SCHLUSSBERICHT

der Universität Siegen zum Projekt SelfAutoDOC

Verbundprojekt:
SelfAutoDOC – Self-healing Automotive Driving platform powered by Organic Computing

Teilvorhaben:
Diagnosemodelle und Selbstheilungsmechanismen auf Basis von Organic-Computing in Fahrzeugen

Allgemeine Angaben

Zuwendungsempfänger: Universität Siegen

Förderkennzeichen: 19A21044C

Laufzeit des Vorhabens: 01.12.2021 – 28.02.2025

I Kurzdarstellung

I.1 Aufgabenstellung

Das Forschungsprojekt SelfAutoDOC hatte das Ziel, innovative fehlertolerante und selbstheilende Architekturen für Fahrzeugfunktionen des autonomen Fahrens zu entwickeln. Während herkömmliche Sicherheitsmechanismen in der Automobilindustrie vor allem auf statische Redundanz setzen, verfolgte SelfAutoDOC einen neuen Ansatz: Organic Computing (OC). Dieser ermöglicht eine dynamische Zuteilung, Rekonfiguration und Selbstheilung von Services, die auf den Steuergeräten der (E/E-)-Fahrzeugarchitektur ausgeführt werden, indem Mechanismen der Selbstorganisation, Selbstoptimierung und aktiven Fehlerdiagnose implementiert werden.

Die Universität Siegen (USI) übernahm im Projekt eine zentrale Rolle in der Entwicklung und Implementierung von Fehlertoleranz- und Diagnoseverfahren für verteilte Steuergeräte. Der Schwerpunkt lag auf der aktiven Fehlerdiagnose und deren Verbindung mit Selbstheilungsmechanismen, um sicherzustellen, dass sicherheitskritische Fahrzeugfunktionen auch im Fehlerfall erhalten bleiben. Hierbei standen insbesondere folgende Forschungsfragen im Fokus: 1. Wie können in einem verteilten System von Fahrzeugsteuergeräten autonom Fehler im Organic Computing erkannt, lokalisiert und kompensiert werden? 2. Welche Algorithmen und Modelle ermöglichen eine adaptive Fehlerdiagnose in Echtzeit? 3. Wie lassen sich Selbstheilungsmechanismen in bestehende sicherheitskritische Fahrzeugarchitekturen integrieren?

Zur Beantwortung dieser Fragen entwickelte USI hormonbasierte Diagnoseverfahren, die es ermöglichen, Fehlerquellen im laufenden Betrieb kontinuierlich zu identifizieren und zu isolieren. Das Verfahren ermöglicht eine autonome Adaptation von Services auf die jeweils zur Verfügung stehenden Rechenressourcen bzw. Steuergeräte.

Ein weiterer Schwerpunkt der Arbeiten von USI war die wissenschaftliche und experimentelle Evaluation der entwickelten Ansätze. Dazu wurden OC-Hormonsimulatoren entwickelt und in Kombination mit Netzwerksimulatoren, sowie Hardware-in-the-Loop (HiL)-Simulationen genutzt, um das Verhalten der Algorithmen unter realistischen Bedingungen zu validieren. Durch gezielte Fehlereinstreuungsexperimente wurde gezeigt, dass durch die entwickelten Verfahren die Zuverlässigkeit und Verfügbarkeit der betrachteten Services des Steuergerätesystems signifikant verbessert werden konnten.

I.2 Voraussetzungen, unter denen das Vorhaben durchgeführt wurde

Der Forschungs- und Entwicklungsauftrag von SelfAutoDOC wurde unter Berücksichtigung mehrerer technischer, organisatorischer und forschungsbezogener Rahmenbedingungen durchgeführt. Die Universität Siegen verfügt über eine leistungsfähige Forschungsinfrastruktur für eingebettete Systeme inklusive zeitsensitiver echtzeitfähiger Netzwerksimulationstechnologien. Auch die bestehenden Hardware- und Softwareplattformen stellten während der gesamten Projektlaufzeit eine solide Grundlage für die Entwicklung und Validierung der Diagnose- und Selbstheilungsmechanismen im Organic Computing dar.

Die enge Zusammenarbeit mit der TTTech Auto Germany GmbH, der Goethe-Universität Frankfurt und der plc2 Design GmbH erfolgte über regelmäßige Abstimmungen. Die interdisziplinäre Vernetzung von Informatik, Elektrotechnik und Fahrzeugtechnik stellte sicher, dass sowohl algorithmische als auch hardwareseitige Aspekte der Diagnose umfassend untersucht wurden. Wissenschaftlich knüpfte das Projekt an bestehende Arbeiten von USI in adaptiver Fehlertoleranz,

Ontologie-basierten Diagnoseverfahren und selbstorganisierenden Rechenarchitekturen an. Erkenntnisse aus den abgeschlossenen Projekten DREAMS, ADISTES und DAKODIS sowie Vorarbeiten zur modellbasierten Fehlerdiagnose und genetischen Algorithmen für sicherheitskritische Netzwerke flossen direkt in die Entwicklung von SelfAutoDOC ein.

I.3 Planung und Ablauf des Vorhabens

Das SelfAutoDOC-Projekt wurde in drei Phasen umgesetzt, wobei der Fokus der Universität Siegen auf der Erforschung, Entwicklung und Integration von Fehlerdiagnose- und Selbstheilungsverfahren lag. In der Anforderungsanalyse definierte USI in Zusammenarbeit mit den Projektpartnern Diagnosemodelle und untersuchte geeignete zeitsensitive/Echtzeit-Netzwerktechnologien. Anschließend erforschte und entwickelte USI adaptive Diagnoseverfahren, die in einem Fehlereinstreuungs-Framework getestet wurden. Dazu entwickelte USI einen sogenannten *Hormon-Guard* zur Kontrolle von (fehlerhaften) Hormonausschüttungen, deren Korrektheit essenziell für das Funktionieren des künstlichen Hormonsystems, also des Schedulers der Tasks und somit der Systemservices ist. In der letzten Phase wurden die Verfahren in einem Use Case beispielhaft in eine Steuergerätearchitektur integriert und in einer Fahrzeug-Demonstrationsplattform sowie in einer simulierten Time-Sensitive-Network-Umgebung für echtzeitfähiges Organic Computing implementiert, wodurch eine erhöhte Fehlertoleranz nachgewiesen wurde. Die Forschungsergebnisse, in den Schwerpunktfeldern von USI wurden in mehreren Publikationen auf Fachkonferenzen veröffentlicht. Zudem stehen zwei im Rahmen des Projektes angefertigte Dissertationen kurz vor der Fertigstellung.

I.4 Wissenschaftlicher und technischer Stand, an den angeknüpft wurde

SelfAutoDOC baute auf bestehenden Ansätzen der Fehlertoleranz und Selbstorganisation in eingebetteten Systemen auf. Zentral waren Konzepte aus dem Bereich des Organic Computing, insbesondere das künstliche Hormonsystem zur dynamischen Zuordnung von Tasks bzw. Services auf Steuergeräte in einem verteilten System und die künstliche DNA zur Beschreibung des Tasks innerhalb der selbstorganisierende Systemarchitekturen, [9, 10, 11, 12]. Zudem wurden zeitgesteuerte Netzwerktechnologien (TSN, TTEthernet) für die deterministische Kommunikation eingebunden [13, 14].

Wissenschaftlich knüpfte die Universität Siegen an Erkenntnisse aus den Projekten DREAMS, ADISTES und DAKODIS an, die sich mit Fehlertoleranz, Echtzeit-Diagnose und adaptiven Rechenarchitekturen befassen, [15, 16]. Als technische Grundlage dienten bekannte Methoden der formalen Verifikation, der Ontologie-basierten Diagnose und des maschinellen Lernens zur Fehleranalyse [17]. Für die Umsetzung wurden einschlägige wissenschaftliche Publikationen zu Organic Computing, selbstorganisierenden Systemen und fehlertoleranter Fahrzeugsteuerung genutzt. Zudem erfolgte eine kontinuierliche Recherche in Datenbanken wie IEEE Xplore, SpringerLink und ScienceDirect, um aktuelle Entwicklungen zu berücksichtigen.

I.5 Zusammenarbeit mit anderen Stellen

Die Zusammenarbeit im SelfAutoDOC-Projekt erfolgte seitens der Universität Siegen ausschließlich innerhalb des Konsortiums, bestehend aus TTTech Auto Germany GmbH, Goethe-Universität Frankfurt, plc2 Design GmbH und der Universität Siegen. Eine Kooperation mit externen Partnern oder weiteren Institutionen fand nicht statt.

II Eingehende Darstellung

II.1 Verwendung der Zuwendung und erzielte Ergebnisse im Einzelnen, mit Gegenüberstellung der vorgegebenen Ziele

Verwendung der Zuwendung Die Zuwendung wurde überwiegend für Personalausgaben verwendet. Zwei Doktoranden arbeiteten kontinuierlich am Projekt und stehen kurz vor dem Abschluss ihrer Dissertationen. Darüber hinaus unterstützten zeitweise weitere wissenschaftliche Mitarbeiter des Lehrstuhls mit ihrer Expertise das Vorhaben. Aufgrund der Vielzahl entstandener wissenschaftlicher Veröffentlichungen wurde ein größerer Anteil der Reisemittel als ursprünglich geplant für Dienstreisen zur nationalen bzw. internationalen Technologie-Konferenzen zur Präsentation unserer Forschungsergebnisse für ein breites Fachpublikum (Dissemination), sowie der Erlangung fachbezogener Rückmeldungen zu unseren Arbeiten und Ergebnissen durch den Austausch und Diskussionen mit anderen Konferenzteilnehmern eingesetzt. Im Gegenzug wurden projektinterne Treffen vermehrt online durchgeführt. Die ursprünglich geplanten Anschaffungen von Equipment konnten aufgrund von Lieferengpässen und Preiserhöhungen nicht zum gewünschten Zeitpunkt durchgeführt werden, sodass stattdessen auf realitätsnahe Simulationswerkzeuge zurückgegriffen wurde und diese weiterentwickelt wurden. Im Rahmen einer genehmigten, kostenneutralen Projektverlängerung um drei Monate wurden nicht genutzte Sachmittel in Personalmittel umgewidmet, um zusätzliche Arbeiten zur Ergebnissicherung umzusetzen.

Erzielte Ergebnisse Im Folgenden werden die im Rahmen des Projekts SelfAutoDOC erzielten Ergebnisse vorgestellt, die sich anhand den in der Vorhabensbeschreibung definierten Zielen und Arbeitspaketen strukturieren.

Die Universität Siegen (USI) hat zu allen Arbeitspaketen beigetragen. In der ersten Projektphase insbesondere zu den Systemanforderungen in Arbeitspaket AP1, sowie zur Gap-Analyse und Komponentenauswahl in AP2. In der letzten Projektphase hat USI am Demonstrator mitgearbeitet und die Fehlereinstreuung und Fehlerdiagnose integriert (AP7 und AP8). Im Kontext der wissenschaftlich-technischen Kernarbeitspakete 4, 5 und 6 hat USI wesentliche Beiträge in drei thematischen Schwerpunktthemen geleistet:

1. **Organic Computing Real-Time Co-Simulation mit OMNeT++ und CARLA** (AP4). Die Entwicklung und Erprobung eines Co-Simulators, der die selbstorganisierenden Mechanismen des Organic Computing mit realitätsnahen Simulationsumgebungen für echtzeitfähige Fahrzeugnetzwerke basierend auf Time-Sensitive-Networking (TSN) und Fahrdynamik (CARLA-Simulator) verbindet.

