss attributable to oversampling and undersampling.
- "Synchronization Timing Constraints" (STCs) are "used to specify a synchronization constraint among the occurrences of two or more timing description events", e.g., for ensuring a consistent time base for the interaction between executable units.
- "Execution Time Constraints" (ExTCs) are "used to specify minimum and maximum execution time constraints of executable entities[31]", e.g., for limiting an executable unit's run time budget.

The last three of them are particularly interesting:

Firstly, LTCs can ensure that certain data is actually read before written again by determining a maximum delay between the triggering of two REs. They can logically correlate with imposed ACs although their semantics differ. Here, LTCs address REs themselves whereas ACs refer to specific variable accesses between them.

Secondly, STCs can enforce that, e.g., two REs are triggered and run "syn-

[31]"Executable Entity" is the generic AUTOSAR term that denotes atomic, schedulable units like Runnable Entities.

chronously with respect to a predefined tolerance" [AUT14d], which affects both their assignment on specific IEUs (and maybe parallel execution) as well as the creation of a valid schedule. And as opposed to LTCs, STCs do not necessarily refer to "connected accesses" of REs (stimulus and response). Besides, it has to be considered if and to what extent EOCs and STCs act against each other in specific setups.

Thirdly, ExTCs set a lower and upper limit for the actual execution of an RE, whereas the peak value for the maximum execution time should logically be smaller than the RE's recurrence (otherwise the ExTC would be unfulfillable). Acting as supplementary timing information, ExTCs can contribute to finding an optimal schedule. Figure 6.2 shows the WSS exemplarily validated without EOCs but by means of imposing four ACs, five LTCs and twelve STCs.

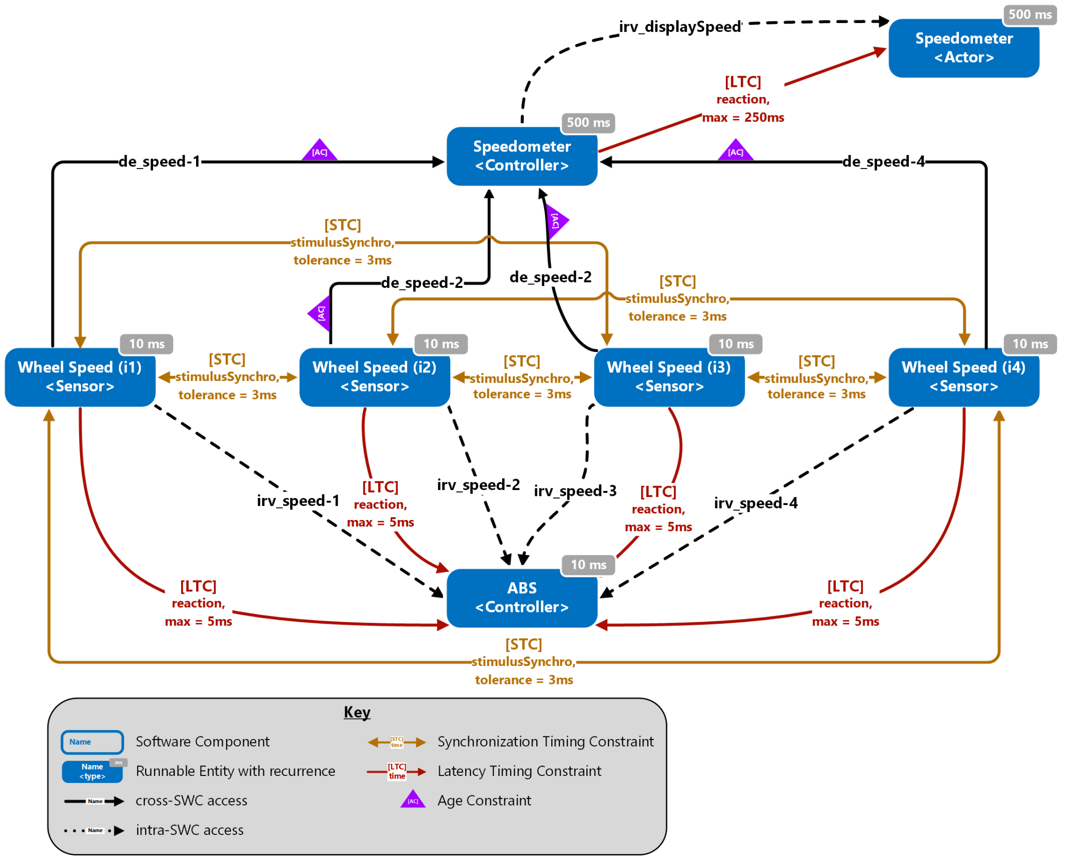

irv_displaySpeed
500 ms
Speedometer <Actor>
[LTC] reaction, max = 250ms
500 ms
Speedometer <Controller>
de_speed-1
[AC]
[AC]
de_speed-4
[STC] stimulusSynchro, tolerance = 3ms
[AC]
de_speed-2
de_speed-2
[STC] stimulusSynchro, tolerance = 3ms
[AC]
10 ms
Wheel Speed (i1) <Sensor>
[STC] stimulusSynchro, tolerance = 3ms
10 ms
Wheel Speed (i2) <Sensor>
[STC] stimulusSynchro, tolerance = 3ms
10 ms
Wheel Speed (i3) <Sensor>
[STC] stimulusSynchro, tolerance = 3ms
10 ms
Wheel Speed (i4) <Sensor>
irv_speed-1
[LTC] reaction, max = 5ms
irv_speed-2
irv_speed-3
[LTC] reaction, max = 5ms
irv_speed-4
10 ms
ABS <Controller>
[LTC] reaction, max = 5ms
[LTC] reaction, max = 5ms
[STC] stimulusSynchro, tolerance = 3ms
Key
Software Component
Runnable Entity with recurrence
cross-SWC access
intra-SWC access
Synchronization Timing Constraint
Latency Timing Constraint
Age Constraint

Figure 6.2: The WSS validated with ACs, STCs and LTCs

Assuming that all REs need 1 ms to be executed, one can determine fixed timing values: The dark red LTCs enforce the target REs ("Speedometer Actor" and "ABS Controller") to react within a certain amount of time. The orange STCs imposed between the four instances of "Wheel Speed Sensor" each demand that if one of them is triggered, the other three are also triggered within 3 ms. For reasons of clarity, bidirectional orange arrows are used to indicate that there are in fact two STCs between each of the six pairs of sensor REs (one for each direction).

Concerning multiple-IEU robustness, including these three types of timing constraints would certainly enable to prepare the model in a more accurate and target-oriented way than up to now. Hence, their inclusion in the concept as well as the implementation seems beneficial – especially for huge, complex models. However, it will be required to think about problematic interactions, advantageous combinations and proper criteria for suggesting solutions by means of specific constraints.

