Enhancing Automotive AUTOSAR Environments with Artificial DNA

Eric Hutter[1]

Abstract: In order to cope with the ever-increasing complexity of automotive embededded systems, bio-inspired techniques can be employed. We propose an organic concept based on artificial DNA (ADNA) and an artificial hormone system (AHS) that can be used to realize highly reliable, robust and flexible automotive systems.

However, computational resources and communication bandwidth are often limited in automotive environments. Additionally, the AUTOSAR Classic Platform as the de facto standard for automotive Electronic Control Units (ECUs) does not support dynamic system behavior. Nevertheless, in this paper we show that the dynamic concept of ADNA and AHS can be successfully applied to a statically configured Classic AUTOSAR environment with moderate computational resource usage. While the communication via CAN bus (Controller Area Network) imposes some limitations, we propose ways of resolving them.

Keywords: Artificial DNA; Artificial Hormone System; Self-organization; Automotive; CAN Bus; AUTOSAR

1 Introduction

Automotive embedded systems are growing more and more complex due to highly demanding application fields (e.g. autonomous driving) that require sophisticated system and software designs. In order to cope with this ever-increasing complexity, bio-inspired techniques like self-organization can be employed.

The previously developed artificial DNA (ADNA) [Br16a] adapts the biological principle of DNA to computing systems: A copy of a system's ADNA is stored in each of its processing elements and serves as the system's blueprint. This allows healthy processing elements to replicate the functions of failed elements.

The ADNA describes the system's tasks and their communication patterns while the distribution of the tasks to the available processing elements is performed decentrally by an artificial hormone system (AHS) [BPR08]. The AHS in turn is inspired by the biological hormone system and uses message-based control loops for the task distribution.

[1] Institute for Computer Science, Goethe-University, Frankfurt am Main, Germany, hutter@es.cs.uni-frankfurt.de

doi:10.18420/inf2019_ws51

The combination of ADNA and AHS makes the described system robust and flexible by realizing several self-X properties such as self-configuration, self-optimization and self-healing. This concept has already been demonstrated, e.g. in an autonomous self-balancing robot vehicle [Br; Br17].

In this publication, we investigate the applicability of the ADNA/AHS concept to automotive environments based on AUTOSAR (Automotive Open System Architecture [AUc]) platforms. Modern cars contain a multitude of processors (electronic control units, ECUs) that perform various tasks to control the car's functions that belong to various domains, e.g. powertrain, driving assistants and board infotainment. Most of these functions have to operate at a very high level of robustness and fault-tolerance. Additionally, fail-operational behavior is a hard requirement for autonomous cars as there is no human driver to fall back to. Thus, the application of system architectures like the ADNA/AHS approach to automotive environments is an interesting field of research.

Previous work has shown that ADNA and AHS can be successfully adapted to common automotive microcontrollers with limited resources by using a minimal operating system (AtomThreads) and a CAN bus for communication [BF19]. However, with AUTOSAR being the de facto standard for automotive software architectures, ADNA-based systems need to integrate with AUTOSAR-based systems in order to be accepted by the automotive industry.

Therefore, this paper deals with the ADNA concept's adaption to AUTOSAR environments and is structured as follows: Section 2 presents related work. The concepts of ADNA and AHS as well as their application to automotive environments are described in Section 3. Section 4 describes how these concepts can be adapted to the AUTOSAR Classic Platform while section 5 shows evaluation results of this adaption. Section 6 concludes this paper.

2 Related Work

Our approach relies on the realization of several self-X properties like self-organization and self-healing in automotive application fields.

In [HKW14], a redundancy scheme for processors in automotive environments is proposed that uses a voting algorithm to determine the validity of the redundant processors' results. This is different than our approach, which better exploits available redundancy using the ADNA: Our concept does not require identical redundant processors in order to enable fail-operational behavior, but can rather migrate failed processors' tasks to other processors.