2. **Formale Verifikation** (AP5). Die Untersuchung und Verifizierung zentraler Systemkomponenten und -prozesse mithilfe formaler Methoden, um die Korrektheit und Zuverlässigkeit des Organic-Computing-Ansatzes sicherzustellen.

3. **Organic-Computing-Diagnose** (AP6). Die Konzeption und Implementierung von Fehlerinjektion sowie Diagnoseverfahren mittels hormonbasierter Mechanismen zur Fehlerdetektion, -identifikation und Rekonfiguration, um eine robuste und fehlertolerante Systemarchitektur zu realisieren.

In den folgenden Abschnitten werden die einzelnen Ergebnisse und ihre Bedeutung für das Projekt SelfAutoDOC im Detail erläutert und mit den im Forschungsantrag angestrebten Zielen verknüpft.

II.1.1 Systemanforderungen und Gap-Analyse

In der ersten Arbeitsphase des Projekts arbeitete USI an der Anforderungserhebung mit (AP1, T1.1). Dabei wurde gemäß Task T1.2 ein Fehlermodell gemäß der Systemanforderungen aufgestellt, das Fehlerfälle identifiziert bzw. auf Fehlerfälle fokussiert, die von den initialen (d.h. zu Projektbeginn vorhandenen) Selbstheilungsmechanismen des Organic Computings noch nicht abgedeckt wurden, die für die weiteren Entwicklungen für Automotive-Anwendungen allerdings essenziell sind. Somit wurde ein Fehlermodell entworfen, das Fehlerfälle im künstlichen Hormonsystem beinhaltet, die direkten Einfluss auf die Task-Allokationen und somit auf den Gesamtsystemservice, bzw. auf dessen Qualität haben. Neben dem speziellen Fehlertyp Hormonfehler deckt das Fehlermodell die klassischen Fehlerdomänen ab, z.B. permenente/transiente Hardware-/Softwarefehler. Die Details zu den betrachteten Fehlerfällen, deren Auswirkungen und dem implementierten Fehlerinjektionsframework werden in Kombination mit der entwickelten Organic-Computing-Diagnose (AP6) in Abschnitt II.1.4 betrachtet.

In Arbeitspaket AP2 hat USI eine umfassende Gap-Analyse durchgeführt, die insbesondere die vollumfängliche Ermittlung des Stands-der-Technik beinhaltet. Die stellt für die Universität die Grundlage zum Verbinden der im Projekt vorgesehenen konkreten Entwicklungen mit dem aktuellen Forschungsbedarf in ebendiesem Kontext dar. Auf die erzielten Forschungsergebnisse wird in den nachfolgenden Abschnitten II.1.2–II.1.4 detailliert eingegangen.

Weiterhin hat USI im AP2 an der Auswahl der Komponenten für den Demonstrator mitgearbeitet und die Konzepte für die Erfassung von Features für die Diagnose auf einem hohen Abstraktionslevel entwickelt (Tasks T2.1, T2.2 bzw. T2.4). Die Analyse von Echtzeit-Kommunikationsprotokollen und Prozessierungshardware im Abgleich mit den aktuellen Trends der Automobilbranche bzw. im Sinne einer zukunftssicheren Unterstützung von Verwertungsmöglichkeiten wurde in SelfAutoDOC das Time-Sensitive Networking (TSN) ausgewählt, das sowohl ereignis- als auch zeitgesteuerte Kommunikation unterstützt. Deteils zur Integration von TSN in die SelfAutoDOC-Architektur finden sich im nachfolgenden Abschnitt II.1.2

II.1.2 Organic Computing Echtzeit-Co-Simulation mit OMNeT++ und CARLA

Im Projekt SelfAutoDOC wurde die Organic-Computing-Architektur maßgeblich weiterentwickelt, sodass sie sowohl die internen, selbstorganisierenden Prozesse als auch die Echtzeit-Kommunikation in Fahrzeugnetzwerken präzise abbildet. Die folgenden Abschnitte gliedern die wesentlichen Weiterentwicklungen und demonstrieren anhand veröffentlichter Ergebnisse, dass die Plattform den Anforderungen autonomer, sicherheitskritischer Systeme vollumfänglich gerecht wird.

Weiterentwicklung der Architektur als Diskret-Event-Simulator

Die Weiterentwicklung der durch den Projektpartner Universität Frankfurt vorhandenen Version eines Organic-Computing-Simulators bzw. Hormonsimulators hatte zum Ziel, die Selbstheilungs- und Selbstkonfigurationsmechanismen des künstlichen Hormonsystems gemäß des Fehlermodells zu erweitern und durch geeignete Schnittstellen mit der Automotive-Hardware des Demonstrators kompatibel zu machen. So wird in unserer ersten projektbezogenen Publikation **[2]** detailliert betrachtet, wie der OC-Simulator hinsichtlich dynamischer Task-Allokation und -Ausführung zu einem Diskret-Event-Simulator umgebaut und erweitert wurde, um TSN-basierte Kommunikation mit Schedules unterstützen zu können. Dazu wurden die bisher in den Rechenknoten (d.h.

Steuergeräten/Processing Elements (PEs)) kontinuierlich ablaufenden, unabhängigen Hormon-Regelschleifen des Organic Computings zur Task-Allokation in sogenannte Hormon-Tasks gekapselt. Zudem wurde ein Task-Dispatcher entwickelt, der lokal als Teil der OC-Middleware läuft und die Ausführungszeiten von Tasks kontrolliert. Auf der Forschungsseite zeigt die Veröffentlichung erstmals die Grundlagen für die Verbindung von hochdynamischem Organic Computing mit der im Automobilbereich notwendigen Berücksichtigung von Schedule-basierter Kommunikation von (sicherheitskritischen) Applikationstasks (d.h. der Systemdienste) und Hormontasks in kombinierten ereignis- und zeitgesteuerten Kommunikationsprotokollen wie TSN.

In einer Diskret-Event-Simulation für Organic Computing müssen die Ausführungszeitpunkte für die Tasks dezentral berechnet werden, sodass die jeweiligen Task-Dispatcher auf den Rechenknoten die Taskausführungen steuern können. Dafür haben wir das Task-Scheduling um die zeitliche Ebene erweitert, da die vorhandene Organic-Computing-Middleware lediglich das Scheduling in der räumlichen Ebene abdecken konnte. Die Ergebnisse wurden in Form einer erweiterten Architekturdefinition auf der Fachkonferenz Automotive meets Electronics & Control (AmEC) präsentiert **[4]**. Für die Demonstration des entwickelten schnellen List-Scheduling-Algorithmus nehmen wir zunächst ein allgemeingültiges logisches Modell einer Applikation ein Form eines gerichteten azyklischen Graphen an. Für das Scheduling wird dann eine sog. Hyperperiode definiert, in der jeweils zunächst die Hormon-Tasks gescheduled werden, bevor die Applikation-Tasks freien Zeitschlitze zugeordnet werden. Somit wird sichergestellt, dass die Organic-Computing-Middleware konfliktfrei auf allen Rechenknoten (ECUs) mit gleicher Periode ausgeführt werden.

Entwicklung eines Co-Simulationsprozesses zur Steuerung von Organic Computing, Fahrzeuganwendung in CARLA und Netzwerkkommunikation

Um die simulierte Organic-Computing-Anwendung in einen realen Kommunikationskontext zu überführen, wurde durch die Integration des Netzwerksimulators OMNeT++ eine OC-Co-Simulationsumgebung geschaffen. Das entwickelte Co-Simulationsframework synchronisiert die diskret-eventbasierte OC-Simulation (s.o.) mit der in OMNeT++ simulierten TSN-Kommunikation und stellt gleichzeitig die Schnittstellen zur Kontrolle und für die Kommunikation mit der Fahrzeuganwendung (d.h. den zugehörigen Tasks) bereit, die im Fahrzeugsimulator CARLA ausgeführt wird. Der Co-Simulationskontroller kommuniziert mit dem OC-Simulator über das Transmission Control Protocol/Internet Protocol (TCP/IP) und mit OMNeT++ über C Anwendungsprogrammierschnittstellen (API) und koordiniert das jeweils als nächstes auszuführende Event, d.h. entweder eine Task in OC oder die Kommunikation zwischen Tasks in OMNeT++.

Die Details der Implementierung finden sich in unserer wissenschaftlichen Veröffentlichung *Discrete-Event Co-Simulation Interface for Time-Triggered Organic Computing* **[7]**, die auf der Intelligent Systems Conference 2024 vorgestellt wurden. Das Papier adressiert auch erneut die Thematik Systemanforderungen aus AP1 und Auswahl der Komponenten für den Demonstrator bzw. Gap-Analyse aus AP2. Die Festlegung auf die Nutzung der TSN-Standards ergibt sich wissenschaftlich bzw. ingenieurtechnisch aus der Tatsache, dass TSN-Mechanismen wie Gate-Control-Lists sich besonders eignen, die im Projekt geforderte deterministische Nachrichtenübertragung in Kombination mit den notwendigen dynamischen Rekonfigurationen der OC-Selbstheilung und -Selbstkonfiguration in Echtzeit zu verbinden.

Dynamische TSN-Rekonfiguration für zeitsensitives Organic Computing

In einem letzten Schritt zur vollständigen Unterstützung der dynamischen Selbst-X-Eigenschaften des Organic Computings wurde das INET Framework von OMNeT++ eingebunden, um dynamische Rekonfigurationen der Nachrichtenkommunikation, (d.h. Zeiten und Pfade) zu realisieren. Die Gesamtarchitektur, für die die Nachrichtenkommunikation zwischen dem Organic-Computing und OMNeT++ über Message Queues implementiert wurde, haben wir in **[8]** mit einem wissenschaftlichen Schwerpunkt auf einer erstmaligen Verbindung grundlegender Eigenschaften der Paradigmen von dynamischem Organic Computing und Zeitdeterminismus veröffentlicht.

Forschungsseitig wird in dieser Veröffentlichung unter anderem die Herausforderung diskutiert, wie die Rekonfiguration bzw. das schnelle dynamische Anpassen der Kommunkationspläne in den Netzwerkknoten (Endsystemen und Routern) ermöglicht werden kann. Dazu wurden die Ergebnisse der heuristischen Planungs-Algorithmen unserer vorangegangenen Arbeiten im Projekt (siehe [4]) genutzt. Die Rekonfigurationen der Router des simulierten Fahrzeug-Kommunikationsnetzwerks, also das dynamische Anpassen (zur Laufzeit) von Scheduling-Tabellen der an der Kommunikation teilnehmenden Netzwerkknoten werden dann mittels TSN Gate-Control-Lists durchgeführt. Somit wurde sichergestellt, dass die Portierung der Implementierung auf die echte Fahrzeughardware im Rahmen des AP7 (Demonstrator) funktioniert und die Industriepartner die Verwertung im echten Fahrzeug vorantreiben können.

