Adaptable Demonstrator Platform for the Simulation of Distributed Agent-Based Automotive Systems

Philipp Weiss
Technische Universität München, Germany
philipp.weiss@tum.de

Sebastian Nagel
Technische Universität München, Germany
s.nagel@tum.de

Andreas Weichslgartner
AUDI AG, Ingolstadt, Germany
andreas.weichslgartner@audi.de

Sebastian Steinhorst
Technische Universität München, Germany
sebastian.steinhorst@tum.de

Abstract

Future autonomous vehicles will no longer have a driver as a fallback solution in case of critical failure scenarios. However, it is costly to add hardware redundancy to achieve a fail-operational behaviour. Here, graceful degradation can be used by repurposing the allocated resources of non-critical applications for safety-critical applications. The degradation problem can be solved as a part of an application mapping problem. As future automotive software will be highly customizable to meet customers' demands, the mapping problem has to be solved for each individual configuration and the architecture has to be adaptable to frequent software changes. Thus, the mapping problem has to be solved at run-time as part of the software platform. In this paper we present an adaptable demonstrator platform consisting of a distributed simulation environment to evaluate such approaches. The platform can be easily configured to evaluate different hardware architectures. We discuss the advantages and limitations of this platform and present an exemplary demonstrator configuration running an agent-based graceful degradation approach.

2012 ACM Subject Classification Computer systems organization → Distributed architectures; Computer systems organization → Redundancy; Computing methodologies → Self-organization

Keywords and phrases fail-operational, graceful degradation, agent-based mapping

Digital Object Identifier 10.4230/OASIcs.ASD.2020.3

Funding With the support of the Technical University of Munich – Institute for Advanced Study, funded by the German Excellence Initiative and the European Union Seventh Framework Programme under grant agreement n° 291763.

1 Introduction

With the introduction of new automotive functionality such as autonomous driving, automotive companies see themselves confronted with increasing customer needs and the demand to frequently deliver new functionalities. To cope with complexity, electronic architectures are undergoing major changes. Instead of adding a new electronic control unit (ECU) for each new functionality, software is being integrated on more powerful ECUs. We expect this consolidation trend to continue such that future electronic architectures will consist of only a few powerful ECUs, similar to consumer electronic devices [4].

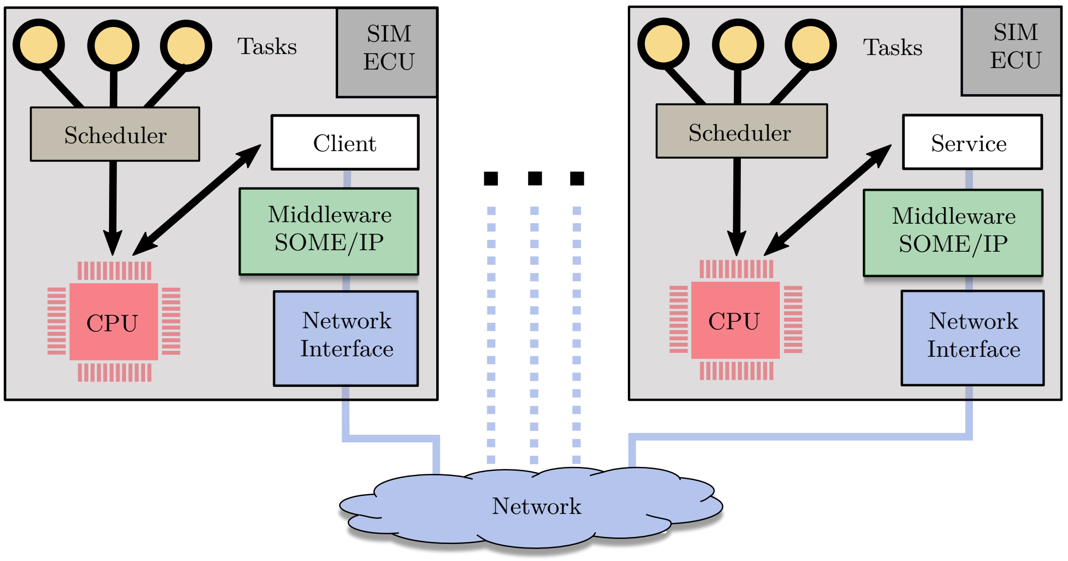

Figure 1 General structure of the simulator platform. The scheduler executes multiple tasks on the CPU. The tasks use the SOME/IP Middleware and the given network infrastructure to communicate. The platform simulates the ECUs instances and uses a physical network for communication.