AUTOSAR defines two automotive system architectures. The AUTOSAR Classic Platform [AUb] describes an embedded automotive system as a composition of individual components that communicate through well-defined ports interconnected by a Virtual Function Bus (VFB). During system generation, each component is statically mapped to a specific ECU. Additionally, a Runtime Environment (RTE) is generated for each of these ECUs as

a specialized implementation of the VFB. While the composition of a system from multiple parts is also a fundamental concept of our approach, the AUTOSAR Classic Platform requires the operating and communication systems to be statically configured and doesn't offer support for dynamic system (re)configuration at run time.

AUTOSAR's second system architecture, the AUTOSAR Adaptive Platform [AUa], is meant to complement the Classic Platform by dealing with scenarios requiring higher degrees of dynamics at run time, e.g. applications for autonomous driving which are out of the Classic Platform's scope. However, even the Adaptive Platform does not offer any support for improved fault-tolerance by dynamic reconfiguration as our approach does.

In [Tr07], the authors propose an organic computing middleware on top of the hardware abstractions specified by the AUTOSAR Classic Platform in order to provide self-configuration and self-healing capabilities in automotive environments. However, they only present behavioral simulations of their architecture and no real implementation is shown. While our concept is also based on middleware to realize self-X properties in automotive applications, we show ways to implement this middleware on top of the AUTOSAR platform and conduct tests on real hardware that include communication via CAN bus.

AutoKonf [Au; Os18] is an ongoing research project that strives to implement fail-safe actor controls for autonomous driving. This is achieved by generic redundant ECUs in hot-stand-by modes that can each replicate one failed specialized ECU. This is different from our approach in multiple ways: The ADNA concept enables a much more fine-grained system composition based on individual tasks in contrast to AutoKonf's composition on the ECU level. Therefore, our approach allows better exploitation of available redundancy as no hot-stand-by ECUs are needed. Instead, a failed ECU's tasks may be migrated individually to the remaining ECUs. However, the AutoKonf project also develops hardware to support switching the actors' controls between different ECUs, while our approach is currently focused on the software side of (re)configuration only.

3 ADNA and Automotive Applications

This section briefly describes the concept of artificial DNA and artificial hormone system (AHS). For detailed information, see [BPR08; Br16a; Br16b].

3.1 Artificial DNA

The idea of artificial DNA is based on the observation that many embedded systems can be composed from a limited number of basic elements (such as controllers, filters, arithmetic logic units (ALUs) etc). Thus, a library consisting of a sufficient amount of such basic elements allows to build embedded real-time systems by simply parameterizing and combining these elements.

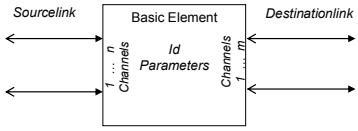

Fig. 1: ADNA basic element structure

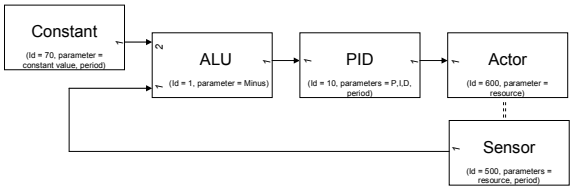

Fig. 2: ADNA realizing a closed control loop

Figure 1 shows the general structure of a basic element. It has two possible types of links to communicate with other elements: The *Sourcelink* is a reactive link that is used to react to incoming requests while the *Destinationlink* is an active link that can be used to initiate such requests. Both links may have multiple channels with different semantics.

Figure 2 shows a simple closed control loop realized as a composition of different basic elements as building blocks: An actor is controlled by a sensor with a constant setpoint value applied. It shall be noted that each basic element describes a *software* component implementing the element's functionality (e.g. reading a sensor's value or performing a calculation), the ADNA does not describe the system's hardware topology.

Given this simple example, it becomes obvious that ADNA-based systems are not programmed. Instead, they can fully be described by an appropriate representation of their composition. Thus, this representation is referred to as the system's ADNA. Further examples as well as details on the specifics of this representation can be found in [Br16a] and [Br17].

3.2 Building the System from its ADNA

A system typically consists of different functions. Using ADNA, each of these functions is further divided into various basic elements. Each of those basic element represents a task. Every processing element in the system stores a local copy of the system's ADNA and thus knows all tasks as well as their interconnections. It passes this information to its local AHS instance to assign the tasks.