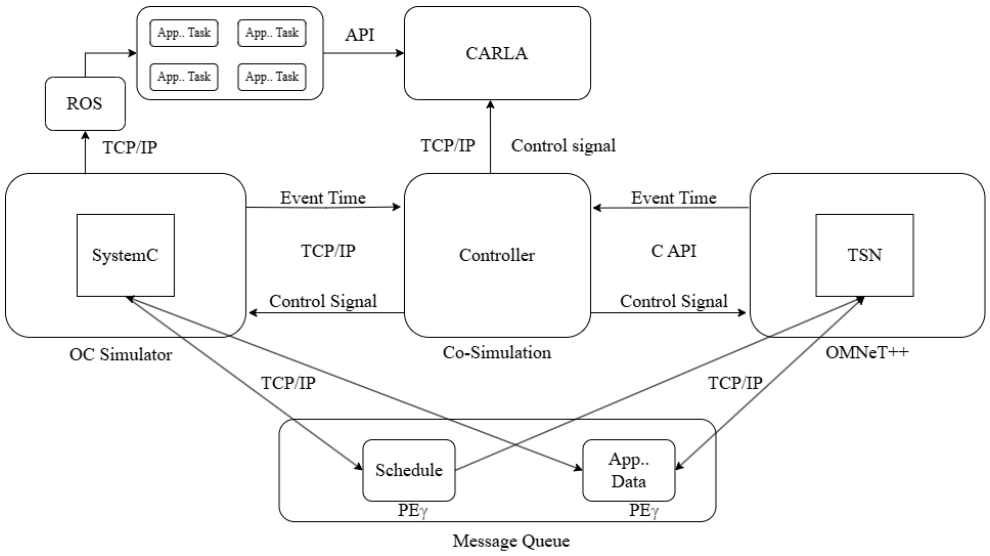

Abbildung 1: Organic Computing Co-Simulationsarchitektur mit CARLA und OMNeT++

Abbildung 1 zeigt die entwickelte SelfAutoDOC-Organic-Computing-Architektur/-Diskret-Event-Co-Simulation als Gesamtübersicht.

Die in den vorangegangenen Abschnitten II.1.2 beschriebenen Arbeiten von USI an SelfAutoDOC beziehen sich insbesondere auf das Arbeitspaket AP4 und haben die Forschungsfrage bzw. das Teilziel **Z1** des Vorhabens, Organic Computing in verteilten Systemen mit zeitgesteuerten Automobilnetzwerken lauffähig zu machen, adressiert. Dabei hat USI durch die entwickelte Architektur mit Co-Simulation, Algorithmik des List-Schedulers und der TSN-Rekonfigurationen maßgeblich die Teilinnovation **I1** (zeitgesteuerte Schedule-Adaptation) vorangetrieben. Die Ergebnisse unter Verwendung des Netzwerksimulators OMNeT++ und des Fahrzeugsimulators CARLA dienen den industriellen Projektpartnern zur Weiterentwicklung ihrer Produkte für die Automobilindustrie.

Die Weiterentwicklung der Organic-Computing-Architektur zu einer Diskret-Event-Co-Simulation demonstriert sowohl intern als auch im Zusammenspiel mit externen Fahrzeugnetzwerken höchste Zuverlässigkeit und Echtzeitfähigkeit. Die erzielten Ergebnisse – validiert durch mehrere begutachtete Konferenzbeiträge – demonstrieren, dass die SelfAutoDOC-Plattform den Anforderungen autonomer, sicherheitskritischer Systeme entsprechend des angestrebten Technology Readiness Level (TRL) 4 des Teilziels **Z1** gerecht wird. Der Versuchsaufbau im Labor bzw. der Hardware-in-the-Loop-Demonstrator unter Verwendung der Hardware (u.a. RazorMotion-Board als Steuergerät im Fahrzeug) der beiden Industriepartner TTTech Auto Germany GmbH und plc2 Design GmbH sowie des Fahrzeugsimulators CARLA wurde dem Projektträger im Rahmen der Abschlussveranstaltung vorgestellt.

II.1.3 Formale Verifikation

Das Ziel von Arbeitspaket 5 war die umfassende formale Verifikation der Organic-Computing-Architektur, insbesondere mittels Model-Checking von ADNA-Bausteinen und deren Zusammenspiel in automobilbezogenen, sicherheitskritischen Szenarien. Im Zentrum standen die formale Modellierung und Überprüfung von Diagnose- und Selbstheilungsalgorithmen sowie deren Integration in ein Zuverlässigkeitsmodell. Darüber hinaus wurden formale Timingmodelle entwickelt, um die Einhaltung notwendiger zeitlicher Anforderungen sicherzustellen.

Insbesondere wurde das Ziel verfolgt, die beweisbare Sicherheit von Organic Computing in Automobilsystemen nachzuweisen (Teilziel **Z3**). Dazu wurde im Rahmen des Projekts ein formales Modell von Organic Computing im Fahrzeug entwickelt (Innovation **I3**). Dieses Modell diente als Grundlage für die formale Überprüfung relevanter Sicherheitseigenschaften wie Deadlock-Freiheit, Zustandserreichbarkeit und Einhaltung von Timingvorgaben, um nachzuweisen, dass sicherheitskritische Anforderungen erfüllt werden können. Diese Maßnahmen zielten darauf ab, die Sicherheit, Robustheit und Zuverlässigkeit der Organic-Computing-Plattform für autonome, fail-operational Fahrzeugsysteme nachzuweisen und bilden damit eine wichtige Basis zur Umsetzung sicherheitskritischer Automobilfunktionen mit erhöhten Zuverlässigkeitsanforderungen.

Verifikation von ADNA-Bausteinen mit UPPAAL

UPPAAL ist ein Modellprüfwerkzeug (Model Checker), das zur Validierung und Verifikation von Systemeigenschaften eingesetzt wird, die als zeitbehaftete Automaten (Timed Automata) modelliert sind. Die Modellprüfung ist ein automatisiertes Verfahren zur Überprüfung formal beschriebener Eigenschaften eines Modells. In UPPAAL werden zeitbehaftete Automaten grafisch modelliert und analysiert. Das Tool erlaubt sowohl die Simulation dieser Modelle als auch die formale Verifikation von Anfragen auf Basis eines Teilbereichs der *Timed Computation Tree Logic* (TCTL). Einzelne zeitbehaftete Automaten werden zu einem Gesamtsystem zusammengefügt und gemeinsam ausgeführt. Dabei interagieren sie miteinander, was die Modellierung komplexerer Strukturen ermöglicht.

Methodik und Vorgehensweise

Ziel der Verifikationsmethodik war es, den Kontrollfluss der einzelnen ADNA-Bausteinen formal auf Korrektheit zu prüfen. Hierzu wurde für jede Komponente ein zustandsbasiertes Modell in

UPPAAL erstellt, das die wesentlichen Funktionen und Kontrollstrukturen der jeweiligen Softwareeinheit abbildete. Die Modellierung erfolgte manuell auf Grundlage des C-Quellcodes (der ADNA-Bausteine) und entsprechender Funktionsdiagramme.

Die Vorgehensweise gliederte sich in mehrere aufeinanderfolgende Schritte: Zunächst wurden die ADNA-Bausteine analysiert, wobei steuerungsrelevante Programmteile wie Schleifen, Bedingungen, Moduswechsel sowie relevante Eingabegrößen und Zustände identifiziert wurden. Anschließend erfolgte die Erstellung eines abstrahierten UPPAAL-Modells, wobei aufgrund fehlender Gleitkommaunterstützung numerische Berechnungen vereinfacht dargestellt werden mussten. Im dritten Schritt wurden spezifische Sicherheits- und Korrektheitskriterien formalisiert und als CTL-Formeln spezifiziert.

Die eigentliche Verifikation wurde mit dem UPPAAL Model Checker durchgeführt. Tritt dabei eine Verifikationsverletzung auf, wurden Gegenbeispiele analysiert und das Modell iterativ angepasst. Nach erfolgreichem Abschluss der Verifikation wurden die Modellgrößen, Verifikationszeiten sowie die überprüften Eigenschaften umfassend dokumentiert, um die Nachvollziehbarkeit der Ergebnisse sicherzustellen.

Modellprüfung mit UPPAAL und TCTL

Die Modellprüfung mit UPPAAL basiert auf einer Teilmenge der Timed Computation Tree Logic (TCTL), also der Erweiterung der CTL um zeitliche Aspekte. Verifikationsanfragen bestehen aus einem Pfadquantor (A, E), einem temporalen Operator (G, F) sowie der zu prüfenden Bedingung. Diese müssen nebenwirkungsfrei und deterministisch auswertbar sein. UPPAAL erlaubt die Analyse von Erreichbarkeits-, Sicherheits-, Lebendigkeits- und Zeiteigenschaften sowie die Prüfung auf Deadlocks und Invarianten. Die Modellierung der ADNA erfolgt über Timed Automata, welche Kontrollstrukturen wie Verzweigungen, Schleifen oder Funktionsaufrufe durch Zustandsübergänge abbilden. Dabei gewährleisten Synchronisationen eine korrekte sequentielle Ausführung. Gleitkommazahlen werden nicht unterstützt, weshalb durch geeignete Approximationen symbolische Validierungen sicherheitskritischer Bedingungen erfolgen.

Nach der Modellerstellung werden komponentenspezifische Eigenschaften formal definiert und mithilfe symbolischer Zustandsraumanalyse überprüft. UPPAAL liefert bei Nichterfüllung Gegenbeispiele zur Fehlersuche.

Abbildung 2 zeigt ein exemplarisches Modell, genauer die Hauptfunktion des `ALU`-Bausteins mit Initialisierung, Modusprüfung und symbolischer Ergebnisverarbeitung. Rücksprünge, Bedingungsprüfungen und Moduswechsel sind strukturell modelliert, wobei keine tatsächlichen Berechnungen durchgeführt, sondern nur deren Bedingungen verifiziert werden.

Evaluation und Verifikationsergebnisse der ADNA-Bausteine

Nachfolgend sind exemplarisch die Ergebnisse der Verifikation für ausgewählte ADNA-Komponenten dargestellt. Vergleiche hierzu Tabelle 1:

Insgesamt wurden acht Basisbausteine mithilfe von UPPAAL modelliert und verifiziert. Ziel war es, kritische Eigenschaften wie Zustandsübergänge, Kontrollflüsse und Modusverhalten zu überprüfen, ohne konkrete Rechenwerte oder Interaktionen zwischen den Komponenten zu berücksichtigen.