As software is being decoupled from the hardware, future automotive software will be designed modular. This separation leads to new system-wide optimization possibilities. At the same time, new software functionality comes with higher resource demands and safety requirements. As there will be no driver as a fallback solution in autonomously driving cars, these systems have to be designed to maintain operation in the presence of critical ECU or software failures. However, such a fail-operational design requires redundancy which imposes an even higher resource demand and therefore cost. An approach to lower cost is graceful degradation where hardware resources which were formerly used by non-critical applications are repurposed for the use of critical applications.

With highly customizable software and unique customer configurations, each software system has to be optimized individually. Furthermore, users will change the configuration frequently and the software requirements might vary with each software update. Thus, design-time solutions to solve the mapping problem and optimize the system are insufficient as they would have to be re-evaluated for each change in the system. Therefore, new solutions are required that solve the mapping problem at run-time as part of the software platform. We presented such an agent-based approach in [5], which ensures fail-operational requirements at run-time using graceful degradation and is able to reconfigure the system after an ECU failure.

To be able to evaluate such emerging approaches, we present an adaptable demonstrator platform in Section 2 that is built on top of a distributed simulation environment. Users of the distributed simulation environment can configure computing components as distinct ECU instances to simulate task execution. For communication a physical network infrastructure can be used. This approach combines the advantages of an easily configurable simulation with a more experimental evaluation. The presented demonstrator platform is easily adaptable to different hardware configurations and provides a fast and accurate way to evaluate agent-based approaches for distributed systems. We present an exemplary demonstrator setup which runs our agent-based graceful degradation approach from [5] in Section 3 and discuss the scalability and limitations of the demonstrator platform in Section 4.

2 Demonstrator Platform

The general structure of our demonstrator platform is presented in Figure 1. We adopted a process-based Discrete-Event Simulation (DES) architecture and selected the SimPy framework [2] to simulate the ECU instances. Any ECU instance includes a task-scheduler to execute multiple tasks and a network interface to connect the ECUs with the deployed middleware. Using the middleware, the ECUs can subscribe to messages from other ECUs or publish messages themselves over the physical network. The system can be specified by the user according to our system model and uses the XML schema for specifications from the OpenDSE framework [3]. In the following sections, we describe the components of the demonstrator platform more precisely.

2.1 Simulation

On a simplified view, SimPy is an asynchronous event dispatcher [2]. SimPy places all events in a heap data structure. The events are then retrieved and dispatched in an event loop one after the other at discrete points in time. In SimPy's default mode the simulations advance as fast as possible, depending on how much computation power is available. As our demonstrator platform uses physical networks, a real-time simulation is required for realistic results. SimPy's real-time mode enforces the time steps of the simulation to be synchronized with the clock of the operating system: Each step of the simulation amounts to a certain amount of real time.

2.2 Task execution

Our software is modelled by independent non-critical and safety-critical applications. Each of the applications might consist of multiple tasks. The communication between the tasks is modelled with messages. Tasks in the system are triggered by incoming messages or periodically by a timer in case they are the anchor task of an application. The access to the CPU is granted by an exchangeable scheduler. The execution time on the CPU is then simulated by passing time according to the resource consumption of the task. Once a task has finished execution, it sends out messages via the network interface.

2.3 Middleware

Our framework uses a middleware, which handles the communication and which is based on SOME/IP [1], an automotive middleware solution. Such a middleware is necessary to enable the dynamic behaviour for moving tasks at run-time on the system. Tasks and agents exclusively communicate via this middleware and are modelled as services and/or clients. The middleware includes a decentralized service-discovery to dynamically find offered services at run-time. Furthermore, a publish/subscribe scheme is used to configure which data is sent between services and clients. Tasks that have outgoing edges in the application graph G_a are offered as a service whose events can be subscribed by the clients with incoming edges. In addition, remote procedure calls can be used for non event-based communication.