6.4.3 Discussing New Constraints

It is an open question if existing constraints are sufficient or if something important is missing with regard to being enable to ensure multiple-IEU robustness. One could argue that it is – attributable to the "multi-core era" – reasonable to consider the introduction of constraints that explicitly express potential parallelism, because the demands arising from the altered circumstances are not enough taken into account by available constraints as parallelism is not a distinct part of the intention behind the timing constructs listed in Subsection 6.4.2 respectively [AUT14d]:

- EvTrCs are designed to specify occurrence patterns of timing events but disregard on which IEU the triggering takes place.
- ExTCs aim at setting bounds to Executable Entities' execution times without taking heed of their possible assignment to different IEUs.
- EOCs are used to define sequences while neglecting timing aspects of the Executable Entities they address. They indirectly indicate possible parallelism when interpreting the REs addressed by them as preferably dedicated to the same IEU.
- ACs, OTCs and LTCs are applied to avoid violating boundary values for data ages or durations between timing description events. Although coping with them is – under certain conditions – facilitated by parallel execution, they do not have a direct correlation with parallelism.

- STCs can demand that several Executable Entities are triggered (virtually) synchronous, but this condition is in general satisfiable by a (fast enough) successive execution on a single EU too (an STC cannot assume several IEUs).

Since these timing constraints do not explicitly address parallelism or multiple-IEU platforms, it seems useful to reflect about possibilities to fill this suspected gap. As constraints usually restrict certain conditions, one could come up with the idea of enforcing parallelism by demanding that a set of Executable Entities is distributed on different IEUs and triggered simultaneously with a certain tolerance. However, this is hardly possible because the number of available IEUs is often unknown and/or not fixed. Thus, it stands to reason to rather give the opportunity of simultaneously processing certain Executable Entities in addition.

One approach towards this would be creating a new type of EOC (a "softened" one), so that the corresponding Executable Entities are allowed to be processed successively as well as in parallel with a certain triggering tolerance. As a result, the semantics would change from the specific preservation of the producer-consumer schema (for "original" EOCs) to only excluding an inverted execution order. For example, a softened EOC from "RE-A" to "RE-B" would merely prevent "RE-B" to be triggered before "RE-A", which seems to be rather inconvenient without creating added value.

An apparently more expedient approach is represented by the combination of STCs and LTCs enriched with parallel semantics, termed "Parallel EOC" (PEOC). A PEOC consist of a source and a target set of Executable Entities. The source set comprises elements that can explicitly be triggered and executed in parallel within a given tolerance (similar to an STC). The target set includes elements that are triggered before a defined maximum reaction time has elapsed (like in an LTC) and after all source elements are executed.

In order to illustrate the usage of this timing constraint, Figure 6.3 shows the WSS validated with one PEOC that replaces twelve STCs and four LTCs compared to Figure 6.2. The PEOC encapsulates the STCs' and LTCs' semantics, facilitates the comprehensibility and visualization (less timing constraints, less edges), underlines potential parallelism (i.e. the possible distribution of source REs on different IEUs) and would simplify an expanded analysis (cf. Section 5.1) compared with particularly analyzing LTCs and STCs as well as separately integrating markers for "concurrent successors".

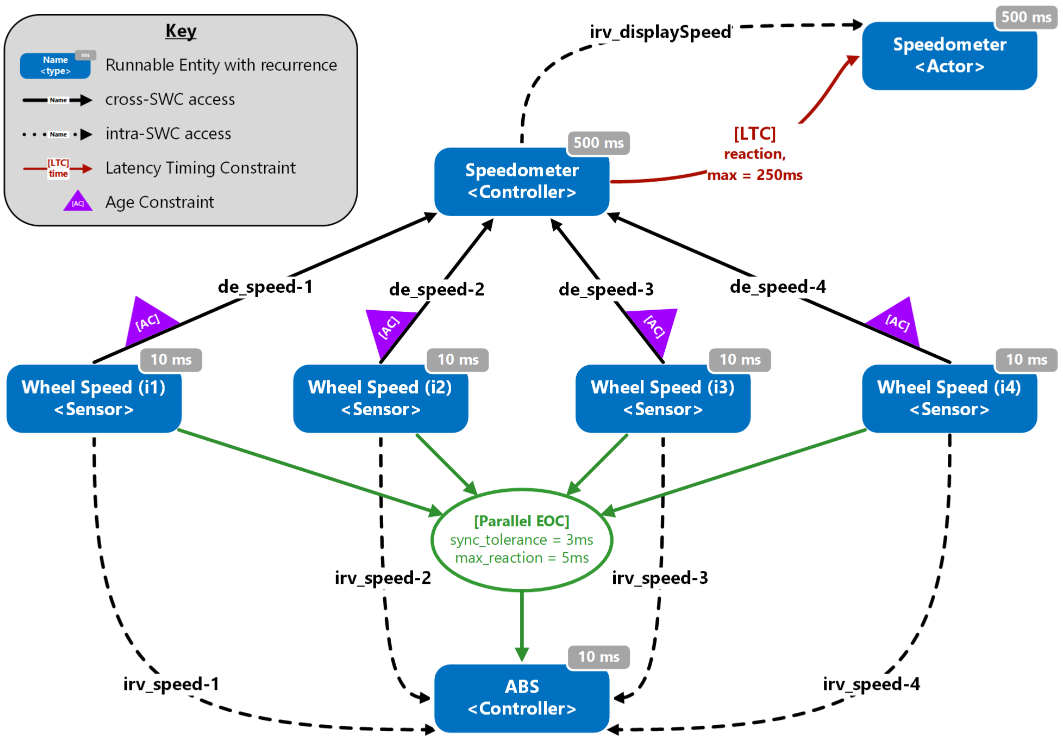


Figure 6.3: The WSS with an imposed "PEOC"

The latter are implicitly included by PEOCs and can be seen as counterpart to the grouping suggestions determined in Subsection 4.3.2. There, the tool searches for isolated regions that are suitable for being mapped as task on a common IEU, whereas a PEOC's source set contains elements that are proper for being distributed. Though both are certainly helpful when trying to overcome the repeatedly addressed tremendous solution space, there is – however – much research work to do: The approaches need to be thoroughly evaluated and tested in order to elaborate a detailed concept and a possible specification suggestion.

6.4.4 The Multi-Rate Problem

As already mentioned in Section 4.2, it is supposed that each Runnable Entity has only one fixed recurrence (triggering frequency) to be taken into consideration. This assumption is made to basically enable the analysis and validation of the models according to the approach (cf. Chapter 4).

However, this is not necessarily the case in every embedded real-time system. This aspect is usually referred to as "Multi-Rate Problem" or "Different Activation Problem". It is important to clearly distinguish the former from the situation found in so-called "multi-rate networks", where the recurrences (sampling rates) of typical producer and consumer REs (e.g. sensors and controllers) can differ to reduce the IEUs' and network's workload, which can possibly lead to data loss or duplication due to over- and undersampling [Mah14, AUT14d].

In contrast to that, the above-mentioned Multi-Rate Problem indicates different recurrence values for one RE: Although each RE runs in only one context, several activations of it by different "Triggering Events" (TEs) are possible: The basic "Timing Event" "causes an RE to be periodically triggered", whereas further TEs "occur as a result of communication activity", e.g., a "Data Received Event" triggers an RE "to receive and process a signal received on a sender-receiver interface" [AUT14c]. If two TEs activate an RE, then the analysis cannot be carried out like explained in Section 4.2, because it is not able to deal with ambiguity. In addition to that, an RE can even have two concrete diverging recurrences (Timing Events) due to several specified variants.

