Safety-Aware Deployment Synthesis and Trade-Off Analysis of Apollo Autonomous Driving Platform

Tarik Terzimehić[1], Simon Barner[1], Yuri Gil Dantas[1], Ulrich Schöpp[1], Vivek Nigam[2], Pei Ke[2]

[1]fortiss – Research Institute of the Free State of Bavaria, Munich, Germany; E-mail: {lastname}@fortiss.org
[2]Huawei Technologies Düsseldorf GmbH, Düsseldorf, Germany; E-mail: {first.lastname}@huawei.com

***Abstract*—The adoption of autonomous cars requires operational critical functions even in the event of HW faults and/or SW defects, and protection of safety-critical functions against security threats. Defining appropriate safe and secure architectures is challenging and costly. In previous work, we have proposed tools to automate the recommendation of safety and security patterns for safety-critical systems. However, safety and security measures may (negatively) influence system performance, besides introducing additional development effort. We present a design space exploration approach, a model-based engineering workflow and tool prototype for automated guidance on trade-off decisions when applying safety and security patterns on a given (unsafe) baseline architecture. Based on models that abstract the vehicle's functionality and its software and hardware components, as well as an engine for the automated pattern recommendation, we investigate the optimization of HW/SW deployments, and provide a trade-off analysis for different architecture candidates. We implemented our approach in an open-source tool and evaluate it with a model of the Apollo autonomous driving platform.**

***Index Terms*—Autonomous vehicles, model-driven development, design space exploration, architecture pattern, safety**

I. Introduction

Autonomous driving has a huge potential to improve the safety of road-based traffic, i.e., to reduce the number of accidents and traffic-related deaths. A prerequisite for achieving this objective is that critical functions of autonomous cars remain operational even in the event of HW faults and/or SW defects, and that safety-critical functions are protected against security threats [1]. Defining appropriate safe and secure architectures is challenging and increases development time and effort [2]. In previous work we have therefore proposed tools to automate the recommendation of safety [3] and security [4] patterns. However, so far we did not consider the performance impact resulting from the application of patterns that typically involve redundancy and therefore cause significant overhead.

The goal of this paper is to close this gap and to perform a trade-off analysis w.r.t. the selection of architecture patterns and deployment decisions, which both heavily impact system performance [5]. Deployment optimization of automotive SW architectures remains intractable for humans despite the trend towards more centralized architectures [6] with fewer high-performance electronic control units (ECUs) [7]. For instance, the increasing adoption of hypervisors providing virtual deployment targets in the form of time-space partitions results in an increase of the complexity. At the same time, tightening time-to-market requirements and frequent in-field software updates require more systematic approaches to architecture definition than experience knowledge.

Our overall contribution is a model-based Design Space Exploration (DSE) approach and a tool prototype to provide automated guidance for trade-off decisions when applying safety and security patterns on a given (unsafe) baseline architecture. The proposed approach leverages the automated recommendation of safety patterns presented in [3] and optimizes SW/HW deployments for the resulting architecture candidates. We evaluate our prototypical implementation in the AutoFOCUS3 (AF3)[1] open-source tool with a model of the Apollo autonomous driving platform[2].

II. Background

A. Apollo Autonomous Driving Platform

This work targets a realistic autonomous vehicle architecture based on the Apollo driving stack. Apollo is an open-source implementation of an autonomous driving system with L4 capabilities [8]. A central part of its implementation is the "*Cyber RT*" middleware, which implements a Service Oriented Architecture (SOA). This middleware manages components of the autonomous driving system, such as *Perception*, *Planning* and *Control*, and provides a publish/subscribe mechanism for the communication between software components.

B. Model-Based Design Space Exploration

The design of software-intensive systems, such as current automotive systems, involves various architectural design decisions (ADDs), such as deploying software components (SWCs) to hardware components (HWCs), applying different architecture patterns, determining the hardware layout (topology), etc. [9]. These decisions satisfy certain constraints (e.g., memory limits of HWCs), and optimize certain optimization goals or objectives (e.g., minimization of hardware costs).

Due to the complexity of current automotive systems, it is difficult to manually make valid or optimal ADDs [7]. DSE techniques are used when manual design decisions are difficult due to a large exploration space. DSE defines an automated process for finding solutions within a constrained solution space [10]. DSE can be used together with Model-based Design (MbD). MbD provides models that not only increase

[1]https://af3.fortiss.org/
[2]https://github.com/ApolloAuto/apollo

the level of abstraction but also provide the semantics needed to automate design tasks. Models describe the system from different viewpoints [11], e.g., the logical viewpoint describes the functionality of the system, while the technical viewpoint describes the SW/HW platform [12].

C. Safety and Security Analysis

During the design of safety-critical systems, safety and security analysis are conducted by engineers to identify and address safety and security artifacts. Examples of safety artifacts are hazards and loss scenarios [13]. A hazard denotes a potential source of harm. A loss scenario denotes the casual factors (e.g., a software fault) that may lead to hazards. Examples of security artifacts are assets and threat scenarios [14]. An asset denotes an object (e.g., software component) that needs protection. A threat scenario denotes the potential actions (or simply attack) to violate the cybersecurity property of assets. These artifacts may be addressed using architectural solutions called safety and security architecture patterns.

A safety architecture pattern is an architectural solution for tolerating loss scenarios/faults in the system architecture [15]. We have proposed a tool – SafPat – to automate the recommendation of safety architecture patterns for safety-critical systems in [3]. SafPat supports several safety architecture patterns, including fault-tolerant patterns (e.g., *heterogeneous duplex*) that provide redundancy enabling the system to continue its operation in the presence of a loss scenario/fault.

