Towards Generic Communication Interfaces of Existing Applications

Hannes Klee*, Michael Buchholz*, Torben Materna†, Klaus Dietmayer*

*Institute of Measurement, Control and Microtechnology, University of Ulm
Albert-Einstein-Allee 41, 89081 Ulm, Germany
Email: {hannes.klee, michael.buchholz, klaus.dietmayer}@uni-ulm.de
†Deutsche Accumotive GmbH & Co. KG, Neue Strasse 95, 73230 Kirchheim u. Teck/Nabern, Germany
Email: torben.materna@daimler.com

***Abstract*—For most hardware components in the automotive area, there is the problem that exchanges can only be done if the components are technically identical. That means not only identical in the interfaces and dimensions but also in the internal properties. These properties are firmly integrated in the software of connected hardware components. In this paper, an approach towards generic interfaces is proposed to identify this information and provide it on the communication interface. Furthermore, the influence of the specific information is reduced by an deployment optimization. Additionally, the communication overhead as well as necessary changes from a given original deployment are considered. A realistic case study is used to demonstrate the practicability of the proposed approach. It shows that it is possible to modify an existing application towards a generic communication interface. This approach proposes a combination of optimization and generalization in one procedure with respect to two acceptance criteria for a future realization. Not only theoretical best solutions are archived, but also solutions with a high chance to be accepted due to organizational changes.**

***Index Terms*—acceptance, evolutionary many-objective optimization, generic interfaces, multi-criterion optimization, search based software architecture optimization, software engineering.**

I. Introduction

Today, a typical car comprises up to 100 Electronic Control Units (ECUs) [1] which are connected through several buses and gateways. In most cases, not only the specific ECU of one hardware component is exactly tailored to the properties, but also several other ECUs which possess certain information about the specific hardware component, e.g. the engine of a car with its characteristics. This information is typically hard coded as data variables during the development process because of the assumption that it is static and, therefore, does not need to be communicated through the interfaces. That is why changes of the hardware affect wide parts of the system. If a component is exchanged with not exactly the same properties as originally specified, the car will malfunction or at least unexpected behavior is possible. Therefore, a component change is associated with additional effort, cost and the risk of unexpected problems. To upgrade a car, the software for every Electronic Control Unit (ECU) must be updated. The necessary software can only be provided by the Original Equipment Manufacturer (OEM), which makes the upgrade idea very unlikely for current cars.

In the last decade, several ideas have been published to enable an easier way to exchange components in a system. In [2], interfaces are defined in profiles which are only dependent on the type of network. These profiles require only to exchange the mapping instead of implementing the whole communication together with drivers in the operating systems. Aue et al. [3] published a patent that suggests a method for providing a generic interface which enables an operation of microcontrollers independent from the specific communication technology. AUTOSAR [4] is an industry standard for automotive E/E (Electric/Electronic) architecture, of which one goal is an easy exchange of hardware or software components. Therefore, a virtual bus is used for the communication and software interfaces are defined on application level [5]. In [6], a functional decomposition of a vehicle control system is suggested. Definitions of the functional interfaces are provided for reuse of the architecture for future technologies. Another idea is published in [7] which binds collections of related components with wrapper interfaces. The wrapper acts as an intelligent interface between the collection and domain applications. The idea is aimed on an effective source code reuse.

In contrast to other approaches, the proposed method in this paper additionally treats the aspect that software today contains specific hard coded information which makes exchange of hardware a difficult task. The second part is related to the software deployment problem where software components (SWCs) are allocated to ECUs. Here, the best mapping of SWCs to ECUs regarding given quality aspects must be found. The problem has been covered from a lot of researches in the past years [8], [9], [10], [11]. The technical survey [12] summarizes different optimization strategies and notices that evolutionary algorithms are the most used ones.

In this paper, an approach is suggested which enables the possibility of changing hardware components within an existing system by building up generic interfaces between the ECUs. Before implementation, the total amount of affected internal parts of the ECUs will be reduced by a deployment optimization. This will decrease the influence of a hardware

change and provide a very flexible system. To demonstrate the difficulties, a subsystem of an electric driven vehicle is used. The practicability of the proposed approach is shown and additionally an existing solution from the automotive area serves as a reference. Then, feasible solutions can be compared with the amount of changes, the resulting communication and the influence of hardware specific information. This idea cannot only be applied in the automotive industry, but also in highly connected systems like a plant or production line, an airplane or even in a software itself. Here, e.g. if software functions are allocated to modules which are connected via a virtual bus an effort on the interfaces of each module or even signals in the software can be saved. The optimization level can be refined from whole components with remodularization [13] up to source code with genetic programming [14].

The rest of the paper is organized as follows: In Section II, the problem is stated in detail on a real world example which serves to derive the tasks of the paper. The solution of the tasks is described in the first part of Section III. The second part is devoted to the description of the proposed algorithm. This method is evaluated on a realistic problem, which is described in Section IV. Experimental results are presented and discussed in Section V. The paper closes with some conclusions as well as an outlook on future work in Section VI.

