C-PO: A Context-Based Application-Placement Optimization for Autonomous Vehicles

Tobias Kain*, Hans Tompits†, Timo Frederik Horeis‡, Johannes Heinrich‡, Julian-Steffen Müller*, Fabian Plinke‡, Hendrik Decke*, and Marcel Aguirre Mehlhorn*
*Volkswagen AG, Wolfsburg, Germany
{tobias.kain | julian-steffen.mueller | hendrik.decke | marcel.aguirre.mehlhorn}@volkswagen.de
†Technische Universität Wien, Vienna, Austria
tompits@kr.tuwien.ac.at
‡IQZ GmbH, Hamburg, Germany
{horeis | heinrich | plinke}@iqz-wuppertal.de

***Abstract*—Autonomous vehicles are complex distributed systems consisting of multiple software applications and computing nodes. Determining the assignment between these software applications and computing nodes is known as the *application-placement problem*. The input of this problem is a set of applications, their requirements, a set of computing nodes, and their provided resources. Due to the potentially large solution space of the problem, an optimization goal defines which solution is desired the most. However, the optimization goal used for the application-placement problem is not static but has to be adapted according to the current context the vehicle is experiencing. Therefore, an approach for a context-based determination of the optimization goal for a given instance of an application-placement problem is required. In this paper, we introduce C-PO, an approach to address this issue. C-PO ensures that if the safety level of a system drops due to an occurring failure, the optimization goal for the successively executed application-placement determination aims to restore the safety level. Once the highest safety level is reached, C-PO optimizes the application placement according to the current driving situation. Furthermore, we introduce two methods for dynamically determining the required level of safety.**

***Index Terms*—autonomous driving, application-placement problem, context-based optimization**

I. Introduction

The *application-placement problem* [1] is a known challenge in the area of autonomous driving. Multiple software applications which are required to operate a vehicle autonomously have to be assigned to the computing nodes installed in the vehicle, whereby various constraints have to be satisfied. Since the quality of an application placement strongly depends on the current context the vehicle is in, a method for optimizing application placements taking the current context into account is required. In this paper, we present such an approach, called C-PO, which is specifically designed for autonomous vehicles. In C-PO, the context comprises both the *internal system state* of a vehicle as well as *external environment observations*, like the current weather or road conditions.

Optimizing the application placement according to the current context is essential as the quality of the application placement directly influences customer satisfaction. Indeed, neglecting the optimization of the application placement will commonly cause that context changes, which may occur as a result of, e.g., hard- or software faults, have to be handled by performing emergency stops in order to guarantee the safety of the passengers and other road users. Such behavior is not desirable since it will cause customer satisfaction to decline [2].

To avoid that occurring context changes require performing emergency stops as pointed out above, the main objective of C-PO is to increase the level of safety with each executed placement optimization. The idea of C-PO is thereby to subdivide the configuration graph—a graph representing the relation between the different application placements—into multiple layers, whereby the layers correspond to safety levels. For each layer, C-PO requires the definition of an optimization goal that aims to reach a better safety level. Once the highest safety level is reached, the optimization goal is determined based on the current driving situation. Besides the general approach of C-PO, we also define a concrete instance of C-PO based on priority levels and show that it fulfills the properties required by C-PO.

As the required level of safety depends on the current context, we introduce two architectures for a dynamic adoption of the safety level: a *dynamic-level approach* and a *dynamic-prioritization approach*. The idea of the former is to adjust the number of levels based on the current context while the latter approach adapts the priority of the applications according to the current situation.

Optimizing the application placement is a well-known research challenge. For instance, Kumar et al. [3] discuss a reconfiguration and placement optimization approach for platoons of autonomous vehicles based on bond-graph modeling. By determining offsets during operation, measures are taken at different levels of the system. The measures can range from switching off individual wheels to switching off entire vehicles. Cooray et al. [4] introduce a framework, called RESIST, for a proactive reconfiguration for situated software systems. This framework aims to maintain the reliability of the systems due to a dynamic optimization of the architectural configuration. Lapouchnian et al. [5] employ a goal-oriented requirements-engineering approach to adapt a system dynamically to react to occurring environmental changes. Finally, Hemmati et al. [6] study the *redundancy-allocation problem*, which faces similar optimization challenges as the application-placement problem.

978-3-9819263-5-4/DATE21/©2021 EDAA

The idea there is to use a multi-objective harmonic search algorithm to determine an optimal redundancy configuration to maximize the mean time to the first failure.

