technische universität dortmund

fakultät für informatik

Fakultät für Informatik
Lehrstuhl für praktische Informatik
Arbeitsgruppe Rechnernetze und verteilte Systeme (RvS)

RECHTZEITIGKEIT UND DIENSTGÜTEFÖRDERUNG VERNETZTER GERÄTEBASIERTER SYSTEME

Dissertation
zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)

vorgelegt von
Dipl.-Inform. Henning Brümmer
geboren am 29.04.1985 in Dortmund

Dortmund, 2021

Tag der mündlichen Prüfung:	Dortmund, 27. Januar 2022
Dekan:	Prof. Dr.-Ing. Gernot A. Fink
Vorsitzender:	Prof. Dr. Peter Buchholz
Erstgutachter:	Prof. Dr. Heiko Krumm
Zweitgutachter:	Prof. Dr. Peter Ulbrich
Viertes Mitglied:	Priv.-Doz. Dr. habil. Frank Weichert

KURZFASSUNG

Hochzuverlässige Systeme müssen sorgfältig entwickelt und gründlich geprüft werden. Sie erbringen sicherheitskritische Funktionen und unterliegen erhöhten Zuverlässigkeitsanforderungen. Eine statische Konfiguration und dedizierte Hardware machen die Verlässlichkeit dieser Systeme möglich. Man spricht auch von sogenannten Insellösungen; d.h. die Systeme sind in sich geschlossen und werden für einen ganz bestimmten Zweck entwickelt. Medizingeräte bilden zum Beispiel solche Systeme. Da diese Systeme für ihre Umgebung aber geschlossen sind, ist eine Fernüberwachung oder eine Automatisierung der Gerätekonfiguration in der Regel nicht möglich. Dies wäre aus Sicht der Verlässlichkeit auch eine potenzielle Gefahrenstelle. Eine ganz andere Systemklasse bilden die vernetzen gerätebasierten Systeme. Diese sind hochdynamisch und können durch die Vielfalt von Geräten, Sensoren, Kommunikationsdiensten, Cloud-Infrastrukturen und Big-Data-Analyseverfahren einen hohen Funktionsumfang anbieten. Sie unterliegen in der Regel jedoch einer begrenzten Verlässlichkeit. Die Fitnessbranche zum Beispiel sammelt und überwacht mittels IoT-Geräten wie Smartphones, Smartwatches und intelligenter Waagen die Gesundheitsparameter ihrer Anwender. Die Frage, die sich nun in der immer stärker wachsenden Welt der vernetzten Systeme stellt, ist, wie Informationen und neu gewonnene Erkenntnisse aus den unverlässlichen vernetzten Systemen auch im Umfeld der zuverlässigen Systeme genutzt werden können.

Die vorliegende Dissertation untersucht die systematische und effiziente Entwicklung von Big Dependable Systems (BDS) als Zusammenführung von nicht verlässlichen vernetzten Systemen und verlässlichen Systemen. Im Rahmen dieser Arbeit wird ein Lösungsansatz entwickelt, der die Verlässlichkeit von BDS unter der Verwendung von unzuverlässigen vernetzten Konsumergeräten weiterhin gewährleistet. Als wichtigste Verlässlichkeitsanforderungen gelten die Funktionssicherheit sowie die Rechtzeitigkeit von BDS. Schwerpunkt der Untersuchung war das Einhalten der Rechtzeitigkeit und Dienstgüte in solch vernetzten gerätebasierten Systemen. Als erstes wurden Fehlertoleranzmuster kombiniert und angepasst, um die Verlässlichkeitsanforderungen in BDS zu erfüllen. Des Weiteren wurden Ansätze untersucht, wie BDS modellbasiert entwickelt und modellbasiert analysiert werden können.

Als Proof-of-Concept wurde ein medizinischer Anwendungsfall untersucht und ein medizinisches BDS am Beispiel des Linksherzunterstützungssystems (LVAD) entwickelt und umfangreich am LVAD-Demonstrator evaluiert. Der Fokus der experimentellen Bewertung lag auf der Vermessung der zeitlichen Ende-zu-Ende Verzögerung und der Dienstgüte der verwendeten Architekturkonzepte sowie der Fehlertoleranzmechanismen.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

1 EINLEITUNG

Die Möglichkeit Informationen aus vernetzten gerätebasierten Systemen auch im Umfeld von sicherheitskritischen Systemen (SkS) zu nutzen, stellt die Informationstechnik (IT) vor eine große Herausforderung. Auf welche Weise können die Verlässlichkeit bzw. die funktionale Korrektheit sowie die Realzeitbedingungen weiterhin gewährleistet sein, obwohl ein unzuverlässiges Geräteensemble von Konsumergeräten Informationen aus der Umwelt erfasst und aufbereitet, um Steuerungssignale für die SkS bereitzustellen.

Dazu wird eine neue Systemklasse die Big Dependable Systems, kurz BDS, entwickelt. In dieser sollen nicht-perfekte Geräte sicherheitskritische Funktionen für verlässliche Systeme bereitstellen.

Im Folgenden wird die Motivation der Arbeit dargestellt und anschließend die Problemstellung und Zielsetzung anhand eines medizinischen Anwendungsfalls erläutert. Im Weiteren werden die gewonnenen Ergebnisse der Forschungsarbeit kurz beschrieben. Abschließend wird der strukturelle Aufbau der Arbeit dargestellt.

1.1 Motivation

Die fortschreitende Miniaturisierung im Digitalsektor sowie die hohe Verbreitung der Sensorik im Konsumentenbereich ermöglichen eine immer bessere automatisierte Überwachung und Steuerung von informationstechnischen Systemen.

Vernetzte gerätebasierte Systeme bzw. das Internet der Dinge, kurz IoT, machen sich diese Eigenschaften bereits zunutze. Mithilfe von unterschiedlichen Sensoren, Cloud-Infrastrukturen, Big-Data-Analysen und mobilen Kommunikationstechnologien können Informationen bzw. Daten der physischen Umwelt gesammelt und geteilt werden. In dieser stark vernetzten Umgebung können digitale Systeme diese Informationen/Daten überwachen und gegebenenfalls anpassen [1].

Abbildung 1 zeigt, wie sich die Anzahl der im IoT vernetzten Geräte in den letzten Jahren entwickelte. Eine wichtige Tatsache ist, dass ein großer Teil der vernetzten Geräte aus Konsumergeräten besteht. Smartphones, Smartwatches, Fitnessarmbänder und smarte Haushaltsgeräte sind Beispiele für diese Gruppe von Konsumergeräten im IoT-Verbund und werden immer häufiger in der Gesellschaft akzeptiert [2]. Neben der Verbreitung spielen aber auch deren hochwertige Sensorik (Bewegungssensoren,

Standortsensoren, Pulssensoren) und vielfältige Kommunikationsdienste (WLAN, Bluetooth, UMTS, LTE oder 5G) eine wesentliche Rolle [1, 3].

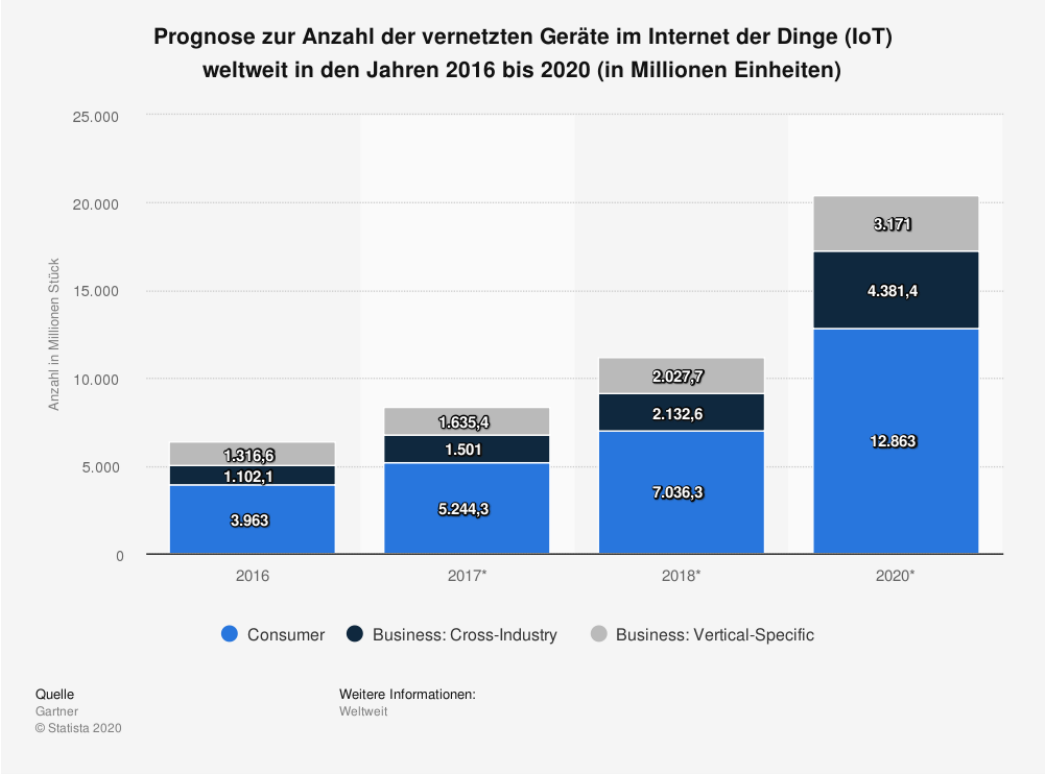


Abbildung 1 - Prognose für die Anzahl der vernetzten Geräte im Internet der Dinge[4]

Diese Möglichkeit, Informationen einfach und kostengünstig zu sammeln, macht den Einsatz von vernetzten Geräten in der Wirtschaft so erfolgreich. Das spiegelt auch der Ansatz der „Industrie 4.0“ wieder [1, 5].

In folgenden Wirtschaftszweigen liefern vernetzte Geräte-basierte Systeme schon jetzt zum Teil einen großen qualitativen Mehrwert:

- Automobilindustrie,
- Telekommunikation,
- intelligente Gebäude,
- Gesundheitswesen.

Diese vernetzten Infrastrukturen unterliegen einer dynamischen Konfiguration, sind nur eingeschränkt verlässlich (engl. dependable) und dienen größtenteils Überwachungs- und Komfortfunktionen [3].

Einige Anwendungsbereiche wie zum Beispiel die Automobilindustrie oder die Medizintechnik unterliegen zusätzlich einer erhöhten Verlässlichkeit und bilden in sich isolierte SkS. Ein Hauptgrund, warum solche Systeme sicherheitskritisch sind und eine hohe Verlässlichkeit aufweisen, ist der Sachverhalt, dass sie direkt mit ihrer Umwelt interagieren und einen unmittelbaren Einfluss auf diese haben [5–7].

Am Beispiel eines selbstbremsenden Fahrzeugs tasten Kamerasensoren und Infrarotsensoren die Umgebung auf Gegenstände ab, die sich im Frontbereich des Fahrzeugs befinden. Diese Umgebungsinformationen werden unter Umständen mit weiteren Daten wie der aktuellen Geschwindigkeit software-basiert ausgewertet und im Notfall wird ein Signal an die Bremse weitergeleitet. Diese wird automatisiert betätigt und bringt das Fahrzeug sicher zum Stehen. Ein solches System ist im gesamten Fahrzeugsystem eingebettet und unterliegt harten Sicherheitsanforderungen. Ein Fehlverhalten in der Verarbeitung oder ein Ausfall kann katastrophale Folgen für den Menschen und seine Umgebung zur Folge haben.

Vorwiegend werden solche hoch verlässlichen SkS mit dedizierter Hardware entwickelt und sind statisch konfiguriert. Solche Systeme werden für eine bestimmte Anwendung entworfen und sind in sich geschlossen. Neben der Verlässlichkeit müssen SkS auch effizient arbeiten und Realzeitbedingungen erfüllen. Diese Charakteristik ist eine völlig andere als bei IoT-Systemen [5, 7].

Die Möglichkeiten und Herausforderungen sind, wie man die Masse an Daten, die in den wachsenden IoT-Systemen gesammelt werden, auch im Kontext solcher SkS verwenden kann. Die Aufgabe besteht in der Verknüpfung unsicherer IT-Dienste aus offenen vernetzten unzuverlässigen Systemen mit geschlossenen verlässlichen SkS. Die Anforderungen an solch entwickelte BDS ist, sicherheitskritische Funktionen auf unzuverlässige Geräte auszulagern. Zielsetzung hierbei ist, die Verlässlichkeit insbesondere die Rechtzeitigkeit und funktionale Korrektheit weiterhin zu gewährleisten.

1.2 Problemstellung und Zielsetzung

In Deutschland starben im Jahr 2017 insgesamt 932272 Menschen. Die häufigste Todesursache war eine Herz-/Kreislauferkrankung. 37,0% aller Sterbefälle waren auf diese Krankheit zurückzuführen [8].

Unter Herz-/Kreislauferkrankungen bildet die Herzinsuffizienz in Deutschland die dritthäufigste Todesursache [9]. Insgesamt starben 38187 Menschen an dieser Krankheit. Zudem bildet die Herzinsuffizienz mit 455700 Behandlungsfällen den zweithäufigsten Anlass für eine stationäre Versorgung im Krankenhaus nach Geburten [10].

Ist die Herzinsuffizienz zu weit vorgeschritten und nicht mehr mittels Pharmakotherapie heilbar, wird die Möglichkeit einer Herztransplantation (HTX) in Betracht gezogen. Das Problem, welches sich aber ergibt, ist, dass der Bedarf an Spenderherzen das Angebot der verfügbaren Transplantate bei Weitem übersteigt. Diese Schere von Angebot und Nachfrage für HTX wird zunehmend größer. 2019 waren in Deutschland 707 Patienten für eine HTX aktiv gelistet. Zur Verfügung standen jedoch im selben Jahr nur 333 Spenderherzen [11, 12].

Um die Wartezeit auf ein Spenderorgan zu überbrücken, werden Herzunterstützungssysteme (engl. ventricular assist devices - VAD) implantiert. Was zu Beginn eine Übergangslösung darstellen sollte, hat sich aktuell zu einer dauerhaften Therapie gegen Herzinsuffizienz entwickelt [13]. Ein Grund dafür ist auch die kontinuierliche Weiterentwicklung von VAD-Systemen.

Abbildung 2 - Größenentwicklung von LVADs in den letzten Jahren[14]

Abbildung 2 zeigt die Miniaturisierung von Linksherzunterstützungssystemen (LVAD) [14]. Durch ein verbessertes Management und eine kontinuierliche Kontrolle sind typische Probleme wie Thrombenbildung, Blutungen, Driveline-Infektionen und neurologische Komplikationen in der stationären Betreuung stark reduziert worden [14]. Anders sieht dies jedoch in der post-stationären Betreuung aus. Auch nach der Entlassung müssen die LVAD-Parameter kontinuierlich überwacht werden, um eine langfristige Betriebsfunktionalität eines LVADs zu gewährleisten. Doch diese umfangreiche Überwachung ist nach der Entlassung aktuell nicht möglich [15]. Von Seiten der Medizin werden daher ein umfassendes Monitoring der LVAD-Parameter sowie weitere Informationen über die Verfassung des Patienten gefordert.

Die aktuelle Problematik ist, dass das LVAD-System als SkS ein abgeschlossenes Medizinprodukt ist. Es gibt keine Möglichkeit von außen dem System zu sekundieren oder Daten vom LVAD in die äußere Umgebung zu übertragen. Damit können das medizinische Fachpersonal oder die einzelnen Herzzentren nicht aus der Ferne auf die LVAD-Parameter zugreifen, um im Falle eines Problems frühzeitig einzugreifen. Des Weiteren wäre eine adaptive Steuerung des LVAD-Systems interessant, um neuartige Optimierungen und Funktionserweiterungen zu ermöglichen. Typische Probleme wie die Thrombenbildung oder die Aortenklappenpathophysiologie wären zu vermeiden.

Diese Möglichkeiten würden auf der einen Seite dem Patienten ein erhöhtes Sicherheitsgefühl vermitteln und zum anderen können Kosten, wie permanente Aufenthalte in Krankenhäusern sowie vermeidbare Operationen, eingespart werden.

Eine verlässliche BDS-gestützte Überwachung und Steuerung von LVAD-Systemen ist von aktuellem Interesse für eine Anschlussheilbehandlung und allgemeine Rehabilitation von Patienten mit einem VAD.

Ziel dieser Arbeit ist es, eine neue Systemklasse zu entwickeln, die eine Verschmelzung von IoT und SkS ermöglicht. Diese neue Systemklasse wird im Laufe der Arbeit als Big Dependable System, kurz BDS, beschrieben. Vernetzte Geräte-basierte Systeme und SkS sollen nicht nebeneinander getrennt existieren, sondern miteinander interagieren. Unzuverlässige Konsumergeräte sollten einen essenziellen Beitrag für kritische Funktionen in SkS leisten können, um eine Verbesserung und Erweiterung des Funktionenumfangs im Gesamtsystem zu ermöglichen.

Die Herausforderung bei der Entwicklung von BDS liegt in der Gewährleistung der Verlässlichkeit und der Qualität des Softwareprodukts. Nach ISO/IEC 9126 [16] wird die Qualität eines Softwareprodukts an folgenden sechs Qualitätsmerkmalen gemessen:

- Wartbarkeit,
- Effizienz,
- Übertragbarkeit,
- Zuverlässigkeit,
- Funktionalität,
- Benutzbarkeit.

Dies verdeutlicht, dass für ein softwarebasiertes Steuerungssystem, wie es ein BDS ist, unter anderem wichtig ist, dass Anforderungen wie die funktionale Korrektheit oder die Rechtzeitigkeit eingehalten werden. Des Weiteren müssen die Maßnahmen, die ergriffen werden, einen qualitativen Mehrwert für das Gesamtsystem haben und effizient zusammenarbeiten. Betrachtet man zum Beispiel die funktionale Korrektheit, so wird diese durch eine erhöhte Redundanz und einen Endscheidungsprozess nach dem Prinzip M-aus-N unterstützt. Wohingegen die Anforderung an die Rechtzeitigkeit ebenfalls auf Redundanz zurückgreift aber beim Endscheidungsprozess nach dem Prinzip 1-aus-N handelt. Dieser Gegensatz bzw. die Konkurrenz zwischen verschiedenen Mechanismen und Eigenschaften benötigt einen wohldefinierten Entwicklungsansatz. Dies stellt eine große Herausforderung an die Entwicklung von BDS dar. Des Weiteren dürfen die eingesetzten Mechanismen sich nicht gegenseitig behindern und müssen weiterhin eine Dienstgüte liefern, die einen funktionalen Mehrwert erbringen. Der Aspekt der funktionalen Korrektheit im Zusammenhang mit BDS wird in einer separaten Dissertation von Herrn Egor Kudrjaschow erforscht und dargestellt.

Schwerpunkt der vorliegenden Dissertation ist die Untersuchung, Entwicklung und Anpassung geeigneter Architekturkonzepte, Fehlertoleranzverfahren und Sicherheitsmechanismen. Passende Entwurfs- und Analyseverfahren für die Rechtzeitigkeit und die Dienstgüteförderung solcher BDS sollen hier geliefert werden.

So könnte ein zukünftiger Zertifizierungsprozess und Einsatz solcher BDS ermöglicht werden.

1.3 Arbeiten und Ergebnisse

Die vorliegende Dissertation ist in Zusammenhang mit dem internationalen Forschungsprojekt "Medical care evolution (Medolution)" entstanden, welches vom Bundesministerium für Bildung und Forschung (BMBF) gefördert wurde[17].

Um die zeitlichen Eigenschaften und die funktionale Korrektheit der BDS-gestützten Datenerfassung und Steuerungsfunktionen zu untersuchen, wurde ein Proof-of-Concept-Demonstrator entwickelt, der die Überwachung und Steuerung von LVADs über unzuverlässige mobile Endgeräte ermöglicht. Es wurde der Entwurfsansatz der modellbasierten Entwicklung mittels AADL gewählt. Dieser ermöglicht es, Verlässlichkeitsanalysen von software-basierten Systemen schon frühzeitig im Entwicklungsprozess durchzuführen. Insbesondere die Entstehung und Ausbreitung von BDS-typischen Fehlern und deren Auswirkung auf das Gesamtsystem wurden genauer untersucht.

Es wurde ein Duplex-Smartphone-System entwickelt, welches in der Lage ist, die Aktivität des Anwenders zu erkennen. Im Falle einer erkannten Ruhephase steuert das Duplex-System das LVAD und leitet einen Aortenklappenspülzyklus ein, indem es periodisch die Umdrehungsgeschwindigkeit der LVAD-Pumpe herabsetzt. Bei erkannter Aktivität wird diese neue Funktionalität beendet und das LVAD-System läuft mit Normalgeschwindigkeit weiter. Hierzu wurden folgende Entwicklungsphasen durchlaufen:

- Es wurde eine Möglichkeit erarbeitet, das LVAD für seine äußere Umgebung zu öffnen. Es wurde der Nachrichtenfluss der originalen LVAD-Steuereinheit untersucht und die Struktur des Nachrichtenaufbaus rekonstruiert. Durch einen zusätzlichen RaspberryPi, der an die LVAD-Steuereinheit angeschlossen wurde, ist eine Bluetooth-Schnittstelle vorhanden, die eine beidseitige Kommunikation zulässt. Relevante LVAD-Parameter wie der Durchfluss, Alarmsignale, der Energieverbrauch der Pumpe, die Drehzahl der Pumpe sowie der aktuelle Akkuzustand der angeschlossenen Batterien können von der äußeren Umgebung erfasst, verarbeitet und angepasst werden.
- Eine software-basierte Androidanwendung wurde entwickelt, um eine Überwachung der LVAD-Parameter und das Steuern der Pumpengeschwindigkeit über die neue Bluetooth-Schnittstelle zu gewährleisten.
- Es wurde eine software-basierte Aktivitätserkennung mittels eines künstlichen neuronalen Netzes entwickelt. Diese verwendet die interne Sensorik der Smartphones (Accelerometer- und Gyroskopsensor), um eine kontinuierliche Aktivitätserkennung des Anwenders zu ermöglichen.

- Ein dynamischer Betriebsmoduswechsel wurde entwickelt, der es ermöglicht, zwischen verschiedenen Betriebsmodi zu wechseln. Es wurden drei Betriebsmodi entwickelt:
  - Betriebsmodus Duplex-Smartphone (optimiert),
  - Betriebsmodus Single-Smartphone (degradiert),
  - Betriebsmodus Fall-Back (fail-safe).
- Parallel zu der Entwicklung der Softwarelösung wurde das System sukzessive mittels AADL modelliert und analysiert. Anpassungen der Systemarchitektur durch passende Fehlertoleranzmuster und Sicherheitsmechanismen wurden in den einzelnen Entwicklungszyklen mittels Fehlerbaumanalysen und Fehlerauswirkungsanalysen ausgewertet.
- Das entwickelte System wurde unter dem Aspekt der Rechtzeitigkeit und der Dienstgüte gründlich evaluiert. Neben Fehlerinjektionen zur Verifizierung der Fehlertoleranzmechanismen wurden auch die Reaktionszeit des Systems unter verschiedenen CPU-Belastungen sowie die Verzögerungszeit aktueller Bluetoothverbindungen untersucht.

Die folgenden geeigneten Architekturkonzepte und Fehlertoleranzmechanismen machen in Kombination einen Einsatz von BDS in Zukunft möglich:

- kleiner Perfektionskern (PK) mit Fall-Back-Funktion,
- dezentrale Datenverarbeitung am Rand der IoT (engl. edge computing),
- Robuster-Software-Entwurf (engl. resilient software design),
- allmählicher Funktionsabbau (engl. graceful degradation),
- Strukturelle-, Funktionale-, und Zeitliche-Redundanz.