II. Problem Description

The motivating example is a subsystem of an electric driven vehicle with focus on the high voltage (HV) battery. The electric powertrain of the vehicle typically consists of an electric engine, which also works as a generator for recuperation, the power electronics, a DC/DC converter, a high voltage battery, the whole cooling circuit (cooling pump, high voltage air condition, valves etc.) and an on-board charger. All these hardware components are controlled by different ECUs which contain the logical functional level. For example, there are the battery management system (BMS), energy and drive management (EMM), thermal management and the operation strategy, as partly shown in Fig. 1. These systems possess software modules which conclude functions that are summarized in a common context. The functions realize algorithms which need specific information about the given hardware components and their properties. Related functions are spread over the whole system and communicate via signals. The signals are specified by a document and are available on a specific interface. Here, messages are transferred over a connection between several ECUs, e.g. a communication bus like CAN or LIN, which are widely spread in the automotive industry. Inside an ECU, the signals are distributed to the required function.

Up to now, the architecture and its hardware components do not change in general, especially if the product is on the market. Before that, only the following reasons force the manufacturer to exchange hardware components:

- problems with the production (amount, time, costs, failure, bankruptcy of a supplier),
- unfulfilled requirements (quality is worse than expected)

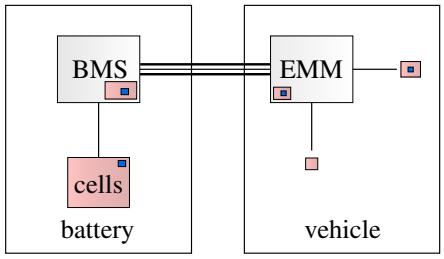

Fig. 1. Example of an electric driven car with focus on the high voltage battery. Specific hardware information of the battery cells is spread over the whole system (red) and specific hardware component information changes in depended ECUs and their functional entities are shown (blue).

After a successful development, further reasons for an exchange of a hardware component are that new alternatives are available. Quality aspects in general are e.g. efficiency, energy, power, costs, weight, safety, low/non-maintenance, end-of-life. In the previously explained example, technical and chemical progress leads to a better performance of batteries which results e.g. in the ability of faster charging, fewer cooling amount, less aging and a wider operation window etc. There are several reasons for an exchange of a component and especially in the sense of the very long lifetime of cars, the industry must concentrate on concepts with generic interfaces to make component exchanges easy.

The biggest challenge, except that the dimension must fit, are the necessary software modifications in the related ECUs. A scenario can be that the exemplary system is changed because of the usage of a new battery cell type. The effect on the whole system is represented in Fig. 1 by the battery/cell information modifications in blue.

The effect of specific hardware component information changes in the depended ECUs and their functional entities are not completely predictable. This leads to the following process:

1) analysis of the system where specific hardware component information is needed,
2) communication of new properties of the hardware component,
3) implementation of changes in the software,
4) verification of the software by testing all affected components themselves and in the interaction with other ECUs.

This must be done in Software/Hardware in the Loop (SiL/HiL) tests, on a test bench and in prototypes or field studies. It can be seen that this process leads to a huge effort for the manufacturer which will result in high cost and makes hardware changes very unlikely.

III. Proposed Approach

The idea is to enable the operation of the vehicle independent of the actual used hardware components (e.g. HV battery in Section II), which means that no specific adjustments of the vehicle related ECUs are necessary if a specific hardware is exchanged. An additional idea related to the problem is that

the influence of the new hardware can be minimized during the design phase by identifying depended software functions and relocate them to the exchangeable hardware, if that is possible. This is part of the software component deployment problem which was treated in a previous paper [11]. Here, the communication overhead and the amount of changes in comparison to an original system is considered. Both aspects cannot be ignored and must be integrated in this new approach.

The necessary step to improve the described process in Section II is that the software functions should be build robust. This means that they must consider the possibility that specific hard coded information can be modified during the lifetime of the system. Additionally, the specific information must be available for all related ECUs. Then, a new hardware component must be tested by itself and its interaction with the vehicle interface, but in contrast to the current practice, not every related component and the respective interaction has to be tested. This means that only testing must be accomplished by the manufacturer of the component.

To provide the suggested process, a higher effort during the development of the software functions in form of analysis and specification is necessary. With these documents, algorithms must be modified to be robust and not custom built for one fixed system configuration. To gain the robustness, an additional error handling must also be considered which reacts on cases if the specific information is not available. Besides that, after a successful implementation, both extensions must be tested concerning the interaction. Despite the one-time necessary effort, the idea promises a lot benefits, e.g. hardware exchanges are easier in late development stages, less total effort by new developments is required which saves money and time, as well as benefits from the improvements of the new components. Furthermore, the approach leads to more competition on the market for the exchange components and a wider use of platform components. Additionally, a higher readiness for owners to invest in their vehicles and not only to maintain them, enables a longer use of resources.