The current solution for these cases (as implemented in AutoAnalyze) is to hint at multi-rated REs and to consider the least respective recurrence as standard while giving the engineer the opportunity to select a different one. This proceeding is grounded on the assumption that REs occurring with maximum frequency pose the most intricate case to process. However, every single altered recurrence can have a considerable impact on the analyzed timing behavior as well as on the final parallelization result.

6.4.5 Logical Execution Time

Considering strong trends like dynamic integration (cf. Subsections 6.3.5 and 6.3.6) and raised effort to master rampant complexity (6.3.4 and 6.4.1), it becomes clear that the predictability (determinism) of timing behavior is becoming an increasingly crucial factor [Lal18, Mad18].

As the timing behavior mainly depends on the underlying communication semantics (i.e. the rules of communicating data across functions [HDK+17]), it is worth taking a closer look at the different ways to implement them. An intuitive approach is that tasks directly access and modify the variables they need without any restrictions.

These read and write accesses on global variables during the task's execution is usually referred to as "explicit communication" [HDK$^+$17]. In this context, typical problems arise from corrupted data consistency, e.g., unclear data age, data races or missing data stability during computation (cf. Subsection 1.1.4).

As opposed to this, the "implicit communication" approach – as employed by AUTOSAR – aims for data consistency by sticking to a strict "read – execute – write" paradigm, i.e., "the task always makes local copies of the shared data it needs at the beginning of its execution, works on the local copies and writes the data back at the end of its execution" [HDK$^+$17]. This proceeding requires local memory and guarantees coherency (processed data has uniform age) as well as stability (input data is steady during computation).

Facing the "design complexity" resulting from challenges like ensuring data consistency, finding a proper scheduling and handling inter-core interferences, the goal to synchronize tasks across multiple IEUs stays highly demanding – especially as communication overhead and limited distribution remain serious obstacles for parallelism in real-time systems [Yip17, Här17].

In order to cope with this, the concept of timed communication has become increasingly important in recent years as it solves many of the mentioned problems "by design" [Här17, Här16]. One specific solution is called "Logical Execution Time" (LET), which – in a nutshell – solves "temporal non-determinism by decoupling computation and communication" [HDK$^+$17]. LET does work on local copies of variables (just like AUTOSAR's implicit communication paradigm), but its strategy is to read data at the start of a task's activation interval and to write it back at its end [Här16, HDK$^+$17]. Thus, the respective time span is distinctly longer than the task's "physical execution time". As a consequence, the defined release and terminate times ensure that the process is executed within a fixed time window – no matter which scheduling strategy is employed [vH18].

Therefore, the LET concept involves several advantageous aspects:

- Determinism: Due to precisely defined points of data exchange, LET enables deterministic parallel execution and communication behavior even with respect to dynamic re-allocation, resulting in a predictable overall system behavior at design and implementation [Mad18, Lal18].
- V&V: The verification and validation process is simplified as ensuring time and value determinism provides for data consistency

(e.g. excluding race conditions) and knowing the duration of event chains in advance allows achieving "correctness by construction" [Lal18, Mad18, vH18]. In addition, the need for defining and taking heed of task priorities is rendered redundant [Yip17].

- Platform Independence: The concept realization is not limited to a specific platform, but is universally applicable [Lal18].
- Maintenance and Development: As re-writing legacy code is not necessary and its handling is facilitated, easy extensibility and refactoring are supported [Lal18, vH18]. In practice, this means that, e.g., "there is no need for complex synchronization mechanisms to handle race conditions or priority inversions" [HDK$^+$17].

On the other hand, there are also some challenges and drawbacks coming along with LET:

- Increased Latencies: LET raises the communication delay and hence results in prolonging the duration of event chains [HDK$^+$17, Lal18, Mad18]. However, there are approaches to mitigate this effect by "filling up the gaps" as well as "seamless LET frame design" [Mad18].
- Special Requirements: In order to fulfill the LET semantics, there a a couple of additional requirements to be met when the mapping is conducted and the schedule is generated. Furthermore, the necessary parametrization of LET tasks cannot be done without in-depth knowledge of the system [Lal18]. Aside from that, there is a need for a "LET language" in order to properly describe integration needs [Mad18].
- Lacking Tool Support: There is a strong demand for tools that support the process of generating LET task parameters, creating expedient task-to-IEUs allocations (mapping) and help at scheduling table and data buffers [Lal18].

In summary, it can be stated that the LET paradigm provides multiple benefits in terms of multi-core software development and is considerably gaining industrial interest [Yip17,Lal18]. Nevertheless, supporting the search for loosely-coupled program parts – like addressed in this PhD thesis – remains very important in order to distribute the calculation (on a fine-grained level) without affecting functional behavior or causing overhead via cross-IEU dependencies [Här16, Mad18, vH18].

7 Annex

Bibliography

[ABG$^+$13] Aldeida Aleti, Barbora Buhnova, Lars Grunske, Anne Koziolek, and Indika Meedeniya. Software Architecture Optimization Methods: A Systematic Literature Review. *IEEE Transactions on Software Engineering*, 39(5):658–683, 2013.

[ABRW93] Neil C. Audsley, Alan Burns, MF Richardson, and Andy J Wellings. *Deadline Monotonic Scheduling – Theory and Application*, volume 1. Elsevier, 1993.

[ACU08] Cevdet Aykanat, B Barla Cambazoglu, and Bora Uçar. Multi-Level Direct k-Way Hypergraph Partitioning with Multiple Constraints and Fixed Vertices. *Journal of Parallel and Distributed Computing*, 68(5):609–625, 2008.

[AK13] Jakob Axelsson and Avenir Kobetski. On the Conceptual Design of a Dynamic Component Model for Reconfigurable AUTOSAR Systems. *ACM SIGBED Review*, 10(4):45–48, 2013.

[APP18] APP4MC Consortium. APP4MC Website. `https://www.eclipse.org/app4mc/help/app4mc-0.8.1/index.html`, 2018. (accessed on December 6th, 2018).

[ARA19a] ARAMiS II Research Project Consortium. E3.2 Design Space Exploration – Achievement (outcome document, version 3). German Aerospace Center (DLR – project executing organisation). 2019, 2019.

[ARA19b] ARAMiS II Research Project Consortium. E3.3 Partitioning of Software Components – Achievement (outcome document, version 3). German Aerospace Center (DLR – project executing organisation). 2019, 2019.

[Arb11] Arbeitskreis Multicore. Relevanz eines Multicore-Ökosystems für künftige Embedded Systems: Positionspapier zur Bedeutung, Bestandsaufnahme und Potentialermittlung der Multicore-Technologie für den Industrie- und Forschungsstandort Deutschland. Arbeitskreis Multicore, 2011.

[Art12] Artop Group. AUTOSAR Tool Platform. `https://www.artop.org/`, 2012. (accessed on July 20th, 2013).

[Asm17] Rinat Asmus. AUTOSAR Adaptive Platform and Classic Platform Multi-Core Improvements. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[Ass12] Association for Standardisation of Automation and Measuring Systems (ASAM). ASAM MDX. `https://wiki.asam.net/display/STANDARDS/ASAM+MDX`, 2012. (accessed on September 5th, 2016).