The AHS is a completely decentralized organic middleware that assigns tasks to distributed computing nodes (see [BPR08]). It uses messages that emulate hormones to assign each task to the most suitable computing node based on the node's capabilities, current load and the tasks' interconnections. Additionally, it can detect node and task failures by missing hormone messages and reassign the corresponding tasks to different nodes, providing self-healing features as long as enough computing power is left in the system.

This approach composes a system at run-time by creating a task for each basic element of the system's ADNA and distributing all tasks to the available computing resources in the best possible way. In case of failures, measures are taken to reassign the affected tasks.

Assignment and re-assignment are both done in real-time with a time complexity of $\mathcal{O}(n)$ where n is the number of tasks (see [BPR08]). Thus, the ADNA approach enables a fine-grained and robust distribution of functions to processors: In contrast to other approaches like AutoKonf, functions are not bound to specific processors but are further divided into tasks that may run on different processors, thus leading to a very fine-grained task distribution.

3.3 Automotive Applications of the ADNA

In automotive applications, the car's various functions are executed by its ECUs. Especially in autonomous driving scenarios, there are many functions for which a failure cannot be tolerated. Thus, a highly robust system design is necessary.

Classical system architectures have used a redundancy concept where each critical ECU is duplicated. More recent approaches like the aforementioned AutoKonf share a single backup ECU in hot-stand-by between multiple different ECUs to reduce the overhead incurred.

In contrast, the ADNA's application to automotive environments further reduces this overhead by means of a much more fine-grained system composition. Here, the car's functions are no longer directly mapped to ECUs. Instead, their individual tasks are dynamically mapped to the available ECUs. This allows a much more flexible task distribution as well as fail-operational behavior without backup ECUs – provided the system's total computational resources suffice to take over the failed ECUs' tasks – or at least graceful degradation.

Of course, this dynamic system behavior incurs a higher overhead compared to the static mappings of functions to ECUs. Thus, the question arises whether typical automotive ECUs are powerful enough in terms of computational, memory, bandwidth and operating system resources to realize this concept. This will be investigated in the next sections.

4 Adaption of the ADNA to the AUTOSAR Classic Platform

This section discusses the steps taken to adapt the ADNA and AHS to the AUTOSAR Classic Platform.[2] Our development target was an implementation conforming to the AUTOSAR Classic Platform specification in version 4.0.3 running on an evaluation board utilizing a Renesas R7F701310EAFP microcontroller which is a common controller for safety-critical ECUs. This controller contains a RH850/P1M 32 bit processor core clocked at 160 MHz, 1 MB ROM and 128 kB RAM [Re].

[2] In the following two sections, the terms "AUTOSAR Classic Platform" and "AUTOSAR" are used interchangeably for the sake of brevity.

The ADNA/AHS system is completely written in ANSI C 90 and could therefore easily be compiled for the target platform using the Green Hills C compiler for this microcontroller family. Mainly, two modules had to be adapted:

- *AHSBasicOSSupport*, which implements the basic multithreading and synchronization mechanisms for AHS and ADNA. The AUTOSAR Classic Platform's operating system component is based on OSEK OS [IS05] and employs a static task mapping specified at compile time. This conflicts with the ADNA's dynamic task concept and thus requires elaborate adaption.
- *AHSBasicCommunication*, which implements all basic communication functions for AHS and ADNA. Since the AUTOSAR Classic Platform requires all communication patterns to be statically defined and forbids arbitrary bus accesses, additional measures had to be taken to adapt this module.

In the following sections, we therefore describe the necessary adaptions in detail.

4.1 Basic Operating System Support

This module implements the ADNA/AHS system's thread model as well as synchronization primitives like mutexes, semaphores and events on top of the underlying operating system.

Instead of building our implementation as a software component on top of the AUTOSAR Runtime Environment (RTE), we decided to directly base it on AUTOSAR's operating system as this approach offered more flexibility by skipping the RTE layer.

Since AUTOSAR's operating system is statically configured, implementing this operating support module was not as straightforward compared to other AHS implementations on e.g. Windows and Linux: On these, the operating system support module can nearly directly map the functions required by ADNA/AHS to the operating system's equivalents. Therefore, we will briefly describe the challenges that had to be overcome in the following.