A few assumptions are necessary to outline the borders of the approach in this paper. One requirement is that an existing system is available, which builds the base of the design optimization. Furthermore, the hardware and ECU structure is fixed and the original deployment of software to ECUs is given. Additionally, a preparatory work is necessary for the optimization in which an analysis of the complete functional level must be done. The functional level is represented by software modules, e.g. the battery management system (BMS) and the energy management (EMM), which are introduced in Section II. The modules consist of functionally grouped blocks (FBs) that contain software components (SWCs) and their connections, see Fig. 2. Exchanged information between ECUs over the communication interface can be found in a bus message description. But this is only a first indicator for the necessary exchange. A closer look in the SWCs provides which hard coded specific information about hardware components is required.

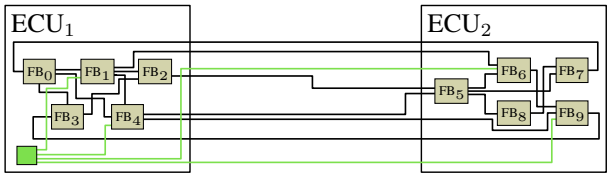

Fig. 2. The system is extended with parameters in green. An example of one parameter is illustrated which is transferred to the requester SWCs. For clarity reasons other parameters of ECU_1 and ECU_2 are not visible.

TABLE I
COMPARISON OF SIGNALS AND PARAMETERS.

	signals	parameters
changes	dynamic	static
information	current values	fixed values
transfer Rate	with frequency f	once
effect	communication overhead	degree of generalization
example	current power	nominal capacity

A. Architecture Extensions

For the transmission of information, two different types are used. Information which is transferred cyclically or per request is described as **signals** because of its dynamically changing content. Static information which does not change during the operation of the vehicle is stated as **parameters**. The parameters represent the previously described hard coded specific hardware component information. Examples are manufacturer specifications, safety limits, type code and initial values. Table I describes the differences between signals and parameters.

The summarization of signals to interactions was published earlier in [11]. This approach is extended to also summarize parameters to interactions which is an important step to reduce the total amount of connections and thereby reduce the amount of SWC duplications before the optimization. Interactions i contain k signals i_{s_k} or parameters i_{p_k}. Interactions can be seen as direct connections between SWCs which provide at least one information. Internal and external information is treated the same way, as it can be changed through a reallocation of a SWC to another host from internal to external information. Signals and Parameters represent only one information. Both are necessary for a distinction of one information with which other SWCs this data is shared. This can be seen in Fig. 3 where e.g. the provided information of SWC_b is shared by five connections which could be required by three SWCs, so the information paths can be reduced to only three interactions. That reduces the amount of SWCs duplicates from five to three, too.

To realize the idea of exchanging components, additional to signals, parameters must be communicated from the components, too. This will be done by transforming the parameters into signals and broadcast them over the system, see Fig. 2 and 3. This can be done e.g. in an initial phase during the first start with the new component. Another solution would be an update of parameter values of the ECUs in a car

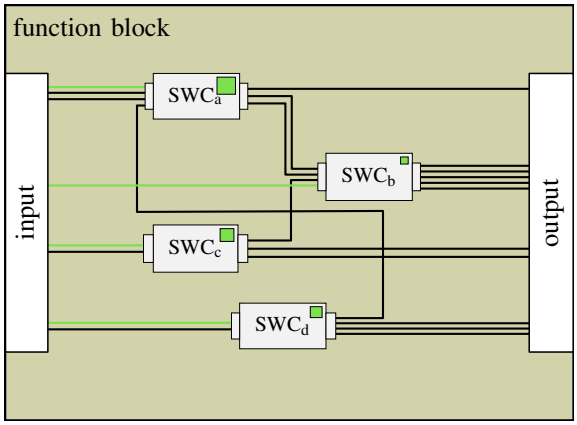

Fig. 3. Representation of a function block (FB) with software components (SWCs), the belonging parameters and signals. Green squares represent an exemplary amount of parameters for each SWC.

repair workshop. In automotive applications in the past, it happened that parameters or at least static information had to be transferred over the interface as signals. This information now can be integrated in the new approach in one single signal and needs to be transmitted only once.

However, in most cases, there is an existing system which is nearly optimal in relation to certain previous quality criteria and improvements can only be achieved by modifications or reallocation of software functions. To gain the best possible solutions, an automated optimization process is required.

B. Optimization Settings

For the stated problem, a search-based optimization method is chosen. The most commonly used one in the field of evolutionary algorithms is the non-dominated sorting genetic algorithm (NSGA-II) [15]. It is a modification of the first version, which both are published by Deb et al. [16]. The second version is easier to use and yields good solutions for multi-objective problems in the shortest time by providing fast non-dominated sorting and an efficient crowding distance metric. In 2014, with NSGA-III [17], a newer version was introduced which is recommended for optimization problems with more than three objective functions. Instead of the crowding distance, uniformly distributed reference points are used to ensure the diversity of the population.