[AUT13] AUTOSAR. *AUTOSAR Methodology*, 2013.

[AUT14a] AUTOSAR. AUTOSAR Basic Information – Short Version. `http://www.autosar.org/fileadmin/files/basic_information/AUTOSARBasicInformationShortVersion_EN.pdf`, 2014. (accessed on October 28th, 2014).

[AUT14b] AUTOSAR. *Glossary*, 2014.

[AUT14c] AUTOSAR. *Specification of RTE*, 2014.

[AUT14d] AUTOSAR. *Specification of Timing Extensions*, 2014.

[AUT14e] AUTOSAR. Technical Overview. `http://www.autosar.org/about/technical-overview/`, 2014. (accessed on August 25th, 2016).

[AUT18a] AUTOSAR. OSEK. `https://www.vectorcast.com/osek`, 2018. (accessed on December 12th, 2018).

[AUT18b] AUTOSAR. SOME/IP Protocol Specification. `https://www.autosar.org/fileadmin/user_upload/standards/foundation/1-0/AUTOSAR_PRS_SOMEIPProtocol.pdf`, 2018. (accessed on December 19th, 2018).

[BJ92] Thang Nguyen Bui and Curt Jones. Finding Good Approximate Vertex and Edge Partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, 1992.

[BKL16] Bert Böddeker, Sebastian Kehr, and Dominik Langen. Methods for Migrating Automotive Control Applications to Multicore ECUs. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[BLH$^+$13] Hans Blom, Henrik Lönn, Frank Hagl, Yiannis Papadopoulos, Mark-Oliver Reiser, Carl-Johan Sjöstedt, De-Jiu Chen, and Ramin Tavakoli Kolagari. *EAST-ADL – An Architecture Descrip-*

tion Language for Automotive Software-Intensive Systems – White Paper, Version 2.1.12, 2013. (accessed on December 19th, 2018).

[BMR04] Bernhard Bauer, Jörg P. Müller, and Stephan Roser. A Model-driven Approach to Designing Cross-Enterprise Business Processes. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pages 544–555. Springer, 2004.

[BMS$^+$16] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. Recent Advances in Graph Partitioning. In *Algorithm Engineering*, pages 117–158. Springer, 2016.

[BRM05] Bernhard Bauer, Stephan Roser, and Jörg P. Müller. Adaptive Design of Cross-Organizational Business Processes Using a Model-Driven Architecture. In *Wirtschaftsinformatik 2005*, pages 103–121. Springer, 2005.

[BSER11] Michael Bohn, Jörn Schneider, Christian Eltges, and Robert Rößger. Migration von AUTOSAR-basierten Echtzeitanwendungen auf Multicore-Systeme. In *Workshop: Entwicklung zuverlässiger Software-Systeme (Stuttgart, Germany)*, 2011.

[CCG$^+$07] Philippe Cuenot, DeJiu Chen, Sebastien Gerard, Henrik Lonn, Mark-Oliver Reiser, David Servat, Carl-Johan Sjostedt, Ramin Tavakoli Kolagari, Martin Torngren, and Matthias Weber. Managing Complexity of Automotive Electronics using the EAST-ADL. In *Engineering Complex Computer Systems, 2007. 12th IEEE International Conference on*, pages 353–358. IEEE, 2007.

[CFJ$^+$07] Philippe Cuenot, Patrick Frey, Rolf Johansson, Henrik Lönn, Yiannis Papadopoulos, Mark-Oliver Reiser, Anders Sandberg, David Servat, Ramin Tavakoli Kolagari, Martin Törngren, et al. The EAST-ADL Architecture Description Language for Automotive Embedded Software. In *Dagstuhl Workshop on Model-Based Engineering of Embedded Real-Time Systems*, pages 297–307. Springer, 2007.

[CHS12] Constantin Christmann, Erik Hebisch, and Oliver Strauß. *Vorgehensweise für die Multicore-Softwareentwicklung*. Fraunhofer Verlag, 2012.

[CJGJ96] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation Algorithms for Bin Packing: a Survey. In *Ap-*

proximation Algorithms for NP-hard Problems, pages 46–93. PWS Publishing Co., 1996.

[CKK$^+$06] Shuchi Chawla, Robert Krauthgamer, Ravi Kumar, Yuval Rabani, and D Sivakumar. On the Hardness of Approximating Multicut and Sparsest-Cut. *Computational Complexity*, 15(2):94–114, 2006.

[Con16] Continental AG. New Engine Controls by Continental Integrates Energy-Flow Management. `http://www.continental-corporation.com/www/pressportal_com_en/themes/press_releases/3_automotive_group/powertrain/press_releases/pr_2012_04_26_energy_management_en.html`, 2016. (accessed on August 28th, 2016).

[Cor13] Daniel Alexander Cordes. *Automatic Parallelization for Embedded Multi-Core Systems Using High Level Cost Models*. PhD thesis, Technische Universität Dortmund, 2013.

[Deb01] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*, volume 16. John Wiley & Sons, 2001.

[Deu15] Michael Deubzer. Multi-Core Software Architecture – Moving Towards Software Engineering. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Deu16] Michael Deubzer. Transition Into a New Era of Automotive Software Engineering. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[DHM$^+$10] Michael Deubzer, Martin Hobelsberger, Jürgen Mottok, Frank Schiller, Reiner Dumke, Markus Siegle, Ulrich Margull, Michael Niemetz, and Gerhard Wirrer. Modeling and Simulation of Embedded Real-Time Multicore Systems. In *Proceedings of the 3rd Embedded Software Engineering Congress*, pages 228–241, 2010.

[DK08] John DeNero and Dan Klein. The Complexity of Phrase Alignment Problems. In *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics on Human Language Technologies: Short Papers*, pages 25–28. Association for Computational Linguistics, 2008.

[DLZV14] Philipp Diebold, Constanza Lampasona, Sergey Zverlov, and Sebastian Voss. Practitioners' and Researchers' Expectations on

Design Space Exploration for Multicore Systems in the Automotive and Avionics Domains – A Survey. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, page 1. ACM, 2014.

[Ebe16] Christos Ebert. Software Development in Collaboratively Engineered Systems. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Ecl09] Eclipse Foundation. Eclipse Modeling Framework Project. `http://eclipse.org/modeling/emf/`, 2009. (accessed on July 15th, 2013).

[Ecl16] Eclipse Automotive Industry Working Group. BTF Specification V2.1.3. `http://wiki.eclipse.org/Auto_IWG#Publications`, 2016. (accessed on April 11th, 2016).

[Eiß12] Thomas Eißenlöffel. *Embedded-Software entwickeln*. dpunkt, 2012.

[EZV+17] Johannes Eder, Sergey Zverlov, Sebastian Voss, Maged Khalil, and Alexandru Ipatiov. Bringing DSE to Life: Exploring the Design Space of an Industrial Automotive Use Case. In *Model Driven Engineering Languages and Systems (MODELS), 2017 ACM/IEEE 20th International Conference on*, pages 270–280. IEEE, 2017.