Firstly, ADNA and AHS need support for dynamic memory management which isn't provided by AUTOSAR. Therefore, a heap was created manually for use with the memory management routines provided by Green Hills' C standard library.

Secondly, the ADNA/AHS system requires functions to dynamically create, manage and stop threads in order to execute the tasks instantiated from the system's ADNA. Mapping this dynamic thread model to AUTOSAR's statically configured operating system was achieved by implementing a thread pool consisting of n tasks. This allows the dynamic creation of up to n threads as long as their stack requirements are satisfied by the statically configured stack sizes of the tasks they shall be mapped to.

Thirdly, threads in ADNA and AHS are synchronized by mutexes, semaphores and events that can also be created dynamically. Contrarily, AUTOSAR's operating system synchronizes tasks using statically configured events and achieves mutual exclusions by acquiring resources and an associated priority ceiling protocol. Since these concepts could not directly be adapted to match the primitives required by ADNA/AHS, those primitives were instead re-implemented on top of AUTOSAR, using a single AUTOSAR event to suspend and resume waiting threads.

Lastly, support for ADNA/AHS timers and waiting for events with a timeout was implemented by utilizing AUTOSAR alarms.

4.2 Basic Communication with CAN Bus

The ADNA/AHS system sends messages and hormones via the AHSBasicCommunication module. Hormones are bundled into telegrams of up to 256 Bytes length. The maximum length of message telegrams is also 256 Bytes. Therefore, the AHSBasicCommunication module has to offer functionality to send and receive telegrams up to that size.

Since the classical CAN bus only supports up to 8 bytes of payload per frame, a transport protocol is required to segment longer telegrams into multiple frames. AUTOSAR's COM module supports the ISO 15765-2 protocol [IS16] (also known as "ISO-TP") for this purpose. However, since this protocol is primarily intended for diagnostic communication via unicasts, it employs flow control for messages split into multiple frames. Therefore, it can't be used for broadcasts as required for the AHS' hormone communication.

As a consequence, our implementation uses a simplified variant of ISO-TP without flow control for transporting telegrams. Figure 3 shows the frame types used in our implementation. The first byte's upper half-byte specifies a frame's type:

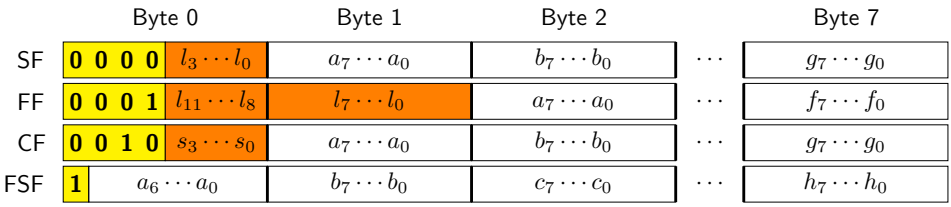

Fig. 3: Frame types used in our transport protocol

- Single Frames (SF) are used to transmit telegrams with lengths of $0 \leq l \leq 7$ bytes.
- Longer telegrams are transported by first sending a First Frame (FF) containing the telegram's total length l and the first 6 bytes of payload. The remaining bytes are sent in chunks of 7 bytes using Consecutive Frames (CF) which also include the

chunk's sequence number s modulo 16, thus allowing to detect a limited number of lost frames.

- Full Single Frames (FSF) are not specified by ISO-TP but rather specific to our protocol. In case a telegram consists of exactly 8 bytes, it can be transmitted using a FSF instead of requiring a FF and a subsequent CF provided that its first byte's MSB is 0. This condition is more often than not satisfied in the ADNA/AHS implementation, so this frame type reduces the protocol's overhead for many telegrams of exactly 8 bytes length.