As input for the optimization a case study must contain SWCs, interactions, ECUs and the bus. The problem is defined as the software component deployment problem where the algorithm is allowed to create different allocations of the SWCs to the ECUs. E.g. if there are two ECUs and 95 SWCs, then the search space is 2^{95} which is an amount of approximately $3.14e+28$ solutions, because each of them can cause different performance values regarding to the quality criteria, see Section III-D. Additionally, constraints are defined for each ECU by a location list, which describes the permission of the SWCs to be allocated to this ECU. The lists divide the SWCs into ECU dependent and independent ones, whereof only independent SWCs can be allocated freely by the algorithm. As output representation, a list of the non-dominated solutions is chosen. This represents the Pareto-optimal set. It contains the solutions which are not dominated by others for at least one of the criteria.

C. Optimization Procedure

The procedure is divided into five stages. Fig. 4 shows the general algorithm, where extensions are highlighted in red. The input of the optimization is extended with the parameters and the deployment of the original solution. During initialization, a population of n solutions is created whereof the original solution is one and the remaining are randomly allocated. All solutions are evaluated on up to five quality aspects (c_{com}, cnt_s, cnt_p, cnt_{mod} and cnt_{changes}), which will be explained in the next section. In the next stage, the optimization begins with the reproduction. This is done by means of typical operations like mutation and crossover, where new solutions are created by winners of the tournament selection from the mating pool. Numerical values can be found in Section IV-B. An acceleration of the process is achieved by the usage of the localization information during the reproduction, i.e. only valid solutions are generated. After the evaluation, the solutions are sorted (by rank and their crowding distance) and only the best n solutions survive. The process is repeated until the stop criterion is met. In this paper, the maximum number of evaluations for one single run is chosen as stop criterion.

D. Objective Functions

The quality objective c_{com} calculates the inter-ECU communication, which is also described as communication overhead. During the calculation, it is checked if an interaction i_j connects two components that are assigned to different ECUs. The method counts the data amount only once per cycle because it is transferred via a bus where the identical information is combined to one signal. The receiving ECUs will manage the distribution of the signals to the SWCs internally. This is done by the collection $list_{transferred}$, which stores the unique identifier of a transferred signal information. Simultaneously, the number of signals cnt_s is calculated by the amount of different information which must be exchanged between different ECUs:

$$c_{\text{com}} = \sum_j \sum_k i_{j,s_{\text{freq},k}} * i_{j,s_{\text{data},k}} * ext_{i_{j,s}} * valid_{s_k}, \quad (1)$$

$$ext_{i_{j,s}} = \begin{cases} 1, \text{ if } i_{j,s_{\text{src}}} \neq i_{j,s_{\text{dest}}} \\ 0, \text{ else} \end{cases}, \quad (2)$$

$$valid_{s_k} = \begin{cases} 1, \text{ if } s_k \notin list_{transferred} \\ 0, \text{ else} \end{cases}, \quad (3)$$

$$cnt_s = \sum_k valid_{s_k}. \quad (4)$$

Neglecting the ECU failure rate, network delay, and reliability of the bus, which all three affect the communication, only the quantitative amount of data is assessed. However, these features could be easily integrated in the proposed method.

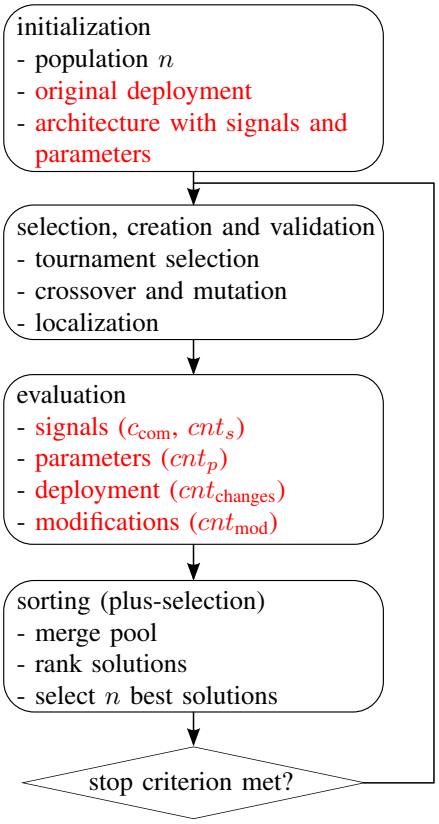

Fig. 4. Optimization procedure of the genetic algorithm with extensions highlighted in red.