[Flä15] Torsten Flämig. Software Architecture Methods for Multicore – Distributed Development and Validation of Architecture in Collaboratively Engineered Multicore Systems. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Fuk18] Takeshi Fukuda. Multi-core Design with Powertrain/ADAS SW and Future Challenges of Highly Self-Driven Vehicles. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[Für15] Simon Fürst. An OEM's Point of View On Multi-Core. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Für16] Simon Fürst. EUROFORUM Elektronik-Systeme im Automobil – AUTOSAR Adaptive Platform for Connected and Autonomous Vehicles. `http://www.euroforum.de/veranstaltungen/elektronik-systeme-im-automobil`, 2016. (accessed on October 4th, 2017).

[GHKF11] Peter Gliwa, Jens Harnisch, Ursula Kelling, and Christoph Ficek.

From Single-Core to Multi-Core Platforms — Systematic Migration of Hard Real-Time Software in AUTOSAR. *Embedded World*, 28:979–992, 2011.

[GLI15] GLIWA embedded systems. An Introduction to Automotive Multi-Core Embedded Software Timing. `https://www.gliwa.com/downloads/Multi-core%20Poster.pdf`, 2015. (accessed on November 13th, 2015).

[Gli18] Peter Gliwa. A Systematic Approach for Timing Requirements. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[GNN$^+$06] Matthias Gehrke, Petra Nawratil, Oliver Niggemann, Wilhelm Schäfer, and Martin Hirsch. Scenario-Based Verification of Automotive Software Systems. In *MBEES Proceedings*, pages 35–42, 2006.

[Gra15] Rudolf Grave. Software Integration Challenge Multi-Core – Experience from Real World Projects. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Gra16] Rudolf Grave. AUTOSAR Multi-Core. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Gra17] Rudolf Grave. Insights in an Automotive Central Computing Cluster. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[Gri04] Matthias Gries. Methods for Evaluating and Covering the Design Space During Early Design Development. *Integration, the VLSI journal*, 38(2):131–183, 2004.

[GRLB09] M. Götz, S. Roser, F. Lautenbacher, and B. Bauer. Token Analysis of Graph-Oriented Process Models. In *13th Enterprise Distributed Object Computing Conference (EDOC)*, pages 15–24, 2009.

[GS12] Urs Gleim and Tobias Schüle. *Multicore-Software*. dpunkt, 2012.

[Gus88] John L Gustafson. Reevaluating Amdahl's Law. *Communications of the ACM*, 31(5):532–533, 1988.

[Här16] Jochen Härdtlein. Distributing Automotive Real-time Systems. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Här17] Jochen Härdtlein. SW Distribution Based on Context-Aware Interoperability. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[HDK$^+$17] Arne Hamann, Dakshina Dasari, Simon Kramer, Michael Pressler, Falk Wurst, and Dirk Ziegenbein. WATERS Industrial Challenge 2017. `https://waters2017.inria.fr/challenge/`, 2017. (accessed on April 3rd, 2017).

[Hen98] Bruce Hendrickson. Graph Partitioning and Parallel Solvers: Has the Emperor No Clothes? In *International Symposium on Solving Irregularly Structured Problems in Parallel*, pages 218–225. Springer, 1998.

[HKI15] Robert Höttger, Lukas Krawczyk, and Burkhard Igel. Model-Based Automotive Partitioning and Mapping for Embedded Multicore Systems. *International Journal of Computer, Control, Quantum and Information Engineering*, 9(1):268–274, 2015.

[HL95] Bruce Hendrickson and Robert W Leland. A Multi-Level Algorithm For Partitioning Graphs. *SC*, 95:28, 1995.

[Hob12] Martin Hobelsberger. *Reusability Evaluation of Component-Based Embedded Automotive Software Systems*. PhD thesis, Otto-von-Guericke-Universität Magdeburg, May 2012.

[HZKL16] Arne Hamann, Dirk Ziegenbein, Simon Kramer, and Martin Lukasiewycz. FMTV 2016 Verification Challenge. *Information Processing Letters*, 2016.

[IAS15] Gesellschaft für Informatik, Verband der Automobilindustrie, and Kompetenz-Cluster SafeTRANS. Automotive Roadmap Embedded Systems. `http://www.safetrans-de.org/documents/Automotive_Roadmap_ES.pdf`, 2015. (accessed on July 20th, 2016).

[INC15] INCHRON GmbH. chronSIM. `http://www.inchron.com/tool-suite/chronsim.html`, 2015. (accessed on November 11th, 2015).

[Inf15a] Infineon Technologies AG. 32-bit TriCore^{TM} Microcontroller. `http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/channel.html?channel=ff80808112ab681d0112ab6b64b50805`,

2015. (accessed on November 16th, 2015).

[Inf15b] Infineon Technologies AG. AURIX TC27xT data sheet. `http://www.infineon.com/cms/en/product/microcontroller/32-bit-tricore-tm-microcontroller/aurix-tm-family/aurix-tm-family-%E2%80%93-tc27xt/channel.html?channel=db3a30433cfb5caa013d01df64d92edc`, 2015. (accessed on November 16th, 2015).

[JPP94] Richard Johnson, David Pearson, and Keshav Pingali. The Program Structure Tree: Computing Control Regions in Linear Time. In *ACM SigPlan Notices*, volume 29, pages 171–185. ACM, 1994.

[KBL17] Sebastian Kehr, Bert Bödekker, and Dominik Langen. Energy-aware Parallelization of AUTOSAR Legacy Applications. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[KCS06] Abdullah Konak, David W Coit, and Alice E Smith. Multi-Objective Optimization Using Genetic Algorithms: A Tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, 2006.

[KHKM11] Jin Kim, Inwook Hwang, Yong-Hyuk Kim, and Byung-Ro Moon. Genetic Approaches for Graph Partitioning: A Survey. In *Proceedings of the 13th annual conference on 13th Annual Genetic and Evolutionary Computation*, pages 473–480. ACM, 2011.

[KK15] Julian Kienberger and Stefan Kuntz. Systematic and Methodical Data-Dependency Analysis for Multiple-IEU Platforms. *ARAMiS Final Report – Continental Automotive GmbH*, pages 41–77, 2015.

[KMKB14] Julian Kienberger, Pascal Minnerup, Stefan Kuntz, and Bernhard Bauer. Analysis and Validation of AUTOSAR Models. In *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, pages 274–281. IEEE, 2014.

[KPQ+16] Sebastian Kehr, Miloš Panić, Eduardo Quiñones, Bert Böddeker, Jorge Becerril Sandoval, Jaume Abella, Francisco Cazorla, and Günter Schäfer. Supertask: Maximizing Runnable-Level Parallelism in AUTOSAR Applications. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 25–30. IEEE, 2016.

[KQBS15] Sebastian Kehr, Eduardo Quiñones, Bert Böddeker, and Günter Schäfer. Parallel Execution of AUTOSAR Legacy Applications on Multicore ECUs with Timed Implicit Communication. In *Proceedings of the 52nd Annual Design Automation Conference*, page 42. ACM, 2015.

[KSKB16] Julian Kienberger, Christian Saad, Stefan Kuntz, and Bernhard Bauer. Efficient Parallelization of Complex Automotive Systems. In *Proceedings of the 7th International Workshop on Programming Models and Applications for Multicores and Manycores (PMAM)*, pages 40–49. ACM, 2016.