A security architecture pattern is an architectural solution for addressing threat scenarios in the system architecture [16]. We have proposed a tool – SecPat – to automate the recommendation of security architecture patterns for safety-critical systems in [4]. SecPat supports several security architecture patterns, including *message authentication* that satisfies the integrity of safety-critical data by encrypting and signing the data. The use of architecture patterns, however, introduces overhead due to, e.g., redundancy for safety reasons or encryption for security reasons that shall be considered by engineers when selecting safe and secure architectural solutions.

III. Methodology

In Section III-A, we describe the overall MbD workflow for analyzing safety, security, and performance of autonomous driving architectures. Steps 1 and 2 summarize the workflow part introduced in previous work [3]. Steps 3 and 4 describe the application of the safety-aware deployment synthesis that is the focus of this paper and thoroughly presented in Section III-B.

A. Overall Workflow

The applied MbD workflow comprises four main steps, which are implemented in the AF3 tool:

Step 1 – Baseline Model. In the first step, a model of baseline architecture without a SW/HW deployment is created (Figure 1 left). The use case model investigated in this paper captures an autonomous driving architecture based on the Apollo driving stack in terms of the following elements[3]:

[3]We follow the modeling approach introduced in [12], but list only the input relevant for the deployment workflow that is in focus of this work.

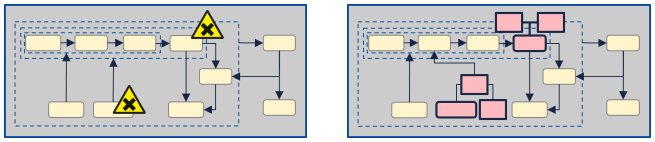


Fig. 1: Abstract baseline architecture with safety artifacts (left) and safety pattern synthesis with additional components (right)

- **Logical architecture** is a hierarchical model of the system components, their interfaces, connections, and communication channels and topics.
- **Task architecture** is a technical view of software tasks (i.e., SWCs) that realize the logical architecture and that will be deployed on the hardware, including task annotations such as RAM requirements, flash requirements, worst case execution time (WCET), etc.
- **Platform architecture** is model of the HWCs (i.e., deployment targets; ECUs in this paper) and their configuration, including annotations such as RAM, flash memory, Automotive Safety Integrity Levels (ASILs), costs, etc.
- **Allocations** are mappings of the elements between the different architectural viewpoints, such as *Components* → *Tasks* (mapping of components to tasks), *Tasks* → *ECUs* (initial mapping of tasks to ECUs), etc.
- **Safety artifacts** are hazard definitions (with severity, controllability, and exposure as defined in ISO 26262), and loss scenarios as potential causes of a hazard.

Our approach also allows to model existing architecture patterns and to map them to the addressed safety artifacts, enabling the extension of existing architectures.

Step 2 – Safety Pattern Synthesis. This step receives as input architectures and the safety artifacts defined Step 1. It addresses the specified safety artifacts by recommending one or several safety architecture patterns (e.g., *acceptance voting*, *heterogeneous duplex*, etc.). For each recommended architecture, safety pattern synthesis provides a set of safety requirements that encode the prerequisites for the application of the selected patterns. The modified architectures (see Figure 1 right) include additional components needed for the implementation of the recommended safety patterns (e.g., monitor, voter), or replicated components from the baseline to ensure redundancy [3]. This step includes the computation of "safety-critical communication topics" that need protection against security threats that could affect safety.

Step 3 – Safety-Aware Deployment Exploration. This step receives as input an architecture with safety patterns, the corresponding safety requirements, and a list of safety-critical topics from Step 2. Next, deployment constraints are derived from the safety requirements generated in Step 2:

- ASIL-compliant allocation constraints
- ASIL decomposition constraints
- Task hardware decoupling constraints

From the list of safety-critical topics, a safety-aware coupling optimization is configured to minimize inter-device safety-critical communication, as well as resource utilization optimization objective to balance load across used ECUs.

A user may define additional constraints, such as memory utilization, fixed allocations of tasks to ECUs (e.g., due to availability of I/Os), and design objectives (e.g., cost minimization) [7]. The input architecture model as well as the deployment constraints and objectives are transformed into a Satisfiability Modulo Theories (SMT) formalization. The Z3 solver [17] processes given SMT formalization and computes a task-to-execution unit deployment and the ASILs for safety pattern components that are subject to ASIL decomposition.

Step 4 – Trade-Off Analysis. This step receives as input an architecture with safety patterns, safety artifacts defined by the user, and the task-to-execution unit table computed in Step 3. It provides the following trade-off analyzes:

- Impact of safety patterns and their deployments on system performance: For generated safety solutions, we calculate different performance criteria, such as communication coupling and resource utilization, and compare them with the baseline architecture without safety patterns.
- Impact of deployment optimization on security patterns: We compute security artifacts based on the defined safety artifacts and determine the necessary security patterns: *access control shared memory*, *message authentication*, *mutual authentication*, *authorization*, and *firewall* (see [4] for more details). We then apply the deployment optimization from Step 3 to reduce the overhead introduced by security patterns. More precisely, we focus on minimizing the effort required to secure communications, a significant factor in data-intensive applications such as autonomous driving [6].

B. Safety-Aware Deployment Synthesis

This section describes the deployment exploration (introduced in Step 3) that is based on an SMT formalization and uses Microsoft's Z3 SMT solver[4] as backend. A user can specify goals (constraints and optimization objectives) by means of goal pattern language [7], [18] that are passed to the Z3 backend using its Java API. Z3 calculates the deployment (or allocation) of tasks to processing units (e.g., ECUs).