According to AUTOSAR, segmentation of telegrams should not be done in the application layer as this would introduce bus-specific properties into bus-independent software components. However, for our initial implementation, we decided to segment telegrams in the application layer nonetheless due to two reasons: On the one hand, implementing a new transport protocol within AUTOSAR would have required appropriate support in the configuration tools in order to properly configure the different signals' routing within AUTOSAR's communication stack. On the other hand, telegram segmentation in the application layer allowed to share a single implementation with other platforms running the ADNA/AHS system on CAN bus and thus providing interoperability between them as required for our evaluation.

Therefore, we implemented telegram segmentation on top of AUTOSAR's COM module. Since communication patterns are also statically configured in AUTOSAR, we defined one CAN signal per ECU for a total of 15 potential ECUs.

The AUTOSAR version we used did not yet support signals of dynamic length to be transmitted, so each signal was configured to exactly 8 bytes payload, thus increasing the protocol overhead slightly for frames that are not completely filled (which can happen for SFs and the last CF of each telegram).

AUTOSAR's communication model is primarily geared to cyclic signal transmission. Unfortunately, this doesn't align nicely with our intended communication pattern, so we configured the COM module's transmission function to be invoked once per millisecond, thus allowing us to send up to 1000 CAN frames per second and AUTOSAR-based ECU.

4.3 Additional Measures

Additionally, we implemented hooks that are called upon entry and exit of AUTOSAR's idle loop to measure the time spent in it using a timer with a sufficiently high precision, thus allowing to calculate the CPU's utilization.

5 Evaluation of the ADNA on the AUTOSAR Classic Platform

Our evaluation setup consisted of four configurations with one to four ECUs each:

- The AUTOSAR-based ADNA/AHS implementation running on the evaluation board described in section 4. The thread pool was configured to 15 tasks, making it possible to run up to 12 basic elements on the AUTOSAR ECU alone (the ADNA/AHS uses 3 threads for internal management and communication). Since the evaluation board was equipped with 128 kB RAM, 2 kB of stack were assigned to each task without any further optimization.
- Zero to three additional virtual ECUs simulated by ADNA/AHS instances running on a Linux PC.

All ECUs were interconnected with each other via CAN bus. Sensors and actors were provided by a simulator that was communicating via UDP while the UDP/COM interconnection was provided by a gateway that also collected statistics about parameters like the CAN bus load (see figure 4).

The CAN bus' bit rate was set to 500 kbit/s as this is the typical rate that is currently used by the automotive industry.

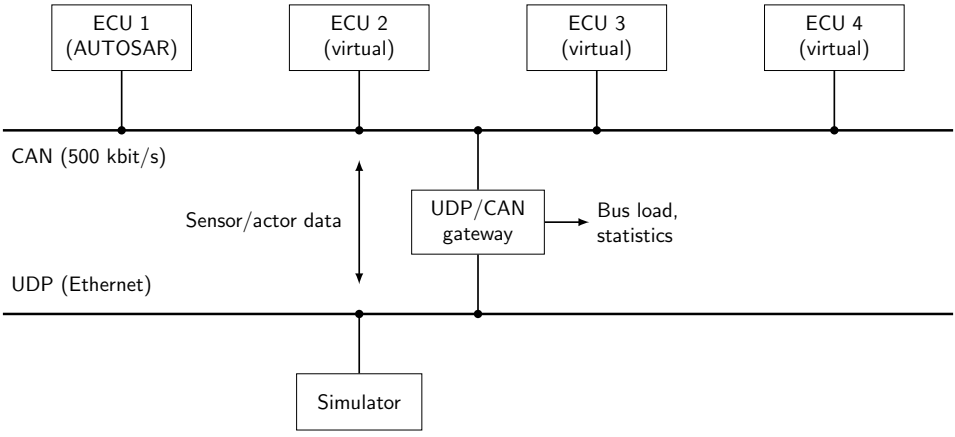

Fig. 4: Overview of evaluation setup

We used several experimental automotive ADNAs realizing various features. The ADNA nomenclature is as follows:

Slow ADNA variant with fastest message cycle time of 60 ms

Med ADNA variant with fastest message cycle time of 30 ms

Fast ADNA variant with fastest message cycle time of 15 ms

Abs ADNA realizes an anti-lock braking system

Tcs ADNA realizes a traction control system

St Maximum steering angle is controlled by velocity

Cr ADNA has a cruise control

Cmp ADNA consists of fewer but less generic basic elements (“compound elements”)

Table 1 shows the evaluation results with the CPU load of the AUTOSAR-based ECU, CAN bus load and a rating of the system’s observed behavior as given by the following scheme:

+ The system was working as intended

○ The system was working as intended, but the CAN bus load exceeded a critical bound of 90%

− The system did not work at all or exhibited incorrect behavior

Empty cells denote that the ADNA consists of too many basic elements to be fully instantiated in the given configuration.