Furthermore, it is assumed that the acceptance of a solution is reverse proportional to the amount of SWC allocation changes based on the original deployment. Changes of the deployment cause organizational effort, e.g. to move either the respective responsibilities and expertise to another team, or to move people from one team to another. The fewer changes cnt_{changes} are necessary, the higher the acceptance will be, see Eq. (5). The amount of changes is calculated by the differences of the current deployment $\boldsymbol{p}$ and the initial deployment $\boldsymbol{p}_{\text{ref}}$:

$$\text{acceptance} \sim \frac{1}{cnt_{\text{changes}}}, \tag{5}$$

$$cnt_{\text{changes}} = \sum_i diff(\boldsymbol{p}[i], \boldsymbol{p}_{\text{ref}}[i]), \tag{6}$$

$$diff(\boldsymbol{p}[i], \boldsymbol{p}_{\text{ref}}[i]) = \begin{cases} 1, \text{ if } \boldsymbol{p}[i] \neq \boldsymbol{p}_{\text{ref}}[i] \\ 0, \text{ else} \end{cases}. \tag{7}$$

To optimize the system towards a generic interface, the number of parameters cnt_p which must be transferred between ECUs should be next to zero. Parameters are treated in the same way as signals, so that the parameter interactions will be checked up on a connection between different ECUs. The information counts only once over the interface, see Eq. (3) and (4). Here, instead of signals, all parameters will be checked whether they are element of the collection $list_{transferred}$.

Another criterion for the acceptance of the solution is the amount of necessary modifications cnt_{mod} of the affected function. It will be counted by the parameter interactions i_p over the interface $cnt_{\text{ext\_interactions},p}$ with the current deployment:

$$cnt_{\text{mod}} = cnt_{\text{ext\_interactions},p}, \tag{8}$$

$$cnt_{\text{ext\_interactions},p} = \sum_j \text{ext}_{i_{j,p}}, \tag{9}$$

$$ext_{i_{j,p}} = \begin{cases} 1, \text{ if } i_{j,p_{\text{src}}} \neq i_{j,p_{\text{dest}}} \\ 0, \text{ else} \end{cases}. \tag{10}$$

Again, parameter interactions differ either in the destination or source SWC, as they are always be provided by one SWC on an ECU. Therefore, the amount of interactions over the interface is equal to the amount of affected SWCs. The distinction between both criteria is necessary to prefer those solutions which affect fewer SWCs by the same amount of parameters and to keep solutions which require less modification but more parameters. For example, in one case, a parameter is required by three SWCs, in another case a different parameter is only required by two SWCs.

The amount of data of the parameters would be another criterion, similar to communication overhead, but is neglected because the transmission has non or only a little effect on the bandwidth, as it must be done only once during initialization (see Section III-A). Further evaluation criteria could be the data transmission reliability, scheduling length, and response time to analyze the real-time ability of a solution, which is not the focus of this contribution.

IV. Experiment

The approach is demonstrated on a subsystem of an electric driven vehicle with focus on the high voltage battery management system (BMS). It represents a real world problem with functions of a plug-in hybrid vehicle. This type is chosen, because it possesses functions for fully electric driving, driving with the combustion engine and the combination as well as charging with an on-board loader. With the usage of the proposed approach, the aim is to make the battery independent from the rest of the vehicle while providing the original functionality.

A. Problem Instances

Besides the BMS, the case study consists of the energy management as well as other logical control units which are typical needed for an EV. The hardware structure is set to two hosts (BMS and the rest of the vehicle) that are connected by one bus, see Fig. 1.

The software architecture consists of 147 SWCs for the functions and two SWCs which contain the specific hardware information. Each SWC has the opportunity to provide and to receive information through in- and outputs. Overall, there are 463 dynamic values available which are outputs of the components. These values are received through 1104 connections, called signals, which build the inputs of the components. In average, the SWCs are connected with more than 7 signals.

Furthermore, the signals are characterized by data amount and frequency. Due to the fact that internal signals are all timed with the same sample rate of their ECU, one fixed rate is set for both, internal signals and signals over the interface.

The hard-coded information which can be found in the components results in 148 static values whereof 129 belong to ECU_1 and 19 to ECU_2. The values are needed at 199 input ports. These connections represent the parameters. In the next step, signals or parameters with the same source and destination component are combined to interactions. The parameter interactions are necessary to evaluate the amount of SWCs which require parameters from another ECU. In total, there are 80 parameter interactions which means that 80 SWCs require at least one parameter (72 with the origin from ECU_1 and 8 from ECU_2). An existing application is used as reference deployment for the optimization which allocates 60 SWCs on ECU_1 and 89 SWCs on ECU_2. 37 SWCs on ECU_1 are hardware dependent due to hardware closely functionality like sensor value interpretation. 17 of the 89 SWCs on ECU_2 are hardware dependent, too. After the subtraction of those SWCs, there are 95 SWCs left which can be allocated freely on both ECUs by the optimization process. The original solution causes a communication overhead of 1160 bits per cycle with 68 signals. Additionally, 81 parameters must be transferred which affect 45 SWCs.

As the used algorithm minimizes the specified quality criteria, no resource restrictions are necessary for the optimization. This fact makes the approach interesting for a phase of system design when the used hardware is not yet selected. Furthermore, a perfect transmission channel is assumed that has no delay, no failure rate, and the transmission rate is unlimited. The approach is not limited to two hosts, but as the amount of ECUs is the base and the amount of SWCs the exponent, in this case 2^{95}, the search space is strongly increasing by considering more than two ECUs.