Abbildung 3 stellt die Abhängigkeiten sowie den Zusammenhang der einzelnen Konzepte und Muster in BDS dar. Die gelb hinterlegten Elemente führen die nicht-sicherheitskritischen Funktionen eines BDS durch. Diese Systemklasse dient unter anderem der Überwachung von technischen Vorgängen, der Datenverarbeitung aus den vernetzten IoT-Geräten und stellt die Cloud-basierte Analyse von Big Data zur Verfügung. Diese Cloud-basierte Funktionalität unterliegt keiner erhöhten Verlässlichkeit. Diese Systemklasse (insbesondere die automatisierte Überwachung und Reparatur von BDS) wird durch geeignete Verwaltungs- und Konfigurationskonzepte in der Dissertation von Anna Litvina beschrieben [18]. Die zweite wichtige Systemklasse in BDS stellen die gerätebasierten Systeme dar und dienen als Grundlage der vorliegenden Dissertation. Die in Abbildung 3 rot hinterlegten Elemente führen sicherheitskritische Funktionen eines BDS durch. Sie beinhalten Fehlerverwaltungsmechanismen, die einer zeitlichen Begrenzung ausgesetzt sind. Sie interagieren direkt mit SkS und unterliegen daher einer erhöhten Verlässlichkeit. Der robuste Systemaufbau ermöglicht es Fehler frühzeitig im System zu erkennen, um diese zu isolieren und zu behandeln. Verschiedene Betriebsmodi machen es möglich, zwischen den IoT-Geräten zu wechseln und im schlimmsten Fall in den sicheren Fall-Back-Modus des SkS zu wechseln.

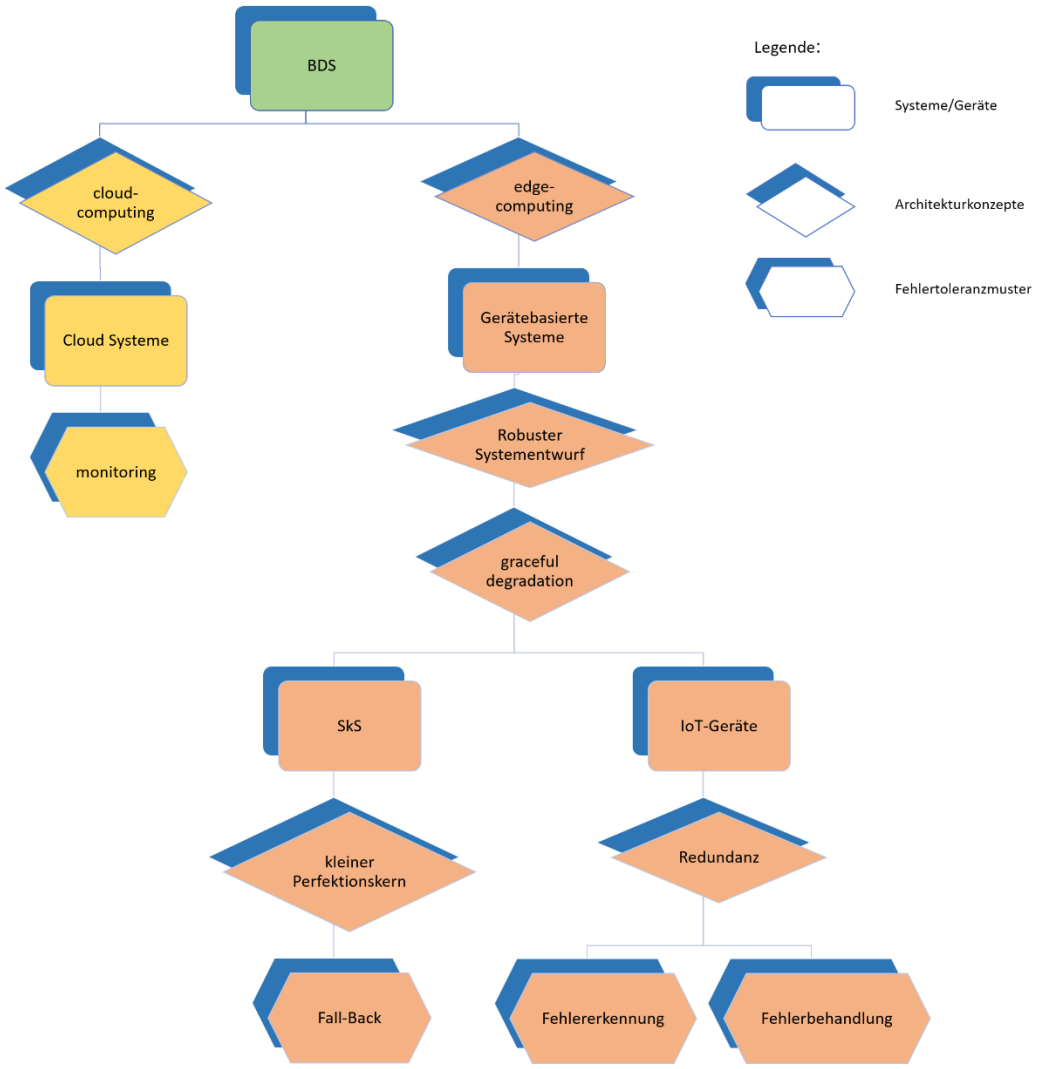


Abbildung 3 - Zusammenhang und Abhängigkeiten der verwendeten Architekturkonzepte und -muster in BDS

Um die Rechtzeitigkeit in dem entwickelten gerätebasierten LVAD-BDS zu untersuchen und die angewandten Fehlertoleranzkonzepte zu bewerten, wurden die Systemmodelle der einzelnen Entwicklungsphasen mittels Fehlerbaumanalyse (FTA) ausgewertet. Dazu wurde das Auftreten eines nicht rechtzeitigen Stellsignals am Aktuator für die einzelnen Systemmodelle untersucht. Tabelle 1 zeigt die Ergebnisse der einzelnen Systemmodelle des entwickelten LVAD-BDS. Diese Systemstrukturen wurden durch AADL analysiert und ergeben folgende Auftrittswahrscheinlichkeiten für ein *NoValue*-Ereignis sowie Fehlerraten und Betriebsmoduswechsel in den Fallback-Modus pro Stunde für ein nicht rechtzeitiges Stellsignal (*NoValue*-Ereignis) an die Steuereinheit des LVADs.

Tabelle 1 - Ergebnisse der Sicherheitsanalyse für ein NoValue-Ereignis in verschiedenen LVAD-BDS Systemen

Systemmodell:	Auftrittswahrscheinlichkeit pro 0.0005 Stunden	Fehlerrate pro Stunde	Wechselt in den Fallback-Modus pro Stunde
Single-Smartphone-LVAD-System ohne Fehlertoleranz	2.9E-1	580	X
Single-Smartphone-LVAD-System mit Fehlertoleranz	1.6E-1	320	0.078
Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranz:	3.4E-3	6.8	4.0E-11

Die Ergebnisse der Sicherheitsanalyse belegen eine deutliche Verbesserung der Dienstgüte durch die eingesetzten Konzepte und Muster für die Entwicklung von BDS Systemen.

1.4 Aufbau der Arbeit

Die vorliegende Dissertation ist wie folgt strukturiert:

Kapitel 2 beschreibt die technischen Grundlagen, die dieser Arbeit zugrunde liegen. Grundbegriffe und Verfahren für verteilte vernetzte Systeme, softwarebasierte SkS, Fehlertoleranztechniken und Modellierungssprachen werden in diesem Kapitel näher betrachtet und kurz beschrieben.

Kapitel 3 gibt einen Einblick in aktuelle Arbeiten, die sich mit Teilaspekten dieser Dissertation beschäftigen. Der aktuelle Forschungsstand im Bereich der Verlässlichkeit in IoT-Systemen, die Aktivitätserkennung mittels mobiler Endgeräte, Folgeerkrankungen durch die LVAD-Unterstützung sowie Modellierung und Analyseansätze für SkS mittels AADL werden dargestellt.

Kapitel 4 beschreibt den wissenschaftlichen Beitrag der Arbeit. Es wird erläutert, wie Architekturkonzepte und Fehlertoleranzmuster gewählt und angepasst werden, um die Einhaltung der Rechtzeitigkeit und Dienstgüte in BDS zu gewährleisten.

Kapitel 5 stellt einen medizinischen BDS-gestützten Lösungsansatz vor. Neben einer kurzen Einführung in LVAD-Systeme werden die medizinischen Anforderungen sowie der Systementwurf und die Softwarelösung erläutert.

Kapitel 6 stellt die Ergebnisse der modellbasierten Analysen für die verschiedenen Entwicklungszyklen dar.

Kapitel 7 repräsentiert den Proof-of-Concept-Demonstrator, der entwickelt wurde, um den medizinischen BDS-Lösungsansatz zu evaluieren. Des Weiteren werden die experimentellen Ergebnisse der Fehlerinjektionen und der CPU-Belastungstests dargestellt.

Kapitel 8 fasst die gewonnen Ergebnisse zusammen und gibt einen Ausblick auf zukünftige Arbeiten und Möglichkeiten, die sich im Bereich der verlässlichen vernetzten Geräte-basierten Systeme ergeben.

2 GRUNDLAGEN

Das folgende Kapitel beschäftigt sich mit den für die Entwicklung von BDS relevanten technischen Grundlagen. Zunächst werden die Systemklassen der verteilten Systeme und SkS näher betrachtet und deren Unterschiede vorgestellt. Weiterhin werden Fehlertoleranzmechanismen, die im späteren Verlauf Verwendung finden, genauer beschrieben. Abschließend beschäftigt sich dieses Kapitel mit der modellbasierten Systementwicklung. Neben der Einführung von Architektur-Beschreibungssprachen (ADLs) werden aktuelle modellbasierte Analysetechniken beschrieben.

2.1 Verteilte Systeme

Nach Tanenbaum [19] ist ein verteiltes System wie folgt definiert:

> *"Ein verteiltes System ist eine Ansammlung unabhängiger Computer, die den Benutzern wie ein einzelnes kohärentes System erscheinen."*

Dies macht deutlich, dass ein verteiltes System aus verschiedenen autonomen Komponenten besteht. Des Weiteren soll der Anwender von der verteilten Struktur nichts bemerken. Die unabhängigen Computer können leistungsstarke Mainframe-Computer, Personal-Computer, Smarte-Konsumergeräte oder auch kleine intelligente Sensoren sein [19]. Ein weiterer wichtiger Aspekt für verteilte System ist in der folgenden Definition dargestellt [20].

> *"A distributed system is one in which components located at networked computers communicate and coordinate their actions only by passing messages."*

Dies zeigt, dass die einzelnen Systemkomponenten untereinander vernetzt sind und mittels Nachrichtenaustausch kommunizieren. Kommunikationsstandards wie Ethernet, WiFi, Bluetooth, ZigBee oder WiMax kommen in verteilten Systemen zum Einsatz [20].

Verteilte Systeme haben folgende Ziele [19]:

- Der Zugriff auf lokale und entfernte Ressourcen soll vereinfacht bzw. erst ermöglicht werden.
- Das System sollte transparent sein. Man spricht von einem transparenten System, wenn der Anwender das System als ein einziges Computersystem

wahrnimmt und von der verteilten Struktur und Dezentralisierung nichts bemerkt. Man unterscheidet zwischen Zugriffstransparenz, Ortstransparenz, Migrationstransparenz, Relokationstransparenz, Replikationstransparenz, Nebenläufigkeitstransparenz sowie Fehlertransparenz.
- Ein weiteres Ziel ist die Offenheit. Offenheit heißt, dass das System interoperabel und flexibel ist. Verschiedene Komponentenkonfigurationen können so ermöglicht werden.
- Zusätzlich soll es in der Größe, geographischen Ausdehnung und der Administration skalierbar sein.

Man unterteilt verteilte Systeme in die folgenden drei Systemklassen:

- **Verteilte Computersysteme** bilden die erste Systemklasse und finden Verwendung bei der Lösung von Hochleistungsaufgaben. Man unterteilt diese Klasse in Cluster- und Grid-Systeme.
  - *Ein Cluster* beschreibt einen lokalen Verbund von Computern, sogenannte Rechenknoten, die durch einen einzigen Computer (Masterknoten) gesteuert und verwaltet werden. Der Masterknoten ist für die Ausführung der Programme und für die Verwaltung des gesamten Clusters verantwortlich. Er ist die zentrale Schnittstelle zum Cluster. Die einzelnen Rechenknoten sind durch ein lokales Hochgeschwindigkeitsnetzwerk miteinander verbunden und erbringen gemeinsam eine Funktion. In der Regel bestehen diese Cluster aus identischen Knoten, den homogenen Clustern [19].
  - *Ein Grid* weist im Gegensatz zum Cluster eine starke Heterogenität auf. Ein Grid ist in hohem Maße dezentral und die einzelnen Rechenknoten sind lose gekoppelt und geografisch verstreut. Im Gegensatz zum Cluster-Computing, welches über ein lokales Netzwerk (LAN) verbunden und zentral angesprochen wird, kommuniziert der Grid-Verbund über das Internet und ist keiner zentralen Hierarchie unterworfen. Die verteilten Ressourcen werden durch offene Standards und Schnittstellen zur Verfügung gestellt [21].
- **Verteilte Informationssysteme** stellen die zweite Systemklasse dar. Abstrakt bestehen Informationssysteme aus drei logischen Schichten (engl. Layers). Die erste Schicht ist die Präsentationsschicht (engl. presentation layer) und dient als Schnittstelle zwischen Mensch-Maschine bzw. Maschine-Maschine. Ein Großteil der Aufgaben dieser Schicht ist es, die Informationen darzustellen und mit dem Informationssystem zu interagieren. Die zweite Schicht, welche direkt unter der Präsentationsschicht angeordnet ist, bildet die Anwendungsschicht (engl. application layer) und fungiert als Verarbeitungsschicht der angeforderten Informationen. In dieser Schicht ist die Businesslogik implementiert, und sie bietet den Service des Informationssystems an. Die Datenschicht (engl. resource management layer) ist die letzte Schicht. Sie befindet sich unter der Anwendungsschicht. In dieser Schicht werden die Daten

für das Informationssystem verwaltet [22]. Im Folgenden werden drei Architekturkonzepte für Informationssysteme kurz dargestellt.

- *1-Schichten-Architektur (engl. One-Tier-Architecture):* Üblicherweise basieren Altsysteme im Informationsbereich auf einer 1-Schicht-Architektur. Diese zentralisierten Systeme bestehen aus einem Mainframe-Computer, der alle logischen Schichten enthält. Die Informationen werden an ein Terminal weiterleitet und anzeigt. Diese Systeme stellen keine Schnittstelle für andere Systeme zur Verfügung.
- *2-Schichten-Architektur (engl. Two-Tier-Architecture):* Klassisch greifen Client-Server-Architekturen dieses Konzept auf und verteilen die logischen Schichten auf zwei unabhängige Geräte. Auf dem Client wird die Präsentationsschicht ausgeführt und entlastet damit den Server. Dort werden nur die Anwendungs- und Datenschicht ausgeführt. Dies ermöglicht es Services über Serverschnittstellen für andere Systeme zur Verfügung zu stellen und bietet eine größere Flexibilität und Erreichbarkeit des Informationssystems.
- *3-Schichten-Architektur (engl. Three-Tier-Architecture):* Bei der 3-Schichten-Architektur werden die drei logischen Schichten modular auf verschiedenen Systemen ausgeführt. Dadurch wird eine klare Trennung der einzelnen Schichten vorgenommen und die Informationsverarbeitung auf verteilte Komponenten ausgelagert. Neben einer verbesserten Skalierung, Performanz und Verfügbarkeit ist eine bessere Wartbarkeit und Erweiterbarkeit gewährleistet.

- **Verteilte Pervasive Systeme** bilden die letzte Systemklasse. Im Gegensatz zu den bisher vorgestellten Klassen, die auf eine hohe Stabilität und zentralisierte Struktur aufbauen, setzen sich pervasive Systeme aus kleinen mobilen und eingebetteten Systemen zusammen. Diese Systeme sind in hohem Maße flexibel und instabil. Ziel solcher durchdringenden Systeme (engl. pervasive Systems) ist es, die Umgebung kontinuierlich zu überwachen, um Veränderungen zu erkennen und darauf zu reagieren. Eine weitere Herausforderung ist es, Ad-hoc-Netze durch Sensoren, Mobiltelefone und Netzwerkinfrastrukturen effizient aufzubauen. Des Weiteren müssen die gewonnenen Informationen einfach im Gesamtsystem zur Verfügung gestellt werden [19]. Im Folgenden werden zwei Anwendungsbeispiele für pervasive Systeme vorgestellt:
  - *Körperbereichsnetzwerke (engl. Body-Area-Networks (BAN))* bilden pervasive Informationssysteme im Gesundheitswesen. Mittels verschiedener Sensoren im und am Körper wird der Gesundheitszustand einer Person überwacht und im Notfall automatisiert Hilfe angefordert. Ein wichtiger Aspekt solcher Systeme ist, dass sie den Benutzer nur minimal behindern und im besten Fall unsichtbar für den Benutzer sind [23].
  - *Sensor-Netzwerke* bilden eine Verbesserung zu herkömmlichen Sensoren. Der stetige Fortschritt der Sensortechnik ermöglicht eine genauere Abtastung, verbesserte Datenverarbeitung und optimierte

Kommunikation von pervasiven Sensorsystemen. Ein solches Sensornetzwerk besteht aus einer Großzahl von kleinen Sensoren, die dicht verteilt sind, um Messdaten aus ihrer Umgebung zu sammeln. Die Masse an Rohdaten der Sensoren können für eine Weiterverarbeitung bzw. Fusionierung direkt an den Anwendungsprozess übermittelt werden. Weiterhin können die gemessenen und empfangenen Daten der Umgebung auch lokal im Sensorknoten verarbeitet werden, und nur relevante schon vorverarbeitete Informationen werden an den Anwendungsprozess weitergeleitet [24].

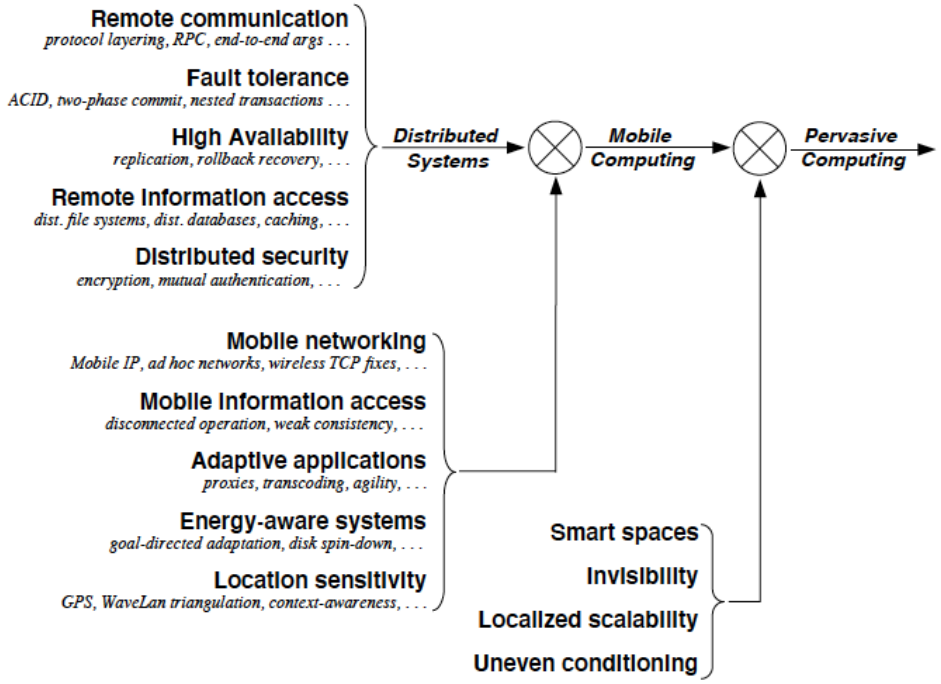


Abbildung 4 - Abhängigkeiten und Zusammenhänge von verteilten Systemen[25]

Abbildung 4 zeigt die Zusammenhänge und Entwicklung von verteilten und mobilen Systemen. Die effizienteren und verlässlicheren Kommunikationstechnologien sowie der immer größer werdende Anteil an smarten Endgeräten führt zu einer stetigen Erweiterung von verteilten Informationssystemen und Steuerungssystemen [25].

2.2 Verlässliche Systeme

Verlässliche Systeme (engl. dependable systems) werden gewöhnlich mit dedizierter Hardware gebildet und sind in ihre Umgebung eingebettet. Sie erbringen eine wohldefinierte, größtenteils sicherheitskritische Funktion und sind statisch konfiguriert. Dies ermöglicht es, das Systemverhalten vorherzusagen, Realzeitbedingungen einzuhalten und eine Qualitätssicherung zu gewährleisten [7].

Oft werden verlässliche SkS auch als monolithische IT-Systeme bezeichnet. Die Entwicklung als eine untrennbare zentralisierte Systemeinheit steht damit im

Gegensatz zu verteilten Systemen und deren Aufteilung in vernetze offene Teilsysteme. Im Folgenden werden die Grundlagen der Systemverlässlichkeit erläutert. Nach Laprie [26] ist die Verlässlichkeit von Computersystemen die Qualität der erbrachten Dienstleistung, sodass man sich zu Recht auf diese Dienstleistung verlassen kann.

Abbildung 5 stellt die drei Eigenschaften der Verlässlichkeit dar. Die Bedrohungen der Verlässlichkeit, rot dargestellt, beschreiben die Faktoren, die die Verlässlichkeit eines Computersystems beeinflussen können. Die Attribute, grün dargestellt, beschreiben die geforderten Anforderungen bzw. Eigenschaften an die Verlässlichkeit von Computersystemen. Die letzte Eigenschaft befasst sich mit den Verbesserungen zur Gewährleistung der Systemverlässlichkeit [27].

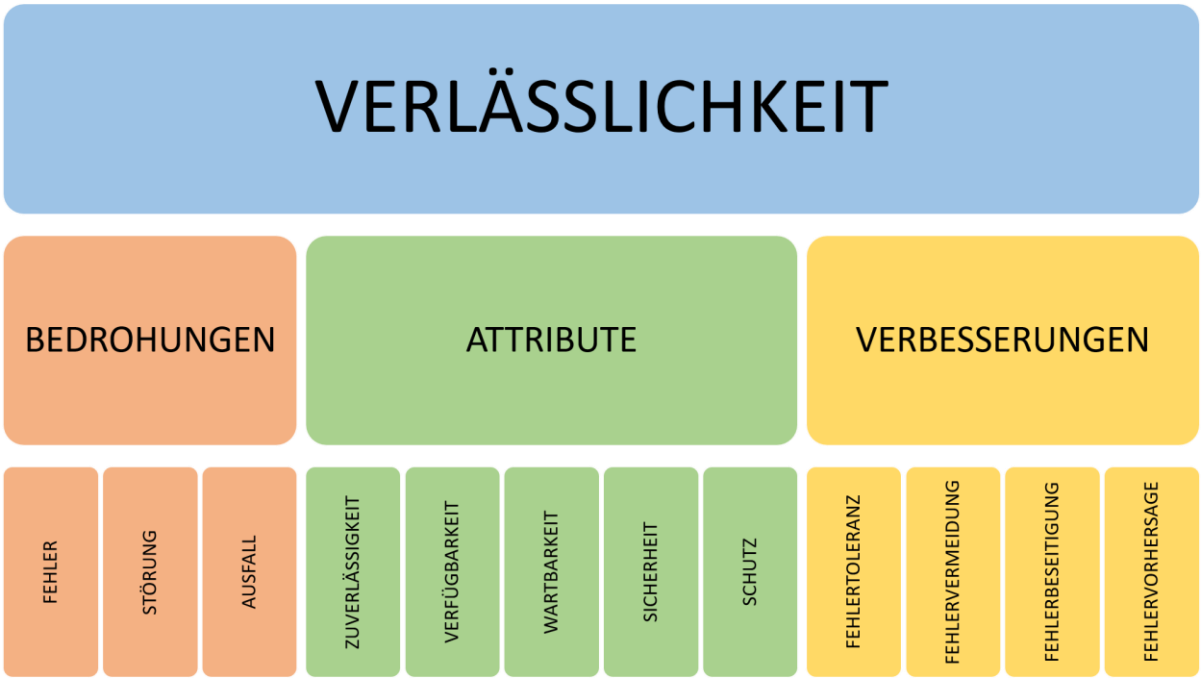


Abbildung 5 - Grundlegende Eigenschaften von Verlässlichkeit[27]

2.2.1 Die Bedrohungen der Verlässlichkeit

Die Verlässlichkeit eines Systems ist die korrekte Bereitstellung von Diensten. Eine Bedrohung dieser korrekten Bereitstellung ist ein Dienstausfall. Ein solcher *Ausfall (engl. Failure)* ist ein Ereignis, bei dem der zu erbringende Dienst von der korrekten Dienstleistung abweicht. Ein Dienst ist eine Folge von externen und internen Zuständen. Dies bedeutet, dass bei einem Dienstausfall mindestens ein Zustand des Systems von der korrekten Leistung abweicht. Eine solche Abweichung wird als *Störung (engl. Error)* bezeichnet. Die Ursache einer Störung wird als *Fehler (engl. Fault)* beschrieben. Fehler können interner oder externer Natur sein. In den meisten Fällen verursacht ein Fehler zunächst eine Störung im Betriebszustand einer lokalen Komponente und kann in der Folge der Störung zu einem Ausfall des Dienstes und damit zu einem Fehlverhalten führen. Nicht jeder Fehler im System führt

zwangsläufig zu einer Störung und damit zu einem Dienstausfall. Man spricht auch von ruhenden bzw. unsichtbaren Fehlern [27].