Das eingesetzte Modellierungskonzept eignet sich besonders für Kontrollfluss-orientierte Bausteine wie `Gate`, `MUX` und `DMUX`. Auch Komponenten wie `Offset`, `ALU` oder `Average`, deren Ablauf nicht

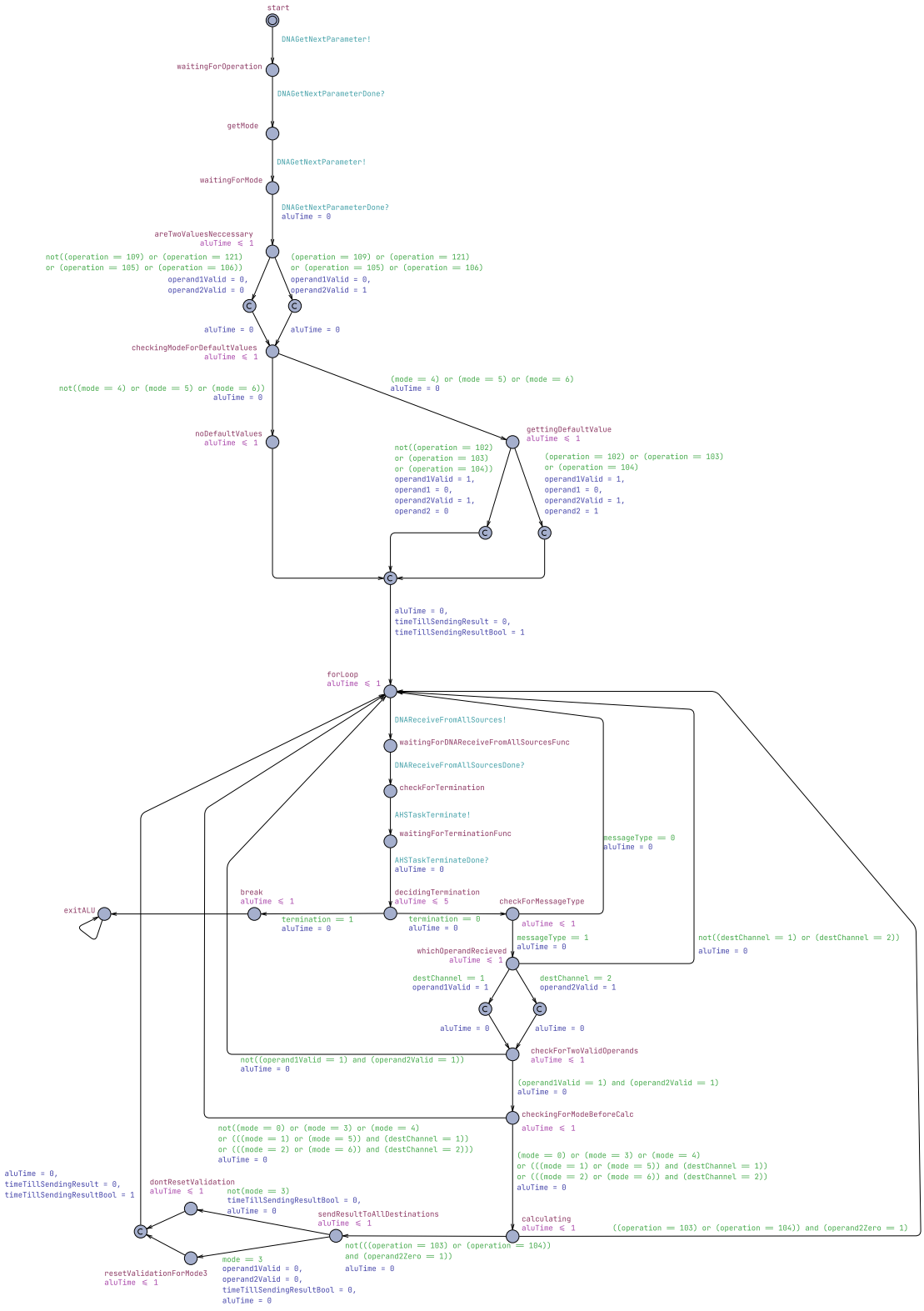

Abbildung 2: Beispiel: UPPAAL-Modell der Hauptfunktion des `ALU`-Basis Bausteins.

Tabelle 1: Ergebnisse der formalen Verifikation ausgewählter ADNA-Komponenten

Komponente	Zustände	Übergänge	Verifikationszeit	Geprüfte Eigenschaften
StateMachine	14	22	2,3 s	Deadlock-Freiheit, gültige Modusumschaltung, kein illegaler Rücksprung
StartProcedure	9	13	1,1 s	Korrekte Initialisierung, kein unerlaubter Reset, sichere Aktivierung
PIDController	18	26	3,6 s	Strukturkorrektheit, Fehlervermeidung bei Grenzwertprüfung, Moduswechsel
SafetyCheck	11	17	1,9 s	Prüfen von Invarianten, korrektes Verhalten bei fehlerhaften Eingaben
LoopHandler	24	37	4,8 s	Schleifenstruktur, Rücksprünge bei Fehlern, Deadlock-Freiheit

unmittelbar vom konkreten Berechnungsergebnis abhängt, konnten durch geeignete Prüfbedingungen erfolgreich analysiert werden. Lediglich bei kontinuierlich rechnenden Komponenten wie dem `PID`-Regler stößt UPPAAL aufgrund fehlender Unterstützung für Gleitkommazahlen an seine Grenzen. Diese Einschränkung ist nicht spezifisch für UPPAAL, sondern gilt für modellprüfungsbasierte Verfahren allgemein.

Die Verifikation konzentrierte sich insbesondere auf:

- das Erreichen kritischer Zustände (z. B. Send-Zustände),
- die korrekte Verarbeitung von Eingaben je nach Betriebsmodus,
- die Absicherung, dass nur gültige Eingangszustände zu Berechnungen führen.

Dabei konnten unter anderem folgende fehlerhafte Implementierungen identifiziert und behoben werden:

`ALU` – fehlerhaftes Verhalten bei unären Operationen Die ADNA-`ALU` verarbeitet zwei Operanden und unterstützt mehrere Modi zur Steuerung ihrer Berechnungslogik. In den Modi 0, 1 und 3 wurde festgestellt, dass der Berechnungszustand auch dann betreten wird, wenn nur ein Operand vorhanden ist – vorausgesetzt, es handelt sich um eine unäre Operation. In Modus 3 stellt dies ein kritisches Problem dar: Nach der ersten Berechnung wird der zweite Operand ungültig gesetzt, was bei einer reinen unären Eingabe weitere Operationen verhindert. Dieser Fehler wurde durch TCTL-Abfragen in UPPAAL erkannt (siehe Abbildung 2) und anschließend in der C-Implementierung des ADNA-Bausteins behoben.

`Average` – falsche Anzahl an Mittelwerten Der ADNA-`Average`-Baustein berechnet den Durchschnitt über eingehende Werte. In einem spezifischen Modus wurde die Berechnung fälschlicherweise über $x + 1$ statt x Werte durchgeführt. Dies kann zu minimalen Rechenfehlern führen oder, im Grenzfall, sogar einen Deadlock verursachen, wenn die erwartete zusätzliche Eingabe nicht erfolgt.

Diskussion und Bewertung des Verifikationsansatzes

Die durchgeführte Modellierung ADNA-Bausteine und die Ergebnisse zeigen, dass die modellbasierte Verifikation mit UPPAAL eine wirkungsvolle Methode zur Identifikation von Designfehlern in eingebetteten Komponenten darstellt. Auch ohne exakte numerische Berechnung lässt sich die Korrektheit des Ablaufverhaltens zuverlässig prüfen. Es kann geschlussfolgert werden, dass der Ansatz einer strukturierten, zustandsbasierten Modellierung auch für eingebettete Komponenten mit komplexem Kontrollfluss praktikabel und zielführend ist und somit die OC Middleware hinreichend prüft. Insbesondere die strikte Trennung von Zuständen und Übergängen sowie die explizite Modellierung von Schleifen, Bedingungen und Funktionsaufrufen ermöglicht eine detaillierte Analyse des Systemverhaltens. Die symbolische Verifikation mit UPPAAL bietet insbesondere im sicherheitskritischen Kontext klare Vorteile. Durch die formale Analyse lassen sich logische Fehler frühzeitig erkennen, die bei klassischen Tests oder Simulationen unentdeckt bleiben könnten. Die Modularisierung der Funktionen in separate Automaten fördert dabei sowohl die Nachvollziehbarkeit als auch die Wiederverwendbarkeit der Modelle. Zustandsinvarianten und synchronisierte Übergänge ermöglichen eine exakte Kontrolle über die Ausführungslogik.

Herausfordernd bleibt der hohe initiale Modellierungsaufwand: Die manuelle Überführung komplexer Kontrollflüsse, speziell bei tief verschachtelten Bedingungen oder iterativen Strukturen, erfordert Erfahrung im Umgang mit UPPAAL und erzeugt teils große Zustandsräume. Darüber hinaus schränkt die fehlende Unterstützung für Gleitkommazahlen die Überprüfbarkeit numerischer Berechnungen ein, wodurch sich die Verifikation vorrangig auf strukturelle und sicherheitsrelevante Aspekte beschränkt. Dennoch zeigt der Ansatz bzw. das beschriebene Vorgehen großes Potenzial, insbesondere für die Industriepartner für die Validierung ihrer Softwarekomponenten für die Fahrzeugsteuergeräte der Zukunft. Eine automatisierte Modellgenerierung aus ADNA-Quellcode könnte den Aufwand signifikant senken, wofür weitere Arbeiten über dieses Projekt hinaus notwending sind. Auch eine Integration in bestehende CI/CD-Pipelines erscheint sinnvoll, um Verifikation und Qualitätssicherung eng zu verzahnen. Perspektivisch wäre zudem eine Erweiterung um datenbasierte Prüfverfahren denkbar, etwa durch symbolische Auswertung oder Intervallarithmetik.

Insgesamt hat sich UPPAAL als wirksames Werkzeug zur Absicherung von Kontrollflüssen bewährt. Die entwickelten Modelle leisten einen belastbaren Beitrag zur Erhöhung der Softwarezuverlässigkeit und ergänzen bestehende Testverfahren auf sinnvolle Weise. Die durchgeführte Verifikation zeigt, dass sich sicherheitskritische Abläufe eingebetteter Software bereits auf Modellebene zuverlässig prüfen lassen. Auch ohne exakte Rechenwerte ermöglicht die modellbasierte Methode eine objektive Bewertung von Zustandsübergängen, Kontrollflüssen und Reaktionsmustern, so wie es im Teilziel **Z3** vorgesehen war.

Besonders hervorzuheben sind dabei:

- die frühzeitige Identifikation von Designfehlern,
- die nachvollziehbare Validierung der Ausführungslogik und
- die Eignung zur Integration in sicherheitskritische Entwicklungsprozesse.

Für datenintensive Komponenten bedarf es ergänzender Verfahren zur Absicherung numerischer Aspekte. Als Werkzeug zur strukturellen Verifikation stellt UPPAAL jedoch ein robustes Element innerhalb moderner Qualitätssicherungsprozesse dar, mit besonderer Relevanz für sicherheitskritische Domänen wie Luftfahrt, Automotive oder Medizintechnik.

Die von den Projektpartnern USI und UGOE erzielten Forschungsergebnisse der formalen Verifikation von künstlicher DNA im Kontext des SelfAutoDOC-Projektes befinden sich aktuell in einer Iteration für die geplante Veröffentlichung bei einer renommierten wissenschaftlichen Fachkonferenz. Wir erwarten die finale Veröffentlichung mit dem Titel *ADNA-based Organic Computing for Fail-Safe Systems Providing High Dependability: Implementation and Verification of ADNA Building Blocks* noch in 2025 und werden die entsprechende Referenz anschließend online stellen.