B. Algorithmic Settings

The proposed approach is integrated within the free available framework ArcheOpterix [18], [19]. The source code of the NSGA-III is taken from [20] and added to the framework. For experiments with three objectives, NSGA-II is used whereas for all five objectives NSGA-II and NSGA-III are compared to decide which one is better for the specific problem instance. The number of divisions in NSGA-III is problem dependent and a short investigation with nine different values shows that the best results can be achieved with six divisions, which result in 210 reference points. For more details on this algorithm see [17]. A second investigation consists of different settings for the remaining algorithm parameters, which can be seen in Table II. The aim is to figure out which algorithm should be used for this specific problem with five quality criteria. The optimization is able to improve solutions within 10000 evaluations until the stop criterion is met. The reproduction rate is set to 0.5 and 10 runs per setting are executed. The mating pool is chosen to a factor of 1.5 of the population size. During the initialization

TABLE II
OPTIMAL ALGORITHM SETTINGS FOR NSGA-II AND NSGA-III.

numerical values	NSGA-II	NSGA-III
evaluations	10000	
reproduction rate	0.5	
mating pool	1.5	
initialization with reference solutions	yes	
population size	20	
mutation rate	1	0.9
crossover rate	0.2	0.1
NSGA-III divisions	-	6

TABLE III
COMPARISON OF NSGA-II AND NSGA-III FOR FIVE OBJECTIVES.

hypervolume	NSGA-II	NSGA-III
without original solution	0.9371	0.8425
with original solution	0.9382	0.9316
with eight Pareto solutions	0.9390	0.9298

it is possible to decide if the original solution is added to the population. Furthermore, it is possible to add other solutions to guide the algorithm. Here, the eight Pareto solutions from the first experiment in Section V are chosen. As the investigation is not the main content of this paper, only the main results will be presented. The hypervolume is chosen as performance metric. It assesses the quality of the solutions found in the genotype (evaluation) space and first appeared in [21]. The quality values are normalized which is why the maximum value of the hypervolume is one. The resulting values can be found in Table III, which represent the average of ten runs with the best found settings for each algorithm, see Table II. A large population size of 500 elements is used for the start and reduced to 100, 50 and finally to 20 elements. Then, the mutation and the crossover rate are modified in a range from 0.01 to 1.

It can be seen that NSGA-II performs better than then the newer version, therefore only this algorithm is used for the following experiments. Here, due to the risk of sub-optimal solutions, the evaluation limit is increased to 50000 and 60 runs will be executed.

V. RESULTS & DISCUSSION

In the first part, the original solution is investigated in relation to the communication effort, signal amount and the necessary deployment changes. In the second part, the parameter amount and the affected SWCs are taken into account together with the previous criteria. The experiments are executed on an Intel Core i7 CPU with 2.67GHz and 18GB RAM. One single run takes about 3 minutes for three criteria and 41 minutes for five criteria. With parallelization by six cores, the whole execution takes less than 7 hours. Table IV summarizes the most important results of both investigations.

TABLE IV
RESULTS OF THE EXPERIMENTS.

evaluation	original	sol_1	sol_2	sol_3
comm. overhead (c_{com})	1160 bits	-1.4%	+0.3%	+176.8%
signals (cnt_s)	68	+1.5%	-5.9%	+163.2%
parameters (cnt_p)	81	+9.9%	0%	-84.0%
modifications (cnt_{mod})	45	47	46	10
changes ($cnt_{changes}$)	0	3	2	43

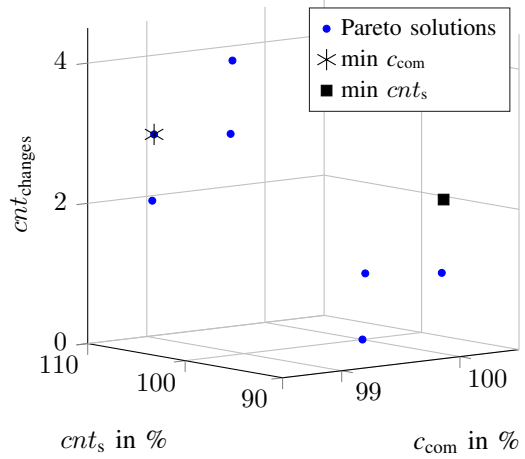

Fig. 5. Results for parameter amount minimization in comparison to signals.

Fig. 5 shows the Pareto solutions for the first test. Even though 60 runs are executed, only eight solutions are found. Solution sol_1 possesses the best communication effort improvement of 1.4% from 1160 to 1144 bits/cycle, but takes a slight increase of the signal amount, just like the change amount which requires three SWCs more on ECU_2. Solution sol_2 possesses the lowest signal amount with only 64 signals instead of 68, which is an improvement of 6%. The solution causes a slight increase of the signal data which leads to a bit more of communication effort and two changes in the deployment are necessary. The used original solution here is already nearly optimal for the communication of dynamic information. Better solutions can be found, but they are only compromises of the given quality criteria and no huge improvements can be achieved. It is a result of the strong connectivity of the SWCs in the ECUs of the original deployment. Only 153 of the 1104 signals are external connections between the two ECUs, which is a proportion of 13.8%. The rest of the 956 signals are communicated internally and lead to an increase of the interface if they are allocated on the other ECU.