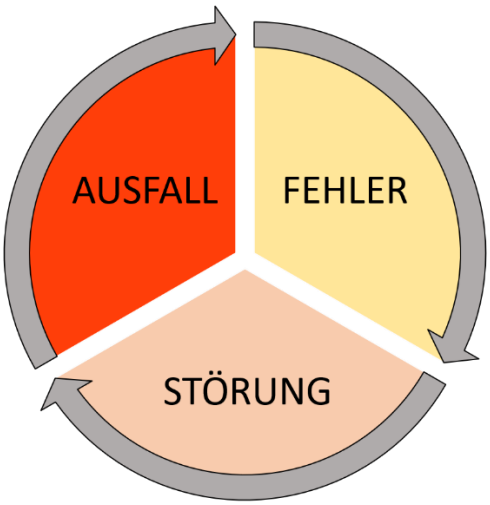


Abbildung 6 - Kausalzusammenhang der Bedrohungen der Verlässlichkeit

Abbildung 6 zeigt die Abhängigkeitskette der einzelnen Bedrohungszustände. Ein ruhender interner Fehler oder ein externer Fehler, der zu einer Störung führt, gilt als aktiver Fehler und löst eine lokale Störung aus. Diese Störung breitet sich in einer Komponente aus und löst an einer Dienstschnittstelle einen Dienstausfall aus. Dieser Ausfall verursacht als externer Fehler in einer anderen Komponente erneut eine lokale Störung. Diese Kausalität führt zu einer Fehlerausbreitung und fehlerhaften Funktionalität des gesamten Systems.

2.2.2 Die Attribute der Verlässlichkeit

Damit ein System verlässlich ist und der Benutzer solch einem System vertrauen kann, hat sich in den letzten Jahrzehnten die Verlässlichkeit als ein integrierendes Konzept der folgenden fünf Eigenschaften entwickelt, welche kurz als "*RAMSS*"-Eigenschaften bezeichnet werden [28]:

Zuverlässigkeit (engl. Reliablity) ist die Wahrscheinlichkeit $R(t)$ eines störungsfreien Betriebs über eine bestimmte Zeitperiode in einer gegebenen Umgebung und für einen bestimmten Zweck. Die Zuverlässigkeit wird oft mithilfe der Rate des Auftretens von Ausfällen ausgedrückt, der Unzuverlässigkeit $Q(t)$. Wenn in einem System durchschnittlich 15 Ausfälle pro 1000 Eingaben in der Stunde auftreten, dann beträgt die Ausfallrate 0.015 pro Stunde und damit eine Zuverlässigkeit von 0,985 pro Stunde (es gilt $Q(t) = 1 - R(t)$) [29, 30].

Verfügbarkeit (engl. Availability) ist die Wahrscheinlichkeit $A(t)$, das ein System zu einem bestimmten Zeitpunkt einsetzbar und in der Lage ist eine geforderte Dienstleistung zu erbringen. Verteilte Systeme, wie beispielsweise ein Web-Server, benötigen die Eigenschaft der Verfügbarkeit. Solche Systeme

können nur eine kurze Unterbrechung der Dienstleistung tolerieren. Die Verfügbarkeit wird in der Regel in der Form von Ausfallzeit pro Jahr angegeben [29, 30].

Wartbarkeit (engl. Maintainability) ist die Wahrscheinlichkeit, dass eine Instandhaltung für ein bestehendes Softwaresystem unter gegebenen Bedingungen innerhalb eines festgelegten Zeitintervalls durchgeführt wird. Zusätzlich beschreibt diese Eigenschaft, dass Software im wirtschaftlichen Sinne angepasst werden muss, um neue Anforderungen bereitzustellen. Diese Anpassungen müssen eine geringe Wahrscheinlichkeit dafür aufweisen, dass neue Fehlerquellen im System induziert werden [29].

Funktionssicherheit (engl. Safety) ist die Wahrscheinlichkeit S(t), dass zum Zeitpunkt t das System seine Funktionen korrekt ausführt (funktionale Korrektheit) sowie im Fall eines Fehlers den Menschen nicht in Gefahr bringt (Schutz des Menschen vor der Maschine). Die Eigenschaft der Funktionssicherheit beschreibt, das ein System, selbst im Falle eines Betriebsausfalls, niemals den Menschen oder seine Umgebung schädigen darf [29, 30].

Datensicherheit (engl. Security) ist das Systemattribut, welches einem System ermöglicht, sich selbst vor externen oder internen Angriffen zu schützen. Ein System darf unter zufälligen oder absichtlichen Angriffen über eine bestimmte Zeitperiode nur mit sehr geringer Wahrscheinlichkeit die Datenintegrität, Informationsvertraulichkeit und Verfügbarkeit verletzen [29].

2.2.3 Die Verbesserungen der Verlässlichkeit

Die Verlässlichkeit von Computersystemen kann durch die folgenden vier Techniken unterstützt werden [27]:

Fehlertoleranz (engl. fault tolerance) bietet einem System die Möglichkeit Software- und Hardwarefehler zu tolerieren. Geeignete Redundanztechniken sowie Designvielfalt geben dem System die Fähigkeit auch bei der Existenz von Fehlern seine korrekte Funktionalität weiterhin zu gewährleisten [27, 31]. Die Fehlertoleranz wird im nächsten Absatz näher betrachtet.

Fehlervermeidung (engl. fault prevention) beschreibt die Technik, die Entstehung oder das Auftreten von Fehlern schon frühzeitig im Entwicklungsprozess zu vermeiden. Dies wird durch geeignete Qualitätskontrollen während der Spezifikation, Implementierung und Fertigung erreicht. Designprüfungen, Komponentenuntersuchung sowie umfangreiche Testphasen für Hardware und Software sind gängige Mittel zur Verifikation [30].

Fehlerbeseitigung (engl. fault removal) kann in zwei Phasen erfolgen. Zum einen während der Entwicklungsphase und zum anderen im laufenden Betrieb. In der Entwicklungsphase besteht die Fehlerbeseitigung aus Verifikation,

Diagnose und Korrektur. Während der Betriebsdauer erfolgt die Fehlerbeseitigung durch regelmäßige Wartung sowie Korrektur von Hardware- und Softwarekomponenten [30].

Fehlervorhersage (engl. fault forecasting) wird durch die Durchführung von Analysen und Bewertungen des Systemverhaltens im Fehlerauftritt ermöglicht. Die Auswertung kann qualitativ oder quantitativ sein. Qualitative Fehlervorhersagen haben das Ziel, Fehlerarten oder Ereigniskombinationen, die zum Systemausfall führen können, zu klassifizieren. Die Fehlermöglichkeitsanalyse sowie die Fehlereinflussanalyse (engl. Failure Mode and Effects Analysis (FMEA)) bietet eine mögliche Methode zur qualitativen Auswertung von Systemen. Die quantitative Bewertung verwendet konkrete Wahrscheinlichkeitswerte bzw. Fehlerraten, um wesentliche Risiken mathematisch darzustellen. Markov-Ketten oder stochastische Petrinetze sind Methoden zur quantitativen Auswertung [27].

2.3 Fehlertoleranz

Grundlage für die Entwicklung von verlässlichen fehlertoleranten Systemen ist die Verwendung von Redundanzen und Designvielfalt (engl. design-diversity) [32, 33]. Echtle beschreibt den Begriff der Redundanz als die Verfügbarkeit von zusätzlichen technischen Mitteln, als tatsächlich für die spezifizierte Funktionalität benötigt werden [34]. Das heißt, dass im fehlerfreien Betrieb die zusätzlichen technischen Mittel teils unnötig sind und nicht gebraucht werden und ausschließlich der Fehlertoleranz dienen. Abbildung 7 zeigt die Unterteilung des Redundanzbegriffes in die verschiedenen Merkmale (blau dargestellt) und die Arten der Aktivierungen (gelb dargestellt).

Man unterscheidet vier Arten von Redundanzen, die strukturelle, die funktionelle, die Informations- und die Zeitredundanz [34].

- Die *strukturelle Redundanz* beschreibt die Zurverfügungstellung von zusätzlichen gleichen oder andersartigen Komponenten im gesamten Systemkontext. Verschiedene Sensoren zur Datengewinnung oder ein 2-von-3 Computersystem sind Beispiele für die Erweiterung der Systemarchitektur durch zusätzliche Hardwarekomponenten. Diese Redundanz beeinflusst unter anderem die Systemkosten. Ebenso kann sie das Gewicht, den Energiebedarf und die Komplexität durch den zusätzlichen Aufwand beeinflussen [30].
- Bei der *funktionellen Redundanz* wird ein System softwareseitig mittels zusätzlicher Funktionalität oder durch redundante Softwareversionen, die mittels Designvielfalt entwickelt werden, erweitert. Diese Funktionen können Zusatzfunktionen sein und nur der Fehlertoleranz dienen. Beispiele für die Zusatzfunktionen sind Testfunktionen, Überwachungsfunktionen oder auch Rekonfigurationsfunktionen. Das N-Versionen-Programming (NVP) Muster ist ein typisches Beispiel für die Entwicklung von diversitärer funktioneller

Redundanz. Dies macht deutlich, dass die funktionelle Redundanz im engen Zusammenhang mit der strukturellen Redundanz steht [34].

- Die *Informationsredundanz* beschreibt die Erweiterung der Datensätze mit zusätzlichen Informationen. Ähnlich wie bei der funktionellen Redundanz sind die zusätzlichen Informationen für die Erkennung, Lokalisierung und wenn möglich Behebung von Fehlern zuständig. Gängige Mittel für die Informationsredundanz sind das Paritätsbit, der Hamming-Code oder die zyklische Redundanzprüfung [30].
- Unter der *Zeitredundanz* versteht man die zusätzliche Zeit, die zur Ausführung der Funktionalität zur Verfügung steht. Wiederholungen von fehlgeschlagenen Berechnungen oder das Umschalten auf Stand-by-Komponenten benötigt zusätzlich Zeit [35].

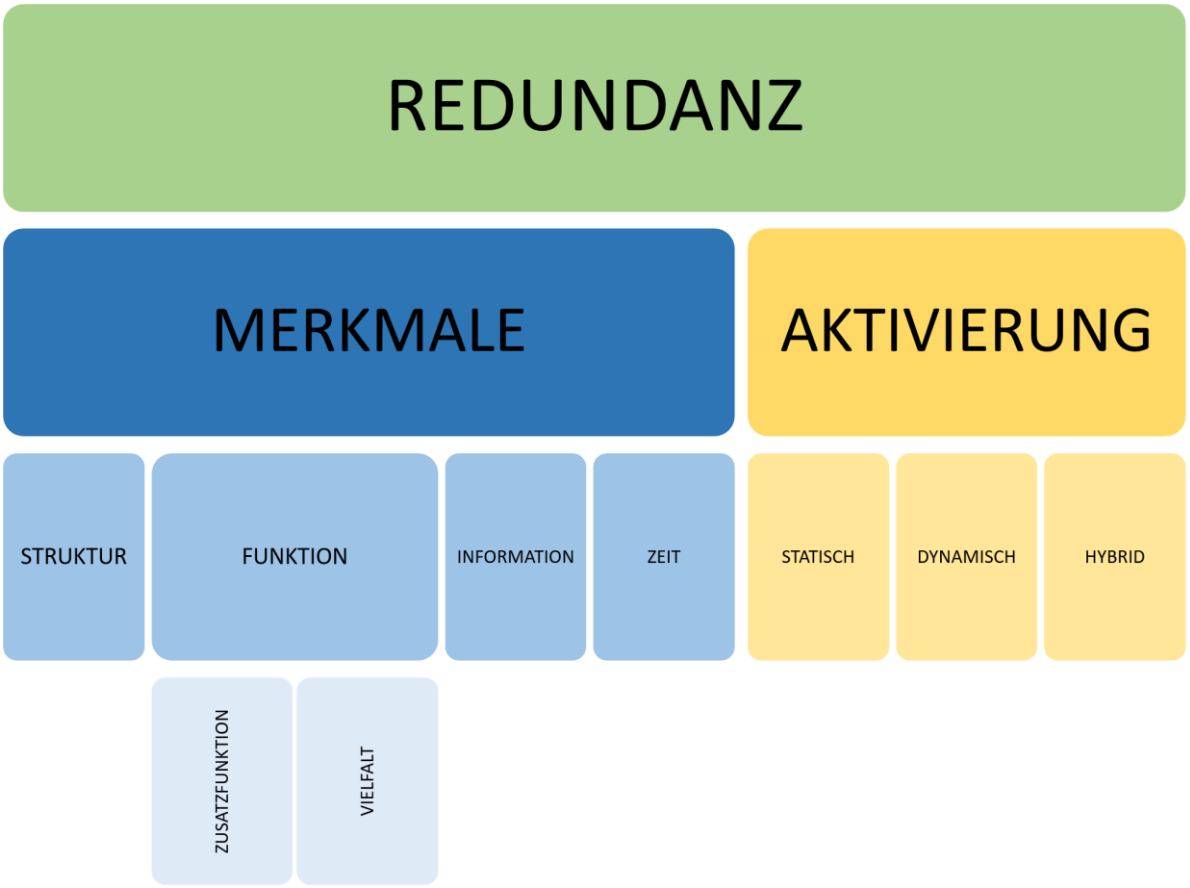


Abbildung 7 - Klassifizierung der Redundanz nach bestimmten Eigenschaften[34]

Die oben beschriebenen Merkmale der Redundanz beschreiben nur das zusätzliche Mittel in Form von Hardware, Software, Daten oder der Zeit zur Verfügung stehen müssen, aber nicht wann und in welcher Form sie im System zur Verfügung stehen. Man unterscheidet zwischen statischen und dynamischen Redundanzen. Bei der *statischen Redundanz* sind die redundanten Mittel während der gesamten Betriebszeit aktiv und erbringen ihre Funktionalität. Dagegen werden die redundanten Mittel bei der *dynamischen Redundanz* erst infolge der Fehlerbehandlung aktiviert und erbringen im fehlerfreien Verlauf keinerlei Funktionalität. Mittel der dynamischen Redundanz stehen dem System als Reserven während der Betriebszeit zur Verfügung. Eine

Reservekomponente kann als heiße Reserve (engl. Hot-Stand-by) auftreten, bei der alle redundanten Mittel eingeschaltet sind und auf eine Aktivierung warten, oder als kalte Reserve (engl. Cold-Stand-by), die bis zu ihrer Aktivierung nicht eingeschaltet sind und erst geladen und konfiguriert werden müssen. In den meisten Fällen kommen statische und dynamische Redundanzen in Kombination vor und bilden die Klasse der hybriden Redundanzen [34].

Betrachtet man die Fehlertoleranz etwas genauer, können die möglichen Redundanztechniken dazu beitragen, Fehler frühzeitig zu erkennen und diese auch zu behandeln. Abbildung 8 bildet die grundlegenden Techniken der Fehlertoleranz ab. Diese werden im Folgenden näher betrachtet.

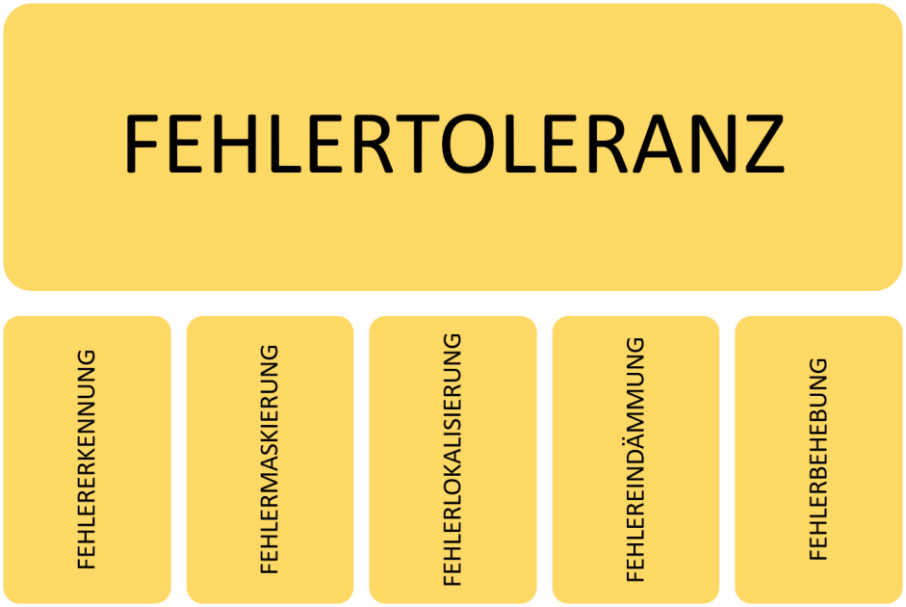


Abbildung 8 - Maßnahmen zur Einhaltung der Fehlertoleranz[30]

Die Grundlage der Fehlertoleranz ist das Erkennen von Fehlern und Störungen, die in einem System auftreten können. Techniken zur *Fehlererkennung* sind Absolut- und Relativtests. Absoluttests (engl. acceptance test) sind Fehlererkennungsmechanismen, die auch für Systeme ohne redundante Komponenten verwendet werden können und nach dem SOLL-IST-Vergleich entscheiden. Ein fehlgeschlagener Absoluttest impliziert immer ein Fehlverhalten. Bei Relativtests (engl. comparison test) werden mehrere replizierte Ergebnisse miteinander verglichen und nach dem Prinzip IST-IST-Vergleich überprüft. Die Ausgaben von mehreren Komponenten werden verglichen, und eine Unstimmigkeit in den Ergebnissen zeigt einen Fehler an [30, 34].

Eine weitere Technik der Fehlertoleranz ist die Kompensation von Fehlern, die auch *Fehlermaskierung* genannt wird. Durch die Maskierung wird sichergestellt, dass trotz des Vorhandenseins eines Fehlers nur korrekte Werte im System weitergeleitet werden. Diese Möglichkeit wird durch das Kompensieren der fehlerfreien redundanten Komponenten gewährleistet. Ein Beispiel für die Maskierung von fehlerhaften Ergebnissen kann durch den Mehrheitsentscheid für N-Komponenten ermöglicht werden, bei dem mindestens (N+1)/2 Ergebnisse gleich seien müssen [30].

Ein Absolut- bzw. Relativtest kann Fehler zwar erkennen, aber im Allgemeinen nicht zur *Fehlerlokalisierung* verwendet werden. Fehlerlokalisierungstechniken dienen der genauen Lokalisierung der Fehlerquelle, um die fehlerhafte Komponente zu isolieren oder eine Rekonfiguration des Systems einzuleiten [30].

Fehler, die im System erkannt und lokalisiert werden, sollen isoliert werden, um eine Fehlerausbreitung im System zu verhindern. Diese Technik wird als *Fehlereindämmung* bezeichnet und verhindert, dass Fehler bzw. Störungen sich im gesamten System ausbreiten und zu einem Ausfall führen [30].

Isolierte fehlerhafte Komponenten sollten aus dem Systemkontext ausgeschlossen werden, solange sie nicht funktionsfähig sind. Eine Rekonfiguration auf Reservekomponenten bzw. der sukzessive Funktionsabbau sind *Fehlerbehebungstechniken* und gewährleisten trotz der Existenz von Fehlern die künftige Systemfunktionalität [30].

Dies setzt für die Gewährleistung der Fehlertoleranz die folgenden zwei Maßnahmen voraus:

- Fehlererkennung,
- Fehlerbehandlung.

Bei der Fehlerbehandlung unterscheidet man zwischen zwei grundlegenden Verfahren, der Rückwärtsfehlerkorrektur sowie der Vorwärtsfehlerkorrektur. Bei der Rückwärtsfehlerbehebung (engl. Rollback) wird das System im Fehlerfall in einen konsistenten Zustand zurückversetzt, der in der Vergangenheit vorherrschte. Das System kann fehlgeschlagene Prozesse noch einmal durchführen und mögliche transiente Fehler korrigieren und ausgrenzen. Im Gegensatz zur Rückwärtsfehlerbehebung wird das System bei der Vorwärtsfehlerkorrektur (engl. Roll-Forward) nicht zurückgesetzt, sondern greift auf redundante Systemkomponenten bzw. Daten zurück. Dies ermöglicht es, Fehler im System zu maskieren bzw. im Eskalationsfall das System in einen sicheren Notbetrieb zu versetzen.

2.3.1 Software-Fehlertoleranztechniken

Im Folgenden werden Software-Fehlertoleranztechniken, insbesondere Grundmuster der Fehlertoleranz sowie erweiterte zusammengesetzte Fehlertoleranzmuster vorgestellt [30, 31, 36, 37]. Tabelle 2 stellt Grundmuster für die softwarebasierte Fehlertoleranz dar. Sie zeigt Mechanismen, die eine Fehlererkennung bzw. Fehlerlokalisierung, Fehlermaskierung und Fehlerbehebung ermöglichen. Diese Muster bilden die Grundlage für die in Tabelle 3 dargestellten Software-Fehlertoleranztechniken und die damit verbundenen Fehlerbehandlungen in fehlertoleranten Systemen.

Tabelle 2 - Grundmuster für softwarebasierte Fehlertoleranzmuster

Grundmuster		Eigenschaft	Zweck
Absoluttest	**Watchdog-Timer**	Timeout-Watchdogs überwachen die zeitlichen Anforderungen und erkennen eine Überschreitung.	Fehlererkennung, Fehlerlokalisierung
	Kodierungsprüfung	CRC-Codes oder Signaturen ermöglichen eine Fehlererkennung bei der Datenübertragung bzw. das Erkennen von manipulierten Daten.	
	Zähler	Durch das Zählen von fehlerhaften Ereignissen kann das Systemverhalten überwacht werden. Dies ermöglicht es zwischen transienten und permanenten Fehlern zu unterscheiden.	
Relativtest	**Vergleich-Voter**	Überwachung von mindestens zwei Komponenten. Die Eingaben werden verglichen und wenn nicht alle Eingaben gleich sind, wird ein Fehlverhalten angenommen.	Entscheidungsträger, Fehlererkennung
	Mehrheitsentscheid-Voter	Überwachung von mindestens drei redundanten Komponenten. Die Eingaben werden wie beim Vergleich-Muster miteinander verglichen und wenn weniger als (N+1)/2 Ergebnisse übereinstimmen, wird ein Fehlverhalten angenommen.	Entscheidungsträger, Fehlererkennung, Fehlermaskierung
	Dynamische-Voter	Überwachung von mindestens zwei Komponenten. Die Eingaben werden dynamisch nach der Gewichtung oder statistischen Mitteln verglichen und ausgewertet.	Entscheidungsträger, Fehlererkennung, Fehlermaskierung
Sicherung	**History**	Daten oder Systemvariablen werden zwischengespeichert und können im Systemablauf zur Fehlerbehandlung verwendet werden.	Fehlerbehebung
	Checkpoint	Ein konsistenter Systemzustand wird periodisch abgespeichert und steht dem System im Fall eines Fehlers als Wiederherstellungspunkt zur Verfügung.	Fehlerbehebung

Struktur	**Redundanz**	Modularisierung von Prozessen im System zur Möglichkeit der Eindämmung von Fehlern. Durch statisches oder dynamisches Vorkommen von Systemkomponenten ist es möglich, Fehler zu maskieren oder zu isolieren.	Fehlereindämmung, Fehlerbehebung

Tabelle 3 stellt softwarebasierte Fehlertoleranztechniken vor, die eine Fehlerbehandlung möglich machen. Die zuvor vorgestellten Grundmuster kommen zur Fehlererkennung, -maskierung und -behebung in diesen Mustern zum Einsatz. Grundlage der vorgestellten Fehlertoleranztechniken sind Redundanzen und Designvielfalt.