C-PO is distinct from these approaches as they do not aim for optimizing the application placement regarding safety. Furthermore, to the best of our knowledge, C-PO is the first approach that introduces an application-placement optimization based on safety levels.

Our paper is organized as follows: Section II discusses the application-placement problem and Section III introduces the general method of C-PO as well as a concrete instance of it. In Section IV, we present two approaches for a dynamic adjustment of the safety level and a comparison of those ideas. Section V concludes the paper with a brief summary and an outline for future work.

II. The Application-Placement Problem

The task of determining the assignment between applications and computing nodes is referred to as the *application-placement problem* [1]. Such an assignment, which we refer to as *configuration*, has to fulfill certain constraints. For instance, the resource demands of the application placed on a computing node have to be met. Furthermore, the required level of redundancy of an application, i.e., the number of application instances that are executed by the system, and the level of hardware segregation, i.e., the number of distinct computing nodes the redundant application instances have to be executed on, have to be fulfilled. Note that if the system executes more than one instance of a specific application, we assume that only one instance interacts with the other applications and actuators. This instance is referred to as the *active instance*. The other redundant instances, which are referred to as *active-hot instances*, are not interacting with the other applications or the actuators of the vehicle.

To discriminate among a potentially large number of solutions for a given instance of an application-placement problem, an optimization function specifies which valid node assignment is desired most. Conceivable optimization goals are, e.g., maximizing the number of computing nodes that execute no applications or maximizing the redundancy of a specific class of applications.

Events that trigger the computation of an application placement are, e.g., the occurrence of a failure or a change in the driving situation. The interrelations of the application placements can be represented in the *configuration graph*, which is a graph where the nodes correspond to configurations and directed edges indicate configuration transitions corresponding to events.

III. The C-PO Approach

As mentioned in the previous section, the application placement can be optimized according to various parameters, whereby, in the case of autonomous vehicles, the optimal application placement strongly depends on the current context. For instance, if the vehicle has encountered safety critical faults, the goal is to determine an application placement that restores the level of safety. On the other hand, if the vehicle is in optimal condition, the optimization goal can be selected based on the current environmental observations.

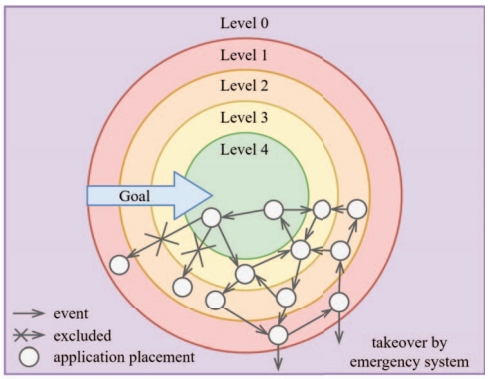

Fig. 1. Visualization of the configuration graph and the layers added by C-PO. The layers on top of the configuration graph subdivides the graph into N levels, where in this example $N = 4$.

In what follows, we introduce our C-PO approach for a context-based determination of the optimization goal for the application-placement problem. The name C-PO stands for "context-based placement optimization".

A. The General Method of C-PO

Fig. 1 illustrates the idea of the context-based application-placement optimization. The approach of C-PO to determine a current optimization goal for the application-placement problem is to add a layer on top of the configuration graph. This layer subdivides the configuration graph into multiple levels, whereby a level number, $x \in \mathbb{N}$, for which $0 \leq x \leq N$ holds, identifies each level, where $N \in \mathbb{N}$ is the number of levels.

The C-PO approach is based on the following four requirements:

(1) The levels need to be defined in such a way that the safety and availability of the system increases as the level number increases. Therefore, level N can be considered as the "best" level, meaning that this level is the most desired one. Accordingly, level 0 is the "worst" level. Since in this level, the minimal safety requirements cannot be satisfied anymore, an emergency system has to take over control of the vehicle and bring it to a safe stop.

(2) For each level greater than 0, specific properties that an application placement of that level has to fulfill need to be defined (like, e.g., minimal redundancy requirements). Furthermore, the levels have to be based on each other, i.e., an application placement of level x has also to fulfill the properties required by all levels y with $0 < y < x$.

(3) Edges are not allowed to intersect more than one level border (note that edges correspond to events). This means that it has to be excluded that an occurring event (e.g., the failure of a computing node) causes a degradation of the level number by two or more. Hence, an event can only cause a drop to the level below the current level, i.e., multilevel jumps are not allowed.