For example, a user can specify a memory utilization constraint that prohibits exceeding the hardware memory limits [18]. In addition to searching for valid deployments, i.e., deployments that satisfy given constraints, a user can specify design objectives, e.g., cost objective [7]. In this case, Z3 will find valid deployment that yields minimal hardware costs.

For this paper, we extended the deployment exploration plugin in AF3 and integrated it with safety and security pattern synthesis. Besides system architecture (task and platform architecture with annotations), the deployment exploration uses safety requirements and information on safety-critical topics from Step 2 (safety pattern synthesis). Deployment exploration generates accordingly safety-related deployment constraints that enforce the underlying safety requirements.

For example, the safety pattern synthesis outputs a requirement "A component *av_monitor(acTestV1, "relative_map")* must be deployed to an ASIL D ECU". Deployment exploration will then automatically derive a formal specification[5] for allocation constraint that will force deployment of the task *Task_av_monitor(acTestV1, relative_map)* to an ASIL D ECU. Besides safety-related constraints, deployment exploration generates safety-aware coupling and memory utilization optimization goals for each safety solution from the safety synthesis step. Applying different optimization goals in deployment exploration, we can improve the overall performance.

After a user selects solution and switches to the deployment viewpoint, corresponding safety-relevant constraints and optimization goals will be listed. In addition to the safety constraints, deployment exploration generates memory utilization constraints to avoid trivial deployment solutions (e.g., deploying all tasks to a single ECU), as well as fixed allocation constraints to ensure deployment of the particular tasks to the ECUs with required hardware sensors, actuators, or communication interfaces. A user can define additional (generic) constraints and optimization goals in the constraint editor and combine them with generated ones.

1) Deployment constraints: Deployment exploration generates different deployment constraints that implement deployment-related safety requirements:

Hardware decoupling constraint: Some safety patterns, such as *triple modular redundancy* pattern, require that particular tasks are hardware decoupled to address hardware faults. This will be achieved by hardware decoupling constraint that will prohibit deployment of specified tasks to the same ECU.

ASIL allocation constraint: Safety-critical tasks, which have been assigned an ASIL, have to be deployed to an ECUs that is certified to at least that ASIL. For example, for an ASIL D task, deployment exploration generates allocation constraints that limit deployment possibilities of the given task to ASIL D ECUs.

ASIL decomposition constraint: Safety synthesis modifies the logical and technical architectures and generates additional components (e.g., redundant components), where for high ASILs, ASIL decomposition [19] according to ISO 26262 is applied. For example, when the *heterogeneous duplex* pattern is recommended, the original SWC is split into two SWCs whose ASIL sum is equal or greater than ASIL from the baseline architecture (e.g., ASIL D = ASIL A + ASIL C = ASIL B + ASIL B). After ASIL decomposition, the ASIL allocation constraints are applied for the resulting SWCs.

2) Deployment optimization goals: Deployment exploration automatically generates optimization goals that can be used to optimize deployment and perform trade-off analysis.

Safety-aware coupling optimization minimizes safety-critical inter-ECU communication. A user can apply safety-aware coupling optimization to reduce:

- attack surface for safety-critical device communication
- cost for message signatures (as measure to ensure integrity of safety-critical communication)
- sensitivity for (communication) failures

[4]https://github.com/Z3Prover/z3

[5]For details on deployment formalization see [7].

This is achieved by making safety-critical communication local (intra-ECU), and thus removing need to sign remote (inter-ECU) communication, which may have a negative impact on costs and the overall performance of the system [20]. However, due to hard constraints (e.g., hardware decoupling required by many architecture patterns), it is not always possible to make safety-critical communication local.

To illustrate safety-aware optimization, we use a subset of the Apollo model. Figure 2a depicts a manual task deployment onto three ECUs, without any optimization. Such a deployment yields a higher safety-critical coupling and, consequently, higher overhead (Figure 2c):

- Eight ports need to generate or verify signatures
- Six tasks need to generate or verify signatures
- Five remote safety-critical signals

Figure 2b depicts a deployment with safety-aware optimization, yielding lower safety-critical coupling and, consequently, lower overhead (Figure 2c):

- Two ports need to generate or verify signatures
- Two tasks need to generate or verify signatures
- One remote safety-critical signal

We calculate critical and non-critical couplings between each two ECUs as the sum of external (or remote) critical and non-critical signals between each two ECUs, respectively. The overall coupling is calculated as weighted sum of critical and non-critical couplings across all ECUs. Critical couplings are weighted 1000-time more. Applying this optimization goal, we minimize the overall coupling to reduce communication load, achieve lower service end-to-end latency (and improve performance in general), increase reliability, etc. [21], [22]. By applying higher weights on critical-coupling, we especially minimize critical remote communication.

Resource utilization optimization balances the relative memory consumption. We consider the additional load for safety- and security-critical tasks (not considered in initial/baseline deployment) and approximate the additional task load as a simple function of task ASILs. A user can apply the resource utilization objective to balance the load of ECUs (no ECU is overloaded) and increase reconfiguration possibilities (there is enough buffer for the dynamic redeployment during runtime). Therefore, we first calculate the weighted relative memory usage for each ECU, and then minimize the variance across all used ECUs. Therefore, we balance the overall ECU utilization: each ECU will be approximately equally loaded.

IV. Evaluation

In this section, we demonstrate the applicability of the proposed workflow on the Apollo driving stack and perform a trade-off analysis to investigate effects of deployment optimization on safety, security, and performance aspects of software-intensive autonomous vehicle systems.