2.4 Modellbasierte Systementwicklung

Neben der Fehlertoleranz sind die Vermeidung, Beseitigung und das Vorhersagen von Fehlern ein weiterer wichtiger Aspekt bei der Entwicklung von SkS, siehe Abbildung 8. Schon während der Entwurfs- und Entwicklungsphase können adäquate Vorgehensmodelle, wie das V-Modell XT [38], dazu beitragen, die Verlässlichkeit zu steigern. Durch das Modellieren, Analysieren und Testen können in der Entwicklungsphase frühzeitig spezifische Eigenschaften des Systems verifiziert und validiert (V&V) [39] werden. Dazu wird in der konzeptionellen Entwurfsphase immer häufiger der Ansatz der modellbasierten Systementwicklung (MBSE) angewendet. Ziel der MBSE ist es, von der dokumentenzentrierten Entwicklung zur modellzentrierten Entwicklung zu kommen. Der internationale Rat für Systemtechnik, kurz INCOSE [40], beschreibt die MBSE als eine abstrakte Sichtweise der Modellierung zur Unterstützung von Systemanforderungen, Entwurfs-, Analyse-, Verifikations- und Validierungsaktivitäten, die den gesamten Entwicklungszyklus einschließen [41]. Ein Modell ist eine abstrakte Sichtweise der Realität wobei die Sicht oder auch Perspektive unterschiedlicher Natur sein kann und damit auch verschiedene Modelle zu einem System existieren können. Jedes dieser Modelle beschreibt somit nur die wesentlichen Merkmale und reduziert damit den Informationsfluss auf ein Minimum der zu abstrahierenden Sichtweise [42].

Im Umfeld der softwarebasierten Systementwicklung hat sich der modellgetriebene Architekturansatz (MDA) als Form der MBSE etabliert [43]. Beim MDA-Ansatz steht das Modell der Systemarchitektur im Mittelpunkt und bedarf einer formellen Sprache zur Beschreibung der Architektur und deren Eigenschaften. Der IEEE Standard 42010 [44] beschreibt Möglichkeiten zur Architekturbeschreibung und dient als Mittelpunkt der architekturzentrierten modellbasierten Systementwicklung.

Tabelle 3 - Software Fehlertoleranztechniken

Fehlertoleranzmuster	Eigenschaft	Zweck	Fehlererkennung	Fehlerbehebung	Ausführung
Recovery Block-Muster (RcB)	Das RcB-Muster verwendet zur Fehlererkennung einen Absoluttest und kann im Fehlerfall das System auf einen vorhandenen Checkpoint zurücksetzen. Die zusätzlichen Softwareblöcke werden dynamisch aktiviert. Die Ausführung und Verarbeitung der einzelnen Reserven benötigt Zeitredundanz.	Fehlereindämmung, Fehlerbehebung	Absoluttest	Rückwärtsbehebung mittels Checkpointing	Ausführung sequentiell diversitär redundanter Software-Komponenten.
N-Version Programming-Muster (NVP)	Das NVP-Muster verwendet zur Fehlererkennung einen Relativtest als Entscheidungsträger. Im Gegensatz zum RcB-Muster, ist das NVP-Muster ein statisches Muster, bei dem die redundanten Komponenten parallel aktiv sind. In der Regel wird ein Mehrheitsentscheid als Entscheidungsträger verwendet.	Fehlermaskierung	Relativtest	Maskierung	Ausführung parallel diversitär redundanter Software-Komponenten.

N-Self-Checking Programming-Muster (NSCP)	Das NSCP-Muster verwendet zur Fehlererkennung einen Absoluttest bzw. einen Vergleich. Die Ergebnisse der sich selber überwachenden Komponenten werden zusätzlich verglichen, und ein Entscheidungsträger in Form eines Relativtests kann Fehler maskieren.	Fehlermaskierung	Absoluttest oder Vergleich sowie Relativtest	Maskierung	Ausführung parallel oder sequentiell diversitär redundanter Software-Komponenten.
Distributed Recovery Blocks-Muster (DRB)	Das DRB-Muster ist eine Kombination aus statischen und dynamischen Mustern. Es bietet Hardware- und Software-Fehlertoleranz.	Fehlereindämmung, Fehlerbehebung	Absoluttest	Rückwärtsbehebung durch RcB in der Leitstation und Vorwärtsbehebung durch aktive redundante Folgestationen	Ausführung parallel und sequentiell diversitär redundanter Komponenten.
Acceptance Voting-Muster (AV)	Das AV-Muster kombiniert Absoluttests und Relativtest zur Fehlererkennung und Maskierung. Parallel ausgeführte Komponenten werden durch Aboluttests überwacht und die Ergebnisse durch einen Relativtest zusätzlich geprüft.	Fehlermaskierung	Absoluttest und Relativtest	Maskierung	Ausführung parallel diversitär redundanter Software-Komponenten.

Fall-Back-Muster (FB)	Das FB-Muster verwendet einen Absoluttest zur Fehlererkennung. Im Fehlerfall kann das FB-Muster auf eine sichere redundante Systemkomponente zurückgreifen. Diese kann gegebenenfalls eine degradierte Notfall-Funktionalität aufweisen.	Fehlereindämmung, Fehlerbehebung	Absoluttest	Vorwärtsbehebung durch Ausführung im ausfallsicheren Modus	Ausführung sequentiell diversitär redundanter Software-Komponenten

Der Standard spezifiziert die Praktiken, Techniken und Darstellungsarten, welche von Software-Architekten zum Modellieren, Analysieren und Präsentieren einer softwarebasierten Architektur verwendet werden können. Es gibt mehrere verbreitete Vorgehensweisen bei der Architekturbeschreibung:

Architektursicht und Architekturblickwinkel unterteilen und spezifizieren die Architektur in Teilbereiche, die für individuelle Anwender von Interesse sind. Eine Architektursicht (engl. architecture view) repräsentiert ein System aus der Perspektive einer konsistenten Menge von Anliegen. Ein Architekturblickwinkel (engl. architecture viewpoint) spezifiziert die Konzepte, Modelle, Analysetechniken und Prinzipien für die Konstruktion einer spezifischen Architektursicht [44].

Architektur-Frameworks legen eine einheitliche Vorgehensweise für das Erstellen, Interpretieren, Analysieren und Verwenden von Architekturbeschreibungen innerhalb eines bestimmten Anwendungsbereichs oder einer Interessengemeinschaft fest. Ziel eines solchen Frameworks ist es, eine gemeinsame Sprache für Architekturbeschreibungen zu gestalten [44]. Bekannte Architektur-Frameworks sind zum Beispiel *The Open Group's Architecture Framework (TOGAF)* [45], das *4+1-Sichtenmodell* [46] oder die *Siemens 4 Sichten Methode* [47].

Architekturbeschreibungssprachen (ADL) sind ein formales Mittel zur Darstellung von Systemarchitekturen. ADLs sollen sowohl für Menschen als auch für Maschinen einfach lesbar und interpretierbar sein. ADLs beschreiben ein System auf einer höheren Abstraktionsebene. Zusätzlich ermöglicht eine Architekturbeschreibung, teils automatisch, eine Analyse und Bewertung hinsichtlich Vollständigkeit, Konsistenz und Leistung. Das *Struktur-Funktions-Modell* [34], die *Systems Modeling Language (SysML)* [42] oder die *Architecture Analysis and Design Language (AADL)* [43] sind verbreitete ADLs im Umfeld der fehlertoleranten Systementwicklung.

Die Möglichkeit komplexe Systemstrukturen strukturiert darzustellen und formal zu beschreiben ermöglicht es SkS sowohl effizient und sorgfältig zu planen als auch zu entwickeln. Im Folgenden werden Architekturbeschreibungssprachen sowie Analysetechniken für die Sicherheitsbewertung beschrieben.

2.4.1 Architekturbeschreibungssprachen

Im Weiteren werden drei grafische Modellierungssprachen näher betrachtet:

- Struktur-Funktions-Modell,
- System-Modellierungssprache SysML,
- Architekturbeschreibungssprache AADL.

2.4.1.1 Struktur-Funktions-Modell

Das Struktur-Funktions-Modell nach Klaus Echtle beschreibt eine grafische Darstellung von Systemen als einen Zusammenhang von Komponenten und der Zuordnung von Funktionen [34]. Die Systemstruktur wird als gerichteter Graph dargestellt. Dieser bildet ab, ob Funktionen (gerichtete Pfeile) durch einzelne Komponenten (Knoten) erbracht oder genutzt werden. Abbildung 9 stellt ein System aus sieben verschiedenen Komponenten dar. Die einzelnen Komponenten können Hardware- oder auch Softwaremodule abbilden. Zusätzlich können Subsysteme definiert werden, die eine Teilmenge von Komponenten und die zugehörigen Funktionen darstellen. Die Komponenten *K1, K2 und K3* mit den zugehörigen Funktionen *f1, f2, f3 und f4* bilden das Subsystem S1 und werden als innere Spezifikation des Subsystems beschrieben. Neben der inneren Spezifikation müssen auch die Schnittstellen des Subsystems S1 definiert werden. Diese werden als äußere Spezifikation dargestellt. Am Beispiel des in Abbildung 9 dargestellten Struktur-Funktions-Modells bildet die äußere Spezifikation für das Subsystem S1 die einzelnen Funktionen *f5, f6 und f7* wobei *f5* und *f6* Funktionalität zur Verfügung stellen und *f7* Funktionalität benötigt.

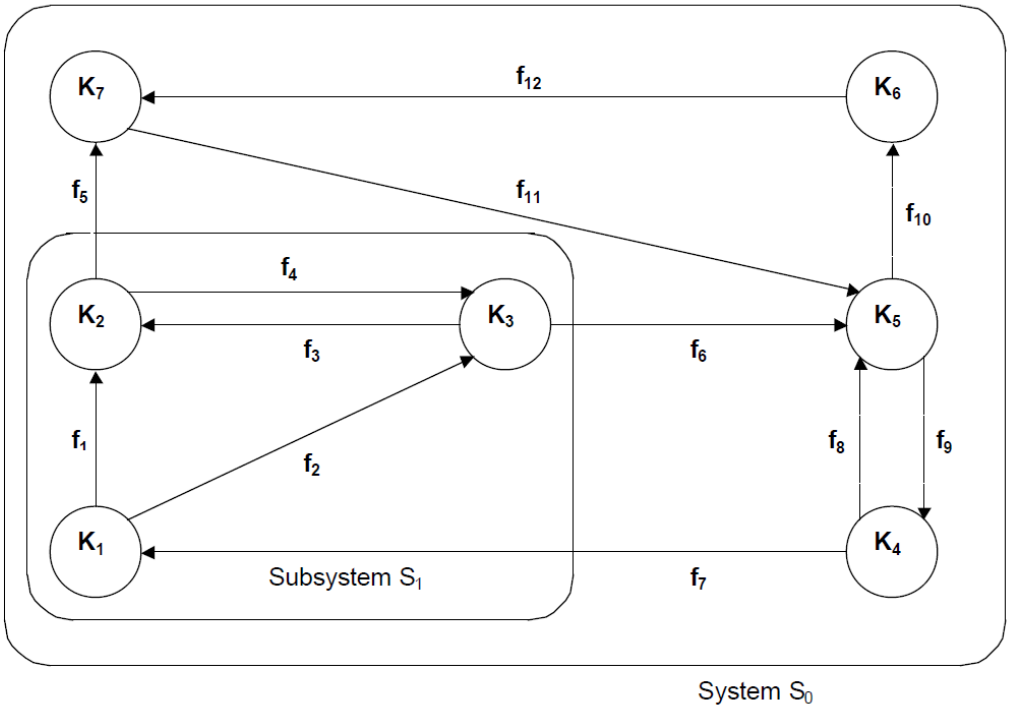


Abbildung 9 - Beispiel eines Struktur-Funktions-Modells[34]

Die abstrakte Darstellung und die Relation zwischen den einzelnen Komponenten durch die Funktionszuordnung kann auf verschiedene Arten interpretiert werden.

Die Funktionszuordnung *f1* aus Abbildung 9 kann wie folgt interpretiert werden [34]:

- *K1* ist Bestandteil der Komponente *K2*,
- *K1* stellt Betriebsmittel zur Verfügung,
- *K2* wird durch *K1* aktiviert,
- *K1* liefert einen Dienst an *K2*.

Dies ermöglicht eine Modellierung aus Sicht der Fehlerentstehung und -ausbreitung sowie eine funktionale Beschreibung von Fehlertoleranzverfahren [34]. Abbildung 10 zeigt ein TMR-System, welches mittels Struktur-Funktions-Modell dargestellt wird. Fehlerbereiche können gemäß der Spezifikation modelliert werden. Am Beispiel eines TMR-Systems ist der Fehlerbereich wie folgt definiert: Es müssen mindestens zwei Komponenten ausfallen. Die Fehlerbereiche in Abbildung 10 bilden die Mengen *{(P1;P2),(P2;P3),(P1;P3)}*, die aufgrund der Übersichtlichkeit nicht abgebildet sind. Da häufig bei Fehlertoleranzverfahren sehr viele Fehlerbereiche zu spezifizieren wären, werden diese vereinfacht in Einzelfehlerbereiche aufgeteilt. Echtle beschreibt den Einzelfehlerbereich wie folgt [34]:

> *"Ein Einzelfehlerbereich ist die maximal große Komponentenmenge, bei der alle Komponenten zu genau den gleichen Fehlerbereichen gehören."*

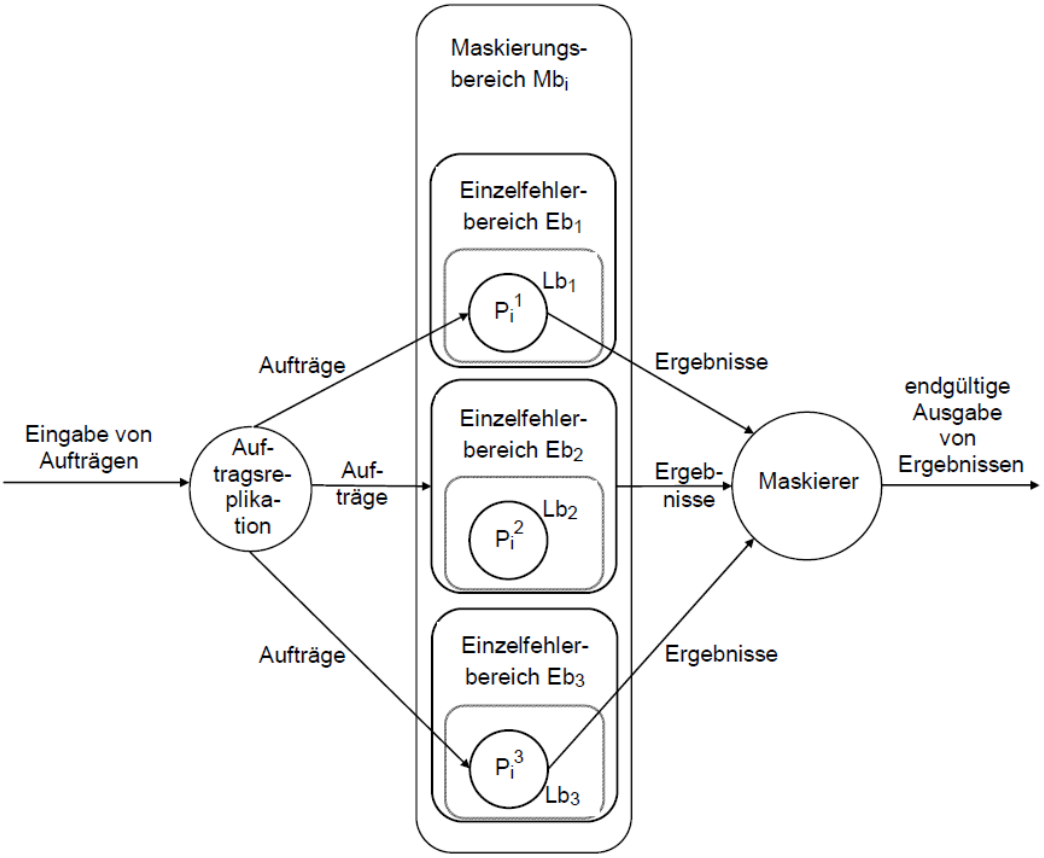


Abbildung 10 - Struktur-Funktions-Modell am Beispiel eines TMR-Systems[34]

Abbildung 10 zeigt die drei Einzelfehlerbereiche *EB1, EB2 und EB3*, welche durch die drei Komponenten *P1, P2 und P3* dargestellt werden. Neben der Darstellung von

Fehlerbereichen bzw. Einzelfehlerbereichen können Komponentenmengen in Form von Lokalisierungs- und Behandlungsbereichen aufgeteilt werden. Behandlungsbereiche können des Weiteren in Ausgrenzungsbereiche, Rückwärtsbehebungsbereiche, Vorwärtsbehebungsbereiche, Maskierungsbereiche und Korrekturbereiche verfeinert dargestellt werden. Das Struktur-Funktions-Modell gibt daher eine gute Möglichkeit, die Komplexität und Größe von fehlertoleranten Systemen anschaulich darzustellen und Sicherheitsbewertungen am Modell durchzuführen.

2.4.1.2 Systems Modeling Language (SysML)

Die System-Modeling-Language, kurz SysML [48], ist eine grafische Modellierungssprache, die auf der Basis von UML2 [49] entwickelt wurde. SysML unterstützt verschiedene Stakeholder bei der Analyse, Spezifikation, Entwurf, Verifikation und Validierung von komplexen Softwaresystemen [50].
Die Sprache hilft dabei, die Systemarchitektur und deren interagierende Komponenten zu spezifizieren. Diese Komponenten können im weiteren Entwicklungsverlauf mit anderen domänenspezifischen Sprachen wie der UML -für den Softwareentwurf- oder VHDL -für den Hardwareentwurf- weiterentwickelt werden. SysML kann Systemarchitekturen wie folgt darstellen [51]:

- Strukturelle Zusammensetzung,
- Systemverhalten,
- Einschränkungen,
- Zuordnungen zwischen Struktur, Verhalten und Einschränkungen,
- Systemanforderungen.

Die Unified Modeling Language (UML Version 2.x) beinhaltet 14 verschiedene Arten von Diagrammen, die hauptsächlich im Umfeld der objektorientierten Softwareentwicklung eingesetzt werden. Die Modellierungssprache SysML, die wie die UML von der Object Management Group (OMG) verwaltet wird, baut auf der UML-Modellierung auf. Einige der UML-Diagrammarten wurden wiederverwendet und basieren auf der gleichen Spezifikation. Andere Diagrammtypen wurden der Systemmodellierung angepasst bzw. verändert und einige werden im Kontext der Systemmodellierung nicht verwendet. Daraus ergibt sich, dass SysML neun verschiedene Arten von Diagrammen beinhaltet. Die OMG klassifiziert die SysML-Diagramme in drei Gruppen [50]. Die erste Gruppe von Diagrammarten beschreibt die Verhaltensdiagramme. Das Systemanforderungsdiagramm bildet die zweite Gruppe und ist ein neues SysML-Diagramm. Die letzte Gruppe bilden die Strukturdiagramme. Abbildung 11 zeigt die neun Diagrammtypen der Systemmodellierungssprache SysML im Detail.

Im Folgenden werden die Diagrammtypen, die gegenüber der UML angepasst und verändert wurden, sowie die neuen SysML-Diagramme näher betrachtet.

Aktivitätsdiagramm

Das Aktivitätsdiagramm (ACT) ist das einzige Verhaltensdiagramm, welches gegenüber der UML-Spezifikation modifiziert wurde. Das ACT zeigt die Daten- und Kontrollflüsse zwischen einzelnen Aktionen. Die wichtigste Veränderung gegenüber der UML-Grundlage ist die Möglichkeit, kontinuierliche und diskrete Informationsflüsse zu modellieren. Die Grundelemente eines ACT sind [50]:

- die Aktionen,
- die Kontrollflüsse und Objektflüsse,
- die Entscheidungsknoten und Kontrolloperatoren,
- Ein- und Ausgangspunkte.

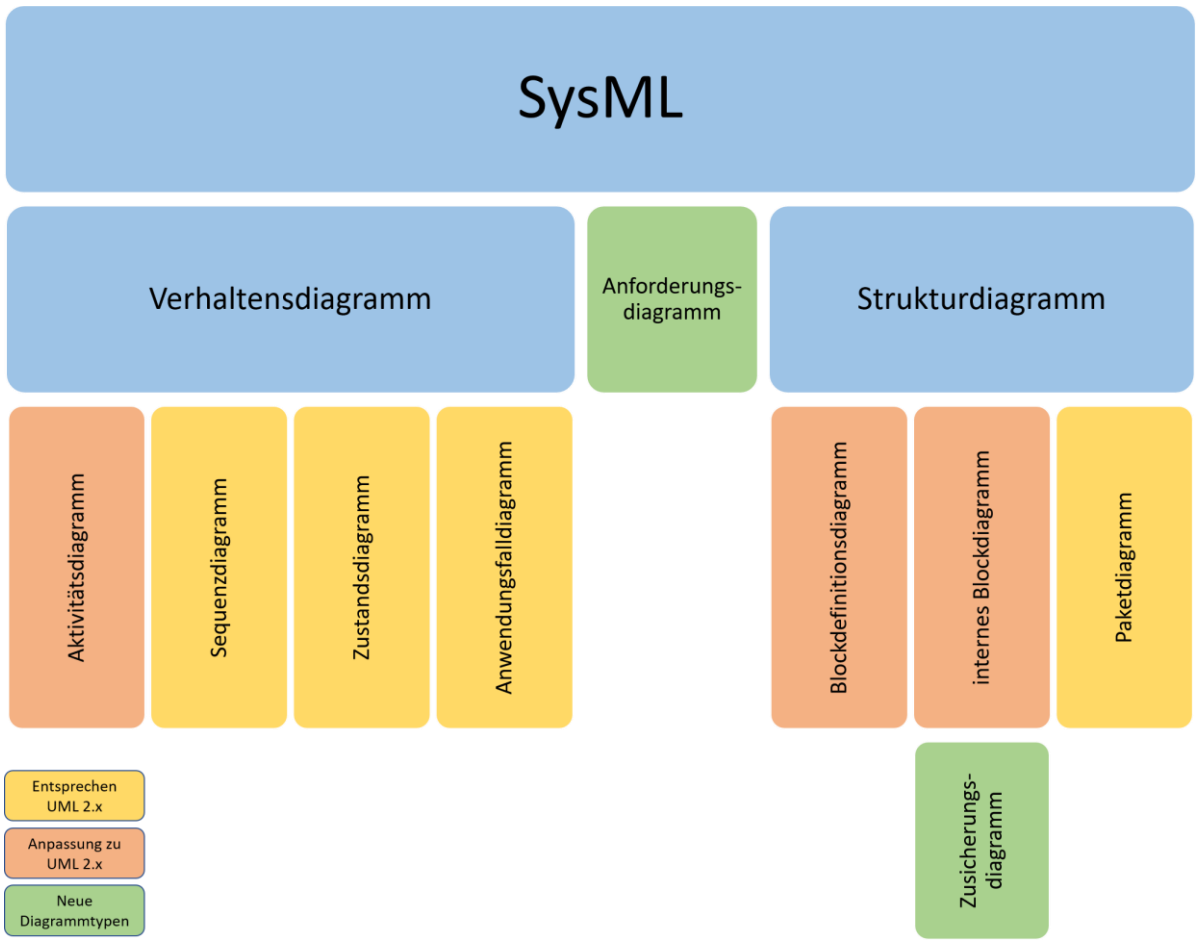


Abbildung 11 - Kategorisierung von SysML-Diagrammtypen[50]

Um die Modellierung von detaillierten Informationsflüssen zu ermöglichen, kann das Modell zwischen kontinuierlichen und diskreten Informationsflüssen unterscheiden und die Flussrichtung darstellen. Mit dem neuen Element des Kontrolloperators ist es möglich, Einfluss auf das Laufzeitverhalten von Aktivitäten und Aktionen zu nehmen. Dadurch kann von Außen durch Kontrollparameter (Objektflüsse) eine Aktivität gestartet, angehalten, pausiert oder fortgesetzt werden [42]. Dies hat den Vorteil, dynamische Abläufe und detailreiche Informationsflüsse über Systemaktivitäten darzustellen und zu bewerten.