II.1.4 Organic-Computing-Diagnose

Modellierung und Simulation von Fehlern in Organic Computing

Zur Erhöhung der Robustheit autonomer Systeme und zur Realisierung des Teilziels **Z4** (experimentelle Evaluation der Zuverlässigkeit beim Einsatz von Organic Computing) wurde im SelfAutoDOC-Projekt ein detailliertes Fehlermodell sowie eine generische Fehlereinstreuungsumgebung für Organic Computing auf Basis des existierenden Hormonsimulators entwickelt. Das Fehlermodell aus AP1 klassifiziert u.a. hormonbasierte Fehler, die im künstlichen Hormonsystem (AHS) auftreten können, in zwei Kategorien: fehlende Hormone und zu hohe bzw. zu niedrige Hormonwerte, wobei fehlende Hormone als ein Spezialfall von zu niedrigen Hormonwerten betrachtet werden können.

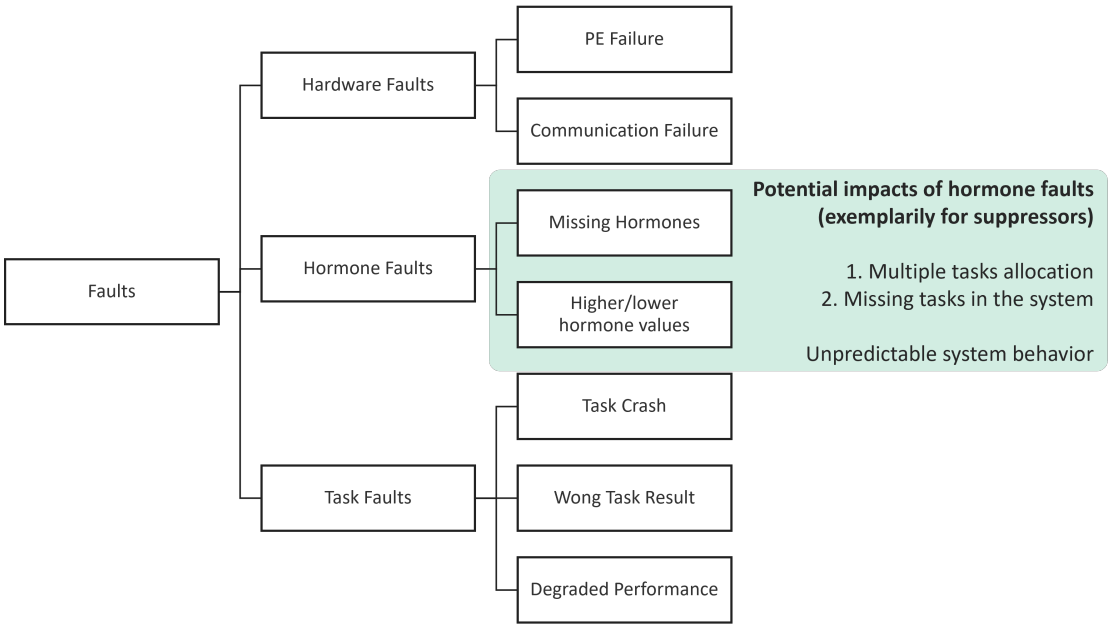

Abbildung 3: Fehlermodell, insbesondere Hormonfehler (vgl. [1])

Wie in Abbildung 3 markiert, können diese Fehlerfälle zu kritischen Systemzuständen führen, indem sie Task-Verteilungen im System massiv stören und somit die Zuverlässigkeit und die Verfügbarkeit des Gesamtsystems bzw. des Systemservices beeinträchtigen. Daher waren ihre Analysen von besonderem Interesse für unsere Forschungen und eine wesentliche Grundlage für Fehlereinstreuungsexperimente.

Eine umfassende Beschreibung der im Rahmen des AP6 implementierten Fehlereinstreuungsumgebung für Organic Computing (Innovation **I4**) und der durchgeführten Analysen und Auswertungen

von Hormonfehlern und deren Auswirkungen auf das System findet sich in unserer begutachteten wissenschaftlichen Veröffentlichung *Fault Injection Framework for Organic Computing Architecture* **[1]**, die in den Proceedings der Intelligent Systems Conference 2023 erschienen ist.

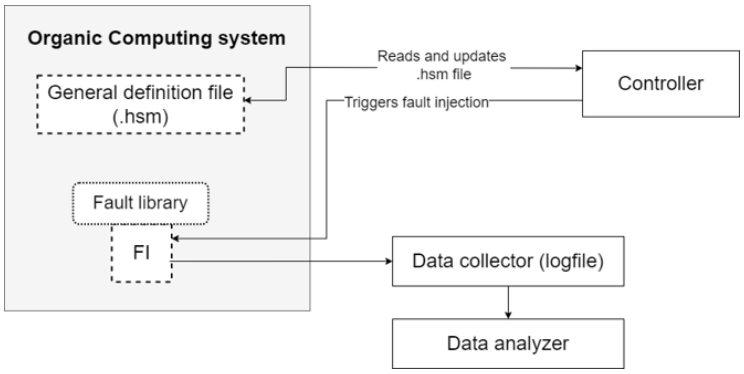

Abbildung 4: Fehlerinjektion

Abbildung 4 abstrahiert die Fehlereinstreuungsumgebung, die aus mehreren Komponenten besteht und mit der OC-Middleware interagiert: Einem *Fault Injector*, der gezielt Fehler gemäß eines spezifischen Fehlerkatalogs injiziert, einer *Fault Library* zur Speicherung der Fehlerszenarien, einem *Controller* zur Steuerung der Fehlerinjektionen sowie einem *Data Collector* und einem *Data Analyzer* zur Erfassung und Auswertung der Experimente. Durch gezielte Simulationen konnten wir nachweisen, dass insbesondere fehlende Hormone in der periodisch auftretenden Offer-Phase (des AHS) dazu führen, dass Tasks mehrfach allokiert und ausgeführt werden, was Ressourcen belastet und zu undefinierten Systemzuständen und Systemausfällen führen kann. Zu hohe Hormonwerte dagegen verhindern, dass Tasks zuverlässig ausgeführt werden, was ebenfalls kritisch ist. Diese Erkenntnisse verdeutlichen, dass klassische Organic-Computing-Architekturen ohne spezielle Erweiterungen nicht ausreichend robust gegenüber solchen hormonbasierten Fehlern oder auch absichtlich herbeigeführten Hormonmanipulationen sind. Basierend auf diesen Erkenntnissen haben einen Hormon-Guard entwickelt und in die Organic-Computing-Laufzeitumgebung integriert, wie im Folgenden im Detail erläutert wird.

Die Entwicklung und Evaluation des Fehlermodells und der generischen Fehlereinstreuungsumgebung zur gezielten Fehlerinjektion und Fehleranalyse entsprechen maßgeblich der Teilinnovation **I4** (des Projektantrags) und legen die Grundlagen für weitere Schutzmechanismen im SelfAutoDOC-Projekt. Dies stellt einen entscheidenden Schritt dar, um autonomes Fahren mittels Organic Computing sicher und zuverlässig zu gestalten.

Diagnose-Framework für Organic Computing

Zur detaillierten Diagnose auf Systemebene hat USI im AP6 ein umfassendes Diagnose-Framework entwickelt, das speziell für die Organic-Computing-Plattform angepasst wurde und über das künstliche Hormonsystem integriert wurde. Das Framework besteht aus zwei wesentlichen Modulen, *LocalDiagnosis* (LocDiag) und *SystemDiagnosis* (SysDiag), wie Abbildung 5 zeigt.

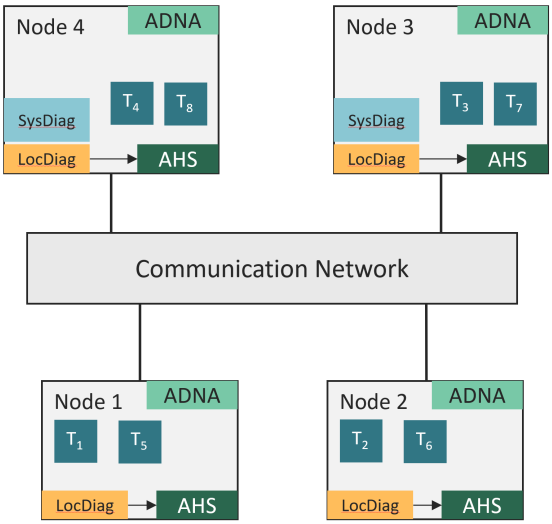

Abbildung 5: Übersicht über die Diagnosearchitektur

LocalDiagnosis überwacht einzelne Rechenknoten (PEs) und sammelt kontinuierlich detaillierte Zustandsinformationen. Zu den Inputs der LocalDiagnosis gehören hormonelle Aktivität, Ressourcenstatus (wie Speicher- und CPU-Auslastung), Task-Ausführungsstatus und Kommunikationsparameter. Diese Eingaben werden in einem strukturierten Diagnoseparameter-Vektor zusammengefasst, der kritische Diagnosedaten eindeutig repräsentiert. Der Output der LocalDiagnosis ist dieser Diagnoseparameter-Vektor, der detaillierte und standardisierte lokale Zustandsinformationen bereitstellt.

SystemDiagnosis ist als redundant laufende Task anzusehen und nutzt diese lokalen Diagnoseparameter-Vektoren als Input und aggregiert sie zu einer globalen Systemansicht (Abbildung 6). Die aggregierten Informationen über die PEs und Tasks werden analysiert, um systemweite Muster und mögliche Fehlzustände zu erkennen. Als Output generiert die SystemDiagnosis umfassende Systemdiagnosen und liefert konkrete Entscheidungsvorlagen für die automatische Systemrekonfiguration und Selbstheilung.

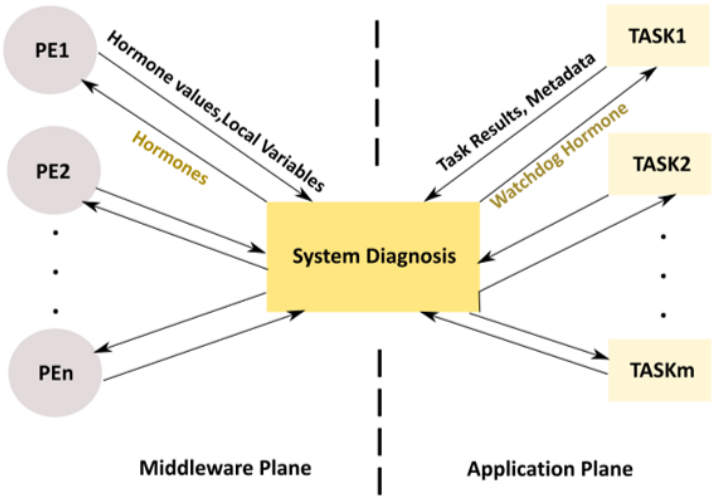

Abbildung 6: Datenfluss für die Systemdiagnose

Die generische Struktur des Diagnoseframeworks von SelfAutoDOC erlaubt es, flexibel auf verschiedene Systemanforderungen und unterschiedliche Arten von Verarbeitungselementen (wie z.B. das RazorMotion Board) einzugehen. Die Diagnosevektoren sind generisch und erweiterbar gestaltet, sodass zusätzliche Parameter je nach spezifischen Anforderungen der jeweiligen Anwendung problemlos integriert werden können. Typischerweise werden Parameter wie CPU-Last, Speicherverbrauch, Kommunikationslatenzen, hormonelle Signalstärken und Task-Erfolgskriterien gesammelt.