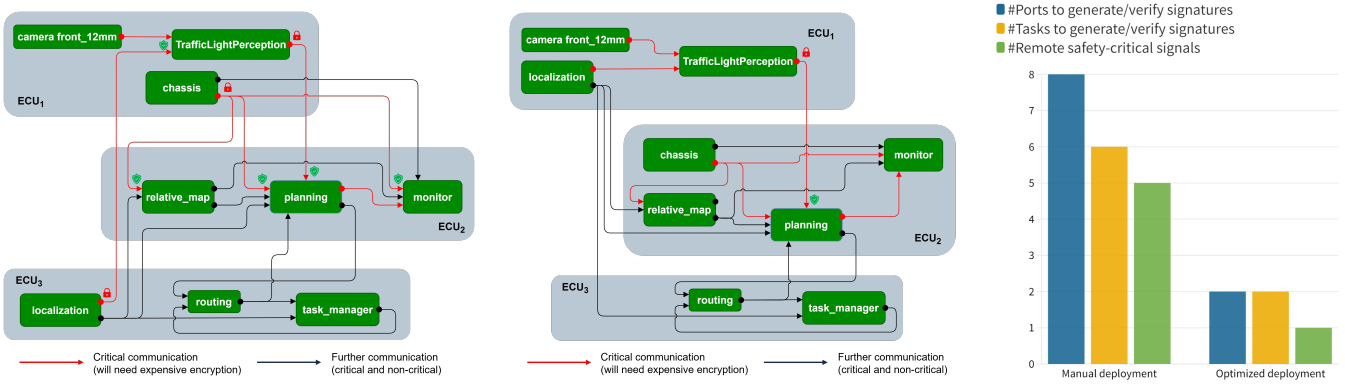
A. Use Case

We modeled the Apollo autonomous vehicle architecture using the AF3 open-source tool. The Apollo AF3 model accounts for functional and logical aspects of the system (Figure 2 in [23]), as well as technical aspects of the SW and HW architecture (Figure 4 in [23]). The Apollo AF3 model comprises 68 logical components (58 software tasks) and 32 hardware components (10 ECUs), representing a baseline architecture and a starting point for the use of our safety-oriented engineering process.

To demonstrate the applicability of the proposed solution and subsequently perform a trade-off analysis, we defined hazards and loss scenarios (i.e., safety artifacts) for the following components of the Apollo architecture:

- **recognition component** is part of the perception component. It receives images from cameras and processes the images to recognize objects on the road.
- **postprocessor** is part of the control component. It outputs control actions (e.g., acceleration) to assist with the steering of a vehicle.
- **relative map** generates a real-time relative map in the body coordinate system and a reference line for planning.

Next, we run safety pattern synthesis to address the specified safety artifacts. As a result, the synthesis recommends 64 safety solutions that address the safety artifacts in different



(a) Manual deployment without safety-aware optimization

(b) Safety-aware deployment optimization

(c) Overhead for remote safety-critical communication

Fig. 2: Example for safety-aware coupling optimization

ways. Each solution contains one or several instances of recommended safety architecture patterns to address the loss scenarios in the specified components. These safety architecture patterns require additional components, e.g., diverse redundant components to take over in the presence of failures. A set of requirements comes along with each solution recommended by the safety synthesis. These requirements shall be implemented during the system development. A subset of such requirements is implemented by deployment synthesis (c.f. Section III-B).

After safety pattern synthesis, one of the generated solutions with the recommended safety patterns is selected for the subsequent deployment synthesis in the exploration perspective of AF3. This perspective then lists the generated safety-related constraints, as well as application-specific constraints (memory utilization constraints and hardware allocation constraints), and different optimization goals. The user selects all constraints and a single or multiple optimization goals (e.g., safety-aware coupling minimization), and starts the deployment optimization. Based on the underlying formalization of the constraints, the Z3 backend determines a deployment that satisfies all selected constraints and minimizes remote safety-critical communication. Thereby, we demonstrate the applicability of the proposed MbD workflow to automatically generate safety-relevant deployment constraints and synthesize deployments that fulfill safety requirements and, therefore, reduce the development effort.

B. Trade-Off Analysis

Safety synthesis yields 64 different solutions that address the specified hazards and loss scenarios in different ways. Table I lists all 64 safety solutions, which apply the following safety patterns:

- Acceptance voting
- Heterogeneous triple modular redundancy
- Heterogeneous duplex
- Simplex architecture

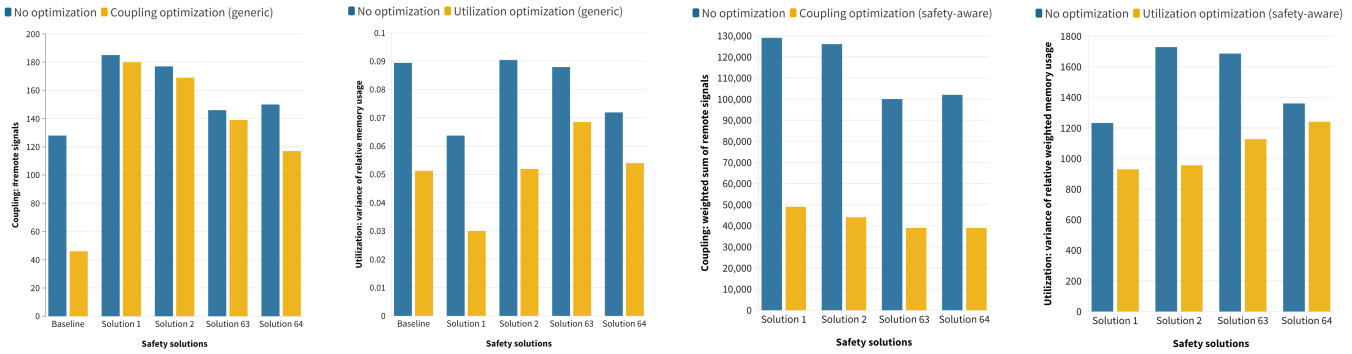
Each safety solution specifies a different number of safety requirements and results in a different number of additional SWCs to fulfill the corresponding safety requirements. These safety requirements and additional components can be understood as the overhead caused by safety synthesis: each requirement and each additional component have to be implemented (additional development effort) and may negatively impact some system parameters such as hardware utilization, software maintainability, communication coupling, etc.