Anforderungsdiagramm

Das Anforderungsdiagramm (REQ) ist der erste neue Diagrammtyp unter SysML. Dieser Typ befasst sich mit der formellen Darstellung von Anforderungen, die an ein System gestellt und geprüft werden müssen. Ein REQ besteht neben dem Namen aus mindestens zwei weiteren Eigenschaften. Erstens einer eindeutigen ID und zweitens einer textuellen Beschreibung der Anforderung [50]. Abbildung 12 zeigt, wie unter SysML das Anforderungsdiagramm grafisch dargestellt wird.

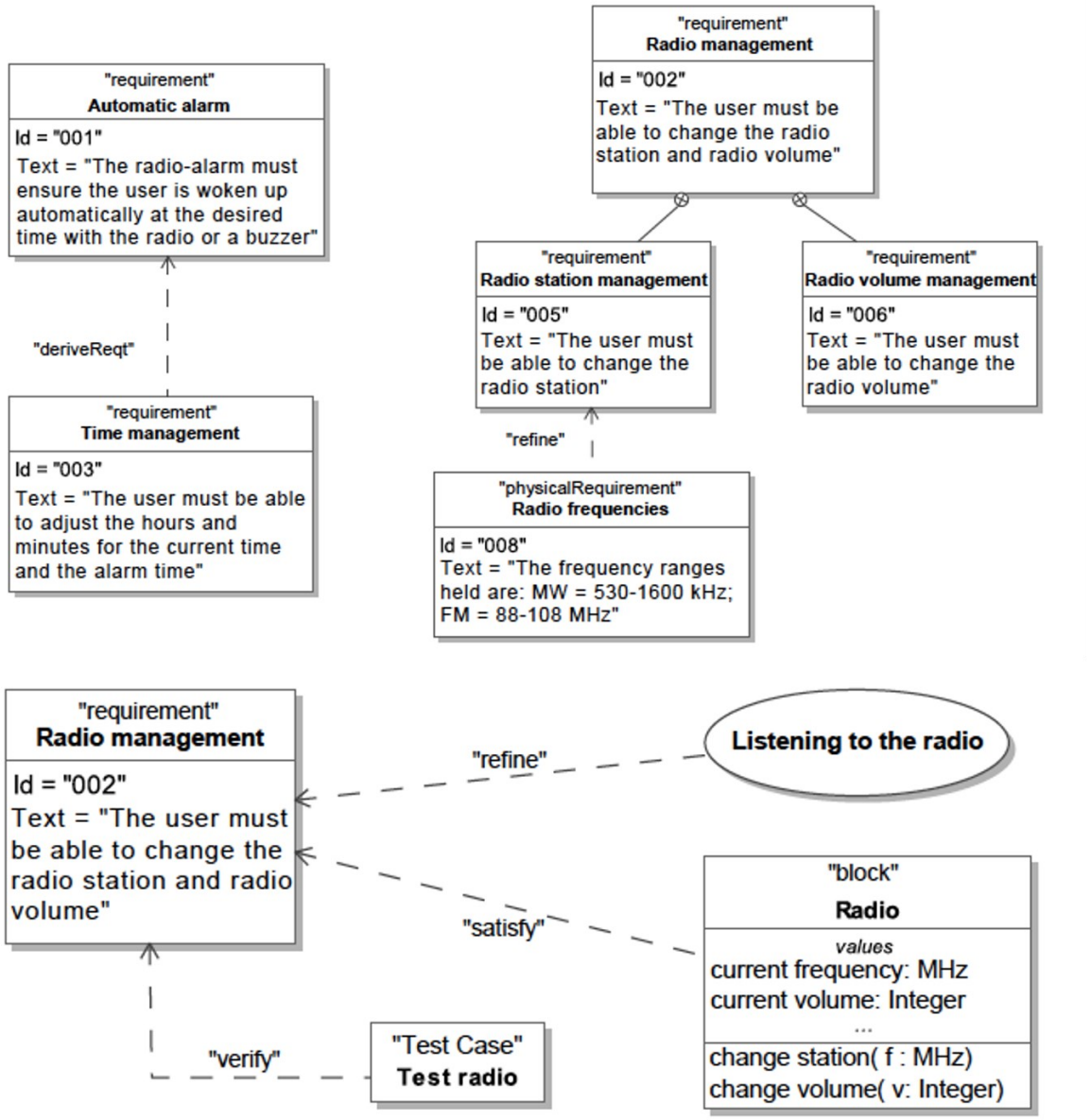


Abbildung 12 - Anforderungsdiagramm unter SysML[50]

Neben der grafischen Darstellung ermöglicht SysML auch die Definition von Relationen zwischen Anforderungsobjekten. Eine Anforderung kann als Kompositanforderung dienen und wird in mehrere Anforderungen aufgeteilt. Dies ist in Abbildung 12 am Beispiel des Radio Managements zu sehen. Die Anforderung wird in zwei detaillierte Anforderungen spezifiziert, eine Anforderung für die Lautstärke

und eine zweite für die Frequenz. Des Weiteren können Anforderungen durch quantitative Eigenschaften verfeinert werden. Dies ist in Abbildung 12 durch die Anforderung der Frequenzbereiche dargestellt und wird durch die Verbindung <*refine*> gekennzeichnet. Diese <*refine*> Beziehung kann nicht nur zwischen verschiedenen Anforderungen existieren, sondern auch zwischen dem REQ-Diagramm und einem Verhaltensdiagramm. Dies ist im obigen Beispiel an dem Anwendungsfall *Radio hören* und der Anforderung *Radio Management* zu sehen. Durch die Verbindung <*deriveReqt*> wird eine Beziehung zwischen zwei Anforderungen dargestellt, in der eine Anforderung von einer anderen abgeleitet werden kann. Das Zeitmanagement ist eine abgeleitete Anforderung der Alarmanforderung, siehe Abbildung 12 links. Eine weitere Beziehung kann dazu benutzt werden, um eine Realisierung der Anforderung dazustellen. Dazu wird ein Blockdefinitionsdiagramm, welches die Anforderung erfüllt über die Verbindung <*satisfy*> mit der Anforderung verknüpft. Die letzte Beziehung verbindet eine Anforderung mit einem Testfall, der diese Anforderung abtestet und durch die Relation <*verify*> markiert ist [42].

Blockdefinitionsdiagramm

Das Blockdefinitionsdiagramm (BDD) gehört zu der Klasse der Strukturdiagramme. Das BDD ist eine Modifikation des UML-Klassendiagramms. Über das BDD kann die Systemstruktur sowie die Hierarchie der Systemkomponenten und -beziehungen dargestellt werden. Es bildet die strukturellen Beziehungen eines Systems anhand der einzelnen Blöcke als ein Blackbox-System ab. Abbildung 13 zeigt auf der linken Seite die Blocknotation eines BDD und auf der rechten Seite das zugehörige BDD mit dargestellten Relationen. Im Gegensatz zum Klassendiagramm der UML, welches nur aus Attributen und Methoden besteht, enthält ein BDD zusätzlich die folgenden Attribute:

- einer Teilmenge von BDDs, in die der Block aufgeteilt ist,
- Beziehungen zu anderen Blöcken, die als Assoziation oder Aggregation definiert sind,
- quantifizierbare Merkmale in Form von Werttypen,
- sowie Funktionen.

Internes Blockdiagramm

Das interne Blockdiagramm (IBD) dient der Beschreibung der internen Struktur eines Blockes. Das IBD besteht erstens aus Instanzen von BDDs, die als Eigenschaftsblock (engl. property block) definiert werden. Zweitens aus Schnittstellen zwischen den Eigenschaftsblöcken, die als Ports dargestellt werden. Drittens aus Verbindungen zwischen den Ports, die als Fluss definiert werden. Abbildung 14 zeigt ein IBD am Beispiel eines PC-Systems. Das System besteht aus einem Monitor, einem PC, einer Maus und einer Tastatur, die als Eigenschaftsblöcke definiert sind. Die einzelnen Blöcke sind durch die Schnittstellen, USB und VGA, miteinander verbunden. Die

Schnittstellen bilden zudem den Informationsfluss ab. Dieser ermöglicht es unidirektionale bzw. bidirektionale Verbindungen darzustellen [42].

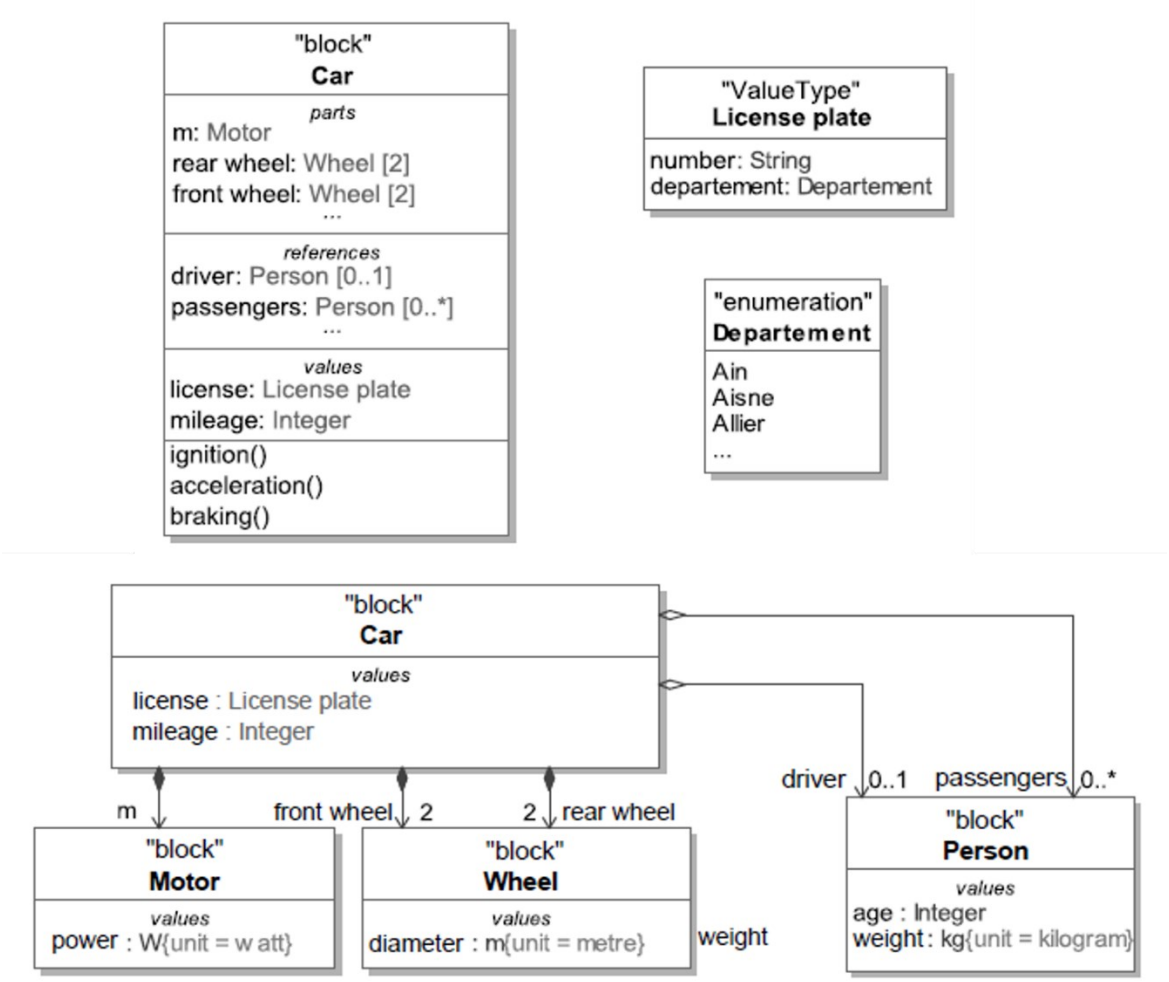


Abbildung 13 - Blockdefinitionsdiagramm unter SysML[50]

Zusicherungsdiagramm

Das parametrische Zusicherungsdiagramm (PAR) ist der zweite neue Diagrammtyp unter SysML und gehört zu der Klasse der Strukturdiagramme. Es dient der Spezifikation von parametrisierten Werten wie der Leistung, Zuverlässigkeit oder der physikalischen Gesetze [50]. Diese Regeln oder mathematischen Gleichungen werden mittels spezieller Beschränkungsblöcke (Constraint-Blöcke) definiert und in einem PAR instanziiert und detailliert dargestellt. Abbildung 15 stellt ein PAR unter SysML grafisch dar.

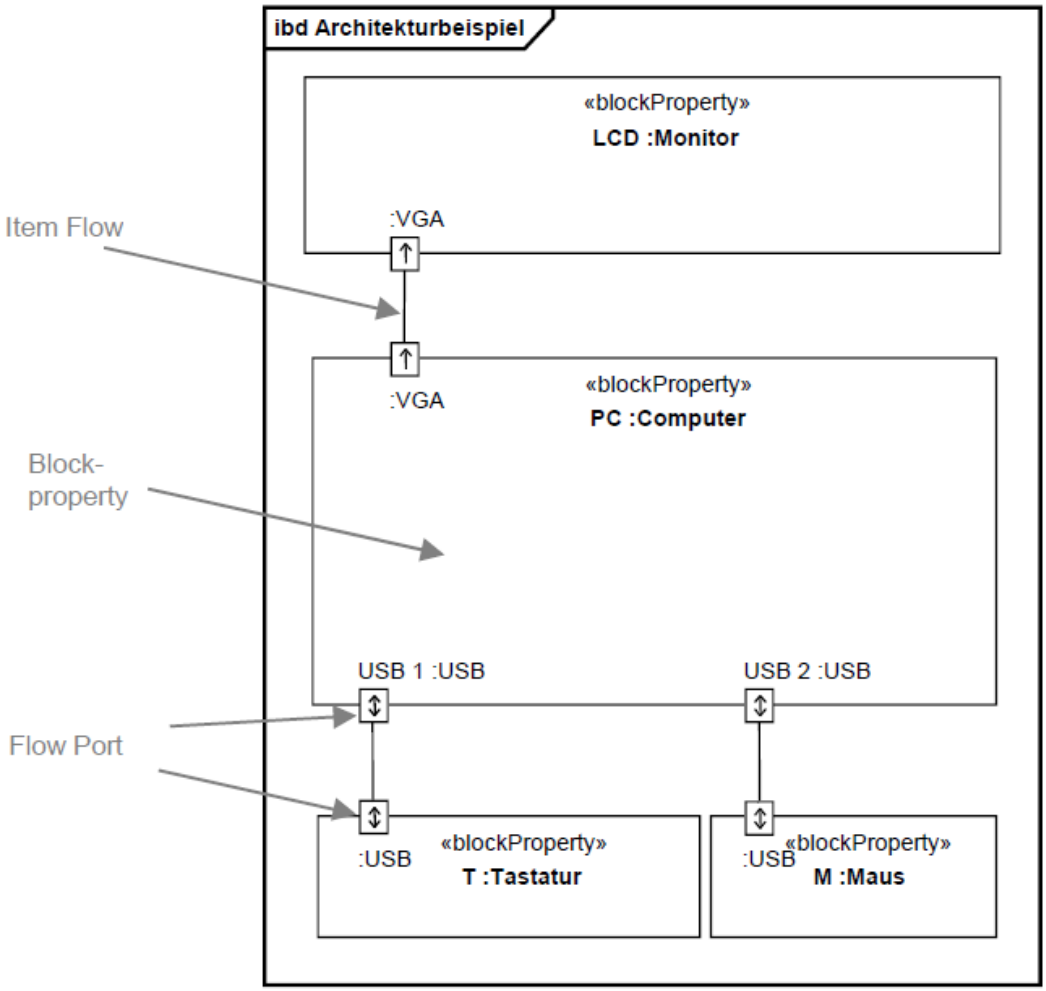


Abbildung 14 - Internes Blockdiagramm unter SysML[42]

Die Parameter einer Beschränkung stellen die Ein- und Ausgaben eines Blocks dar. Die Regel oder Funktion des Blocks wird durch die Funktion einer Beschränkung festgelegt. Die parametrischen Diagramme sind dazu gedacht, formale Zusammenhänge mit dem Ziel zur Unterstützung der frühzeitigen Systemanalyse abzubilden. Die Darstellung des PAR basiert auf dem IBD, welches die interne Struktur der Blöcke über Ports abbildet und durch Constraints-Blöcke erweitert wird [42].

2.4.1.3 Architecture Analysis & Design Language (AADL)

Die Architecture Analysis & Design Language, kurz *AADL* [52], ist ein internationaler Standard der SAE [53] für die formelle Beschreibung von Systemarchitekturen. AADL bietet die Möglichkeit, umfangreiche Hardware- und Software-Architekturen zu modellieren. Des Weiteren können die einzelnen AADL-Modelle durch relevante Sicherheitsinformationen angereichert werden, um automatisierte Verlässlichkeitsanalysen durchzuführen. Im Anwendungsgebiet der sicherheitskritischen eingebetteten Systeme gewinnt die in 2004 durch Peter Feiler entwickelte Architekturbeschreibungssprache immer mehr an Popularität [43].

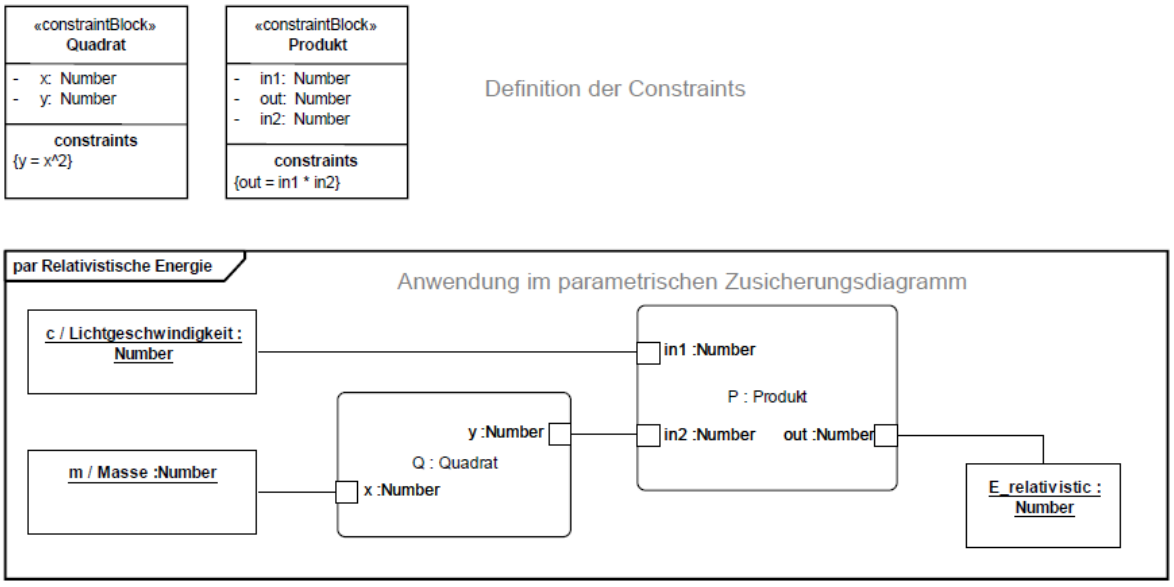


Abbildung 15 - Parametrische Zusicherungsdiagramm unter SysML[42]

AADL ist eine grafische und textuelle Beschreibungssprache. Die zentralen Komponenten bei AADL besitzen einen Typ und eine oder mehrere Implementierungen dieses Typs. Dieser Zusammenhang ist unter SysML durch das BDDs und das IBDs dargestellt. Ein AADL-Typ repräsentiert die funktionale Schnittstelle der Komponente und ist als Blackbox zu betrachten. Sie dient als Referenzmodell für verschiedene Implementierungen. Die Implementierung beschreibt die interne Struktur einer Komponente und verfeinert vorhandene Eigenschaften. AADL-Komponenten können in vier Typklassen unterteilt werden [43]:

- Softwarekomponenten,
- Hardwarekomponenten,
- Verbundkomponenten,
- generische Komponenten.

Softwarekomponenten können zum Beispiel vom Typ *Thread, Prozess* oder *Daten* sein. Hardwarekomponenten stellen unter anderem *Prozessoren*, *Kommunikationsmedien* oder *Geräte* dar. Ein Verbund oder auch Kompositmodell ist eine Systemabbildung, die eine Integration von *Software-*, *Hardware-* und anderen *Systemkomponenten* darstellt. Die letzte Kategorie stellt die a*bstrakten Modelle* dar und dient der konzeptionellen Abbildung von Systemkomponenten, die von neutraler Natur sind oder noch keiner spezifischen Klasse angehören [54].

Abbildung 16 zeigt oben die textuelle Spezifikation eines Softwareprozesses und unten die dazugehörige grafische Darstellung. Die im Beispiel dargestellte AADL-Komponente besteht aus einem Typ (*process*), einer Schnittstellendefinition (*features*) und einer Beschreibung des Datenflusses (*flows*). Die Schnittstellen werden über Ports definiert und können Daten-, Event- oder auch Event-Data-Ports darstellen. Des

Weiteren beschreibt der Port die Flussrichtung der Schnittstelle. Die Flow-Definition ist eine abstrakte Darstellung, die anzeigt, wie sich der Informationsfluss in einer Komponente ausbreitet. Man unterscheidet zwischen der *Quelle (engl. flow source)*, dem *Pfad (engl. flow path)* und der *Senke (engl. flow sink)* eines Datenflusses. Der Informationsfluss in Abbildung 16, stellt einen Datenpfad von Porteingängen auf den Portausgang dar.

```
process Data_Sampling
    features
        ACC_in: in data port;
        Gyro_in: in data port;

        Sample_out: out data port;
    flows
        sampling_path_gyro: flow path Gyro_in -> Sample_out;
        sampling_path_acc: flow path ACC_in -> Sample_out;
end Data_Sampling;
```

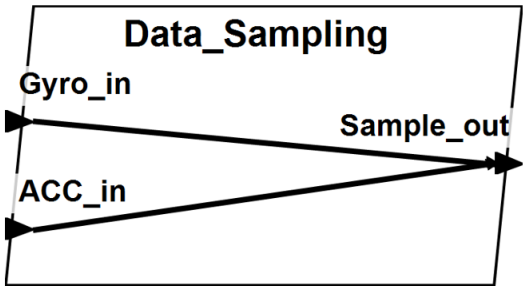


Abbildung 16 - Textuelle und grafische Darstellung eines Prozesses unter AADL

Eine detaillierte Spezifikation des Prozesses kann durch eine oder mehrere Implementierungen erfolgen. Abbildung 17 zeigt eine mögliche Implementierung. Eine Implementierung besteht aus Subkomponenten (*subcomponents*), die die interne Struktur und Hierarchie der AADL-Komponente abbilden. Die einzelnen Subkomponenten verweisen wiederum auf Implementierungen anderer AADL-Komponenten. In unserem Beispiel besteht die *Data_Sampler.detail* Implementierung aus drei Threads, je einer Überwachungseinheit (Watchdog) für die Eingangssignale und einem Thread zur Erstellung der Samples. Um den abstrakten Informationsfluss aus Abbildung 16 zu spezifizieren, müssen die Schnittstellen aller Komponenten miteinander verbunden werden. Dafür werden Port-Verbindungen (*connections*) definiert. Der Datenfluss kann somit durch Angabe von Port-Verbindungen und Datenflüssen der Subkomponenten implementiert werden.

```
process implementation Data_Sampling.detail
    subcomponents
        WD_acc: thread SW_thread_DS::Watchdog_Datasampling.impl;
        WD_Gyro: thread SW_thread_DS::Watchdog_Datasampling.impl;
        Data_Sampling: thread SW_thread_DS::Datasampling.impl;
    connections
        con_gyro_in: port Gyro_in -> WD_Gyro.WD_in;
        con_gyro_wd_out: port WD_Gyro.WD_out -> Data_Sampling.Gyro_in;
        con_acc_in: port ACC_in -> WD_acc.WD_in;
        con_acc_wd_out: port WD_acc.WD_out -> Data_Sampling.ACC_in;
        con_sample_out: port Data_Sampling.sample_out -> Sample_out;
    flows
        sampling_path_acc: flow path ACC_in -> con_acc_in -> WD_acc.WD_flow_path ->
        con_acc_wd_out -> Data_Sampling.acc_flow_path -> con_sample_out -> Sample_out;
        sampling_path_gyro: flow path Gyro_in -> con_gyro_in -> WD_Gyro.WD_flow_path ->
        con_gyro_wd_out -> Data_Sampling.gyro_flow_path -> con_sample_out -> Sample_out;
end Data_Sampling.detail;
```

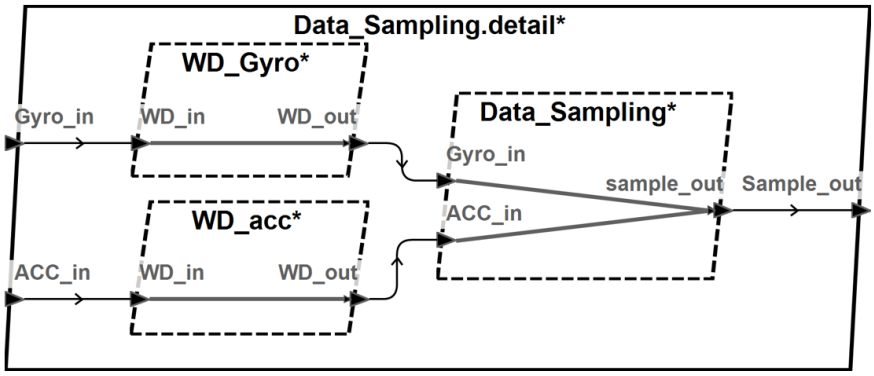


Abbildung 17 - Implementierung eines Prozesses unter AADL

Neben der Syntax und Semantik der AADL-Grundsprache gibt es zusätzliche Erweiterungen, die der Beschreibungssprache weitere Funktionalität und domänenspezifische Eigenschaften bieten.