Tab. 1: Evaluation results

ADNA		**1 ECU**			**2 ECUs**			**3 ECUs**			**4 ECUs**		
Name	# Tasks	CPU	CAN	Rtg	CPU	CAN	Rtg	CPU	CAN	Rtg	CPU	CAN	Rtg
SlowCr	18				10%	43%	+	10%	43%	+	10%	50%	+
FastCr	18				16%	70%	+	13%	76%	+	15%	89%	+
SlowAbsCr	26				13%	60%	+	13%	72%	−	13%	70%	+
MidAbsCr	26				17%	84%	+	15%	94%	−	17%	100%	−
FastAbsCr	26				18%	100%	−	15%	100%	−	16%	99%	−
FastAbsTcsStCr	35							18%	99%	−	20%	100%	−
SlowAbsTcsCmp	9	9%	34%	−	9%	45%	+	8%	44%	+	9%	44%	+
MidAbsTcsCmp	9	9%	35%	−	11%	65%	−	11%	65%	+	11%	65%	+
FastAbsTcsCmp	9	9%	35%	−	13%	89%	−	13%	90%	−	13%	95%	−
SlowAbsTcsStCrCmp	11	10%	34%	−	10%	50%	+	10%	55%	+	10%	54%	+
MidAbsTcsStCrCmp	11	10%	34%	−	11%	68%	−	12%	77%	+	12%	74%	+
FastAbsTcsStCrCmp	11	9%	34%	−	13%	92%	−	15%	98%	○	14%	97%	−

It can be observed that the AUTOSAR ECU’s CPU utilization is rather low and poses no issue at all. However, most configurations are rated with ‘−’ even though the CAN bus load does not reach a critical limit. This is due to the fact that the AUTOSAR ECU’s throughput is limited to 1000 sent CAN frames per second (which corresponds to approx. 22% bus load) as explained in section 4. Depending on the actual task distribution achieved by the AHS, the inter-ECU communication load may require the AUTOSAR ECU to send more

than 1000 frames per second and its failure to do so results in instable or incorrect system behavior. The communication load's dependence on the resulting task distribution can for example be seen for the "SlowAbsCr" ADNA which is not working as intended when exactly three ECUs are present.

5.1 Evaluation with Optimized Communication Implementation

The aforementioned throughput limitation imposed by our communication module implementation made nearly all real-world applications of the ADNA concept on AUTOSAR impossible. Additionally, no configuration consisting of only a single AUTOSAR ECU was sufficient to completely realize any system.

Therefore, we decided to adapt our implementation by deviating from the AUTOSAR standard even further in order to overcome this limitation: Instead of sending CAN frames by routing signals through AUTOSAR's communication stack, we invoked its CAN driver directly. However, such bus write requests may fail (if e.g. no hardware transmit buffer is available). Since we did not want to maintain our own queue (that would have interfered with AUTOSAR's internal message queues), we used our first implementation as a fallback method in these cases. Based on our experiments, this way of sending CAN frames improved the maximum throughput to about 2400 frames sent per second (corresponding to approx. 53% bus load and thus an improvement by a factor of 2.4). While not being standards-compliant at all, this alternative implementation provided an easy way of improving the throughput for a second evaluation run. The results of this run are shown in table 2.