[KSS^{+}17] Julian Kienberger, Stefan Schmidhuber, Christian Saad, Stefan Kuntz, and Bernhard Bauer. Parallelizing Highly Complex Engine Management Systems. *Concurrency and Computation: Practice and Experience – Special Issue*, 29(15), 2017.

[Kun17] Stefan Kuntz. System Architecture – The importance of hardware models in developing applications for MSoCs. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[KWF15] Lukas Krawczyk, Carsten Wolff, and Daniel Fruhner. Automated Distribution of Software to Multi-core Hardware in Model Based Embedded Systems Development. In Giedre Dregvaite and Robertas" Damasevicius, editors, *Information and Software Technologies*, pages 320–329. Springer International Publishing, 2015.

[Lal18] Erjola Lalo. Challenges of Migration Automotive Event-Triggered Systems to LET paradigm. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[Lan16] Jochen Langenwalter. Self-Driving Car Supercomputer. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[LL04] Deming Liu and Yann-Hang Lee. Pfair Scheduling Of Periodic Tasks With Allocation Constraints On Multiple Processors. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International*, page 119. IEEE, 2004.

[LLP^{+}09] Rongshen Long, Hong Li, Wei Peng, Yi Zhang, and Minde Zhao. An Approach to Optimize Intra-ECU Communication Based on Mapping of AUTOSAR Runnable Entities. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages

138–143. IEEE, 2009.

[LW82] Joseph Y-T Leung and Jennifer Whitehead. On the Complexity of Fixed-Priority Scheduling of Periodic, Real-Time Tasks. *Performance Evaluation*, 2(4):237–250, 1982.

[Mac15] Harald Mackamul. AMALTHEA – An Open Source Development Platform for Embedded Multi- and Many-Core Systems. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Mad15] Ralph Mader. Timing and Design Tool Support in Continental Powertrain Multi-Core Platform. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Mad18] Ralph Mader. Logical Execution Time Implementation in Classic AUTOSAR. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[MAE13a] MAENAD Consortium. EAST-ADL Introduction – EAST-ADL Overview. `http://www.maenad.eu/public_pw/conceptpresentations/1_Overview_EAST-ADL_Introduction_2012.pdf`, 2013. (accessed on September 25th, 2013).

[MAE13b] MAENAD Consortium. EAST-ADL Introduction – Relation to AUTOSAR. `http://www.maenad.eu/public_pw/conceptpresentations/2_RelationToAUTOSAR_EAST-ADL_Introduction_2012.pdf`, 2013. (accessed on September 23th, 2013).

[Mah14] Magdi S Mahmoud. *Control and Estimation Methods over Communication Networks*. Springer, 2014.

[May16] Albrecht Mayer. From Safe Driving to Safer Autonomous Driving. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[McD07] Ryan McDonald. A Study of Global Inference Algorithms in Multi-Document Summarization. In *European Conference on Information Retrieval*, pages 557–564. Springer, 2007.

[Mel15] Alessandra Melani. Global Scheduling in Multiprocessor Real-Time Systems. `http://retis.sssup.it/~giorgio/slides/cbsd/mc3-global-2p.pdf`, 2015. (accessed on July 20th, 2013).

[MFCM16] Lothar Michel, Torsten Flämig, Denis Claraz, and Ralph Mader. Shared SW Development in Multi-Core Automotive Context. In *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[MNBSL12] Aurélien Monot, Nicolas Navet, Bernard Bavoux, and Françoise Simonot-Lion. Multisource Software on Multicore Automotive ECUs — Combining Runnable Sequencing with Task Scheduling. *IEEE Transactions on Industrial Electronics*, 59(10):3934–3942, 2012.

[Moy13] Bryon Moyer. *Real World Multicore Embedded Systems*. Newnes, 2013.

[MPS07] Burkhard Monien, Robert Preis, and Stefan Schamberger. Approximation Algorithms for Multilevel Graph Partitioning. In Teofilo F Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*, chapter 60. Emerald Group Publishing Limited, 2007.

[NBN09] Farhang Nemati, Moris Behnam, and Thomas Nolte. Efficiently Migrating Real-Time Systems to Multi-Cores. In *2009 IEEE Conference on Emerging Technologies & Factory Automation*, pages 1–8. IEEE, 2009.

[Neu16] Moritz Neukirchner. Efficient Communication and Synchronization in Multi-Core Systems. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[NNB10] Farhang Nemati, Thomas Nolte, and Moris Behnam. Blocking-Aware Partitioning for Multiprocessors. *Digitala Vetenskapliga Arkivet*, 2010.

[OG18] Maximilian Odendahl and Sebastian Gerstl. Softwaredesign für die AUTOSAR Adaptive Platform. `https://www.embedded-software-engineering.de/softwaredesign-fuer-die-autosar-adaptive-platform-a-688631/`, 2018. (accessed on 28th June, 2018).

[OLKY05] Won Hyun Oh, Jung Hee Lee, Hyoung Geun Kwon, and Hyoung Jin Yoon. Model-Based Development of Automotive Embedded Systems: a Case of Continuously Variable Transmission (CVT). In *11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 201–

204. IEEE, 2005.

[OO84] Karl J Ottenstein and Linda M Ottenstein. The Program Dependence Graph in a Software Development Environment. In *ACM Sigplan Notices*, volume 19, pages 177–184, 1984.

[Pat10] David Patterson. The Trouble with Multi-core. *Spectrum, IEEE*, 47(7):28–32, 2010.

[PD13] Frank Padberg and Oliver Denninger. Multicore-Softwarefehler im Visier: Automatische Fehlererkennung in Entwürfen paralleler Programme. *OBJEKTspektrum*, 20(1):72–76, 2013.

[PH15] Uwe Pohlmann and Marcus Hüwe. Model-Driven Allocation Engineering. In *30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 374–384. IEEE, 2015.

[PKQ$^+$14] Miloš Panić, Sebastian Kehr, Eduardo Quiñones, Bert Böddeker, Jaume Abella, and Francisco Cazorla. RunPar: an Allocation Algorithm for Automotive Applications Exploiting Runnable Parallelism in Multicores. In *Proceedings of the 2014 International Conference on Hardware/Software Codesign and System Synthesis*, page 29. ACM, 2014.

[QCLT11] T. N. Qureshi, D.-J. Chen, H. Lönn, and M. Törngren. From EAST-ADL to AUTOSAR Software Architecture: A Mapping Scheme. In *Software Architecture*, volume 6903, pages 328–335. Springer, 2011.

[Qui16] Sophie Quinton. WATERS Community Forum. `http://ecrts.eit.uni-kl.de/forum/viewtopic.php?f=27&p=69#p79`, 2016. (accessed on March 23rd, 2016).

[Rad17] Franz-Josef Radermacher. Automotive Megatrends – Just a Hype? Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[RBM06] Stephan Roser, Bernhard Bauer, and Jörg P. Müller. Model- and Architecture-Driven Development in the Context of Cross-Enterprise Business Process Engineering. In *Services Computing, 2006. SCC'06. IEEE International Conference on*, pages 119–126. IEEE, 2006.