Thereby, particular safety patterns lead to different overhead, as shown in the heat-map in Table I. We can see that the application of the *acceptance voting* pattern has the biggest influence on the safety overhead: the more *acceptance voting* pattern instances are used, the more safety requirements are specified, and the more additional components are needed. This is confirmed by the Pearson correlation [24], which indicates a significant large positive relationship between *acceptance voting* pattern instances and number of additional components ($r(62) = 0.968, p < 0.001$), as well as a significant large positive relationship between *acceptance voting* pattern instances and the number of safety requirements ($r(62) = 0.916, p < 0.001$). The Pearson correlation, applied on safety solutions without *acceptance voting* pattern instances, indicates significant large positive relationships between *heterogeneous triple modular redundancy* pattern instances and the number of additional components ($r(25) = 1, p < 0.001$), and between *simplex architecture* pattern instances and the number of safety requirements, ($r(25) = 0.945, p < 0.001$).

TABLE I: Safety solutions apply different patterns and yield different number of additional components and requirements.

Solution	#Safety Pattern Instances				#Components	#Requirements
	Acceptance Voting	HTMR	Heterogeneous Duplex	Simplex Architecture		
1	3	0	0	0	18	53
2	2	1	0	0	15	46
3	2	1	0	0	15	46
4	2	1	0	0	15	46
5	2	0	0	1	14	49
6	2	0	0	1	14	49
7	2	0	0	1	14	49
8	2	0	1	0	14	47
9	2	0	1	0	14	47
10	2	0	1	0	14	47
11	1	2	0	0	12	39
12	1	2	0	0	12	39
13	1	2	0	0	12	39
14	1	1	0	1	11	42
15	1	1	0	1	11	42
16	1	1	0	1	11	42
17	1	1	0	1	11	42
18	1	1	0	1	11	42
19	1	1	0	1	11	42
20	1	1	1	0	11	40
21	1	1	1	0	11	40
22	1	1	1	0	11	40
23	1	1	1	0	11	40
24	1	1	1	0	11	40
25	1	1	1	0	11	40
26	1	0	0	2	10	45
27	1	0	1	1	10	45
28	1	0	1	1	10	45
29	1	0	0	2	10	43
30	1	0	0	2	10	43
31	1	0	1	1	10	43
32	1	0	2	0	10	43
33	1	0	2	0	10	43
34	1	0	2	0	10	42
35	1	0	1	1	10	41
36	1	0	1	1	10	41
37	1	0	1	1	10	41
38	0	3	0	0	9	32
39	0	2	0	1	8	35
40	0	2	0	1	8	35
41	0	2	0	1	8	35
42	0	2	1	0	8	33
43	0	2	1	0	8	33
44	0	2	1	0	8	33
45	0	1	0	2	7	38
46	0	1	0	2	7	38
47	0	1	0	2	7	38
48	0	1	1	1	7	36
49	0	1	1	1	7	36
50	0	1	1	1	7	36
51	0	1	1	1	7	36
52	0	1	1	1	7	36
53	0	1	1	1	7	36
54	0	1	2	0	7	34
55	0	1	2	0	7	34
56	0	1	2	0	7	34
57	0	0	0	3	6	41
58	0	0	1	2	6	39
59	0	0	1	2	6	39
60	0	0	1	2	6	39
61	0	0	2	1	6	37
62	0	0	2	1	6	37
63	0	0	2	1	6	37
64	0	0	3	0	6	35

To analyze the impact of safety synthesis on the system performance in more detail, we chose the two solutions with the highest overhead (solutions 1 and 2), and the two solutions with the lowest overhead (solutions 63 and 64). Solutions 1 and 2 specify the highest number of safety requirements, 53 and 46 requirements, respectively, and yield up to 31% more tasks and up to 40% more signals compared with the baseline architecture without safety measures. On the other



(a) Optimization of generic coupling

(b) Optimization of generic utilization

(c) Optimization of safety-aware coupling

(d) Optimization of safety-aware utilization

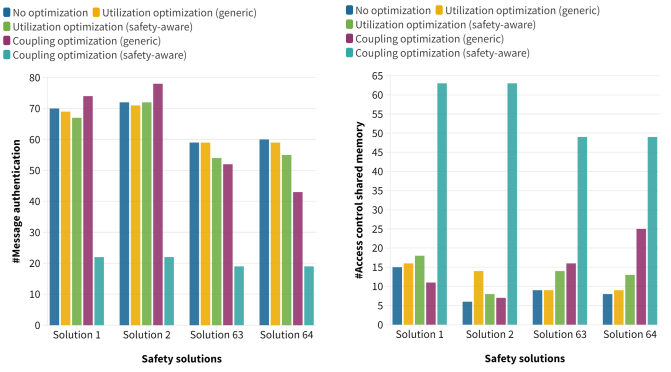
Fig. 3: Impact of applied safety measures on system performance

hand, solutions 63 and 64 implement 37 and 35 requirements, respectively, and yield 14% more tasks and up to 15% more signals compared with the baseline architecture.