Tab. 2: Evaluation results of second run

ADNA		**1 ECU**			**2 ECUs**			**3 ECUs**			**4 ECUs**		
Name	# Tasks	CPU	CAN	Rtg	CPU	CAN	Rtg	CPU	CAN	Rtg	CPU	CAN	Rtg
SlowCr	18				11%	46%	+	10%	43%	+	11%	49%	+
FastCr	18				18%	76%	+	15%	77%	+	17%	89%	+
SlowAbsCr	26				15%	60%	+	14%	65%	+	14%	66%	+
MidAbsCr	26				19%	88%	+	18%	88%	+	18%	97%	○
FastAbsCr	26				19%	93%	○	17%	100%	−	18%	98%	−
FastAbsTcsStCr	35							19%	100%	−	21%	100%	−
SlowAbsTcsCmp	9	10%	46,7	+	9%	43%	+	9%	46%	+	9%	49%	+
MidAbsTcsCmp	9	14%	62,0	+	12%	66%	+	12%	66%	+	12%	66%	+
FastAbsTcsCmp	9	18%	80,4	−	15%	95%	○	15%	96%	○	15%	97%	○
SlowAbsTcsStCrCmp	11	12%	45,8	+	11%	57%	+	11%	53%	+	11%	59%	+
MidAbsTcsStCrCmp	11	16%	65,3	+	14%	73%	+	14%	78%	+	14%	77%	+
FastAbsTcsStCrCmp	11	19%	80,2	−	15%	96%	○	16%	100%	−	15%	97%	○

As can be seen, the AUTOSAR ECU is now able to run several "Cmp" ADNAs by itself. Many configurations work fine, only most of the "Fast" variants as well as one "Med" configuration still have either higher send throughput demands than can be met using our new communication module implementation or higher total bandwidth demands than can be satisfied by the CAN bus.

Generally, most ratings improved from ‘−’ to ‘∘’ or even ‘+’. Only the “FastAbsTcsStCrCmp” ADNA’s results for 3 ECUs (that already only just worked before) worsened from ‘∘’ to ‘−’.

Overall, the CAN bus load is still quite high. The CPU utilization remains rather low, albeit being tendentially higher than in the previous evaluation run.

These results show that it is possible to dynamically map tasks to AUTOSAR ECUs despite its static operating system model. Inter-ECU communication however poses more problems, at least when utilizing a classical CAN bus and segmenting telegrams to CAN frames in the application layer rather than directly inside AUTOSAR’s communication stack. Thus, with regard to further research concerning the ADNA/AHS concept in automotive applications, it is inevitable to integrate the transport protocol directly into AUTOSAR’s core modules in order to maximize the achievable throughput.

Additionally, real-world applications require a sufficient safety margin with regards to the bus load. As our results show, the classical CAN bus with a bit rate of 500 kbit/s cannot satisfy this requirement. Thus, the use of CAN FD or at least classical CAN with its maximum bit rate of 1 Mbit/s seems to be crucial for our concept’s applicability in the automotive domain.

6 Conclusion

In this paper we have shown that it is possible to apply the self-organizing ADNA/AHS concept to the AUTOSAR Classic Platform despite its static system configuration. Due to its self-healing capabilities, this approach can improve the fail-operational behavior and flexibility of automotive systems.

Our basic operating system support module uses a statically configured task pool to map the ADNA/AHS’s dynamic thread model and synchronization primitives to the Classic Platform’s static operating system. This works remarkably well and results in rather low CPU load. However, since all tasks’ stack sizes also have to be configured statically, the number of threads that can be started by the ADNA/AHS is limited by this configuration and not the threads’ actual stack requirements at runtime, thus possibly limiting the system’s dynamic behavior.

Our communication module is built on top of the Classic Platform’s communication stack and limits the maximum achievable throughput, thus preventing most system configurations from working correctly. Therefore, we resorted to sending CAN frames directly through AUTOSAR’s CAN driver if possible. This improves the throughput sufficiently and allows most of the tested ADNAs to work.

With regard to real-world applications of our concept, our custom transport protocol will need to be integrated into AUTOSAR's core modules for full compliance to the AUTOSAR standard as well as improved reliability.

Further research will need to be conducted as we have shown that the classical CAN bus can quickly become a bottleneck, even for rather small ADNAs. Therefore, more powerful buses like CAN FD have to be evaluated.