Die Diagnosearchitektur und erste Evaluationsergebnisse haben wir auf der GI/ITG International Conference on Architecture of Computing Systems (ARCS) präsentiert **[3]**. Anhand einer autonomen Fahrzeugsimulation in begrenzter Umgebung wurde gezeigt, wie eine getriggerte Rekonfiguration und ADNA-Degradation die Verfügbarkeit des Systemservices in Fehlerfällen (u.a. Ausfall von Umgebungssensorik) erheblich verbessert hat. Die Fähigkeiten der Organic-Computing-Diagnose werden mit Einbindung des Hormon-Guards (auf lokaler Ebene) und des Stress-Hormon-Algorithmus (auf Systemebene) vervollständigt und werden in den folgenden Abschnitten erläutert.

Das Diagnose-Framework als ein Teil der Innovation **I2** (Organic Computing Diagnose und Integration von Diagnose mit Selbst-Heilung) trägt durch seine Skalierbarkeit und Integration in die OC-Middleware direkt zum Teilzeil **Z2** bei, indem es den Grundstein für getriggerte Rekonfigurationen legt. Somit können, um hinreichende Zuverlässigkeit sicherzustellen, Systemservices nach Kritikalität im Fehlerfall proaktiv aufrecht erhalten werden.

Hormon-Guard: Absicherung der Task-Allokation und Hormonwerte

Ein weiterer wesentlicher Beitrag zum Teilziel **Z2** des SelfAutoDOC-Projekts – Erreichung hinreichender Zuverlässigkeit für sicherheitskritische Fail-Operational- Automobilsysteme – stellt die Entwicklung und Implementierung des sogenannten Hormon-Guards dar. Das Ziel des Hormon-Guards ist es, fehlerhafte hormonelle Signale innerhalb des künstlichen Hormonsystems (AHS) zu erkennen und deren negative Auswirkungen auf die Systemfunktion zu verhindern oder deutlich zu reduzieren.

Der Hormon-Guard wurde im Rahmen von AP6 entwickelt und baut auf unseren Forschungsergebnissen zu Fehleranalysen und -effekten aus unserer ersten Publikation [1] auf. Der Hormon-Guard wurde als ein Baustein von LocalDiagnosis des Diagnose-Frameworks in die OC-Middleware integriert, d.h. in die ADNA-Prozessoren, die auf der Fahrzeughardware laufen. Er arbeitet mit zwei grundlegenden Methoden: Schwellenwertprüfungen und Plausibilitätsanalysen. Die Schwellenwertprüfung stellt sicher, dass Hormone innerhalb vordefinierter, sicherer Grenzen bleiben, wobei die Schwellwerte automatisiert aus der künstlichen DNA, also aus dem Systembauplan extrahiert werden. Bei Überschreiten dieser Grenzen werden die Signale automatisch verworfen um das System vor fehlerhaften Taskverteilungen zu schützen. Die Plausibilitätsprüfungen analysieren zusätzlich die logische Konsistenz der Hormone der unterschiedlichen Rechenknoten mit dem Wissen über deren jeweils aktuelle Performance-Fähigkeiten (z.B. welche und wie viele Tasks ausgeführt werden können) und erkennt inkorrekte Zuordnungen und widersprüchliche Signale.

Unsere experimentelle Evaluation zeigt, dass der Hormon-Guard insbesondere in Szenarien mit überhöhten Suppressorwerten äußerst wirksam ist. So konnte gezeigt werden, dass die Systemstabilität selbst unter mehrfachen Fehlereinspeisungen aufrechterhalten wird, und die Aufgabenverteilung weiterhin fehlerfrei funktioniert. Dieses realitätsnahe Anwendungsszenario wurde dem Projektträger im Rahmen der Abschlussveranstaltung anhand des Demonstrators mit echter Fahr-

zeughardware der beteiligten Industriepartner vorgestellt.

Unsere zugehörigen wissenschaftlichen Ergebnisse wurden nach erfolgreicher Begutachtung auf der MECO-Konferenz 2024 vorgestellt **[5]**. Die Publikation demonstriert den Hormon-Guard als weiteren wesentlichen und innovativen Baustein des OC-Diagnose-Frameworks (**I2**).

Adaptive Fehlerdiagnose auf Systemebene durch Stresshormon-Propagation

Als weiterer Beitrag zur adaptiven und zuverlässigen Fehlerlokalisierung wurde ein hormonbasierter Mechanismus entwickelt, der auf der Propagation sogenannter Stresshormone beruht und dadurch die Adaptivität der Diagnose gemäß des dynamischen Characters des Organic Computing unterstützt. Diese Methode nutzt OC-inspirierte Prinzipien, um Fehlerzustände effektiv zu erkennen und zu lokalisieren. Unter dem Titel *Application Level Fault Diagnosis in Organic Computing Using Stress Hormone Propagation* haben wir unsere Forschungsergebnisse zur adaptiven Fehlerdiagnose auf dem International Symposium on Multidisciplinary Studies and Innovative Technologies präsentiert **[6]**. Ziel dabei war, den Diagnose-Algorithmus skalierbar für die Automobil-Industrie bzw. die beteiligten Industriepartner zu machen. Es sollte gewährleistet sein, dass spezifische Diagnosetechniken in Form von in ADNA kodierten Tasks jederzeit angepasst oder ersetzt werden können, wenn die Anwendung oder das unterliegende System mit seiner u.U. speziellen Sensorik dies erforderlich macht. Der Fokus lag somit auf der Integration des entwickelten sog. Stresshormon-Algorithmus in das Diagnose-Framework. Die Fähigkeiten der Fehlerdiagnose entfalten sich dann durch das proaktive Triggern von Rekonfigurationen über das AHS, sodass bspw. Tasks proaktiv umverteilt werden, um z.B. Rechnenknoten vor Totalausfall zu schützen oder um eine hinterlegte Degradations-ADNA zu aktivieren, die bei zu geringer verbleibender Gesamtrechenleistung im System die Kernfunktionalitäten des Systemservices aufrechterhält.

Im Organic Computing ist die Systemanwendung in Form von künstlicher DNA kodiert, d.h. Applikationsmodell (auch logisches Modell der Anwendung) liegt in Form eines azyklischen Graphen vor.

Das Ziel der Fehlerdiagnose ist die Erkennung der ursprünglichen fehlerhaften Task (wobei die Tasks entsprechend Sensorik abstrahieren, deren Daten sie verarbeiten) und entsprechende Abhilfemaßnahmen, z.B. Alarme, Neustart von Tasks, Migrationsaufgaben, Rekonfigurationen.

Die Kernidee unseres Diagnosealgorithmus ist, verteilten Konsens über die fehlerhaften Tasks in einem ADNA-Task-Graphen zu erzielen. Dazu soll folgendes gelten:

1. Annahme: Wenn der Input einer Task falsch ist, dann ist auch der Output falsch.
2. Jede nachfolgende Task (Kind-Task) führt eine Fehlererkennung am Output ihrer übergeordneten Task (Eltern-Task) durch, z.B. eine Eingabevalidierung.
3. Jede Kind-Task kann fehlerhafte Eltern-Tasks mit einer bestimmten Wahrscheinlichkeit erkennen.
4. Implikation: Alle Kinder-Tasks einer fehlerhaften Tasks sind fehlerhaft.

Der Stresshormon-Algorithmus beginnt mit der Fehlererkennung durch klassische Verfahren wie Input-Validation und Plausibilitätsanalyse. Wird eine Diskrepanz festgestellt, generiert der betroffene Rechenknoten (PE) Stresshormone. Die generierte Stressmenge wird durch mehrere Faktoren bestimmt: die Konfidenz der Fehlerannahme, die Gewichtung der Kommunikationsverbindungen zwischen Tasks und die Anzahl der direkt abhängigen Tasks (Out-degree). Anschließend erfolgt

eine gezielte Propagation dieser Stresshormone zu übergeordneten Tasks, wobei ein Normalisierungsfaktor berücksichtigt wird, um die Stressübertragung zu skalieren und zu gewichten. Zusätzlich werden sogenannte Relaxationshormone übertragen, welche die Ausbreitung des Stresssignals kontrollieren und eingrenzen, um eine präzise Lokalisierung der Fehlerursachen sicherzustellen. Diese Kombination von Stress- und Relaxationshormonen verleiht dem Algorithmus seine adaptiven Eigenschaften und erlaubt eine dynamische Reaktion auf wechselnde Systemzustände.

Die Erkennung der fehlerhaften Task erfolgt wie folgt:

Die Task mit dem höchsten Stresslevel wird als fehlerhafte Task identifiziert:

$$FaultyTask = \text{argmax}(StressLevel(Task)).$$

Der Stresslevel einer Task wird aus den von den Kinder-Tasks produzierten Hormonen berechnet:

$$StressLevel(Task) = \sum_{Children} (StressHormone) - \sum_{Children} (RelaxorHormone).$$

Die Experimente und Ergebnisse aus der entwickelten Stresshormon-basierten Fehlerdiagnose zur Erkennungsgenauigkeit finden sich in unserer o.g. Veröffentlichung [3] In einer Vielzahl von Simulationsszenarien wurden zunächst mehrere gerichtete azyklische Graphen (DAGs, d.h. logische (Task-) Modelle) für eine bestimmte Knotenanzahl generiert. Für jeden DAG wird jeder Knoten unabhängig über 1000 Simulationen hinweg als fehlerhaft betrachtet, sodass insgesamt etwa 20.000.000 Simulationen für die Darstellung der Erkennungsrate durchgeführt wurden. Die Erkennungsrate für jede untergeordnete Task wird mit 0,90 angenommen. Diese ergibt sich in einem konkreten Anwendungsfall als ein systemspezifischer Parameter. Die Auswertung zeigt, dass je mehr Knoten der Anwendungsgraph enthält, umso geringer ist die Varianz. Es konnte eine mittlere Genauigkeit von etwa 95% erreicht werden, was besser ist als die initial angenommene Erkennungsrate. Dies erklärt sich durch die Aggregation der Stresshormone von mehreren Kinder-Knoten des Graphen (Abbildung 7, vgl. [6]).

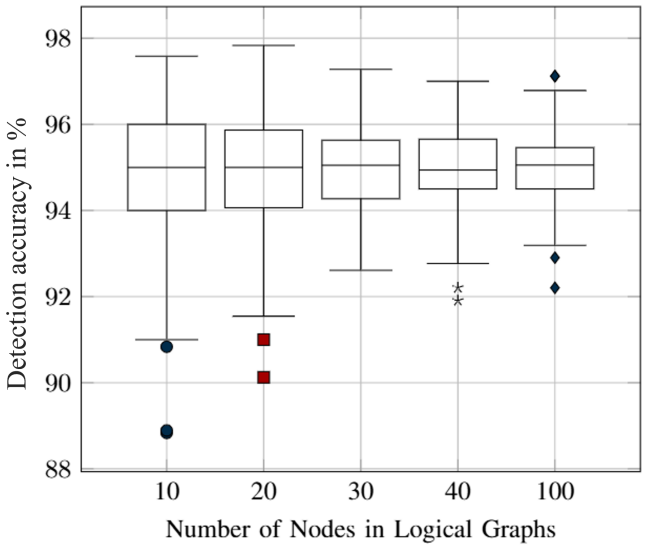

Abbildung 7: Fehlererkennungsraten mit dem Stresshormon-Algorithmus

Die Wirksamkeit und Selbst-Adaptivität dieses Ansatzes wurde durch umfangreiche Simulationen mit Graph-basierten Szenarien (SNAP-Datensätze) evaluiert. Die Ergebnisse zeigen hohe Lokalisierungsgenauigkeiten (88-98%) und belegen die Robustheit des Ansatzes auch bei zunehmender Komplexität der Netzwerkszenarien. Diese Resultate bestätigen, dass der Stresshormon-Algorithmus entscheidend dazu beiträgt, die geforderte Zuverlässigkeit für sicherheitskritische Fail-Operational Automobilsysteme (Teilziel **Z2**) zu erreichen und stellt zugleich eine wesentliche Innovation im Bereich der dezentralen und autonomen Fehlerlokalisierung mittels hormonaler Kommunikation (Teilinnovation **I5**) dar.