The application of safety and security patterns will not only improve the safety and security of the given system, but it will also introduce additional overhead. Besides the development effort for additional SWCs, safety and security measures may (negatively) influence the system performance. Therefore, we performed several experiments using the Z3 SMT solver [17] in terms of deployment syntheses optimizing different objectives:

- **Generic coupling** optimization minimizes all inter-ECU communication.
- **Safety-aware coupling** optimization minimizes especially safety-critical inter-ECU communication.
- **Generic resource utilization** optimization balances relative memory consumption across all ECUs.
- **Safety-aware resource utilization** optimization balances relative memory consumption across all ECUs, considering additional load for safety- and security-critical tasks.

Due to the high complexity of the deployment synthesis problem – the most complex architecture of safety solution 1 contains 76 tasks (deployable units), 200 signals, and 10 ECUs (deploying targets) – we introduced a calculation timeout of six hours, which leads to sub-optimal deployment solutions.



(a) Safety-aware coupling optimization reduces usage of expensive *message authentications*

(b) Safety-aware coupling optimization favors usage of cheaper *access control shared memory* pattern

Fig. 4: Reducing overhead with deployment optimization

Figure 3a and Figure 3b show results for generic coupling and resource utilization optimizations, respectively. Optimized (but not optimal) generic coupling (i.e., number of remote, inter-ECU, communications) has increased by applying different safety patterns (Figure 3a). Initial (not optimized) communication coupling increased from 14% (when safety solution 63 is applied) to 45% (when safety solution 1 is applied) compared to the baseline architecture without safety measures. The impact of safety solutions on optimized coupling is even higher: optimized communication coupling increased from 154% (when safety solution 64 is applied) to 291% (when safety solution 1 is applied) compared to the optimized baseline architecture. This negative impact of the safety solutions is caused by the higher number of tasks and signals in safety solutions compared to the baseline architecture. On the other hand, the application of the safety patterns did not have much influence on the variance of the ECU utilization (Figure 3b). Additional tasks introduced by different safety solutions were uniformly distributed across available ECUs, having no significant impact on the variance of the relative memory utilization.

Figures 3c and 3d show results for safety-aware coupling and resource utilization optimizations, respectively. Here we had to skip comparisons with the baseline architecture, as it lacks necessary safety-relevant information, such as the criticality of communication caused by the defined loss scenarios. The safety-aware coupling optimization reduced the weighted sum of remote signals quite equally, by 62.5% on average, whereas the safety-aware utilization optimization had a different impact on the safety solutions and reduced the variance of relative weighted memory usage by 9% for solution 64, and by 45% for solution 2.

Besides analyzing the influence of safety patterns on performance (i.e., coupling and utilization), we also analyzed whether the safety-aware deployment optimization leads to less safety and security overhead. Figure 4 shows the bar charts for the number of *message authentication* and *access control shared memory* patterns, respectively, with different deployment optimizations. We can conclude that the application of safety-aware coupling deployment optimization decreases the

security overhead by reducing the number of expensive *message authentications* (needed for inter-ECU communication) in favor of the cheaper *access control shared memory* pattern.

V. Related Work

Numerous works investigate the application and synthesis of safety and security architecture patterns, like, for example in [25]–[28]. However, those works neglect the interference of the applied safety and security architecture patterns with the system performance and do not consider DSE. On the other side, optimization of different quality attributes (QAs) through deployment synthesis is thoroughly investigated as well (e.g., in [5], [29]–[31]) but most often not in conjunction with the application and synthesis of safety and security architecture patterns. Only a few works, beside ours, investigate the application of DSE in conjunction with safety and security aspects.

Ebner *et al.* investigate the application of DSE on a safety-critical autonomous driving systems [32]. They generate design variants (deployments or allocations) and analyze them according to possible erroneous system states. The approach is demonstrated on a simplified electric drive-train example with four functionalities (i.e., SWCs) and four HWCs (although additional hardware resources might be required for redundancy safety measures). For the applied example, the proposed approach seams applicable. However, the question arises, whether the proposed safety analysis is feasible for all possible design variants. For example, the baseline Apollo AF3 model, which we use in our paper, contains 58 tasks (i.e., SWCs) and 10 ECUs (i.e., HWCs), leading to 10^{58} possible deployments [33]. Performing safety analysis for each of those, like proposed in this paper, seems unrealistic. Further, it is unclear, how the design variants are generated, i.e., whether they were optimized towards particular optimization goals, which methodology was used, etc.

Hu *et al.* [34] proposed a heuristic to calculate schedules while satisfying safety requirements and optimizing development costs. Development costs are approximated by the applied ASILs – higher ASIL costs more – and can be reduced by ASIL decomposition as "a great option for balancing system conflicting performance requirements" [34]. The authors modeled the task architecture of an automotive system as a directed acyclic graph and evaluated the approach on a benchmark. This work, however, focuses solely on ASIL decomposition, which is infeasible for COTS components with fixed ASILs provided by suppliers [35].

Similarly to our work, Obergfell *et al.* [36] advocate the application of SOA in the automotive domain and apply DSE to synthesize deployment candidates and help designers to make sound early-stage design choices. Thereby, they focus on minimizing the utilized ECU cores and verifying timing requirements using a simulation framework. This work considers safety requirements as well, and formulates them as deployment constraints, but it lacks an impact analysis of deployment optimization on safety overhead, and vice versa.

Yasaweerasinghelage *et al.* [20] apply a genetic algorithm to optimize architectures regarding cost and performance (i.e., response time) while satisfying security constraints. They evaluate the proposed approach on a rather simple example (9 SWCs and 3 HWCs) to find out that the resulting secure architecture "tends to be more expensive and have inferior performance" [20]. Our work goes one step further towards supporting architects in early design decisions by analyzing the impact of deployment optimization on security overhead.