For the next test, the genotype space is extended to five objectives by the amount of parameters that must be transferred over the interface and the affected SWCs which must be modified to be robust against parameter changes. The original solution requires 81 parameter transfers and 43 modified SWCs to gain a generic interface. This is quite a large number and means a lot of effort for implementation and testing. Solution sol_1 of the first part has a negative effect of the

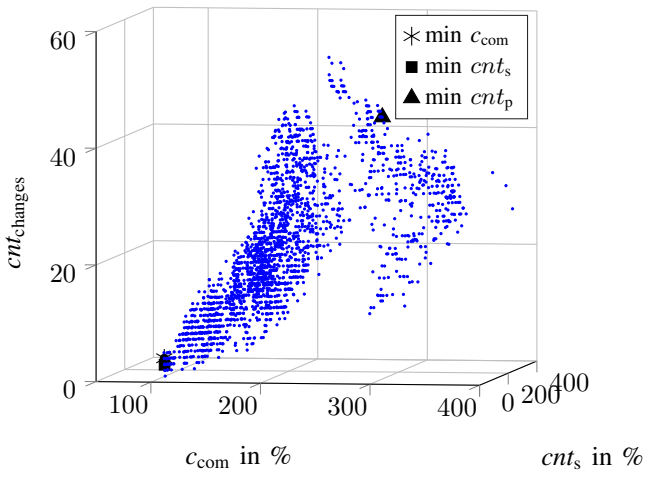

Fig. 6. Pareto optimal solutions for five quality criteria shown for the deployment changes and the signal interface aspects.

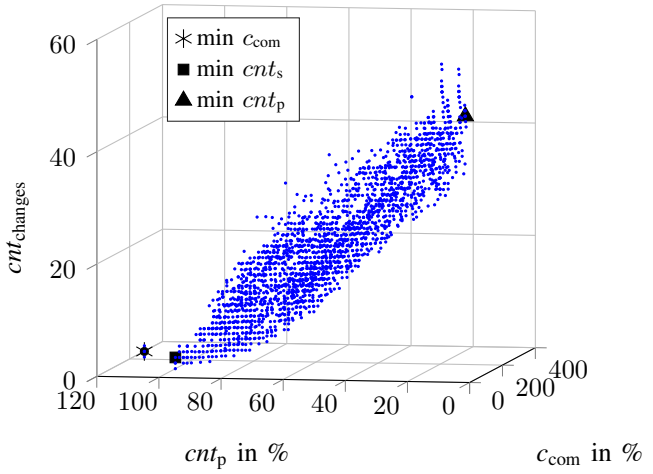

Fig. 7. Pareto optimal solutions for five quality criteria shown for the deployment changes, parameter minimization and the communication overhead.

required parameter amount. Eight parameters more must be transferred, which is an increase of nearly 10%. Here, even 47 SWCs have to be modified, whereas only three SWCs must be moved to the other ECU. The few changes mean a little organizational effort, but the high number of modifications leads to an improbable acceptance. Solution sol_2, however, does not affect the number of parameters over the interface, but one SWC requires a parameter which is already transferred, while 46 SWCs must be modified. Fig. 6 and 7 show the 2223 Pareto optimal solutions found for the allocation and the interface criteria during the 60 runs. It can be seen that there are many solutions which reduce the total parameter amount, but also cause a huge increase of the dynamic communication. It would be expected that a solution with zero parameters can be achieved, but only solution sol_3 is found which possesses the best value for the parameter minimization. Here, 13 parameters instead of 81 must be communicated which is an improvement of 84%. This solution represents a deployment

where all parameter affected hardware independent SWCs are allocated to ECU_1. The remaining parameters over the interface result from SWCs that cannot be moved (5) or would require more parameters from the other ECU (8) which is why sol_3 really represents the minimal parameter deployment. 10 interactions contain the parameters which means that only 10 SWCs must be modified for robustness regarding parameter changes. Unfortunately, the other criteria are almost 1.8 times more data and 1.6 times more signals that must be transferred. Furthermore, the reduction of the parameters leads to a huge increase of the deployment changes with 43 SWCs from ECU_2 to ECU_1. Even though the solution possesses the minimum effort for modifications and testing, the great organizational effort as well as the higher utilization of the bus lead to a very low acceptance. As mentioned before, this extreme solution is not really necessary because parameters can be transferred without affecting the bus load.

To decide which solution should be used for a generic communication interface, both aspects, the necessary effort for robustness and move of SWCs, must be evaluated in more detail. Then, a limit for the bus load must be set and it must be checked if the algorithms can use available information from the new host which could make further reductions possible.