II.1.5 Demonstrator und Use Cases

Abbildung 8 gibt eine Übersicht über den Demonstrationsaufbau in Verbindung mit den Beiträgen der jeweiligen Partner (Firmenlogos in der Abbildung). USI hat eng mit UGOE zusammengearbeitet, da die Integration der USI-Beiträge hauptsächlich in der Organic-Computing-Middleware erfolgte.

Die Integration von Fehlereinstreuung und das Ausgleichen von Fehlerfällen mittels der entwickelten Diagnosemechanismen wurden im Rahmen von AP7 und AP8 (Demonstrationsfallstudie und Integration) in den CARLA-basierten Hardware-in-the-Loop Demonstrator integriert. Aussagekräftige Testfälle wurden dem Förderträger beim SelfAutoDOC-Abschlusstreffen am 27.02.2025 demonstriert. Die zugehörigen Analysen und Evaluationen sowie die wissenschaftlichen Beiträge der Universität Siegen wurden in begutachteten Konferenzbeiträgen veröffentlicht.

Die Use Cases sahen u.a. eine Interaktion von Benutzern mittels eines Lenkrades und Pedalen vor, um ein simuliertes Fahrzeug in der CARLA-Umgebung auch manuell steuern zu können. Für die dezentrale Ausführung der in ADNA erstellten Fahrzeuganwendungen wurden RazorMotion-Boards des Projektpartners TTTech auto eingebunden, die die echte Fahrzeughardware bzw. die Steuergeräte darstellen und als ADNA-Prozessesoren fungieren. Weiterhin wurde die FPGA-Hardware des Partners PLC2 eingebunden, um die Hardware-optimierten „Grabber-Tasks“ für Video-Datenverarbeitung zu demonstrieren.

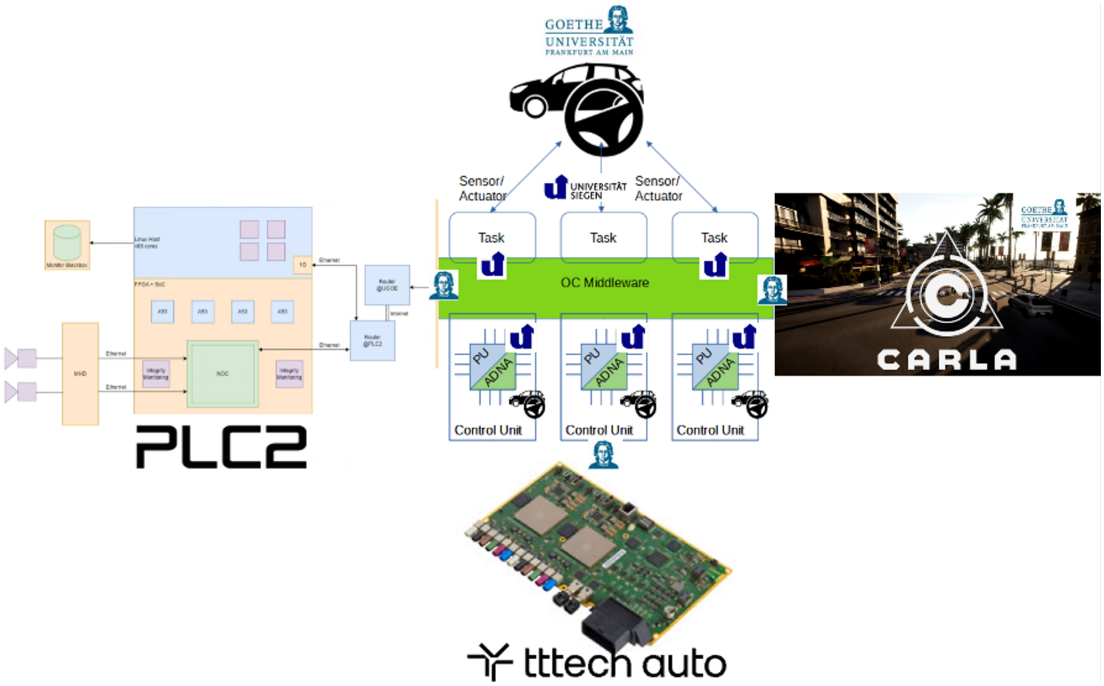

Abbildung 8: Übersicht über den Demonstrationsaufbau

Die folgende Auflistung fasst die gezeigten Demonstrationsfälle zusammen:

1. Organic Computing mit PC-ADNA-Prozessoren und einer manuellen Fahr-ADNA in CARLA mit externer Steuerhardware.

   Dieser Use Case veranschaulicht die Implementierung einer Fahrzeugsteuerung in Form einer künstlichen DNA sowie die Interaktionen der weiterentwickelten Organic-Computing-Middleware mit der Fahrzeugsimulationsumgebung CARLA und der externen Steuerhardware (Lenkrad und Pedale) beispielhaft.

2. Organic Computing mit ADNA-Prozessoren auf einem RazorMotion-Board und verschiedenen ADNAs (e.g. autonomes Fahren, Auto-Pilot) in CARLA.

   Dieser Use Case demonstriert die Einbindung des RazorMotion-Boards von TTTech auto sowie der entsprechenden Kommunikationsschnittstellen mit Ausführung der OC-Middleware und der ADNA-Tasks auf dem Fahrzeugsteuergerät.

3. Fehlerinjektionsframework und Selbstheilungsfähigkeiten bei PE-Ausfällen und Task-Ausfällen (CARLA/DynamicAutoSim).

   Dieser Use Case präsentiert die Möglichkeiten des implementierten Fehlerinjektionsframeworks und veranschaulicht die Selbst-X-Eigenschaften des Organic Computing bei zufälliger Aktivierung verschiedener Fehlerfälle während unterschiedlicher Fahrszenarien in CARLA/DynamicAutoSim.

4. Proaktive OC-Selbst-Heilung basierend auf Diagnoseparameter-Vektor in CARLA/DynamicAutoSim.

   Dieser Use Case fokussiert auf den Diagnoseparameter-Vektor und zeigt die Möglichkeiten der proaktiven Selbst-Heilung auf PE-Ebene auf.

5. Fehlereinstreuung von Hormonfehlern, Fehlerdiagnose und Fehler-Recovery mit Hormon-Guards in CARLA.

   Dieser Use Case führt die um Hormon-Guards erweiterte Organic-Computing-Middleware ein und demonstriert die Fehlerdiagnose sowie die Fehlerbehebung bei Auftreten verschiedener Hormonfehler.

6. Network-on-Chip Integration: Kamara-ADNA auf PLC2-FPGA-System mit hardwareoptimierten *Grabber-Tasks*.

   Dieser Use Case demonstriert das Zusammenspiel von Organic Computing und dem FPGA-System von PLC2 anhand hardwareoptimierter Grabber-Tasks für Videoanwendungen im Fahrzeug am Beispiel einer Lane-Assist-ADNA.

Diese Use Cases vermitteln einen zusammenfassenden Eindruck von den im Projekt SelfAutoDOC entwickelten neuen Fähigkeiten des Organic Computing für sichere Automotive-Anwendungen. Die Universität Siegen hat maßgeblich an den Use Cases 1–5 mitgearbeitet, wobei die Use Cases 3–5 die Projektanteile von USI im Rahmen der OC-Fehlerdiagnose in besonderer Weise hervorheben. Die zugrundeliegenden wissenschaftlichen Analysen und Evaluationen wurden in unseren Projektveröffentlichungen dargelegt und diskutiert: [1, 2, 3, 4, 5, 6, 7, 8].

II.2 Wichtigste Positionen des zahlenmäßigen Nachweises

Die im Projekt eingesetzten Mittel verteilten sich im Wesentlichen auf die folgenden Hauptpositionen:

Personalmittel:
Beantragt: **257.994,09 €**
Verausgabt: **271.858,66 €**

Der größte Anteil der Zuwendung entfiel auf Personalkosten für zwei wissenschaftliche Mitarbeiter (Doktoranden), die über die gesamte Projektlaufzeit hinweg kontinuierlich am Vorhaben gearbeitet haben. Weitere wissenschaftliche Mitarbeiter wurden zeitweise zur Unterstützung einzelner Arbeitspakete eingesetzt. Im Rahmen der genehmigten, kostenneutralen Projektverlängerung um drei Monate wurden nach Absprache mit dem Förderträger nicht genutzte Sachmittel in zusätzliche Personalmittel umgewandelt, um die o.g. Mitarbeiter für die zusätzlichen Arbeiten länger finanzieren zu können.

Reisemittel:
Beantragt: **6.900,00 €**
Verausgabt: **9.133,64 €**

Aufgrund der hohen Zahl an qualitätsgesicherten Veröffentlichungen wurden zusätzliche Mittel für die Teilnahme an nationalen und internationalen Fachkonferenzen zur Präsentation der Projektergebnisse verwendet. Die erhöhten Ausgaben dienten insbesondere der wissenschaftlichen Dissemination und dem gezielten Austausch mit Fachkolleginnen und -kollegen durch die im Projekt beschäftigten Mitarbeiter. Im Gegenzug wurden projektinterne physische Treffen reduziert und vermehrt virtuell durchgeführt.

Sachmittel (Equipment):
Beantragt: **22.812,19 €**
Verausgabt: **0 €**

Die ursprünglich geplanten Anschaffungen von Hardware-Equipment konnten durch Marktveränderungen (Lieferengpässe, Preisanstiege insb. während der Pandemiezeit) nicht wie geplant realisiert werden. Stattdessen wurde auf simulative Ansätze und (Open-Source-) Softwaretools zurückgegriffen (u.a. CARLA, OMNet++), die projektintern weiterentwickelt wurden. Die hierfür vorgesehenen Mittel wurden im Zuge der Projektverlängerung teilweise in Personalmittel umgewidmet.

Gesamtkosten:
Beantragt: **287.706,28 €**
Verausgabt: **280.992,30 €**

Die Mittelverwendung entsprach in Summe weitgehend den Zuwendungsbedingungen und wurde projektadäquat angepasst, um die wissenschaftlichen Ziele und die Qualitätssicherung der Projektergebnisse sicherzustellen. Weiterführende Details finden sich in den eingereichten Verwendungsnachweisen der Universität Siegen.