Apvrille *et al.* [37] also focus on the interaction of safety and security during design and analyze how safety, security and performance requirements conflict or support each other. They extended an open-source framework for UML/SysML-based design of embedded systems by adding safety and security features (operators, formal proof) to diagrams. They conduct a qualitative analysis of the performance, security and safety requirements interference, whereas our trade-off analysis is quantitative and relates well-known safety and security architecture patterns with performance QAs.

A quantitative trade-off analysis is given in [38], where five parameters, e.g., cost and communication load, were analyzed in conjunction with safety measures. This work compares four different architecture solutions to achieve redundancy for fault-tolerant and fail-operational systems, using three example applications (modeled as graphs). The main findings of the conducted trade-off analysis can be used as guidance or recommendations for the architects. For example, virtualization with a zone-based architecture should be applied to implement redundancy if the reduction of costs and failure probabilities are the main criteria. Our trade-off analysis complements this study and considers deployment design decisions with other safety and security architecture measures beside redundancy.

VI. Conclusion

We found that early ADDs, such as deployment synthesis of SWCs (i.e., tasks) to HWCs (i.e., ECUs), can reduce the overhead of applying safety and security architecture patterns: i) our MbD workflow was able to automatically generate safety-relevant deployment constraints and synthesize deployments that fulfill safety requirements and, therefore, reduce the development effort, ii) the application of the proposed safety-aware coupling optimization decreases the security overhead by reducing the number of expensive *message authentications* and replacing them with the cheaper *access control shared memory* pattern. Although our work is not the first one that performs DSE in conjunction with safety and security requirements (see, e.g., [20], [32]), to the best of our knowledge, we are the first to implement such a complete MbD workflow that 1) considers performance, safety and security system aspects, 2) automatically generates deployment constraints out of synthesized safety requirements, and 3) optimizes deployment synthesis towards security overhead reduction.

Furthermore, we found out that the application of the *acceptance voting* safety pattern leads to the highest development overhead, i.e., this safety pattern requires the most additional components and safety requirements, followed by the *simplex architecture* and the *heterogeneous triple modular redundancy* patterns. The application of the *heterogeneous duplex* pattern

leads to the lowest development overhead. For future work, we will analyze further safety patterns (e.g., the *monitor-actuator* and *homogeneous duplex* patterns) to obtain a complete picture of the safety patterns' overhead and provide adequate guidance to the architects.

Our trade-off analysis confirmed the expected negative impact of the safety patterns on the communication coupling due to additional SWCs (i.e., tasks) and their connections (i.e., signals). However, the application of different safety patterns did not affect the variance of the ECU utilization. Additional tasks introduced by different safety solutions were uniformly distributed across available ECUs, having no significant impact on the variance of the relative memory utilization. We plan to expand our analysis on further QAs to assist the user in selecting the system architecture with adequate architecture patterns for the QAs of interest.

The benefits of our work are twofold. First, we extended an existing open-source DSE implementation in the AF3 tool and integrated it with the reasoning engine for the safety and security patterns recommendations [3], [4]. Thereby, we demonstrated the applicability and benefits of this model-based design process for the considered subsystem of the Apollo autonomous driving stack. Second, we provide a trade-off analysis of deployment design decisions in conjunction with the application of safety and security patterns, which can serve as a guideline for comparing different possibilities (e.g., application of different deployment optimization goals, application of different architecture patterns, etc.) and selecting the appropriate ones for the given context.

REFERENCES

[1] "Safety first for automated driving," Aptiv; Audi; Baidu; BMW; Continental; Daimler; Fiat Chrysler Automobiles; Infineon; Intel; Volkswagen;, Tech. Rep., 2019. [Online]. Available: https://www.daimler.com/documents/innovation/other/safety-first-for-automated-driving.pdf

[2] S. Jasser, "Enforcing architectural security decisions," in *2020 IEEE International Conference on Software Architecture (ICSA)*, 2020.

[3] Y. Dantas, T. Munaro, C. Carlan, V. Nigam, S. Barner, S. Fan, A. Pretschner, U. Schöpp, and S. Tverdyshev, "A Model-based System Engineering Plugin for Safety Architecture Pattern Synthesis," in *Int. Conf. on Model-Driven Eng. and Software Development*, 2022.

[4] Y. G. Dantas and V. Nigam, "Automating safety and security co-design through semantically-rich architecture patterns," *ACM Trans. CPS*, 2022.

[5] A. Koziolek, H. Koziolek, and R. Reussner, "PerOpteryx: Automated Application of Tactics in Multi-Objective Software Architecture Optimization," in *ACM SIGSOFT conference on Quality of software architectures (QoSA) and architecting critical systems (ISARCS)*, 2011.

[6] V. Bandur, G. Selim, V. Pantelic, and M. Lawford, "Making the case for centralized automotive E/E architectures," *Trans. Veh. Technol.*, 2021.

[7] J. Eder, S. Zverlov, S. Voss, M. Khalil, and A. Ipatiov, "Bringing DSE to Life: Exploring the Design Space of an Industrial Automotive Use Case," in *Int. Conf. on Model Driven Eng. Languages and Sys.*, 2017.

[8] Society of Automotive Engineers, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles," Tech. Rep., 2021.

[9] T. Terzimehić, K. Dorofeev, and S. Voss, "Exploring architectural design decisions in industry 4.0: A literature review and taxonomy," in *Int. Conf. on Model Driven Engineering Languages and Systems*, 2021.