[RN16] Stefan Rathgeber and Michael Niklas. AUTOSAR meets new Use Cases – The AUTOSAR Adaptive Platform. Embedded

Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[RTS16] Hans-Leo Ross, Frank Tränkle, and Stefan Schwarzkopf. Safety Concept of Road Vehicles with System Controllers. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Saa09] Christian Saad. Model Analysis Framework. `http://www.informatik.uni-augsburg.de/en/chairs/swt/ds/projects/mde/maf/`, 2009. (accessed on July 20th, 2013).

[Saa15] Christian Saad. *Data-flow based Model Analysis: Approach, Implementation and Applications*. PhD thesis, Universität Augsburg, June 2015.

[Sar87] Vivek Sarkar. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors*. PhD thesis, Stanford University, April 1987.

[SB13] Christian Saad and Bernhard Bauer. Data-Flow Based Model Analysis and Its Applications. In *Model-Driven Engineering Languages and Systems*, pages 707–723. Springer, 2013.

[SBR10] Jörn Schneider, Michael Bohn, and Robert Rößger. Migration of Automotive Real-Time Software to Multicore Systems: First Steps Towards an Automated Solution. In *22nd EUROMICRO Conference on Real-Time Systems*, 2010.

[Sch15a] Benjamin Schatz. Challenge Multi-Core TCU: An Applied Example of Multi-Core Migration. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Sch15b] Rolf Michael Schneider. The ARAMiS Automotive LSSI Demonstrators and the Lessons Learned. Embedded Multi-Core Conference. In *EMCC 2015 Proceedings*, 2015.

[Sch18] Christopher Schwager. Central Vehicle Computer: Achieving Freedom From Interference by Temporal Software Separation. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[SDM$^+$14] S. Schmidhuber, M. Deubzer, R. Mader, M. Niemetz, and J. Mottok. Towards the Derivation of Guidelines for the Deployment of Real-Time Tasks on a Multicore Processor. In *Model-Based Safety and Assessment*, pages 152–165. Springer, 2014.

[Sei09] Michael Seibt. Architekturmodellierung mit EAST-ADL2 und AUTOSAR. *Hanser Automotive Spezial*, 9:38–41, 2009.

[Sie16] Rudolf Sieber. Applying AUTOSAR in a Powertrain Dynamic Architecture Using Multicore ECUs – Seamless Dynamic Behavior for Inhomogeneous Applications. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Sin11] Purnendu Sinha. Architectural Design and Reliability Analysis of a Fail-Operational Brake-by-Wire System from ISO 26262 Perspectives. *Reliability Engineering & System Safety*, 96(10):1349–1359, 2011.

[SKK00] Kirk Schloegel, George Karypis, and Vipin Kumar. *Graph Partitioning for High-Performance Scientific Simulations*. Army High Performance Computing Research Center, 2000.

[SKS10] Kathrin Danielle Scheidenmann, Michael Knapp, and Claus Stellwag. Load Balancing in AUTOSAR-Multicore-Systemen. *WEKA Fachmedien GmbH*, 2010.

[SMD$^+$10] Angela C Sodan, Jacob Machina, Arash Deshmeh, Kevin Macnaughton, and Bryan Esbaugh. Parallelism Via Multithreaded and Multicore CPUs. *Computer*, 43(3):24–32, 2010.

[SS13] Peter Sanders and Christian Schulz. Think Locally, Act Globally: Highly Balanced Graph Partitioning. In *Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13)*, volume 7933 of *LNCS*, pages 164–175. Springer, 2013.

[SS15] Peter Sanders and Christian Schulz. KaHIP v1.00 — Karlsruhe High Quality Partitioning User Guide. `http://algo2.iti.kit.edu/schulz/software_releases/kahipv1.00.pdf`, 2015. (accessed on July 20th, 2016).

[SSH$^+$16] Andreas Sailer, Stefan Schmidhuber, Maximilian Hempe, Michael Deubzer, and Jürgen Mottok. Distributed Multi-Core Development in the Automotive Domain – A Practical Comparison of ASAM MDX vs. AUTOSAR vs. AMALTHEA. In *ARCS 2016; 29th International Conference on Architecture of Computing Systems; Proceedings of*, pages 1–8. VDE, 2016.

[SSRB12] John A Stankovic, Marco Spuri, Krithi Ramamritham, and Giorgio C Buttazzo. *Deadline Scheduling for Real-Time Systems: EDF* *and Related Algorithms*, volume 460. Springer Science & Business Media, 2012.

[Ste16] Bärbel Steininger. Migration to Multi-Core Technology for Innovative Powertrain Solutions. Embedded Multi-Core Conference. In *EMCC 2016 Proceedings*, 2016.

[Sut05] Herb Sutter. The Free Lunch Is Over: A Fundamental Turn Toward Concurrency in Software. *Dr. Dobb's Journal*, 30(3):202–210, 2005.

[SVZ15] Bernhard Schätz, Sebastian Voss, and Sergey Zverlov. Automating Design-Space Exploration: Optimal Deployment of Automotive SW-Components in an ISO26262 Context. In *Proceedings of the 52nd Annual Design Automation Conference*, page 99. ACM, 2015.

[SWL$^+$09] Chihhsiong Shih, Chien-Ting Wu, Cheng-Yao Lin, Pao-Ann Hsiung, Nien-Lin Hsueh, Chih-Hung Chang, Chorng-Shiuh Koong, and W.C. Chu. A Model-Driven Multicore Software Development Environment for Embedded System. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 261–268, 2009.

[Sym15] Symtavision GmbH. SymTA/S and TraceAnalyzer. `https://www.symtavision.com/products/symtas-traceanalyzer/`, 2015. (accessed on November 11th, 2015).

[SZ10] Jörg Schäuffele and Thomas Zurawka. *Automotive Software Engineering*. Springer DE, 2010.

[The15] The AMALTHEA4public Consortium. Deliverable: D3.1 – Analysis of State of the Art V&V Techniques, 2015.

[The16] The AMALTHEA4public Consortium. Deliverable: D2.2 – Concept for Partitioning, Mapping, and Tracing for Multi- and Many-core Systems, 2016.

[TIM09] TIMMO Project. TIMMO Website. `https://itea3.org/project/timmo.html`, 2009. (accessed on March 7th, 2017).

[TIM10] TIMMO-2-USE. Timing Model – TOols, algorithms, languages, methodology, USE cases. `https://itea3.org/project/timmo-2-use.html`, 2010. (accessed on November 16th, 2015).

[Tim15] Timing-Architects Embedded Systems GmbH. Timing Architects Tool Suite. `http://www.timing-architects.com/ta-tool-suite/simulator/`, 2015. (accessed on November 11th, 2015).

[Tip95] Frank Tip. A Survey of Program Slicing Techniques. *Journal of Programming Languages*, 3(3):121–189, 1995.

[vH18] Hermann von Hasseln. Migration of Legacy Software to Multicore: LET as Enabler. Embedded Multi-Core Conference. In *EMCC 2018 Proceedings*, 2018.

[WAT16] WATERS – 7th International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems. FMTV Verification Challenge. `https://waters2016.inria.fr/challenge/`, 2016. (accessed on March 23rd, 2016).