2.4 Communication

The SOME/IP protocol specification recommends UDP as a transport layer protocol for cyclic data and for hard latency requirements in case of errors [1]. Therefore to receive messages from the underlying network, we use a multi-threaded UDP Server. The UDP

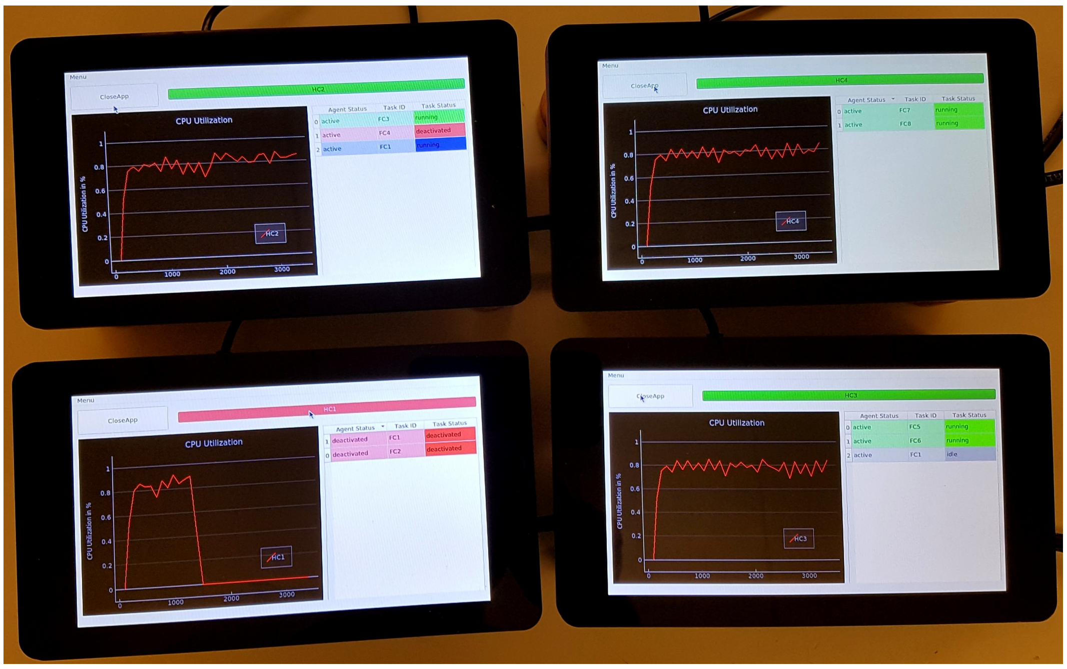

Figure 2 Our demonstrator setup consisting of 4 Raspberry Pis, which are connected over a switch and Ethernet links. Each of the hardware nodes is running a single simulation instance with our agent-based graceful-degradation approach from [5]. The graphical user interface presents the simulated CPU utilization and the status of the agents and tasks.

server does not handle the incoming messages synchronously but instead dispatches a handler thread whenever it receives a new message. The handler thread post-processes the message and schedules the message in the SimPy simulation of the ECU. The correct scheduling of the incoming message is essential to avoid time deviations in the communication between the ECUs. To transmit data on the link-level, the sending entity serializes the data upfront in the network interface, and the receiving entity deserializes the data in the receive handler.

3 Case Study

We implemented our platform with the 4 simulation instances $HC1 - HC4$, each running on a dedicated Raspberry Pi 4, which are depicted in Figure 2. The Raspberry Pis are connected via an Ethernet link to a central switch. The graphical user interface presents the simulated CPU utilization and the status of agents and tasks on the corresponding simulation instance.

The presented example is running our agent-based graceful degradation approach from [5]. The system, which consists of a safety-critical task $FC1$ and seven non-critical tasks $FC2 - FC8$, depicts the situation after the failure of simulation instance $HC1$ and the first degradation. The safety-critical agent is marked with blue, while its passive task agent is marked with grey.

In the first phase of our approach, each task gets assigned to an agent that allocates required CPU and link resources for it, migrates to the corresponding ECU and starts the task. This ramp-up can be observed with the CPU utilization at the beginning of the simulation.