[10] B. Schätz, F. Hölzl, and T. Lundkvist, "Design-Space Exploration through Constraint-Based Model-Transformation," in *17th IEEE Int. Conf. and Workshops on Eng. of Comp. Based Syst.*, 2010.

[11] A. Bucaioni and P. Pelliccione, "Technical architectures for automotive systems," in *Int. Conference on Software Architecture (ICSA)*, 2020.

[12] S. Barner, F. Chauvel, A. Diewald, F. Eizaguirre, Ø. Haugen, J. Migge, and A. Vasilevskiy, *Modeling and Development Process*. Distributed Real-Time Architecture for Mixed-Criticality Systems, CRC Press, 2018.

[13] ISO26262, "ISO 26262, road vehicles — functional safety — part 6: Product development: software level," 2018.

[14] ISO/SAE 21434, "Road vehicles - cybersecurity engineering." 2020.

[15] A. Armoush, "Design patterns for safety-critical embedded systems," Ph.D. dissertation, RWTH Aachen University, 2010.

[16] B. H. C. Cheng, B. Doherty, N. Polanco, and M. Pasco, "Security Patterns for Automotive Systems," in *Int. Conf. on Model Driven Engineering Languages and Systems Companion*, 2019.

[17] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems*, 2008.

[18] J. Eder, S. Voss, A. Bayha, A. Ipatiov, and M. Khalil, "Hardware architecture exploration: automatic exploration of distributed automotive hardware architectures," *Software and Systems Modeling*, 2020.

[19] A. Frigerio, B. Vermeulen, and K. Goossens, "Component-level ASIL decomposition for automotive architectures," in *Int. Conf. on Dependable Systems and Networks Workshops*, 2019.

[20] R. Yasaweerasinghelage, M. Staples, H.-Y. Paik, and I. Weber, "Optimising architectures for performance, cost, and security," in *Software Architecture*, T. Bures, L. Duchien, and P. Inverardi, Eds., 2019.

[21] T. Terzimehić, S. Voss, and M. Wenger, "Using Design Space Exploration to Calculate Deployment Configurations of IEC 61499-based Systems," in *Int. Conf. on Automation Science and Eng.*, 2018.

[22] T. Terzimehić, M. Wenger, S. Voss, S. Grüner, and H. Elfaham, "SMT-Based Deployment Calculation in Industrial Automation Domain," in *Int. Conf. on Emerging Technologies and Factory Automation*, 2019.

[23] Y. G. Dantas, S. Barner, P. Ke, V. Nigam, and U. Schoepp, "Automating vehicle SOA threat analysis using a model-based methodology," in *Int. Conf. on Information Systems Security and Privacy ICISSP*, 2023.

[24] L. Ming-Chang, "Software Quality Factors and Software Quality Metrics to Enhance Software Quality Assurance," *J. Appl. Sci. Technol.*, 2014.

[25] C. Preschern, N. Kajtazovic, and C. Kreiner, "Security analysis of safety patterns," in *Conf. on Pattern Languages of Programs*, 2013.

[26] ——, "Building a safety architecture pattern system," in *European Conf. on Pattern Languages of Program*, 2015.

[27] H. Martin, Z. Ma, C. Schmittner, B. Winkler, M. Krammer, D. Schneider, T. Amorim, G. Macher, and C. Kreiner, "Combined automotive safety and security pattern engineering approach," *Reliability Engineering and System Safety*, 2020.

[28] I. Sljivo, G. J. Uriagereka, S. Puri, and B. Gallina, "Guiding assurance of architectural design patterns for critical applications," *Journal of Systems Architecture*, 2020.

[29] U. Pohlmann and M. Hüwe, "Model-driven allocation engineering," in *30th IEEE/ACM Int. Conf. on Automated Software Eng. (ASE)*, 2015.

[30] S. Zverlov, M. Khalil, and M. Chaudhary, "Pareto-efficient deployment synthesis for safety-critical applications in seamless model-based development," in *Euro. Cong. on Embedded real-time SW and Sys.*, 2016.

[31] A. Busch, D. Fuchss, and A. Koziolek, "PerOpteryx: Automated Improvement of Software Architectures," in *Int. Conf. on Software Architecture - Companion*, 2019.

[32] C. Ebner, K. Gorelik, and A. Zimmermann, "Model-Based Design Space Exploration for Fail-Operational Mechatronic Systems," in *Int. Symposium on Systems Engineering*, 2021.

[33] S. Malek, N. Medvidović, and M. Mikic-Rakic, "An extensible framework for improving a distributed software system's deployment architecture," *IEEE Transactions on Software Engineering*, vol. 38, 2012.

[34] B. Hu, S. Xu, Z. Cao, and M. Zhou, "Safety-Guaranteed and Development Cost- Minimized Scheduling of DAG Functionality in an Automotive System," *Trans. on Intelligent Transportation Sys.*, 2022.

[35] K. L. Lu and Y. Y. Chen, "Safety-Oriented System Hardware Architecture Exploration in Compliance with ISO 26262," *Applied Sciences*, 2022.

[36] P. Obergfell, S. Kugele, and E. Sax, "Model-Based Resource Analysis and Synthesis of Service-Oriented Automotive Software Architectures," in *Int. Conf. on Model Driven Eng. Languages and Sys.*, 2019.

[37] L. Apvrille and L. W. Li, "Harmonizing Safety, Security and Performance Requirements in Embedded Systems," in *Design, Automation and Test in Europe Conf. and Exhibition*, 2019.

[38] A. Frigerio, B. Vermeulen, and K. Goossens, "Isolation of redundant and mixed-critical automotive applications: Effects on the system architecture," in *IEEE Vehicular Technology Conference*, 2021.