(4) For each level greater than 0, a *goal function* needs to be defined. As already mentioned before, level N is the most

desired level. Therefore, the goal of all the other levels is to get as fast as possible to level N. This can be achieved by a goal function that prioritizes application placements that fulfill as many properties requested by the next level as possible. Once level N is reached, the application placement can be optimized based on the current driving situation.

B. An Instance of C-PO based on Priorities

Above, we laid down the criteria that the levels defined by C-PO have to fulfill. We now describe a concrete instance of the C-PO approach using different application priority classes to define the levels and prove that the specified levels meet the required criteria. We refer to this instance as a *static approach* as the number of levels is fixed. In Section IV, we introduce two *dynamic architectures* for realizing C-PO instances.

The application priority classes used for this instance are the following: $HIGHEST$, $HIGH$, LOW, and $LOWEST$. We assume that each application is of exactly one priority category. In accord to the four priority classes, we set N, the number of levels, to 4. The levels are specified as follows:

- *Level 0*: At least one application of priority $HIGHEST$ cannot be executed anymore.
- *Level 1*: All applications of priority $HIGHEST$ can be executed. The system cannot run at least one application of priority $HIGH$.
- *Level 2*: For each application of priority $HIGHEST$, there is at least one redundant instance, whereby each redundant instance of an application is executed by a different computing node (the level of hardware segregation is two). For each application of priority $HIGH$, at least one instance is executed by the system. The system cannot run at least one application of priority LOW.
- *Level 3*: For each application of priority $HIGHEST$, there are at least two redundant instances, and the level of hardware segregation is three. For each application of priority $HIGH$, there is at least one redundant instance, and the level of hardware segregation is two. For each application of priority LOW, at least one instance is executed by the system. The system cannot run at least one application of priority $LOWEST$.
- *Level 4*: For each application of priority $HIGHEST$, there are at least three redundant instances, and the level of hardware segregation is four. For each application of priority $HIGH$, there are at least two redundant instances, and the level of hardware segregation is three. For each application of priority LOW, there is at least one redundant instance, and the level of hardware segregation is two. For each application of priority $LOWEST$, at least one instance is executed by the system.

Next, we show that (i) for any level $x > 0$, an application placement of level x has also to fulfill the properties required by all levels y, for $1 \leq y < x$, and (ii) multilevel jumps are excluded.

The first property is trivial to prove since the minimum redundancy and segregation requirements are increasing as the level number rises.

To prove that multilevel jumps are excluded, we have to show, for all possible events, that the level gets degraded at most to the level below the current level. In total, we consider two events: (i) a failure of an application instance and (ii) a failure of a computing node; all other events can be reduced to these two.

Assume that the system is currently in level N, for $N > 0$. Therefore, the minimum number of instances and the minimal level of hardware segregation per application per priority category are given as follows: (i) $HIGHEST$: N, (ii) $HIGH$: $\max(N-1, 0)$, (iii) LOW: $\max(N-2, 0)$, and (iv) $LOWEST$: $\max(N-3, 0)$.

The minimum number of instances and the minimal level of hardware segregation per application per priority category of level $N-1$ are given as follows: (i) $HIGHEST$: $N-1$, (ii) $HIGH$: $\max(N-2, 0)$, (iii) LOW: $\max(N-3, 0)$, and (iv) $LOWEST$: $\max(N-4, 0)$.

In case an application instance fails, the number of instances of the corresponding application is reduced by one. If the application exactly matches the minimal required number of instances of level N, then the system will drop to level $N-1$. Otherwise, if the application exceeds the minimal required number of instances, the system remains in level N.

A failure of a computing node causes that all application instances executed by this computing node will also fail. Since, for each application, the minimal level of hardware segregation is equal to the minimum number of application instances, we can assume that for applications that match exactly the minimal requirements of level N, at most one instance is affected by the failure of the computing node. Therefore, this type of failure can be reduced to the failure of an application instance. Thus, multilevel jumps are also excluded in case of a computing node failure.

We have shown that for both types of events, the current level the system is in gets reduced by at most one. Therefore, multilevel jumps are excluded.

Note that multilevel jumps cannot be excluded if the instances of an application share the same dependencies, like, e.g., depending on the same sensor input. Consequently, application instances have to be implemented diversely.