Der Verhaltensmodell Anhang (*Behavior Model Annex*) bietet eine Möglichkeit, das Systemverhalten und damit dynamische Prozessabläufe abzubilden [55]. Ein Verhaltensmodell stellt einen Zustandsautomaten dar. Die Zustände eines Verhaltensmodells bilden verschiedene Betriebsmodi ab, und die Transitionen zwischen den Zuständen stellen Ereignisse im System dar. Dies hat zum Ziel, dynamische Rekonfigurationen zu modellieren und Ereignisse abzubilden, die die Systemübergänge aktivieren. Abbildung 18 zeigt die grafische Darstellung eines Verhaltensmodells unter AADL. Der im Beispiel dargestellte LVAD-Kontroller hat zwei Betriebsmodi, den Fall-Back- und den Betriebsbereit-Modus. Um einen Betriebsmoduswechsel in AADL darzustellen, dienen Eventports der Subkomponenten als Aktivierung einer Zustandsüberführung im Verhaltensmodell. Der Watchdog-Prozess dient als Überwachungskomponente und kann den LVAD-Kontroller im Falle eines Fehlers vom Betriebsbereit-Modus in den Fall-Back-Modus schalten.

Die Unterstützung von geeigneten Werkzeugen zur Erstellung von AADL-Modellen, wie die Open Source AADL Tool Environment (OSATE) [56], ermöglicht die visuelle

Darstellung von aktiven bzw. inaktiven Komponenten in verschiedenen Betriebsmodi. Im unteren Beispiel werden die Softwarekomponenten und Verbindungen, die im Betriebsmodus *Fall_Back_Mode* aktiv sind, rot dargestellt.

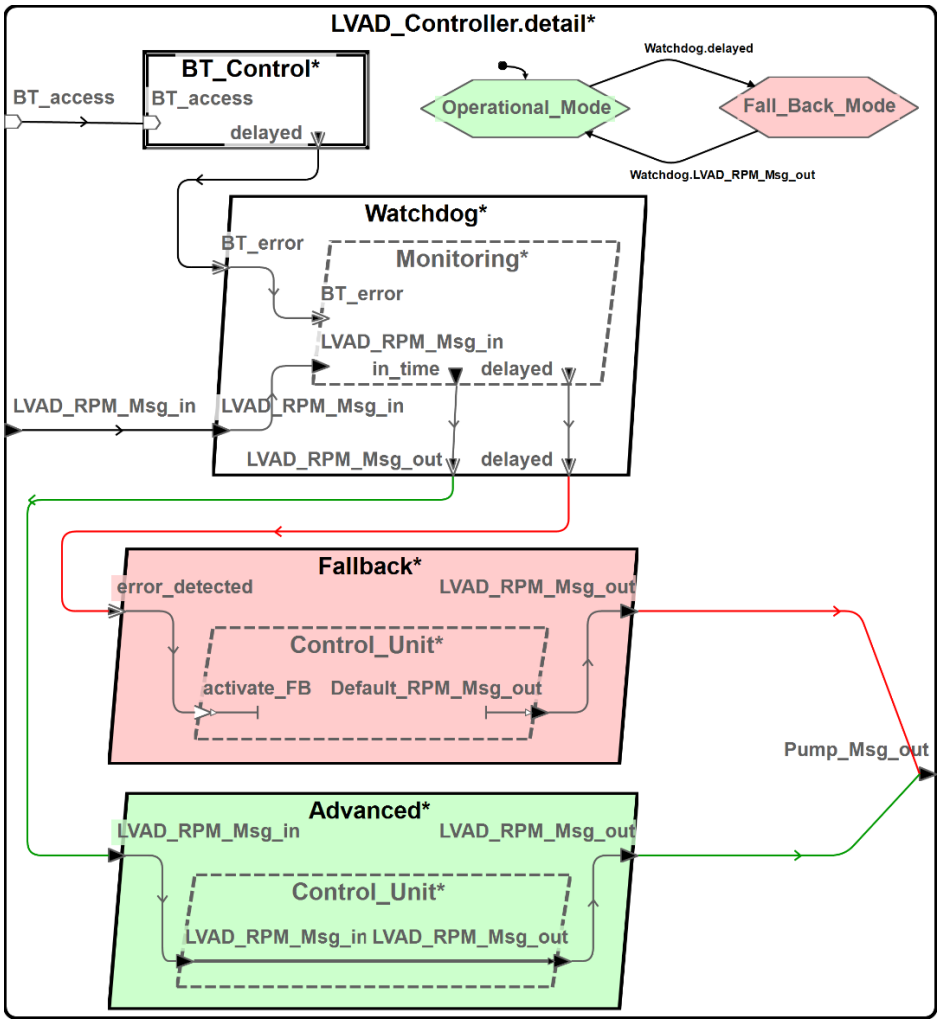


Abbildung 18 - Beispiel einer dynamischen Rekonfiguration unter AADL

Eine zusätzliche Erweiterung der AADL-Sprache ist der Fehlermodell-Anhang (*Error Model Annex - EMV2*) [57]. Durch diesen Anhang können Fehlerbibliotheken für die AADL-Modelle definiert werden. Diese Bibliotheken bestehen aus der Spezifikation von Fehlertypen und Fehlermodellen. AADL bietet eine Vielzahl von vordefinierten Fehlertypen [58]:

- Servicefehler,
- Wertefehler,
- Zeitfehler,
- Fehler bei der Replikation.

Abbildung 19 stellt die formelle Definition eines Fehlermodells unter AADL dar. Error Typen können definiert werden und im gesamten Modell verwendet werden. Zusätzlich kann man generische Zustandsautomaten definieren, die den Modellen zur Verfügung stehen. Diese bestehen aus Fehlerzuständen, die eine Komponente

annehmen kann [59]. Im Beispiel gibt es den fehlerfreien Zustand (*operational*) sowie einen fehlerhaften Zustand (*failed*). Transitionen (*t1, t2*) spezifizieren die Zustandsüberführung durch das Auftreten von Fehlerereignissen (*failure, repair*). Ähnlich der Flow-Spezifikation besitzen Fehler im Modell einen Entstehungspunkt (*error source*), verbreiten sich (*error propagation*) und haben eine Senke (*error sink*). Abbildung 20 zeigt die Watchdog-Komponente aus Abbildung 17, die in ihrer Implementierung durch ein Fehlermodell erweitert wird. Neben den Fehlertypen greift die Komponente auf das Fehlerverhalten des spezifizierten Fehlerautomaten *simple* zurück. Abstrakt werden Fehlerverbreitung und der interne Fehlerfluss definiert. Fehler können weitergeleitet, isoliert oder in andere Fehler transformiert und weitergeleitet werden. Die Implementierung im Beispiel legt fest, dass dann, wenn sich die Komponente im Fehlerzustand *operational* befindet und ein verspäteter Sensorwert (*LateSensorData*) ankommt, der Sensorwert nicht weitergeleitet wird. Der Fehler wird somit zu einem *NoSensorData* Fehler transformiert und im System weitergeleitet. Dies ermöglicht eine Fehlerisolation und Fehlerbehandlung. Sollte im Zustand *operational* ein falscher Sensorwert (*WrongSensorData*) am Port anliegen, wird dieser durchgereicht und nicht behandelt.

```
package ErrorLib
public

annex EMV2{**

    error types
        LateValue: type;
        WrongValue: type;
        NoValue: type;
        ServiceError: type;
        NoPower: type;
        LateSensorData: type;
        NoSensorData: type;
        WrongSensorData: type;

    end types;

    error behavior simple
        events
            failure: error event;
            repair: error event;
        states
            operational: initial state;
            failed: state;
        transitions
            t1: operational -[failure]-> failed;
            t2: failed -[repair]-> operational;
    end behavior;

**};

end ErrorLib;
```

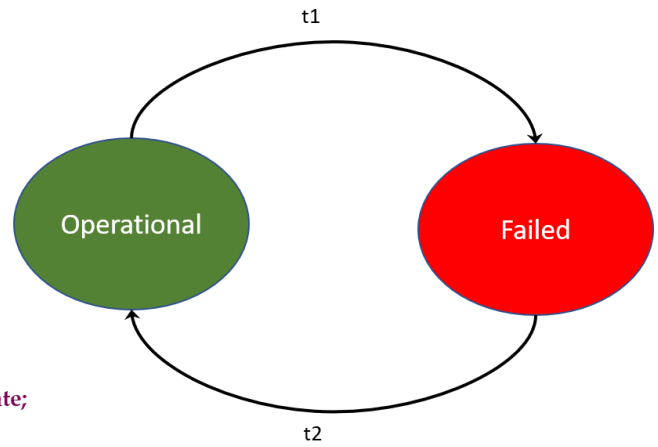


Abbildung 19 - Fehlermodell unter ADDL

Durch den strukturellen Aufbau von softwarebasierten Systemen, die Modellierung von dynamischen Operationszuständen sowie die Anreicherung der Systemarchitektur durch sicherheitsspezifische Informationen bietet AADL die

Grundlage zur Entwicklung und Validation von softwarebasierten SkS. Im Zusammenhang mit dem Modellierungswerkzeug OSATE können grafische Darstellungen der Systemarchitektur und deren Dynamik dargestellt sowie automatisierte Sicherheitsanalysen über den Modellen durchgeführt werden. Eine Auflistung und kurze Beschreibung der von OSATE unterstützten Analysetechniken wird im Folgenden gegeben.

```
thread Watchdog_Datasampling
        features
                WD_in: in data port;
                WD_out: out data port;
        flows
                WD_flow_path: flow path WD_in -> WD_out;
end Watchdog_Datasampling;

thread implementation Watchdog_Datasampling.impl
        annex EMV2 {**
                use types ErrorLib;
                use behavior ErrorLib::simple;

                error propagations
                        WD_in: in propagation {LateSensorData, WrongSensorData};
                        WD_out: out propagation {NoSensorData, WrongSensorData};
                flows
                        ep_wd1: error path WD_in {LateSensorData} -> WD_out {NoSensorData};
                        ep_wd2: error path WD_in {WrongSensorData} -> WD_out {WrongSensorData};
                end propagations;

                component error behavior
                        propagations
                                prop_wd_lv: operational -[WD_in {LateSensorData}]-> WD_out {NoSensorData};
                                prop_wd_wv: operational -[WD_in {WrongSensorData}]-> WD_out {WrongSensorData};
                end component;
        **};
end Watchdog_Datasampling.impl;
```

Abbildung 20 - Verwendung von Fehlermodellen unter AADL

2.4.2 Analysetechniken

Um eine Systembewertung aus Sicht der Verlässlichkeit durchzuführen, erfordert es eine qualitative und quantitative Analyse des Systemverhaltens. Die qualitativen Analysetechniken untersuchen die Beschaffenheit des Systems. Sie dienen der Informationsgewinnung über Systemstrukturen, Interaktionen oder der Interpretation von Systemteilen. Die quantitativen Analysetechniken beschäftigen sich mit messbaren Größen des Systems. Sie dienen der Informationsgewinnung über die Häufigkeit oder die Wahrscheinlichkeit des Auftretens von Ereignissen.

Einen Leitfaden für das Vorgehen, die Techniken und die Dokumentation von Sicherheitsanalysen beschreiben unter anderem der U.S. Militärstandard *MIL-STD-882E* [60], der *ARP4761* [61] Standard für die Entwicklung von Luftfahrtsystemen sowie der *DIN-EN-50126-2* [28] Standard für die Entwicklung von Bahnanwendungen. Sicherheits- und Risikoanalysen unterstützen die Systemverlässlichkeit durch folgende Techniken [62, 63]:

- Reliability Block Diagramm (RBD),
- Fault Tree Analysis (FTA),
- Event Tree Analysis (ETA),
- Failure Mode and Effects Analysis (FMEA),
- Functional Hazard Analysis (FHA),
- Petri Net Analysis (PNA),
- Markov Analysis (MA).

Das AADL-Entwicklungswerkzeug OSATE unterstützt die Sicherheits- und Risikoanalysen für sicherheitskritische eingebettete Systeme durch den *MIL-STD-882 und ARP4761* Standard [64]. Es können durch die Verwendung des EMV2-Fehlermodells die folgenden Sicherheitsanalysen über AADL-Modelle durchgeführt werden:

- Functional Hazard Analysis (FHA) [64],
- Failure Mode and Effects Analysis (FMEA) [64],
- Fault Tree Analysis (FTA) [65],
- Flow Latency Analysis (FLA) [66].

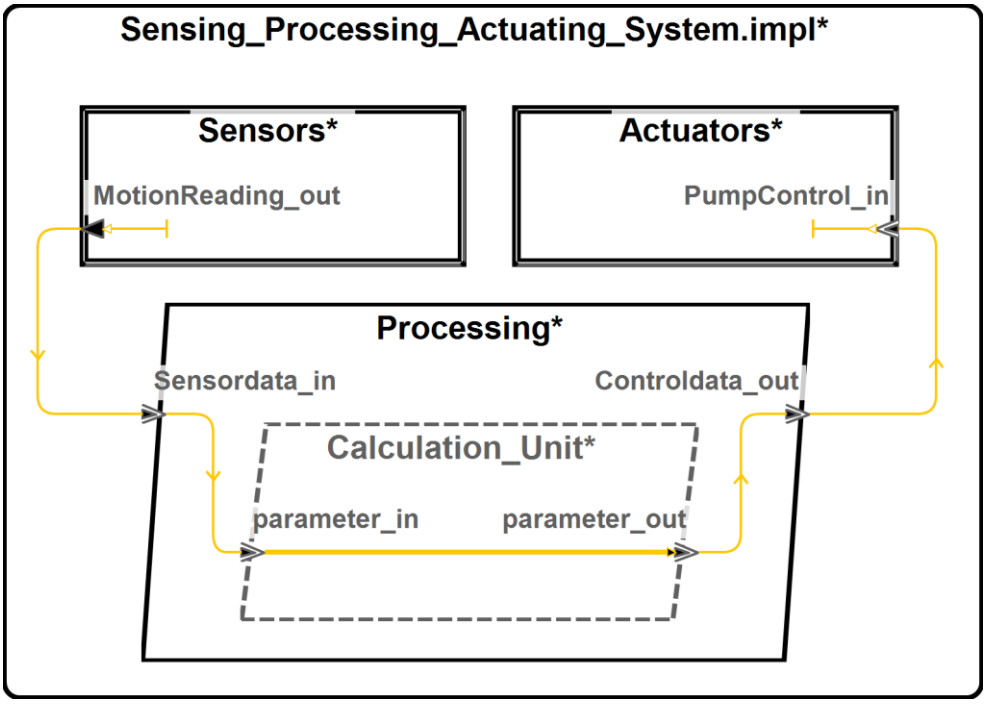


Abbildung 21 - AADL Beispiel eines Steuerungssystems

Abbildung 21 zeigt ein einfaches Steuerungssystem, welches durch AADL modelliert wurde. Das Steuerungssystem besteht aus zwei Geräten, einem Sensor und einem Aktuator, sowie einer softwarebasierten Kontrolleinheit. Im Folgenden werden die von OSATE zur Verfügung gestellten Sicherheitsanalysen sowie die benötigten EMV2-Eigenschaften an diesem Beispiel beschrieben.

2.4.2.1 Functional Hazard Analysis (FHA)

Die Gefahrenanalyse (FHA) dient der Bewertung und Dokumentation von potenziellen Gefahren in einem System. EMV2 bietet eine Vielzahl an Eigenschaften, die einem AADL-Modell hinzugefügt werden können [64]. Abbildung 22 zeigt, wie ein Device-Modell unter AADL durch Hazard-Eigenschaften erweitert werden kann.

```
device Motion_Sensor
    features
        MotionReading: out event data port;
    flows
        flow_source_1: flow source MotionReading;
    properties
        period => 50ms;
        Deadline => 9ms;
        Compute_Execution_Time => 1ms .. 3ms;
end Motion_Sensor;

device implementation Motion_Sensor.impl
    properties
        dispatch_protocol => periodic;
    annex EMV2 {**
        use types ErrorLib;
        use behavior ErrorLib::Simple;

        error propagations
            MotionReading: out propagation {ServiceError};
        flows
            es1: error source MotionReading {ServiceError} when Failed;
        end propagations;

        properties
            EMV2::OccurrenceDistribution => [ProbabilityValue => 1.0e-02;Distribution => poisson;] applies to failure;

            EMV2::severity => 1 applies to es1.ServiceError;
            EMV2::likelihood => C applies to es1.ServiceError;
            EMV2::hazards => ([
                failure => "Loss of sensor readings";
                description => "No motion readings due to sensor failure";
                comment => "Becomes major hazard if no redundant sensor";
            ]) applies to es1.ServiceError;
    **};
end Motion_Sensor.impl;
```

Component	Error Model Element	Description	Failure	Severity	Likelihood	Comment
Sensors	"ServiceError on es1"	"No motion readings due to sensor failure"	"Loss of sensor reading	1	C	"Becomes major hazard if no redundant sensor"
Actuators	"failure on failure"	"Pump does not react and work correctly"	"Actuator failure"	1	E	
Processing/ Calculation_Unit	"WrongValue on es1"	"Classification of the movement incorrectly"	"error in calculation"	1	D	"Without additional components no detection and correction of the error possible"

Abbildung 22 - Definition von Hazard-Eigenschaften des Sensor-Modells und FHA-Analyse durch OSATE

Die Hazard-Eigenschaften werden mit einer Fehlerquelle, einem Fehlerzustand oder einem Fehlerereignis verknüpft. Fehlerquellen, die in einem System entstehen, listet die FHA auf. Die Analyse setzt unter AADL eine Fehlerquelle (error source) sowie eine Gefahrenbeschreibung (error properties) in einem Modell voraus. Im unteren Beispiel werden dem ServiceError neben einer Beschreibung und Kommentaren ein Schweregrad (engl. severity) sowie eine mögliche Wahrscheinlichkeit (engl.

likelihood) für das Auftreten angegeben. Dies ermöglicht ein frühzeitiges Verständnis über die Fehlerentstehungen in einem System. OSATE kann aus den zusätzlichen Informationen ein Hazard-Dokument anfertigen, welches im Systemumfeld von SkS und dem möglichen Zertifizierungsprozess erforderlich ist, siehe Abbildung 22 unten.

2.4.2.2 Failure Mode and Effects Analysis (FMEA)

Die Fehlermöglichkeits- und -einflussanalyse (FMEA), auch Auswirkungsanalyse (engl. Fault Impact Analysis) genannt, wertet die gesamte Fehlerausbreitung in einem System aus. Sie erweitert die FHA durch die zusätzlichen Informationen des Fehlerflusses und der Fehlersenken. FMEA ist eine induktive Fehlerauswirkungsanalyse und verfolgt den bottom-up Ansatz. Die FMEA listet alle Komponenten auf, die durch eine Fehlerquelle betroffen sind und bildet somit eine Fehlerausbreitung im System ab und dient der Vorwärtsfehleranalyse [62].
Dies ermöglicht eine strukturierte Auflistung aller Komponenten, die durch einen Fehler betroffen sind und dient der frühzeitigen Sicherheitsbewertung. Abbildung 23 zeigt das automatisiert erstellte FMEA-Dokument und listet diejenigen Fehler aus der FHA auf, die sich im System über vorhandenen Schnittstellen ausbreiten können.

Component	Initial Failure Mode	1st Level Effect	Failure Mode	second Level Effect	Failure Mode
Sensors	failed	{ServiceError} MotionReading_out -> Processing. Calculation_Unit: parameter_in	Processing. Calculation_Unit {ServiceError}	{NoValue} parameter_out -> Actuators: PumpControl_in	Actuators {NoValue} [Masked]
Processing. Calculation_Unit	failed	{WrongValue} parameter_out -> Actuators: PumpControl_in	Actuators {WrongValue} [Masked]		

Abbildung 23 - Beispiel einer FMEA-Analyse durch OSATE

Die einzelnen Level der Analyse beschreiben die Komponenten, die auf dem Pfad einer Fehlerausbreitung involviert sind. Eine Fehlersenke wird als maskierter (Masked) Fehler dargestellt. Dies ermöglicht eine Bewertung und Darstellung von Fehlertoleranzmaßnahmen mit dem Ziel der Fehlereingrenzung bzw.- isolation.

2.4.2.3 Fault Tree Analysis (FTA)

Die Fehlerbaumanalyse (FTA) basiert im Gegensatz zur FMEA-Analyse auf dem top-down Ansatz. FTA ist eine deduktive Fehlerauswirkungsanalyse, die sich darauf konzentriert, alle potenziellen Fehler, die zu einem Systemausfall führen, zu identifizieren und darzustellen und dient der Rückwärtsfehleranalyse [65]. Ein Fehlerbaum (FT) besteht aus Fehlerereignissen, die die Blätter eines FT darstellen und logischen Verknüpfungen, die die inneren Knoten des FT repräsentieren. Die Wurzel bildet den zu untersuchenden Ausfall oder einen Fehlerzustand ab. Die logischen Verknüpfungen basieren zum Beispiel auf Und-Verknüpfungen, Oder-

Verknüpfungen, Entweder-Oder-Verknüpfungen oder Priorisierten-Und-Verknüpfungen [67].

Abbildung 24 stellt einen Fehlerbaum für das in Abbildung 21 dargestellte Steuerungssystem. Das Gesamtsystem befindet sich im Fehlerzustand *Failed* wenn der Aktuator ausfällt oder der Sensor bzw. die Kalkulation fehlerhaft ist. Jedes Fehlerereignis hat eine Fehlerrate, mit der die Häufigkeit des Auftretens angegeben wird.