The self-X properties realized by our approach already add completely new qualities to the AUTOSAR Classic Platform. Since the AUTOSAR Adaptive Platform is designed to complement the Classic Platform rather than replace it, implementing a single ADNA/AHS network spanning across Classic and Adaptive AUTOSAR ECUs could facilitate even more interesting use cases. Thus, we plan to apply our concept to the AUTOSAR Adaptive Platform as well.

Besides these adaptions to automotive environments, we plan to conduct more research on the ADNA/AHS approach itself. A priority-based task assignment strategy could be used to implement a system's most important features in case too many failures occurred to realize the system completely. Alternative paths of execution could be used to implement features with either specialized and more efficient basic elements on suitable processor cores or generic but less efficient basic elements on other cores in case those cores fail.

References

[AUa] AUTOSAR: Adaptive Platform, URL: https://www.autosar.org/standards/adaptive-platform/, visited on: 04/12/2019.

[AUb] AUTOSAR: Classic Platform, URL: https://www.autosar.org/standards/classic-platform/, visited on: 04/12/2019.

[AUc] AUTOSAR: Home Page, URL: https://www.autosar.org/, visited on: 04/12/2019.

[Au] AutoKonf: Homepage, URL: http://www.autokonf.de/, visited on: 04/12/2019.

[BF19] Brinkschulte, U.; Fastnacht, F.: Applying the Concept of Artificial DNA and Hormone System to a Low-Performance Automotive Environment. In (Schoeberl, M.; Hochberger, C.; Uhrig, S.; Brehm, J.; Pionteck, T., eds.): Architecture of Computing Systems – ARCS 2019. Springer International Publishing, Cham, pp. 87–99, 2019.

[BPR08] Brinkschulte, U.; Pacher, M.; von Renteln, A.: An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware. In: Organic Computing. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 261–283, 2008.

[Br] Brinkschulte, U.: Video of the ADNA Controlled Robot Vehicle, URL: http://www.es.cs.uni-frankfurt.de/index.php?id=252, visited on: 06/17/2019.

[Br16a] Brinkschulte, U.: An artificial DNA for self-descripting and self-building embedded real-time systems. Concurrency and Computation: Practice and Experience 28/14, pp. 3711–3729, 2016.

[Br16b] Brinkschulte, U.: Prototypic Implementation and Evaluation of an Artificial DNA for Self-Descripting and Self-Building Embedded Systems. In: 2016 IEEE 19th International Symposium on Real-Time Distributed Computing (ISORC). Pp. 10–18, May 2016.

[Br17] Brinkschulte, U.: Prototypic implementation and evaluation of an artificial DNA for self-descripting and self-building embedded systems. Springer EURASIP Journal on Embedded Systems/, Feb. 2017.

[HKW14] Hong Yi, C.; Kwon, K.; Wook Jeon, J.: Method of improved hardware redundancy for automotive system. In: 2014 14th International Symposium on Communications and Information Technologies (ISCIT). Pp. 204–207, Sept. 2014.

[IS05] ISO: Road vehicles – Open interface for embedded automotive applications – Part 3: OSEK/VDX Operating System (OS), Standard ISO 17356-3:2005, Geneva, CH: International Organization for Standardization, Nov. 2005.

[IS16] ISO: Road vehicles – Diagnostic communication over Controller Area Network (DoCAN) – Part 2: Transport protocol and network layer services, Standard ISO 15765-2:2016, Geneva, CH: International Organization for Standardization, Apr. 2016.

[Os18] Oszwald, F.; Becker, J.; Obergfell, P.; Traub, M.: Dynamic Reconfiguration for Real-Time Automotive Embedded Systems in Fail-Operational Context. In: 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW). Pp. 206–209, May 2018.

[Re] Renesas: RH850/P1M, URL: https://www.renesas.com/eu/en/products/microcontrollers-microprocessors/rh850/rh850p1x/rh850p1m.html, visited on: 01/09/2019.

[Tr07] Trumler, W.; Helbig, M.; Pietzowski, A.; Satzger, B.; Ungerer, T.: Self-Configuration and Self-Healing in AUTOSAR. In: SAE Technical Paper. SAE International, Aug. 2007.