As the goal of levels 1, 2, and 3 is to get to level 4, the optimization goals of those levels are defined as in Table I.

To illustrate the priority-based C-PO instance, let us consider the following scenario: Consider a vehicle with six computing nodes, as shown in Fig. 2. In total, four applications (*App 1* to *App 4*) are executed by the system, where *App 1* belongs to priority class $HIGHEST$, *App 2* to $HIGH$, *App 3* to LOW, and *App 4* to $LOWEST$.

In its initial configuration, the system is in optimal condition since it fulfills all the requirements requested by level 4.

Next, we assume that the active instance of *App 1* fails. Through a switchover to one of the active-hot instances of *App 1*, a total loss of *App 1* can be avoided. Note that level 4 requires that for each application of priority $HIGHEST$, it holds that the minimal number of instances, as well as the minimal level of hardware segregation, is four. Due to the failure of the active instance, *App 1* misses one application

TABLE I
CONTEXT-BASED APPLICATION-PLACEMENT OPTIMIZATION GOALS.

Level	Optimization Goal
1	Try to run one redundant instance of each application of priority $HIGHEST$, where the level of hardware segregation is two.
2	Try to run one additional redundant instance of each application of priority $HIGHEST$, where the level of hardware segregation is three. Try to run one redundant instance of each application of priority $HIGH$, where the level of hardware segregation is two. Run as many applications of priority LOW as possible.
3	Try to run one additional redundant instance of each application of priority $HIGHEST$, where the level of hardware segregation is four. Try to run one additional redundant instance of each application of priority $HIGH$, where the level of hardware segregation is three. Try to run one redundant instance of each application of priority LOW, where the level of hardware segregation is two. Run as many applications of priority $LOWEST$ as possible.
4	Optimization based on the driving situation.

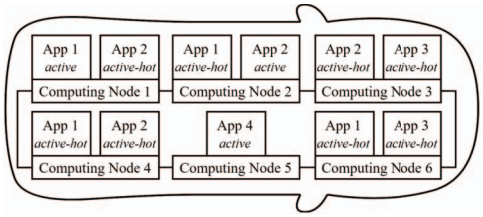

Fig. 2. Initial configuration that satisfies the requirements of level 4.

instance to satisfy the requirements of level 4. Therefore, the successor configuration is not an element of level 4 but rather belongs to level 3.

As mentioned before, the goal of all the non-optimal levels is to perform recovery operations so that the system gets to the highest possible level. Level 4 can be recovered by starting a new active-hot instance of *App 1*.

Once the system is again in level 4, optimizations based on the current driving situations can be performed. Assuming that the goal is to extend the range of the vehicle, a switch to an application placement which utilizes only a subset of all available computing nodes is preferred. Therefore, the active instance of *App 4* is moved to another computing node so that *Computing Node 5* can be shut down. Apart from that optimization, further energy-saving measures can be performed as long as the system is in level 4.

This example shows that, due to a context-based determination of the optimization goals for the application-placement problem, a failure, which caused a drop of the safety level, is handled so that in the end the system is again in the optimal safety level. Fig. 3 shows the path through the configuration graph of the discussed example.

IV. DYNAMICALLY ADJUSTING THE LEVEL OF SAFETY

C-PO allows an optimization based on the current driving situation only if the system has reached level N. In the priority-based level definition of the C-PO instance introduced in Subsection III-B, the highest level requires running many redundant application instances. However, some driving situations, like,

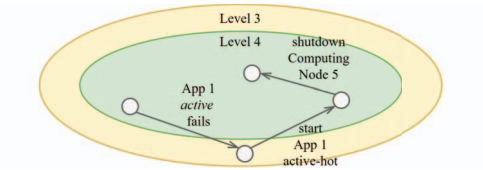

Fig. 3. A path starting and ending in level 4 due to a context-based optimization.

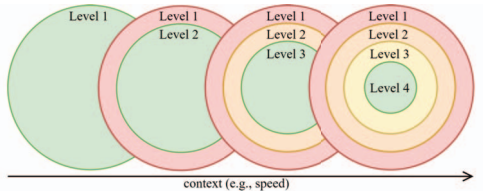

Fig. 4. Dynamic adjustment of the number of levels based on the context.

for example, cruising at low speed in a traffic jam, do not require a highly redundant software architecture.