II.3 Notwendigkeit und Angemessenheit der geleisteten Arbeit

Die im Projekt SelfAutoDOC geleistete Arbeit war sowohl notwendig als auch in ihrer Ausgestaltung und ihrem Umfang angemessen, um die angestrebten Ziele zu erreichen. Die Entwicklung hochzuverlässiger, fehlertoleranter Fahrzeugarchitekturen erfordert neue Ansätze jenseits klassischer Redundanzmechanismen. Die Kombination von Organic-Computing-Prinzipien mit aktiver Diagnose und Selbstheilung stellt hierfür einen innovativen und vielversprechenden Lösungsweg dar. Die wissenschaftliche Bearbeitung der dafür erforderlichen Konzepte, darunter verteilte Steuergerätearchitekturen, künstliche DNA, hormonbasierte Taskverteilung sowie integrierte Selbstdiagnose und -rekonfiguration, erforderte ein hohes Maß an interdisziplinärer Systemkenntnis, formaler Modellierung und simulativer Evaluierung. Die kontinuierliche Arbeit zweier wissenschaftlicher Mitarbeiter sowie die Unterstützung durch weitere Teammitglieder erlaubten eine tiefgehende Bearbeitung dieser Aspekte. Die entstandenen Veröffentlichungen, Algorithmen und Demonstratoren zeigen, dass die Projektarbeit zielgerichtet, forschungsadäquat und im Hinblick auf zukünftige Anwendungen im Automobilbereich hochrelevant war. Die 39-monatige Projektlaufzeit ermöglichte den beiden Doktoranden die Anfertigung ihrer Dissertationen im Rahmen ihrer wissenschaftlichen Weiterqualifizierung. Beide Arbeiten werden voraussichtlich im laufenden Jahr abgeschlossen.

Die Projektergebnisse werden über die Laufzeit hinaus weiterwirken. Die Industriepartner TT-Tech Auto und PLC2 beabsichtigen, die in SelfAutoDOC gemeinsam entwickelten Konzepte und Mechanismen in zukünftigen Versionen ihrer Produkte, insbesondere des RazerMotion-Boards als Fahrzeugsteuergerät, weiterzuverwerten. Seitens der Universität Siegen als einem der akademischen Partner fließen die im Projekt erarbeiteten Inhalte systematisch in die Lehre ein. Sie dienen als Grundlage für neue Lehrmodule bzw. als Anreicherung bestehender Veranstaltungen im Bereich fehlertoleranter eingebetteter Systeme und Organic Computing. Darüber hinaus bilden sie die Basis für studentische (Abschluss-) Arbeiten und die wissenschaftliche Qualifizierung des Nachwuchses. Der Wissenstransfer in Ausbildung und Praxis ist somit nachhaltig sichergestellt.

II.4 Voraussichtlicher Nutzen und Verwertbarkeit der Ergebnisse

Die im Projekt SelfAutoDOC entwickelten Konzepte, Methoden und Werkzeuge bieten ein erhebliches Verwertungspotenzial sowohl für die wissenschaftliche Community als auch für die industrielle Praxis.

Die beteiligten Universitäten Siegen und Goethe-Universität Frankfurt planen, die Ergebnisse unmittelbar in die akademische Lehre und Forschung zu integrieren, sowie für (gemeinsame) Nachfolgeprojekte einzusetzen. Die entwickelten Konzepte des Organic Computings, wie etwa künstliche DNA (ADNA), künstliche Hormonsysteme (AHS) und selbstorganisierende Diagnose- und Selbstheilungsmechanismen, werden in Lehrveranstaltungen zu eingebetteten Systemen, autonomen Fahrzeugsystemen und Systemzuverlässigkeit eingesetzt. Dies motiviert Studierende für innovative Themen an der Schnittstelle von Informatik, Elektrotechnik und up-to-date Automotive-Technologie und qualifiziert sie für eine spätere Tätigkeit in der Industrie.

Konkret fließen auch die Methoden der formalen Verifikation (z.B. mit UPPAAL) und die entwickelten Simulations- und Co-Simulationswerkzeuge (z.B. für TSN und OMNeT++) in studentische Laboraufgaben und Arbeiten ein, wie im Projektverlauf bereits durch mehrere Abschlussarbeiten demonstriert wurde. Dadurch werden Studierende aktiv in aktuelle Forschung eingebunden, wissenschaftlich geschult und für künftige F&E-Vorhaben im Bereich Organic Computing.

Die industriellen Projektpartner TTTech Auto und PLC2 Design werden auf den prototypischen Implementierungen aufbauen und die im Projekt entwickelten Technologien in zukünftigen Produktgenerationen vermarkten. Insbesondere die im Projekt realisierten Konzepte zur autonomen Fehlerdiagnose, zur selbstorganisierenden Rekonfiguration im Fehlerfall sowie zur Integration von Self-X-Eigenschaften in die Fahrzeug-(Kommunikations-)Architekturen und Steuergeräte von morgen ermöglichen einen Paradigmenwechsel in der Entwicklung sicherheitskritischer Fahrzeugsystemen.

Durch die Nutzung von OC-Ansätzen wird die bisher erforderliche Hardware-Redundanz reduziert und durch softwarebasierte Fehlertoleranz ersetzt werden, was sowohl Kosten spart als auch die Flexibilität erhöht. TTTech kann die entwickelten Architekturen mit dynamischer TSN-Rekonfiguration direkt in seine Plattformen für automatisiertes Fahren einbringen, während PLC2 Design die entwickelten Hormon-Guards und deren Middleware-Komponenten in die Entwicklung skalierbarer SoC-Plattformen einfließen lassen kann.

Das Projekt liefert damit einen signifikanten Beitrag zur industriellen Umsetzung hochverfügbarer, fail-operational Fahrzeugsysteme mit reduziertem Entwicklungsaufwand und gestiegener Adaptivität, was ein entscheidender Wettbewerbsvorteil für die beteiligten Unternehmen im stark wachsenden Markt für autonome Fahrzeuge ist.

II.5 Während der Durchführung des Vorhabens dem ZE bekannt gewordenen Fortschritts auf dem Gebiet des Auftrags bei anderen Stellen

Uns sind keine Ergebnisse oder Arbeiten Dritter bekannt, die den erfolgreichen Abschluss des Projekts negativ beeinflusst haben bzw. einer weiteren Verwertung der Ergebnisse durch die Industriepartner entgegenstehen.

Der Fortschritt auf dem Gebiet des Einsatzes von Organic Computing im automobilen Kontext wurde im Rahmen des Verbundvorhabens durch regelmäßigen Austausch mit den Industriepart-

nern, sowie durch kontinuierliche Recherchen des wissenschaftlichen Umfelds durch die Universitäten Siegen und Frankfurt bewertet. Alle Projektpartner haben aktuelle Marktbedarfe, Trends und Technologieentwicklungen in das Projekt eingebracht, wodurch relevante externe Entwicklungen unmittelbar beachtet werden konnten. Zur Projektlaufzeit wurden auch keine konkurrierenden Forschungsaktivitäten identifiziert, die den angestrebten Zielen oder dem Innovationsgrad des Vorhabens entgegengestanden hätten. Vielmehr hat die fortlaufende Analyse des Stands der Technik die hohe Relevanz des entwickelten Ansatzes zur Kombination von Organic Computing mit deterministischer Kommunikation (z.B. Time-Sensitive Networking) hervorgebracht, insbesondere im Hinblick auf Selbstheilung, Fehlertoleranz und Echtzeitfähigkeit sicherheitskritischer Systeme (siehe auch Zwischenberichte der Universität Siegen; u.a. beschriebene Forschungsbedarfe).

II.6 Veröffentlichungen der Ergebnisse der Universität Siegen

[1] S. Nahar, S. Meckel und R. Obermaisser. „Fault Injection Framework for Organic Computing Architecture“. In: *Intelligent Systems and Applications: Proceedings of the 2023 Intelligent Systems Conference (IntelliSys)*. Springer. 2023.

[2] M. Qosja, S. Meckel und R. Obermaisser. „Simulator for Time-Triggered Organic Computing“. In: *Procedia Computer Science* 220 (2023).

[3] U. Raj, S. Meckel, A. Koschowoj, M. Pacher, R. Obermaisser und U. Brinkschulte. „Self-adaptive diagnosis and reconfiguration in adna-based organic computing“. In: *International Conference on Architecture of Computing Systems*. Springer. 2023.

[4] M. Qosja, S. Meckel und R. Obermaisser. „Time-Triggered Organic Computing Architecture for Autonomous Driving Vehicles Using List Scheduling“. In: *AmEC 2024–Automotive meets Electronics & Control; 14. GMM Symposium*. VDE. 2024.

[5] S. Nahar, U. Raj, S. Meckel und R. Obermaisser. „Enhancing reliability in organic computing using hormone guard“. In: *2024 13th Mediterranean Conference on Embedded Computing (MECO)*. IEEE. 2024.

[6] S. Nahar, U. Raj, S. Meckel und R. Obermaisser. „Application Level Fault Diagnosis in Organic Computing Using Stress Hormone Propagation“. In: *2024 8th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE. 2024.

[7] M. Qosja, S. Meckel und R. Obermaisser. „Discrete-Event Co-Simulation Interface for Time-Triggered Organic Computing“. In: *2024 IEEE 12th International Conference on Intelligent Systems (IS)*. IEEE. 2024.

[8] M. Qosja, U. Raj, S. Meckel und R. Obermaisser. „Dynamic TSN Reconfiguration for Time-Triggered Organic Computing“. In: *Procedia Computer Science* (2025).

Literatur

[9] U. Brinkschulte, M. Pacher und A. von Renteln. „An Artificial Hormone System for Self-Organizing Real-Time Task Allocation in Organic Middleware“. In: *Organic Computing*. Springer, 2008.

[10] U. Brinkschulte. „An artificial DNA for self-descripting and self-building embedded real-time systems“. In: *Wiley Journal on Concurrency and Computation: Practice and Experience* (2015).

[11] U. Brinkschulte. „Prototypic implementation and evaluation of an artificial DNA for self-descripting and self-building embedded systems“. In: *Springer EURASIP Journal on Embedded Systems* (2017).

[12] U. Brinkschulte, R. Obermaisser, S. Meckel und M. Pacher. „Online-Diagnosis with Organic Computing based on Artificial DNA“. In: *1st IEEE International Conference on Societal Automation, SA 2019, Krakow, Poland*. 2019.

[13] R. Obermaisser. *Time-triggered communication*. CRC Press, 2011.

[14] M. Pahlevan und R. Obermaisser. „Genetic Algorithm for Scheduling Time-Triggered Traffic in Time-Sensitive Networks“. In: *23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE. 2018.

[15] R. Obermaisser, H. Ahmadian, A. Maleki, Y. Bebawy, A. Lenz und B. Sorkhpour. „Adaptive Time-Triggered Multi-Core Architecture“. In: *Designs, MDPI* 3.1 (2019).

[16] R. Obermaisser, R. I. Sadat und F. Weber. „Active diagnosis in distributed embedded systems based on the time-triggered execution of semantic web queries“. In: *17th Int. Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*. IEEE. 2014.

[17] U. Brinkschulte, R. Obermaisser, S. Meckel und M. Pacher. „Online-Diagnosis with Organic Computing based on Artificial DNA“. In: *2019 First International Conference on Societal Automation (SA)*. IEEE. 2019.