VI. Conclusions

In this paper, the problem of hard coded specific hardware component information of an existing system was addressed. The goal was to modify the system that generic interfaces between the vehicle and exchangeable hardware components are possible. An approach was presented which transforms the specific static information into parameters and sent them over the interface.

An exemplary case study needs 45 SWCs which must be modified to gain a robust generic interface. On the basis of the expected effort for the modification, a software component deployment optimization was used to reduce the total amount of parameters over the interface. The best solution achieves a reduction of about 84% with only 10 SWCs to be modified, but results in 43 changes of SWC from one host to another, which will be expected as unlikely to be accepted due to the required company reorganization.

Future work will comprise a detailed estimation of the effort for generalization and deployment changes. Additionally, a method for optimizing algorithms with internally available information is a desired goal which enables the possibility to reduce interface communication after deployment changes.

References

[1] R. N. Charette, "This Car Runs on Code," *IEEE Spectrum*, Feb. 2009. [Online]. Available: http://spectrum.ieee.org/transportation/systems/this-car-runs-on-code

[2] IEC CD 61800-7, Adjustable speed electrical power drive systems, Part 7: Generic interface and use of profiles for power drive systems. International Electrotechnical Commission. [Online]. Available: http://www.iec.ch/

[3] A. Aue and E. Becker, "Method for providing a generic interface and microcontroller having a generic interface," Dec. 4 2014, uS Patent App. 14/288,028. [Online]. Available: http://www.google.com/patents/US20140359180

[4] [Online]. Available: http://www.autosar.org

[5] [Online]. Available: http://www.autosar.org/about/basics/

[6] A. M. Phillips, "Functional decomposition in a vehicle control system," in *American Control Conference, 2002. Proceedings of the 2002*, vol. 5, 2002, pp. 3713–3718 vol.5.

[7] H. M. Haddad, "Integrated Collections: Approach to Software Component Reuse," in *Third International Conference on Information Technology: New Generations (ITNG'06)*, April 2006, pp. 28–33.

[8] P.-Y. R. Ma, E. Lee, and M. Tsuchiya, "A Task Allocation Model for Distributed Computing Systems," *IEEE Transactions on Computers*, vol. 31, no. 1, pp. 41–47, 1982.

[9] I. Moser and S. Mostaghim, "The automotive deployment problem: A practical application for constrained multiobjective evolutionary optimisation," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, July 2010, pp. 1–8.

[10] A. Aleti and I. Meedeniya, "Component Deployment Optimisation with Bayesian Learning," in *Proceedings of the 14th International ACM Sigsoft Symposium on Component Based Software Engineering*, ser. CBSE '11. New York, NY, USA: ACM, 2011, pp. 11–20.

[11] H. Klee, M. Buchholz, T. Materna, and K. Dietmayer, "Acceptance-Based Software Architecture Deployment for Improvement of Existing Applications," in *Computational Intelligence, 2015 IEEE Symposium Series on*, Dec 2015, pp. 1832–1837.

[12] A. Aleti, B. Buhnova, L. Grunske, A. Koziolek, and I. Meedeniya, "Software Architecture Optimization Methods: A Systematic Literature Review," *IEEE Trans. Software Eng.*, vol. 39, no. 5, pp. 658–683, May 2013.

[13] W. Mkaouer, M. Kessentini, A. Shaout, P. Koligheu, S. Bechikh, K. Deb, and A. Ouni, "Many-Objective Software Remodularization Using NSGA-III," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 3, pp. 17:1–17:45, May 2015.

[14] W. B. Langdon and M. Harman, "Optimizing Existing Software With Genetic Programming," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 1, pp. 118–135, Feb 2015.

[15] A. S. Sayyad and H. Ammar, "Pareto-optimal search-based software engineering (POSBSE): A literature survey," in *Realizing Artificial Intelligence Synergies in Software Engineering (RAISE), 2013 2nd International Workshop on*, May 2013, pp. 21–27.

[16] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, April 2002.

[17] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 4, pp. 577–601, Aug 2014.

[18] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya, "ArcheOpterix: An extendable tool for architecture optimization of AADL models," in *ICSE Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES)*, May 2009, pp. 61–71.

[19] I. Meedeniya, A. Aleti, I. Avazpour, and A. Amin, "Robust ArcheOpterix: Architecture optimization of embedded systems under uncertainty," in *ICSE Workshop on Software Engineering for Embedded Systems (SEES)*, June 2012, pp. 23–29. [Online]. Available: http://mercury.it.swin.edu.au/g_archeopterix/experiments/SEES2012/

[20] D. Hadka, "MOEA Framework - A Free and Open Source Java Framework for Multiobjective Optimization. Version 2.9," 2015. [Online]. Available: http://www.moeaframework.org/.

[21] E. Zitzler and L. Thiele, *Multiobjective optimization using evolutionary algorithms — A comparative case study*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 292–301.