To address this issue, we introduce in what follows two approaches which allow a dynamic adjustment of the required level of safety. These approaches require that the current context the vehicle is experiencing, e.g., the weather conditions or the asphalt type, can be identified correctly. In case that the context cannot be perceived, a static approach of C-PO, as the one introduced in Subsection III-B, can be utilized.

A. Dynamic-Level Approach

One approach for dynamically defining the safety level is to adjust the number N of levels based on the current context, i.e., levels are added and removed depending on the current situation.

Recall that C-PO requires that an optimization based on the driving situation is only allowed in level N. On the other hand, the optimization goal of all the other levels is to fulfill the requirements requested by the next highest level. Therefore, dynamically adjusting the number of levels causes that the level in which an optimization based on the driving situation is allowed varies. Fig. 4 illustrates the idea of a dynamic adjustment of the number of levels. Note that all levels have to fulfill the properties required by C-PO, i.e., the levels have to be based on each other and multilevel jumps have to be excluded. Moreover, the definitions of the levels do not change dynamically.

Since many different contexts are possible, conflicts regarding the adjustment of the number of levels can emerge. For example, assume a vehicle driving at a very low speed on a road where the sidewalks are filled with pedestrians. Assume further that, based on the present low speed, the required number of levels is two. However, since the vehicle is surrounded by many pedestrians, the minimal requirement is $N = 3$. This means that the latter context demands a higher number of levels than the former. In such a case, always the higher requirements are applied. This ensures that the minimal demands of each context are satisfied.

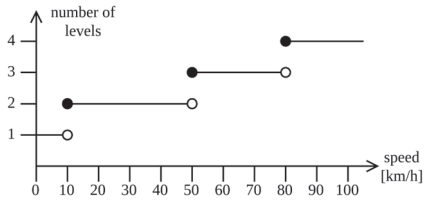

Fig. 5. Function that specifies the number of levels required at different speeds.

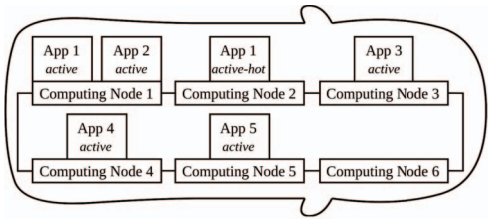

Fig. 6. Configuration of a parked vehicle. The configuration fulfills the requirements of the maximum level, i.e., level 1, in the current context.

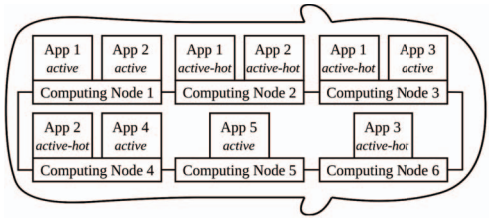

Fig. 7. Configuration of a vehicle driving 70 km/h. The configuration fulfills the requirements of the maximum level, i.e, level 3, in the current context.

TABLE II
SPECIFICATION OF THE PRIORITY CLASS PER APPLICATION AT DIFFERENT SPEED CLASSES. THE SPEED v IS GIVEN IN KM/H.

	$v < 10$	$10 \leq v < 50$	$50 \leq v < 80$	$v \geq 80$
App 1	$LOWEST$	LOW	$HIGH$	$HIGHEST$
App 2	$HIGHEST$	$HIGH$	LOW	$LOWEST$
App 3	$HIGH$	$HIGHEST$	$HIGH$	LOW
App 4	$LOWEST$	LOW	LOW	LOW
App 5	LOW	$HIGH$	$HIGH$	$HIGHEST$

To illustrate the dynamic-level approach, let us consider an example in which the number of levels is adjusted based on the speed of the vehicle. To keep this example manageable, we do not consider any other context besides speed. Furthermore, we reuse the level definition introduced in Subsection III-B. The function displayed in Fig. 5 specifies the correspondence between speed and the number of required levels.

Assuming a car is parked, i.e., the speed in 0 km/h, according to Fig. 5, the number N of levels equals 1. Furthermore, we define that in total five applications (*App 1* to *App 5*) are executed by the system, where *App 1* and *App 2* belongs to priority class $HIGHEST$, *App 3* to $HIGH$, *App 4* to LOW, and *App 5* to $LOWEST$. The configuration depicted in Fig. 6 shows a valid application placement that satisfies all the properties required by level 1.