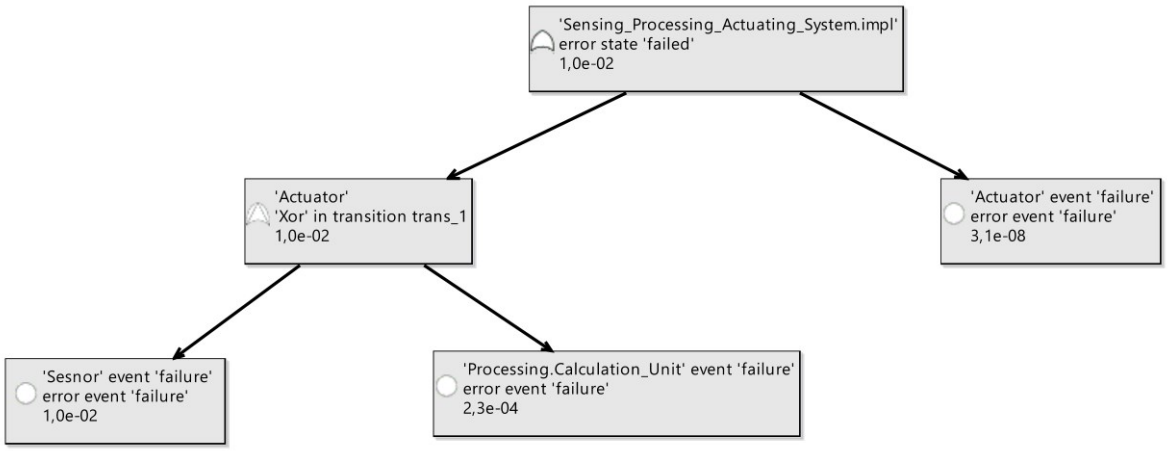


Abbildung 24 - Beispiel einer FTA-Analyse durch OSATE

Über die mathematischen Regeln der logischen Verknüpfungen ergibt sich eine Fehlerrate des Auftretens des zu untersuchenden Ausfalls. In AADL werden die Fehlerraten durch die Eigenschaft *OccurrenceDistribution* angegeben. Die Implementierung des Aktuator-Modells ist in Abbildung 25 dargestellt. Neben der quantitativen Analyse, die die Fehlerrate berechnet, liegt das Interesse auf der Berechnung der minimalen Schnittmengen (engl. minimal cut set), die zu einem Top-Fehlerereignis führen. Dieser minimale Schnitt ist eine qualitative Darstellung und wichtiger Bestandteil einer Sicherheitsanalyse [67].

2.4.2.4 Flow Latency Analysis (FLA)

Die Latenzanalyse (FLA) dient der Bewertung der Reaktionszeit bzw. Verzögerungszeit eines Systems. Neben der Definition von Datenflüssen in einem AADL-Modell müssen zusätzliche Angaben von zeitlichen Abläufen, wie die Ausführungszeit, Grenzwerte oder die Periode der Ausführung angegeben werden, so dass OSATE diese Zeitinformationen gegenüber der geforderten Latenzzeit bewerten kann [66]. Um eine FLA durch OSATE durchzuführen, muss ein Ende-zu-Ende-Datenfluss im System definiert sein. Dieser besteht aus einer Quelle und führt über mehrere Pfade zu einer Datensenke. Wichtig ist, dass es sich hier um Datenflüsse und Schnittstellenverbindungen handelt und nicht um Fehlerflüsse des EMV2. Abbildung 21 stellt den Ende-zu-Ende-Fluss des Steuerungssystems gelb dar.

```
device Pump_Actuator
	features
		PumpControl: in event data port;
	flows
		flow_sink_1: flow sink PumpControl;
	properties
		period => 50ms;
		deadline => 9ms;
		Compute_Execution_Time => 1ms .. 3ms;
end Pump_Actuator;

device implementation Pump_Actuator.impl
	properties
		Dispatch_Protocol => Aperiodic;
	annex EMV2 {**
		use types ErrorLib;
		use behavior ErrorLib::Simple;

		error propagations
			PumpControl: in propagation {NoValue, WrongValue};
		flows
			es1: error sink PumpControl {NoValue, Wrongvalue};
		end propagations;

		component error behavior
			transitions
				trans_1: operational -[PumpControl {NoValue} or PumpControl {WrongValue}]-> Failed;
		end component;

		properties
			EMV2::OccurrenceDistribution => [ProbabilityValue => 3.1e-08;Distribution => poisson;]
			applies to failure;

			EMV2::severity => 1 applies to failure;
			EMV2::likelihood => A applies to failure;
			EMV2::hazards => ([
				failure => "Actuator failure";
				description => "Pump does not react and work correctly";
			]) applies to failure;

			EMV2::severity => 1 applies to es1.NoValue;
			EMV2::likelihood => C applies to es1.NoValue;
			EMV2::hazards => ([
				failure => "No pump control";
				description => "No control signal for the actuator";
			]) applies to es1.NoValue;
	**};
end Pump_Actuator.impl;
```

Abbildung 25 - Beispiel eines Aktuator-Modells unter AADL

Die Implementierung des Ende-zu-Ende-Flusses ist in Abbildung 26 dargestellt und hat eine Anforderung der Latenz von maximal 30ms. Durch die genaue Spezifikation des Datenflusses kann OSATE die einzelnen Komponenten und Verbindungen auf zeitliche Abhängigkeiten überprüfen und analysieren.

Abbildung 27 bildet die durch OSATE durchgeführte FLA-Analyse und die daraus folgende Latenzbewertung ab. Die Analyse bezieht sich hier auf die Worst-Case-Analyse über die Grenzwerte (Deadline) der jeweiligen Komponenten. Jede Komponente, die auf dem Ende-zu-Ende-Datenfluss liegt (Sensor, Kalkulation und der Aktuator) wird mit den zeitlichen Eigenschaften aufgelistet.

```
system Sensing_Processing_Actuating_System
end Sensing_Processing_Actuating_System;

system implementation Sensing_Processing_Actuating_System.impl
        subcomponents
                Sesnor: device Sensor::Motion_Sensor.impl;
                Actuator: device Actuator::Pump_Actuator.impl;
                Processing: process Controlling::Control_Unit.impl;
        connections
                c1: port Sesnor.MotionReading -> Processing.Sensordata_in;
                c2: port Processing.Controldata_out -> Actuator.PumpControl;
        flows
                etef_1: end to end flow Sesnor.flow_source_1 -> c1 -> Processing.flow_path_1 -> c2 -> Actuator.flow_sink_1
                {latency => 10ms..30ms;};

        annex EMV2 {**
                use types ErrorLib;
                use behavior ErrorLib::simple;

                composite error behavior
                        states
                                [Actuator.Failed]-> Failed;
                end composite;
        **};
end Sensing_Processing_Actuating_System.impl;
```

Abbildung 26 - Beispiel eines Ende-zu-Ende-Datenfluss im AADL-Systemmodell

Die Werte werden aufsummiert und mit dem erwarteten Latenzwert verglichen. Das im Beispiel spezifizierte System erfüllt die geforderte Anforderung aber aus zeitlichen Gründen nicht.

Result	Min Actual	Min Method	Max Actual	Max Method
device Sensors	0.0ms	first sampling	0.0ms	first sampling
device Sensors	1.0ms	processing time	9.0ms	deadline
connection Sensors.MotionReading_out -> Processing.Calculation_Unit.parameter_in	0.0ms	no sampling/ queuing latency	0.0ms	no sampling/ queuing latency
thread Processing.Calculation_Unit	3.0ms	processing time	15.0ms	deadline
connection Processing.Calculation_Unit.parameter_out -> Actuators.PumpControl_in	0.0ms	no sampling/ queuing latency	0.0ms	no sampling/ queuing latency
device Actuators	1.0ms	processing time	9.0ms	deadline
Latency Total	5.0ms		33.0ms	
Specified End To End Latency	10.0ms		30.0ms	
End to end Latency Summary				
WARNING	Minimum actual latency total 5,00ms less than expected minimum end to end latency 10,0ms (faster actual minimum response time)			
ERROR	Maximum actual latency total 33,0ms exceeds expected maximum end to end latency 30,0ms			
WARNING	Jitter of actual latency total 5,00..33,0ms exceeds expected end to end latency jitter 10,0..30,0ms			

Abbildung 27 - Beispiel einer FLA-Analyse durch OSATE

Zwar ist das System im besten Fall schneller als erwartet, im schlimmsten Fall kann jedoch die zeitliche Einschränkung von 30ms nicht garantiert werden. Siehe hier auch Abbildung 27 Warnungen und Error. Diese Analysetechnik ermöglicht die Systembewertung auf Grundlage von zeitlichen Beschränkungen und kann die Verzögerungszeit für eine Prozesskette frühzeitig im Entwicklungsprozess gegen geforderte zeitliche Anforderungen prüfen [68].

3 VERWANDTE ARBEITEN

Dieses Kapitel beschäftigt sich mit aktuellen Forschungsgebieten und -arbeiten, die thematisch im Zusammenhang mit der vorliegenden Dissertation stehen. Zuerst werden Problemstellungen und Lösungsansätze aus dem Bereich der Verlässlichkeit von vernetzten IoT-Systemen dargestellt. Anschließend wird das Problem der Aortenklappeninsuffizienz (AI) durch die langfristige Unterstützung von LVAD-Systemen beschrieben. Im Anschluss daran werden aktuelle Arbeiten aus dem Bereich der körperlichen Aktivitätserkennung durch mobile sensorunterstützte IoT-Geräte dargestellt. Abschließend werden aktuelle Arbeiten aus dem Bereich der modellbasierten Entwicklung durch AADL beschrieben.

3.1 Verlässlichkeit in IoT-Systemen

Kuan-Hsun Chen berichtet in seiner Dissertation über die Ausführung von verlässlicher Anwendungssoftware unter der Verwendung unzuverlässiger Hardware [69]. Schwerpunkt der Forschung liegt auf der Reduzierung von transienten Hardwarefehlern, so genannten Soft-Errors, im Umfeld von eingebetteten Systemen. Chen entwickelt geeignete Scheduling-Ansätze und Scheduling-Analysen, um Realzeit-Anforderungen im Umfeld von Echtzeitsystemen und *Software-Implemented Hardware Fault Tolerance (SIHFT)* Techniken weiterhin zu gewährleisten.

Carlo Alberto Boano u.a. beschreiben Methoden und Werkzeuge, um die verlässliche Kommunikation im IoT-Umfeld zu erhöhen [70]. Sie untersuchen in Ihrer Arbeit die drahtlose Vernetzung von Low-Power IoT-Geräten und geben eine Möglichkeit, auch in kritischen Umgebungen mit ressourcenarmen IoT-Plattformen, eine verlässliche Kommunikation zu gewährleisten.

Yousef Jasemian und Lars Arendt-Nielsen beschreiben in Ihrer Arbeit einen frühen Vorgänger von BDS im medizinischen Umfeld [71]. Sie entwickeln ein mobiles Telemedizinsystem zur Überwachung von Vitalparametern. Dazu untersuchen Sie die drahtlose Kommunikation zwischen den Sensoren und einem Smartphone, welches mittels des Bluetooth-Protokolls realisiert wird. Die Arbeit analysiert die verlässliche Kommunikation anhand der *Paketfehlerrate (PER)* sowie der *Paketverlustrate (PLR)* für die entwickelte Bluetooth-Verbindung.

Daniel Macedo u.a. untersuchen die Verlässlichkeit für IoT-Plattformen mit dem Schwerpunkt der aktiven bzw. passiven Redundanz [72]. Sie analysieren die

Zuverlässigkeit und Verfügbarkeit von IoT-Anwendungen durch Markov-Modelle. Die *mittlere Betriebsdauer bis zum Ausfall (MTTF)* wird für verschiedene Szenarien zwischen den aktiven und passiven Redundanztechniken verglichen und bewertet.

3.2 Pathophysiologie der Aortenklappe durch die Unterstützung von LVAD-Systemen

Die langzeitige Unterstützung durch LVAD-Systeme kann zu einer Veränderung der Aortenklappe im menschlichen Körper führen. Die Forschungsarbeiten von [73–75] untersuchen die Aortenklappenveränderung durch pulsatile und nicht-pulsatile LVAD-Systeme. Die Ergebnisse zeigen, dass bei Verwendung von kontinuierlichen LVAD-Systemen die Druckbelastung in der linken Herzkammer abnimmt und es damit zu einer verkürzten Klappenöffnung kommt. Dieses Problem führt langfristig zu einer dauerhaften Verformung der Aortenklappe und damit zu deren Insuffizienz.

Nikolay Dranishnikov beschreibt in seiner Dissertation, dass es durch eine Aortenklappeninsuffizienz (AI) zu einer Rückführung des Bluts aus der Aorta in den linken Ventrikel und damit zu einer erhöhten Belastung des Ventrikels kommen kann [76]. Dieses Verhalten führt zu einer zunehmenden Blutstauung im Ventrikel und damit zu einer reduzierten Belastbarkeit des Patienten. Des Weiteren wird die Leistung des LVAD-Systems deutlich vermindert. Dieser zyklische Blutfluss, welcher als *totes Volumen* bezeichnet wird, führt zu einem erhöhten Leistungsverbrauch der LVAD-Pumpe.

Um dem Effekt der AI entgegenzuwirken und die Funktionalität des LVAD-Systems auch im langzeitigen Betrieb zu gewährleisten, untersuchen [73, 77] die Veränderung der Druckverhältnisse und der resultierenden Aortenklappenöffnung durch periodische Anpassung der Pumpengeschwindigkeit. Die Ergebnisse der Forschungsarbeiten zeigen, dass eine kurzzeitige Verringerung der Pumpengeschwindigkeit die Druckverhältnisse regulieren, eine Öffnung der Aortenklappe ermöglichen und eine Insuffizienz der Aortenklappe verhindern kann.

3.3 Bewegungserkennung mittels smarter IoT-Sensorik

Die Arbeiten [78, 79] geben einen Überblick der menschlichen Aktivitätserkennung (HAR) durch die Verwendung von tragbarer Sensorik und Maschinenlernverfahren. Sie beschreiben die einzelnen Methoden, die aktuell für maschinelles Lernen zur Aktivitätserkennung Stand der Technik sind:

1. Auswahl der Aktivitäten, die klassifiziert werden sollen,
2. Datenerhebung,
3. Merkmale extrahieren,
4. Training und Klassifikation.

Die Arbeiten zeigen einen aktuellen Stand der Technik im Bereich der Sensordaten durch Body-Sensoren und Umgebungssensoren. Des Weiteren werden aktuelle Frameworks, online wie auch offline, und Algorithmen zur Extrahierung von Aktivitätsmerkmalen vorgestellt.

Media Anugerah Ayu u.a. entwickeln ein HAR-System, welches auf die Accelerometer-Daten eines Smartphones zurückgreift [80]. Es können fünf verschiedene Aktivitäten klassifiziert werden. Zum Trainieren und Klassifizieren der Merkmale wird das WEKA-Data-Mining-Tool verwendet. Dieses bietet eine Vielzahl an vordefinierten Algorithmen zur Klassifikation von Daten. Das HAR-System wird durch zwei verschiedene Methoden untersucht. Erstens durch den Nächste-Nachbarn-Klassifikator und zweitens durch den Naive-Bayes-Klassifikator.

Die Arbeit von Robert-Andrei Voice u.a. beschreibt eine weitere Möglichkeit der Aktivitätserkennung durch die Verwendung von Smartphone-Sensorik [81]. Neben Accelerometer-Daten werden zusätzlich Gyroskop- und Gravitations-Daten zur Erkennung verwendet. Das HAR-System kann zwischen sechs Aktivitäten unterscheiden. Der Algorithmus zum Trainieren und Analysieren der Klassifikation basiert auf einem künstlichen neuronalen Feedforward-Netzwerk (MLP).

3.4 MBSE anhand von AADL

Brian Larson u.a. untersuchen in Ihrer Arbeit, am Beispiel eines kleinen Medizingeräts, wie die AADL-Modellierung bei der Entwicklung von SkS hilfreich ist [82]. Als Anwendungsfall wird die Systemarchitektur eines Säuglingsinkubators (Isolette), explizit die Wärmeregulierung, modelliert und untersucht. Zum Einsatz kommt der EMV2-Anhang von AADL, um Fehler und das Fehlerverhalten der Isolette-Architektur zu bewerten. Durch die Modellierung von Fehlerzuständen und Fehlerausbreitungen, werden FMEA-, FHA- und FTA-Risikobewertungen sowie Sicherheitsanalysen durchgeführt. Die Arbeit zeigt, dass Modellierungssprachen wie AADL, in Kombination mit der Unterstützung von Werkzeugen wie OSATE, eine konsistente und wiederverwendbare Technik liefern, um automatisierte und integrierte Gefahrenanalysen über Systemarchitekturen durchzuführen.

Ein weiteres Beispiel zur Unterstützung der Gefahrenanalyse durch AADL im Bereich der eingebetteten Systeme zeigt die Arbeit von Julien Delange [83]. In seiner Arbeit wird die Systemarchitektur für eine automatische Geschwindigkeitsregulierung im autonom fahrenden Auto bewertet. Die Arbeit zeigt, dass verschiedene alternative Architekturen auf Basis der geforderten Anforderungen und Aspekte modelliert und verglichen werden können. Die Ergebnisse verdeutlichen schon frühzeitig in der Entwicklung, ob Sicherheitsanforderungen erfüllt und genügend Ressourcen vorhanden sind. Dies ermöglicht den Stakeholdern die Kosten, die Komplexität sowie die Verlässlichkeit eines Systems zu bewerten und frühzeitig Entscheidungen über die Struktur der Architektur zu treffen.

Systeme und Anwendungen aus der Raumfahrt unterliegen in der Regel hohen sicherheitskritischen Anforderungen. Die Arbeiten von Michela Muñoz Fernández [84] sowie Peter Feiler u.a. [85] beschreiben den Vorteil von AADL beim Design von softwarebasierten Flugsystemen. Fernández beschreibt die Verwendung von AADL im Juno-Projekt. Das AADL-Modell umfasste die Kommunikationskomponenten, das Command & Data-handling System sowie die softwarebasierte Flugsteuerung. Eine Ende-zu-Ende-Flussanalyse sowie Daten und Speicheranalysen werden auf dem AADL-Modell durchgeführt und offenbaren Kommandofehler, die im Juno-System aufgetreten sind. Die Arbeit von Feiler u.a. beschreibt die Verwendung von AADL im Zusammenhang mit dem SARP-Projekt. Hierzu wird das Mission Data System (MDS) in AADL abgebildet, um nicht-funktionale Eigenschaften, wie Sicherheit und Leistung der eingebetteten Softwaresysteme zu überprüfen.

4 UNTERSTÜTZUNG VON RECHTZEITIGKEITS - UND DIENSTGÜTEANFORDERUNGEN IN BDS

In einem BDS arbeiten verlässliche und unzuverlässige Systemkomponenten im Verbund zusammen. Die unzuverlässigen Systemkomponenten bilden ein gerätebasiertes System (GBS), erweitern den eingeschränkten Funktionsumfang der verlässlichen Systemkomponenten und führen sicherheitskritische Funktionen aus. Das BDS bildet ein Kontrollsystem, in dem unzuverlässige IoT-Sensoren die Umgebungsparameter messen und von unzuverlässigen IoT-Geräten verarbeitet werden. Diese unzuverlässigen IoT-Geräte steuern und regeln die zuverlässigen Systemkomponenten und deren Aktuatoren. Somit unterliegt ein BDS einer erhöhten Funktionssicherheit und stellt an die Aktuatoren die folgenden Zuverlässigkeitsanforderungen:

- Die Stellsignale, die durch das GBS erbracht werden, müssen *rechtzeitig* bei den Aktuatoren ankommen und
- *funktional korrekt* an den Aktuatoren zur Verfügung stehen.

Damit die Aktuatoren zuverlässig und rechtzeitig angesteuert werden können, wird ein Perfektionskern (PK) mit Fall-Back-Funktion im Systemkontext eines BDS benötigt. Dieser PK befindet sich zwischen dem GBS und den Aktuatoren. Abbildung 28 stellt die einzelnen Systemkomponenten eines BDS abstrakt dar.

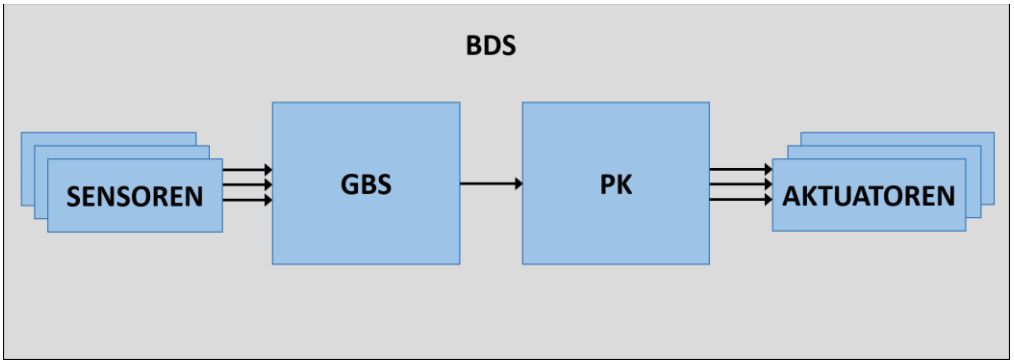


Abbildung 28 - Komponentenbasierte Darstellung eines BDS

Neben den Anforderungen der Zuverlässigkeit muss der erweiterte Funktionsumfang des GBS, trotz PK mit Fall-Back-Funktion, den Aktuatoren so oft und so lange wie möglich zur Verfügung stehen. Diesbezüglich unterliegt ein BDS mit PK zusätzlich einer gewissen Dienstgüte (*engl. Quality of Service (QoS)*). Die Dienstgüteanforderung an ein BDS mit PK und Fall-Back-Funktion ist wie folgt definiert:

- Im Betrieb sollte sich das BDS so *wenig und so kurz* wie möglich im Fall-Back-Modus befinden, um einen Mehrwert und die maximale Funktionalität des GBS zu unterstützen.

Um die Zuverlässigkeitsanforderung und damit die rechtzeitige und korrekte Steuerung der Aktuatoren zu gewährleisten, wird ein PK zwischen das GBS und den Aktuatoren geschaltet. Dieser PK sollte möglichst klein und nach dem KISS-Prinzip (engl. **K**eep **I**t **S**imple and **S**traightforward) entwickelt sein. Eine Fall-Back-Funktion im PK ermöglicht die rechtzeitige Steuerung der Aktuatoren im Falle eines nicht rechtzeitig ankommenden Stellsignals. Dies erlaubt dem BDS zwischen dem funktionserweiterten GBS-Betrieb und dem degradierten Funktionsumfang des ausfallsicheren Betriebs (*engl. Fail-Safe*) zu wechseln.

Um die Dienstgüte eines BDS mit PK zu gewährleisten und zu verbessern, müssen geeignete Maßnahmen vor dem PK angewandt werden. Dies setzt voraus, dass das GBS geeignete Fehlertoleranzmechanismen zur Verfügung stellt, um die Anforderungen an den Aktuatoren bezüglich der *funktionalen Korrektheit* sowie *Rechtzeitigkeit und Dienstgüte* zu erfüllen. Dazu werden im Folgenden geeignete Architekturkonzepte sowie Fehlertoleranzmuster mit dem Schwerpunkt der Rechtzeitigkeit und Dienstgüte in BDS vorgestellt.

4.1 Architekturkonzept

Wie zuvor dargestellt besteht ein BDS aus Sensoren, einem GBS, einem kleinen PK mit Fall-Back-Funktion sowie verschiedenen Aktuatoren. Aus Sicht der Systemarchitektur ergeben sich folgende Eigenschaften und Herausforderungen an diese neue Systemklasse der BDS und deren Zuverlässigkeit:

- Unzuverlässige IoT-Geräte erbringen sicherheitskritische Funktionen für ein zuverlässiges System und erweitern den Funktionsumfang bei Aufrechterhaltung der Systemzuverlässigkeit und Dienstgüte.
- Die einzelnen Systemkomponenten kommunizieren untereinander über unzuverlässige Kommunikationsdienste (Bluetooth, WLAN, GSM).
- Verzögerungen der Abtastrate der unzuverlässigen Sensorik können auftreten.
- Pre- und Postprocessing-Ereignisse unterliegen einer zeitlichen Verzögerung der Datenverarbeitung in den GBS.
- Die zeitliche Abweichung der Stellsignale an den Aktuatoren kann variieren.

- Der gesamte Datenfluss des Kontrollsystems von der Sensorik hin zur Aktuatorik unterliegt einer Ende-zu-Ende-Verzögerung.

Trotz dieser Herausforderungen und Fehlerquellen unterliegt ein BDS Realzeitbedingungen und muss rechtzeitig auf ein Ereignis innerhalb einer vorgegebenen Reaktionszeit ein Stellsignal erbringen.