[WGG10] Klaus Wehrle, Mesut Günes, and James Gross. *Modeling and Tools for Network Simulation*. Springer Science & Business Media, 2010.

[Wir11] Loring Wirbel. Embedded Multicore Goes Mainstream. `http://www.designnews.com/author.asp?section_id=1386&doc_id=231676`, 2011. (accessed on July 15th, 2013).

[WMM$^+$13] Ernest Wozniak, Asma Mehiaoui, Chokri Mraidha, Sara Tucci-Piergiovanni, and Sébastien Gerard. An Optimization Approach for the Synthesis of AUTOSAR Architectures. In *2013 IEEE 18th Conference on Emerging Technologies & Factory Automation (ETFA)*, pages 1–10. IEEE, 2013.

[Yip17] Eugene Yip. Pulling the Trigger – Practical Considerations When Applying the Time-Triggered Approach to Embedded Multi-Core Systems. Embedded Multi-Core Conference. In *EMCC 2017 Proceedings*, 2017.

[YPS13] J Youn, I Park, and M Sunwoo. Heuristic Resource Allocation and Scheduling Method for Distributed Automotive Control Systems. *International Journal of Automotive Technology*, 14(4):611–624, 2013.

[ZG11] Ming Zhang and Zonghua Gu. Optimization Issues in Mapping AUTOSAR Components to Distributed Multithreaded Implementations. In *2011 22nd IEEE International Symposium on Rapid System Prototyping*, pages 23–29. IEEE, 2011.

[ZZZ$^+$12] Qi Zhu, Haibo Zeng, Wei Zheng, Marco DI Natale, and Alberto Sangiovanni-Vincentelli. Optimization of Task Allocation and Priority Assignment in Hard Real-Time Distributed Systems. *ACM Transactions on Embedded Computing Systems (TECS)*, 11(4):85, 2012.

Listings

List of Figures

List of Tables

Glossary

ABS Anti-lock Braking System. 5, 12, 36, 71, 75, 79, 90

AC Age Constraint. 71, 73–75, 82–84, 95, 105, 106, 142–146, 175

AND Average Node Degree. 81, 86, 87

Artop AUTOSAR Tool Platform. 33, 59, 60, 104

ASL Allocation Specification Language. 46

ASW Application Software. 27, 30, 36

ATP Automatic Task Parallelization. 117

AUTOSAR AUTomotive Open System ARchitecture. iii–vi, 10, 15, 16, 26–33, 36–38, 45, 47, 49, 55, 58, 60–63, 66, 69–71, 82, 83, 94–97, 99, 103, 104, 107, 113, 115, 119, 122–124, 132, 134, 138–141, 143, 144, 150, 175, 177

BD Backward Dependency. 72–74

BSW Basic Software. 27, 30, 36

CLD Cross-Linking Degree. 81, 86, 88, 89

CP Critical Path. 49, 50, 54, 93

EAST-ADL Electronic Architecture and Software Tools – Architecture Description Language. 31, 32, 175

ECU Electronic Control Unit. iii, v, 2–6, 10, 11, 26–29, 36, 40, 41, 45–47, 56–60, 69, 70, 72, 73, 95, 132, 137

EDF Earliest Deadline First. 49, 54, 96, 115, 119, 120

EMS Engine Management System. 3, 6, 8–11, 40, 41, 51, 63, 87–89, 107, 113, 114, 117–120, 122–124, 126–128, 132, 175

EOC Execution Order Constraint. 69, 71, 73–75, 79, 82–84, 89, 90, 95, 104–106, 142–147

ESS Earliest Start Scheduling. 49, 54

EU Execution Unit. 33, 34, 38, 51, 62, 68, 132, 134, 137, 147

EvTrC Event Triggering Constraint. 144, 146

ExTC Execution Time Constraint. 144–146

FD Forward Dependency. 72, 74

GA Genetic Algorithm. 45, 47, 48, 54, 100, 116, 128

HLCM High-Level Cost Model. 48, 54

HTG Hierarchical Task Graph. 48, 52, 175

ICC Inter-Core Communication. 8, 93, 98, 116, 117, 120, 125, 128

IEU Independent Execution Unit. iii–vi, 3–8, 10, 12, 14–17, 19, 27, 29, 31, 33–36, 38, 39, 41, 46, 47, 50, 51, 55–58, 62, 65, 73–75, 77–82, 90–97, 104, 106, 107, 120, 132, 134, 135, 137, 142, 145–151, 175

ILP Integer Linear Programming. 45, 46, 48, 50, 54

IRV Inter-Runnable Variable. 58, 70, 175

ISR Interrupt Service Routine. 98, 102

KaHIP Karlsruhe High Quality Partitioning. 44

LET Logical Execution Time. 30, 51, 54, 94, 150, 151

LTC Latency Timing Constraint. 144–147, 175

MAF Model Analysis Framework. 104, 106

MILP Mixed Integer Linear Programming. 45, 48

MLP Multi-Level Partitioning. 40, 41, 44, 54, 64, 84

mNRT maximum Normalized Response Time. 98, 116, 117, 125, 128

OCL Object Constraint Language. 45

OEM Original Equipment Manufacturer. 26, 136–138

OS operating system. 27, 28, 30, 61, 64, 66, 122, 123

OTC Offset Timing Constraint. 144, 146

PA Process Allocation. 116, 117

PCP Priority Ceiling Protocol. 30

PEOC Parallel EOC. 147, 148, 175

PSOA Periodic Stimulus Offset Assignment. 117

RE Runnable Entity. 10, 27–29, 31, 36, 37, 41, 42, 45–47, 49–51, 54, 58, 61–64, 69–75, 77–86, 91, 93–96, 99, 102, 104–107, 113–115, 119, 120, 122, 123, 126, 139, 142–149, 175

REI Runnable Entity Instance. 58, 69, 70, 72, 74, 75, 83, 86, 87

RS Runnable Sequencing. 116, 117

RTE Runtime Environment. 27

SER Single-Entry Region. 86

SERA Single-Entry Region Analysis. 81, 84, 86, 88, 89, 103, 106, 107, 114, 128

SMDS Software Methodologies for Distributed Systems. ix, 21

STC Synchronization Timing Constraint. 144–147, 175

SW-C Software Component. 27–31, 36, 37, 45, 47, 58, 61, 62, 69, 70, 73, 90, 94, 95, 105, 106, 113, 120, 122, 123, 142, 143, 175

TATS TA Tool Suite. 60, 95, 115, 116, 120, 122, 128

TE Triggering Event. 149

TIC Timed Implicit Communication. 51, 54

TS-EM Task Splitting with Enforced Migration. 116, 117

TS-IPA Task Splitting with Inter Process Activation. 116, 117

V&V Verification and Validation. 5, 68, 138, 140, 150

VA variable access. 33, 35, 62, 64, 70, 72, 82, 83, 95, 106, 113, 114

VFB Virtual Functional Bus. 27

WCET Worst Case Execution Time. 5, 51

WSS Wheel Speed System. 12, 13, 36, 37, 58, 75, 76, 78, 79, 90, 91, 145, 147, 148, 175