Assuming that the vehicle accelerates to 30 km/h, the number N of levels increases to 2. As a result, the optimization goal of level 1 is to bring the system to level 2. This can be achieved by starting an active-hot instance of *App 2* on any computing node except for the one that already executes the active instance.

Next, the vehicle increases its speed to 70 km/h. Consequently, the number N of levels gets updated again to 3. The newly introduced level requires that at least three instances of all applications of priority $HIGHEST$ are executed by the system. Furthermore, level 3 requires the execution of at least two instances of all applications of priority $HIGH$. Therefore, to upgrade the system to level 3, additional active-hot instances of *App 1*, *App 2*, and *App 3* are required. The resulting configuration is displayed in Fig. 7.

This example shows that the vehicle is reconfigured once a certain speed threshold is exceeded. However, it is also conceivable to extend the dynamic-level approach by a policy that defines that the maximum level of an aimed speed has to be fulfilled before reaching that speed. Hence, such a policy ensures that a vehicle is only allowed to drive at a certain speed if the highest required safety level for that speed is fulfilled. In case the safety level drops and the highest safety level cannot be fulfilled anymore, the speed of the vehicle has to be reduced.

Consider, for instance, a vehicle driving 40 km/h and a configuration that fulfills the requirements of level 2. The described policy enforces that the vehicle has to be reconfigured to satisfy the requirements of level 3 before the vehicle's speed exceeds 50 km/h if the function depicted in Fig. 5 is assumed.

B. Dynamic-Prioritization Approach

Another approach for a dynamic adjustment of safety levels is one based on the C-PO instance of Subsection III-B where the number of levels, their definitions, and optimization goals are fixed while the priority class an application belongs to varies dynamically. This means that for each application and each context, the corresponding priority has to be defined. In case that two contexts define conflicting priorities, always the higher priority class gets applied. Therefore, it is guaranteed that the minimum requirements of all contexts are satisfied.

To clarify the dynamic-prioritization approach, consider a system that runs five applications. Furthermore, we set the number N of levels to 4 and reuse the level definition introduced in Subsection III-B. For simplicity, we again only consider speed as context in this example. Table II defines the priority classes for all applications based on the different speed intervals.

Assuming that a vehicle is moving at a speed of less than 10 km/h, according to Table II, *App 1* and *App 4* both belong to the priority class $LOWEST$, *App 5* is of priority LOW, *App 3* of priority $HIGH$, and *App 2* belongs to the priority class $HIGHEST$. Furthermore, we assume that the configuration of the vehicle fulfills exactly the minimal requirements of level 4, i.e., the system executes one application instance of *App 1* and *App 4*, two application instances of *App 5*, where the level of hardware segregation is two, etc.

Next, the vehicle accelerates to a speed of 40 km/h. Consequently, the priority of the applications change. According

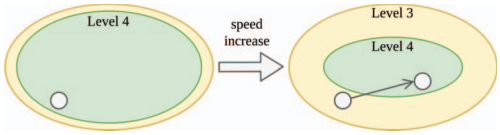

Fig. 8. Visualization of a level drop that was caused due to an speed increase. Due to the context change the level boundaries change.

to Table II, the priorities change as follows: (i) the priorities of *App 1* and *App 4* get upgraded from $LOWEST$ to LOW, (ii) the priority of *App 2* gets downgraded to priority $HIGH$, (iii) the priority of *App 3* gets upgraded to $HIGHEST$, and (iv) the priority of *App 5* gets upgraded to $HIGH$.

Since *App 1*, *App 3*, *App 4*, and *App 5* do not fulfill requirements of level 4 anymore, the level of the current configuration drops to level 3. To get to level 4 again, additional instances of *App 1*, *App 3*, *App 4*, and *App 5* are required such that the hardware segregation requirements have to be respected. Since the priority of *App 2* dropped from $HIGHEST$ to $HIGH$, one redundant instance of *App 2* can be stopped.

Note that the dynamic-prioritization approach causes that the level boundaries are flexible, i.e., a context change can cause a drop or a lift of the level of the system. Fig. 8 illustrates the decline from level 3 to level 4 and the reconfiguration that again fulfills the requirements of the maximum level.

Note that in order to not violate the multilevel-jump property defined in Subsection III-A, it must be ensured that the priority of an application does not get upgraded by more than one priority level in case of a context change. This means that, for example, a lift of *App 1* from priority $LOWEST$ to priority $HIGH$ in case that the speed exceeds the 10 km/h threshold has to be excluded.