To ensure fail-operational behaviour, our approach uses passive redundancy combined with graceful degradation. In a critical ECU failure scenario, the passive redundant tasks of safety-critical applications can be reactivated. Using graceful degradation, safety-critical

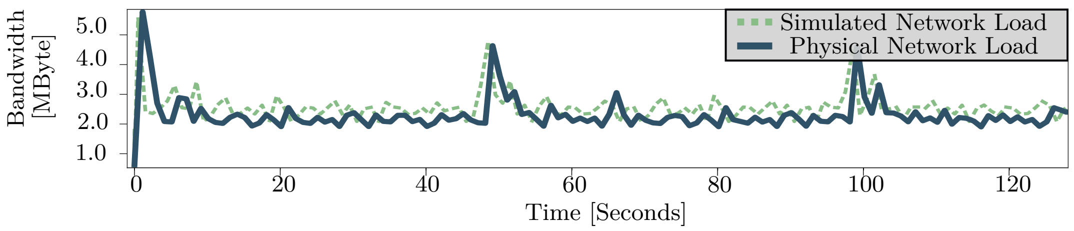

Figure 3 Measurements of the physical and simulated network traffic to and from a simulated ECU. The simulated link traffic and the physical link traffic show almost identical characteristics.

tasks can then take over the resources of non-critical tasks, which have to be shut down. As it has to be ensured that sufficient resources are available prior to an ECU failure, the agents of the passive tasks can reserve resources at the ECU and the agents of non-critical tasks. The reservation process ensures that it can be predicted if sufficient resources are available prior to a failure scenario. In our example the agent of $FC1$ has cloned itself at the beginning of the simulation and the corresponding passive task agent then reserved the resources at the agent of task $FC4$. After the failure of $HC1$, the passive task agent on $HC2$ detected the failure by a heartbeat timeout, claimed its resources at the agent of task $FC4$ and reactivated $FC1$. The degradation can be observed as $FC4$ is marked with red and the task status 'deactivated'.

After an ECU failure and the immediate failure reaction, the fail-operational behaviour of the safety-critical tasks can be re-established. The advantage of the agent-based approach is that no additional algorithm is required for the reconfiguration and the same mapping procedure can be repeated. In our example, after the immediate failure reaction on $HC2$, the task agent of $FC1$ cloned itself again to re-establish its fail-operational behaviour. The corresponding passive task agent is marked with grey on $HC4$. The agent status 'active' indicates that it has successfully reserved its resources and the system is able to endure another ECU failure without loosing its safety-critical functionality.

4 Discussion

We have compared simulated network traffic from our framework described in [5] with the physical network traffic of the demonstrator platform (Figure 3). The results validate that the previously simulated network is realistic and comes close to what we observe when using real network infrastructure. However, the simulated and observed behaviour is not entirely the same. For our future and current studies, the more realistic approach ensures that we do not miss any critical network behaviour, which we did not capture in our simulation.

The demonstrator platform is independent of the hardware, the operating system (OS) and the network architecture that the internet protocol suite (IP) uses on the link layer. It is extensible to any common computing platform. Also, the platform can simulate multiple ECU instances on a single hardware node in the network. The platform is, therefore, very flexible and scalable to assist in the rapid exploration and evaluation of distributed systems on various hardware setups and network topologies.

However, the scalability of the simulation is limited by the available computational power the nodes have: If a hardware node is not able to process its entire simulation workload within the foreseen real-time interval, the real-time constraint is violated leading to simulation delays.

5 Conclusion

In this paper we have introduced an adaptable demonstrator platform where multiple simulation instances can be used to simulate a distributed automotive system. Using a physical network, the platform combines the advantages of a rapid simulation configuration with a more realistic communication. The simulation instances provide a middleware for communication and a task execution model for computation. An exemplary implementation has been presented which runs an agent-based graceful degradation approach to achieve fail-operational behaviour. As long as the real-time factor of the simulations can be met, the demonstrator platform can be used to easily demonstrate emerging decentralized approaches on different hardware architectures.

References

1 AUTOSAR. *SOME/IP Protocol Specification R19-11.* URL: `https://www.autosar.org/fileadmin/user_upload/standards/foundation/19-11/AUTOSAR_PRS_SOMEIPProtocol.pdf`.

2 Ontje Lünsdorf and Stefan Scherfke. *SimPy Discrete Event Simulation Library for Python, Version 3.0.9.* URL: `https://simpy.readthedocs.io`.

3 Felix Reimann, Martin Lukasiewycz, Michael Glaß, and Fedor Smirnov. *OpenDSE – Open Design Space Exploration Framework*, 2019. URL: `http://opendse.sourceforge.net/`.

4 Selma Saidi, Sebastian Steinhorst, Arne Hamann, Dirk Ziegenbein, and Marko Wolf. Future automotive systems design: Research challenges and opportunities: Special session. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2018.

5 Philipp Weiss, Andreas Weichslgartner, Felix Reimann, and Sebastian Steinhorst. Fail-operational automotive software design using agent-based graceful degradation. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, 2020.