C. Dynamic-Level vs. Dynamic-Prioritization Approach

At first glance, the two approaches discussed above seem quite similar. However, on closer examination, it turns out that each approach has different characteristics.

The dynamic-level approach is a general method that is not bounded to a specific manner of defining levels. On the other hand, the dynamic-prioritization approach is less general. As indicated by its name, priorities are a key aspect of this idea. In case that the levels are not defined using priorities, this approach might not be applicable.

However, the dynamic-prioritization approach allows a more detailed specification of how the redundancy requirements of an application vary in case of a context change. For example, with this approach, as illustrated in Subsection IV-B, it is expressible that the redundancy requirements of an application rise as the speed increases, while, on the other hand, the redundancy requirements of another application decrease as the speed increases. Due to this flexible priority classification, computing resources can be utilized more efficiently.

Although the dynamic-prioritization approach requires that the developer of an application specifies for each context the corresponding priority level, this can be a cumbersome and challenging task, especially if many different contexts are considered. The dynamic-level approach, on the other hand, requires just to define the minimum number of levels that are demanded by the different contexts, which is considered to be an easier task than the challenge described before. Therefore, such an approach might be easier to implement.

V. Conclusion

In this paper, we introduced C-PO, an approach for a context-based determination of the optimization goal that defines the most desirable solution of the application-placement problem. Furthermore, we presented a concrete instance of C-PO based on priorities and discussed two methods to adjust the required safety level dynamically. The dynamic-level approach adapts the number of levels based on the current context and the dynamic-prioritization approach adjusts the priority of the applications according to the current situation.

Concerning future work, we plan to evaluate C-PO using a simulation tool [7] which injects failures into a predefined system architecture. Furthermore, we plan to integrate C-PO into a framework for a context-based system architecture for autonomous vehicles [8]. This framework defines three layers: the *context layer*, the *reconfiguration layer*, and the *architecture layer*. The task of the context layer is to detect context changes, derive requirements from these observations, and report them to the reconfiguration layer. The reconfiguration layer then determines reconfiguration actions, which are executed by the architecture layer, that fulfill those requirements. The idea is to integrate C-PO in the context layer to derive the requirements from the the context observations that are used as input for the reconfiguration layer.

References

[1] T. Kain, H. Tompits, J.-S. Müller, M. Wesche, Y. A. Martinez Flores, and H. Decke, "Optimizing the Placement of Applications in Autonomous Vehicles," in *Proceedings of the 30th European Safety and Reliability Conference* (*ESREL 2020*), 2020.

[2] P. Koopman and M. Wagner, "Autonomous Vehicle Safety: An Interdisciplinary Challenge," *IEEE Intelligent Transportation Systems Magazine*, vol. 9, no. 1, pp. 90–96, 2017.

[3] P. Kumar, R. Merzouki, B. Conrard, and B. Ould-Bouamama, "Multilevel Reconfiguration Strategy for the System of Systems Engineering: Application to Platoon of Vehicles," in *Proceedings of the 19th International Federation of Automatic Control World Congress*, 2014, pp. 8103–8109.

[4] D. Cooray, S. Malek, R. Roshandel, and D. Kilgore, "RESISTing Reliability Degradation Through Proactive Reconfiguration," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering* (*ASE 2010*), 2010, pp. 83–92.

[5] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu, "Towards Requirements-Driven Autonomic Systems Design," *ACM SIGSOFT Software Engineering Notes*, vol. 30, no. 4, 2005.

[6] M. Hemmati, M. Amiri, and M. Zandieh, "Optimization Redundancy Allocation Problem with Nonexponential Repairable Components using Simulation Approach and Artificial Neural Network," *Quality and Reliability Engineering International*, vol. 34, 02 2018.

[7] T. F. Horeis, T. Kain, J.-S. Müller, F. Plinke, J. Heinrich, M. Wesche, and H. Decke, "A Reliability Engineering Based Approach to Model Complex and Dynamic Autonomous Systems," in *Proceedings of the 3rd International Conference on Connected and Autonomous Driving* (*MetroCAD 2020*), 2020.

[8] T. Kain, J.-S. Müller, P. Mundhenk, H. Tompits, M. Wesche, and H. Decke, "Towards a Reliable and Context-Based System Architecture for Autonomous Vehicles," in *Proceedings of the 2nd Workshop on Autonomous Systems Design* (*ASD 2020*), 2020.