Um die Realzeitbedingungen in einem BDS zu erfüllen, insbesondere die Gewährleistung der Rechtzeitigkeit für die Stellsignale an den Aktuatoren, stellt Abbildung 29 einen Architekturvorschlag dar, welcher die Gewährleistung der Rechtzeitigkeit und Dienstgüte in BDS unterstützt.

In einem BDS überwachen und messen verschiedene Sensoren die Umgebungsparameter. Diese Sensordaten werden mit einer bestimmten Abtastrate vom GBS (*siehe Abbildung 29 Device Based System*) verarbeitet. Das GBS besteht aus verteilten redundanten IoT-Geräten, die multimodal miteinander interagieren können. Die IoT-Geräte können den Systemkontext des BDS dynamisch erweitern bzw. einschränken. Die einzelnen Geräte bilden mehrere unabhängige Subsysteme, die sich gegenseitig funktional ersetzen und in verschiedenen Betriebsmodi auftreten können (*siehe Abbildung 29 Operation_Mode_0-M*). Die einzelnen IoT-Geräte im GBS kommunizieren bidirektional und tauschen Systeminformationen untereinander aus (*siehe Abbildung 29 processing_data_inout*). Des Weiteren wird der Datenfluss des gesamten GBS (*siehe Abbildung 29 control_signal_out*) durch das Leitstation-Folgestation-Prinzip (*engl. Master-Slave*) geregelt. Die Rolle der Leitstation sowie der Folgestationen regelt das GBS bzw. regeln die IoT-Geräte dynamisch untereinander (*siehe Abbildung 29 Mode_Switching_Unit*). Die gewählte Leitstation verarbeitet die zusätzlichen redundanten Systeminformationen der Folgestationen für eine mögliche Fehlertoleranz. Die zusätzlichen Folgestationen befinden sich im aktiven Stand-by. Dieses Konzept setzt die Ansätze des allmählichen Funktionsabbaus (*engl. Graceful Degradation*) sowie des robusten Softwareaufbaus (*engl. Resilient Software Design*) um.

Das GBS sendet Steuerungsinformationen an den PK (*siehe Abbildung 29 Reliable_Computing-Base*). Dieser PK enthält eine Steuereinheit (*siehe Abbildung 29 Control_Unit*) sowie die zu steuernden Aktuatoren. Die Steuereinheit des PKs arbeitet nach dem Fall-Back-Muster. Ein Watchdog-Timer überwacht die Kommunikationsschnittstelle und kann zeitliche Verzögerungen und Zeitüberschreitungen feststellen. Detektiert der Watchdog-Timer eine zu große Verzögerung und damit kein rechtzeitiges Stellsignal, so aktiviert dieser den Fall-Back-Modus und die Aktuatoren werden rechtzeitig mit einem korrekten funktionssicheren Stellsignal versorgt. Kommt ein Stellsignal rechtzeitig am PK an, arbeitet die Steuereinheit im Operational-Modus und das Stellsignal wird an die Aktuatoren weitergeleitet, um die erweiterte Funktionalität des GBS umzusetzen.

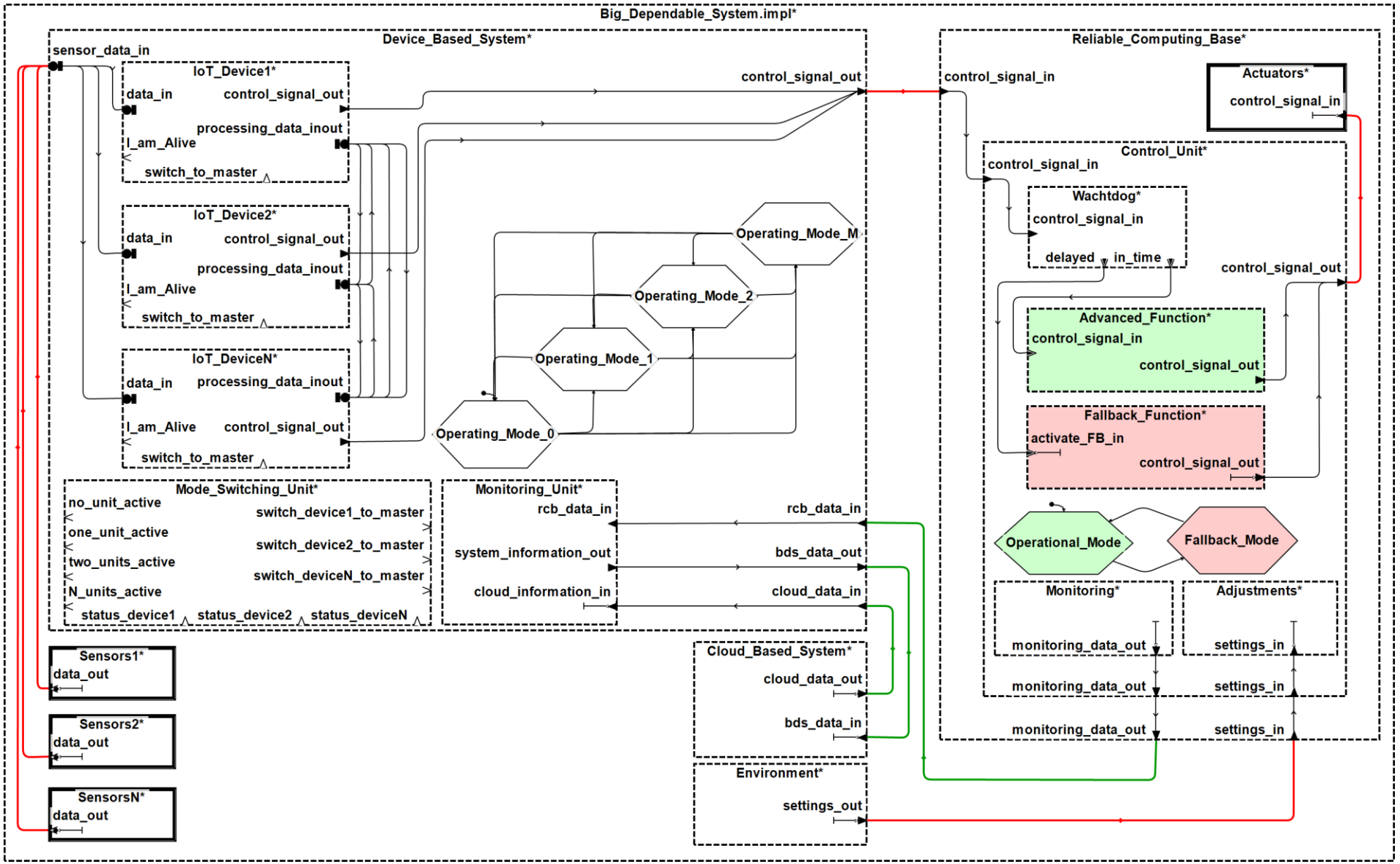


Abbildung 29 - Abstraktes AADL-Architekturkonzept für die Entwicklung von BDS

Zusätzlich zu dem Kontrolldatenfluss zwischen Sensoren und Aktuatoren kann ein Datenfluss zwischen einem Verwaltungssystem (*siehe Abbildung 29 Environment*) und dem PK bestehen. Diese Verbindung ermöglicht es Aktuatorparameter für den Fall-Back-Zustand im Laufe der Zeit anzupassen (*siehe Abbildung 29 settings_in*). Des Weiteren können Aktuatorinformationen (*siehe Abbildung 29 monitoring_data_out*) an das GBS gesendet werden, um diese in den IoT-Geräten zur Laufzeit zu verarbeiten.

Neben den einzelnen Subsystemen GBS, PK und Verwaltungssystem kann ein Cloud-System (*siehe Abbildung 29 Cloud_Based_System*) im BDS enthalten sein. Dieses steht bidirektional mit dem GBS in Verbindung und tauscht Aktuator- und GBS-Informationen aus. Diese Systeminformationen können in dem Cloud-System abgespeichert und für eine cloudbasierte Analyse sowie eine abgesetzte Überwachung verwendet werden.

Die in Abbildung 29 dargestellten Datenflüsse eines BDS unterliegen sicherheitskritischen und sicherheitsunkritischen Anforderungen. Die roten Datenflüsse setzen eine erhöhte Verlässlichkeit voraus und müssen Rechtzeitigkeitsanforderungen einhalten. Diese umschließen den Kontrollprozess durch die Sensoren über das GBS und den PK zu den Aktuatoren sowie den Datenfluss zwischen Verwaltungssystem und PK. Die grünen Datenflüsse stellen Zusatzfunktionen dar, die keiner erhöhten Verlässlichkeit unterliegen. Diese Funktionen beeinflussen die Aktuatoren nicht und setzen im Umfeld der Funktionssicherheit keine Rechtzeitigkeitsanforderungen voraus.

Der vorgestellte Architekturvorschlag eines BDS ist geprägt durch die folgenden Architekturkonzepte, um die Rechtzeitigkeit und Dienstgüte zu unterstützen:

- Unzuverlässige und zuverlässige Geräteensemble bilden die Grundstruktur eines BDS und erweitern den Funktionsumfang der zuverlässigen Geräte.
- Ein BDS bildet ein Kontrollsystem nach dem Prinzip *Sensorik – GBS – PK – Aktuatorik.*
- Ein kleiner PK nach dem KISS-Prinzip mit Fall-Back-Funktion wird dem GBS vorgeschaltet, um das rechtzeitige Ansteuern der Aktuatoren zu gewährleisten.
- Ein verteiltes dynamisches IoT-Geräteensemble bildet die Grundstruktur des GBS.
- Die einzelnen IoT-Geräte arbeiten in multimodalem Betriebswechsel dynamisch nach dem Graceful Degradation Ansatz zusammen, um die Dienstgüte des BDS zu verbessern.
- Der Kontrollzugriff im BDS bzw. die Kommunikation des GBS basiert auf dem Leitstation-Folgestation-Prinzip.

Im weiteren Verlauf dieser Arbeit wird der sicherheitskritische Kontrolldatenfluss zwischen Sensoren, GBS, PK und Aktuatoren näher betrachtet.

4.2 Auswahl und Anpassung von Fehlertoleranzmustern

Der kleine PK mit Fall-Back-Funktion, der zwischen GBS und Aktuatoren agiert, gewährleistet die Zuverlässigkeit und steuert die Aktuatoren rechtzeitig an. Dieses Konzept und Fehlertoleranzmuster alleine reichen nicht aus, um die Dienstgüte der BDS-Services zu gewährleisten. Die binäre Umstellung zwischen betriebsbereit und funktionssicherem Fall-Back-Betrieb versetzt das BDS oft in den Fall-Back-Modus und verweilt in diesem für eine lange Zeitspanne. Um diesem Verhalten entgegenzuwirken und um die Dienstgüte zu verbessern, wird der Graceful Degradation Ansatz im Systemkontext der GBS angewendet. Zur Unterstützung der Rechtzeitigkeit und Dienstgüte des GBS wurden die in Tabelle 4 skizzierten Fehlertoleranzmuster entwickelt. Es handelt sich dabei um spezielle Kombinationen bekannter Muster, welche an die Anforderungen der hier vorgestellten GBS-Architektur angepasst sind.

Tabelle 4 - Fehlertoleranzmuster zur Unterstützung der Rechtzeitigkeit und Dienstgüte in gerätebasierten Systemen

Fehlertoleranzmuster	Zweck	Fehlererkennung	Fehlerbehebung
Distributed-Redundancy-Dynamic-Voter-Muster (DRDV-Muster)	Fehlermaskierung, Fehlereindämmung, Fehlerbehebung von Software- und Hardwarefehlern	Watchdog-Timer, Relativtest	Fehlermaskierung sowie Vorwärtsbehebung durch redundante Back-up-Komponente
Recovery-Block-Acceptance Voting - Muster (RcB-AV-Muster)	Fehlermaskierung, Fehlereindämmung, Fehlerbehebung von Softwarefehlern	Watchdog-Timer, Relativtest	Fehlermaskierung sowie Rückwärtsbehebung mittels Checkpointing
Leaky-Bucket-History-Muster (LBH-Muster)	Fehlereindämmung, Fehlerbehebung von Software- und Hardwarefehlern	Watchdog-Timer, Zähler	Vorwärtsbehebung durch gespeicherte History-Daten
N-Self-Checking-Programming-Acceptance-Voting-Muster (NSCPAV-Muster)	Fehlereindämmung, Fehlerbehebung, Fehlermaskierung von Softwarefehlern	Watchdog-Timer, Vergleichtest, Relativtest	Fehlermaskierung sowie Vorwärtsbehebung durch redundante Back-up-Komponente

Das DRDV-Muster ist eine Kombination aus dem N-Modularem Redundanzmuster (NMR) [30] mit dynamischem Mehrheitsentscheider [31]. Das Muster dient der Fehlertoleranz von Software- sowie Hardwarefehlern. Die N-redundanten Hardwarekomponenten interagieren untereinander nach dem Leitstation-Folgestation-Prinzip [86] und ermöglichen durch die aktiven Back-up-Komponenten eine Vorwärtsfehlerbehebung. Das RcB-AV-Muster ist ein Fehlertoleranzmuster zur

Behebung von Softwarefehlern. Es kombiniert das RcB- und AV-Muster [30, 31, 37] miteinander. Verschiedene redundante Softwaremodule dienen der Rückwärtsbehebung durch Checkpointing und verbessern die Verfügbarkeit. Jedes dieser Softwaremodule kann durch zusätzliches N-Version-Programming mit separaten Absoluttests und einem Relativtest Fehler maskieren und verbessert die Korrektheit der Softwaremodule. Ein weiteres wichtiges Fehlertoleranzmuster zur Einhaltung der Rechtzeitigkeit ist das LBH-Muster. Dieses Muster setzt sich aus den folgenden zwei Fehlertoleranzmustern zusammen: dem Leaky-Bucket-Counter-Prinzip sowie dem Riding-Over-Transients-Muster [31, 36]. Dieses Fehlertoleranzmuster toleriert kurzfristige Verzögerungen und Datenverluste von Hardwareausfällen und/oder Softwarefehlern und verändert den Systemzustand nicht. Das NSCPAV-Muster ist eine Konfiguration aus dem NSCP-Muster [37] mit integriertem AV-Muster [31] und dem Pair-and-a-Spare-Muster [30]. Dieses Fehlertoleranzmuster dient zur Erkennung und Behebung von Softwarefehlern. Verschiedene Softwaremodule überwachen sich untereinander selbst. Im Fehlerfall können redundante Back-up-Module zur Entscheidungsfindung verwendet werden, um eine Fehlermaskierung und Vorwärtsbehebung zu gewährleisten.

Durch die Verwendung dieser kombinierten Muster im GBS können die Verfügbarkeit und das rechtzeitige Steuern des PK deutlich verbessert werden. Im Folgenden werden die Muster und ihre Eigenschaften sowie Zusammensetzungen im Detail beschrieben.

4.2.1 DRDV-Muster

Das DRDV-Fehlertoleranzmuster basiert auf der klassischen passiven Hardware-Redundanz [30] und dient der Fehlertoleranz von Software- und Hardwarefehlern. Abbildung 30 zeigt den abstrakten DRDV-Musteraufbau. Mehrere Hardwarekomponenten stehen bidirektional in Verbindung und bilden ein dynamisches verteiltes GBS. In diesem verteilten System stellt eine explizite Hardwarekomponente die Leitstation (*primary device*) dar und die restlichen Hardwarekomponenten dienen als Folgestationen (*Back-up-device*) [86].

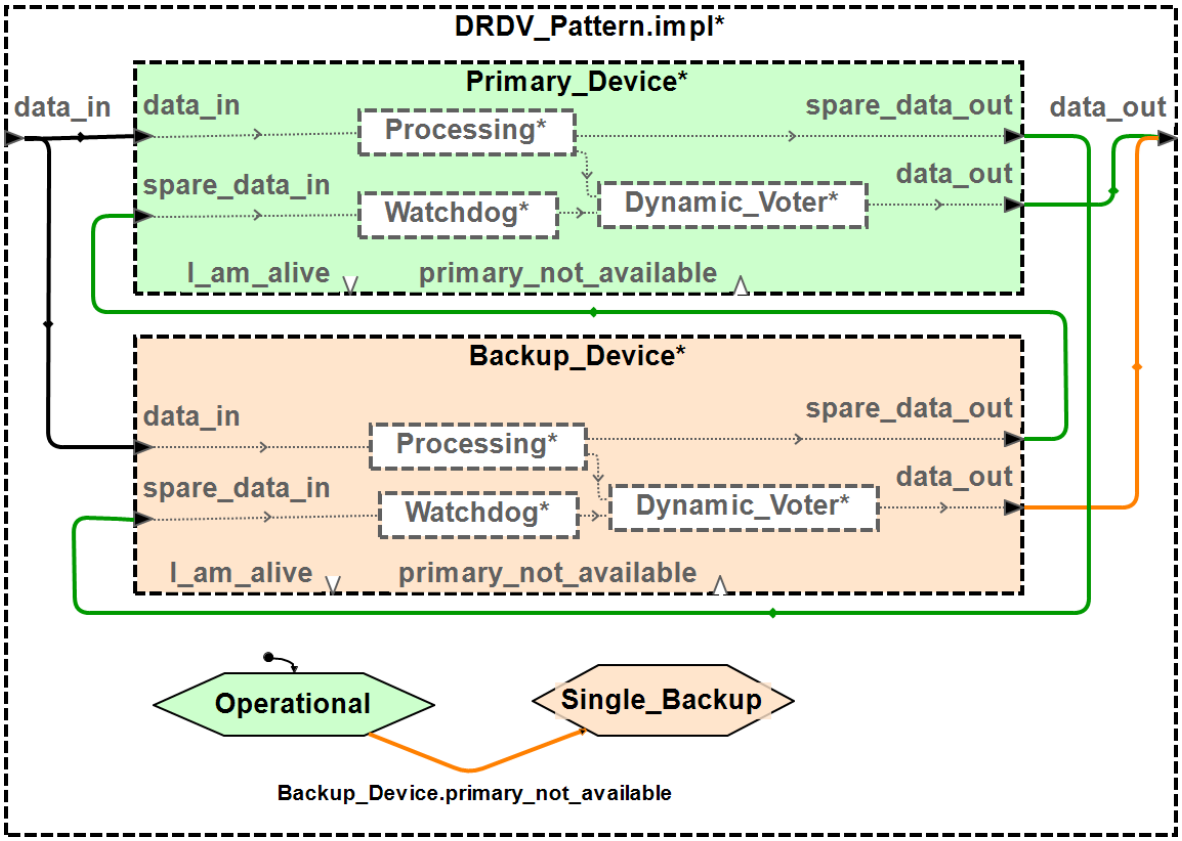


Abbildung 30 - Abstraktes AADL-Modell des DRDV-Muster zur Fehlerbehebung von Software- und Hardwarefehlern

Die im Back-up arbeitenden aktiven Stand-by-Komponenten dienen zusätzlich als Informationsquelle für eine mögliche Fehlermaskierung der internen Softwareprozesse in der Leitstation. Die Leitstation kann neben den intern gewonnenen Daten auch auf alle verfügbaren redundanten Daten der Back-up-Komponenten zugreifen. Jede Hardwarekomponente überwacht die Kommunikationsschnittstelle über einen Watchdog-Timer [36] und kann zeitliche Fehler oder Verbindungsfehler erkennen. Aufgrund der Dynamik des verteilten Systems verwendet das DRDV-Muster einen dynamischen Entscheidungsträger [31] nach dem M/N-Prinzip (M aus N). Angesichts der Anzahl der Informationsquellen ist die M/N-Logik dynamisch anpassbar. Stehen dem System $N \geq 3$ Hardwarekomponenten zur Verfügung, arbeitet der Entscheider (*dynamic voter*) nach dem Prinzip der Mehrheitsentscheidung mit $M \geq (N+1)/2$. Dies ermöglicht das Maskieren von fehlerhaften Hardwarekomponenten sowie Softwarefehlern und dient der Verbesserung der Korrektheit. Stehen dem System nur $N = 2$ Komponenten zur Verfügung, arbeitet der Voter als Vergleichstest mit $M = N$ und erhöht die Korrektheit. Sollte eine Leitstation ausfallen oder das Geräteumfeld verlassen, übernimmt eine Back-up-Komponente die Rolle der Leitstation und das System kann weiterhin zeitgerecht unterstützt werden. Solange noch mindestens eine Hardwarekomponente zur Verfügung steht und reagiert, kann ein Stellsignal rechtzeitig verarbeitet und weitergeleitet werden [30].

Die Komponenten des DRDV-Musters sind in Tabelle 5 dargestellt.

Tabelle 5 - Zusammensetzung und Konfiguration des DRDV-Fehlertoleranzmusters

Muster und Prinzipien	Zweck	Fehlererkennung und -behebung
passive Hardware Redundanz: **NMR-Muster** [30]	N-verschiedene Hardwarekomponenten arbeiten parallel zusammen. Ein Voter verarbeitet die Ergebnisse und entscheidet über die Ausgabe. Verbesserung der funktionalen Korrektheit.	Relativtest (in der Regel Mehrheitsentscheider) und Maskierung von einzelnen Fehlern.
Relativtest: **Dynamischer Mehrheitsentscheider** [31]	Dynamisch anpassbare Logik auf der Grundlage der zur Verfügung stehenden Eingaben (Back-up-Komponenten). Wird im Kontext des NMR-Musters angewandt, um die Zuverlässigkeit und Verfügbarkeit zu gewährleisten.	Dynamischer Relativtest und Maskierung von einzelnen Fehlern.
Zugriffskontrolle: **Leitstation-Folgestation-Prinzip** [86]	Eine explizite Hardwarekomponente des NMR-Musters stellt die Leitstation dar und alle anderen zur Verfügung stehenden Komponenten dienen als Folgestation (Back-up). Die Komponenten entscheiden dynamisch, wer welche Rolle einnimmt. Verbesserung der Verfügbarkeit des GBS.	Watchdog-Timer und I-Am-Alive-Signal können Ausfälle der Leitstation erkennen. Fehlervorwärtsbehebung durch Übernahme der Rolle als Leitstation durch eine Back-up-Komponente.
aktive Hardware Redundanz: **Hot-Stand-by-Redundanz** [30]	Eine Hardwarekomponente stellt die Leitstation dar. Fällt diese aus, kann eine Back-up-Komponente die aktive Rolle der Leitstation übernehmen. Verbesserung der Verfügbarkeit.	Fehlervorwärtsbehandlung durch redundante Stand-by-Hardwarekomponenten. Mindestens eine aktive Hardwarekomponente muss als Leitstation agieren, sonst fällt das DRDV-Muster aus.

Der Kompromiss (*engl. Trade-Off)* zwischen den zwei Aspekten der funktionalen Korrektheit (*dynamic voter*) und der Rechtzeitigkeit (*aktive Back-up-Komponenten*) liegt bei diesem Muster vorrangig in der Einhaltung der Korrektheit und der Dominanz des dynamischen Entscheidungsträgers. Um die Anforderung der Rechtzeitigkeit und Dienstgüte weiter zu verbessern, müssen zusätzliche Fehlertoleranzmuster vor und nach dem Voter angewendet werden.

4.2.2 RcB-AV-Muster

Das RcB-AV-Fehlertoleranzmuster dient der Behebung von Softwarefehlern. Abbildung 31 stellt das Muster abstrakt dar. Das Fehlertoleranzmuster arbeitet klassisch nach dem RcB-Muster [36]. Es stehen dem Prozess (N-1)-verschiedene Softwaremodule passiv zur Verfügung. Ein Checkpoint verwaltet die Eingabedaten und kann im Fall eines Fehlers die zusätzlichen passiven redundanten Stand-by-Komponenten aktivieren, um einen wiederholten Prozess durchzuführen. Eine Monitoring-Komponente überwacht die zeitlichen Eigenschaften [36] sowie die schon durchgeführten Wiederholungen. Des Weiteren kann gemäß der noch zur Verfügung stehenden Zeitredundanz dem Checkpoint die Aktivierung eines neuen Durchlaufs durch das Monitoring-Modul übermittelt werden. Zusätzlich agiert in jedem Modul das AV-Muster [31] mit Absoluttests (*Watchdog-Timer*) und einem Relativtest (*Voter*). N-diversitäre Softwarekomponenten arbeiten parallel und jede Komponente wird durch einen Watchdog-Timer zeitlich überwacht. Der Relativtest entscheidet nach der Mehrheitsentscheidung M/N mit M ≥ (N+1)/2.

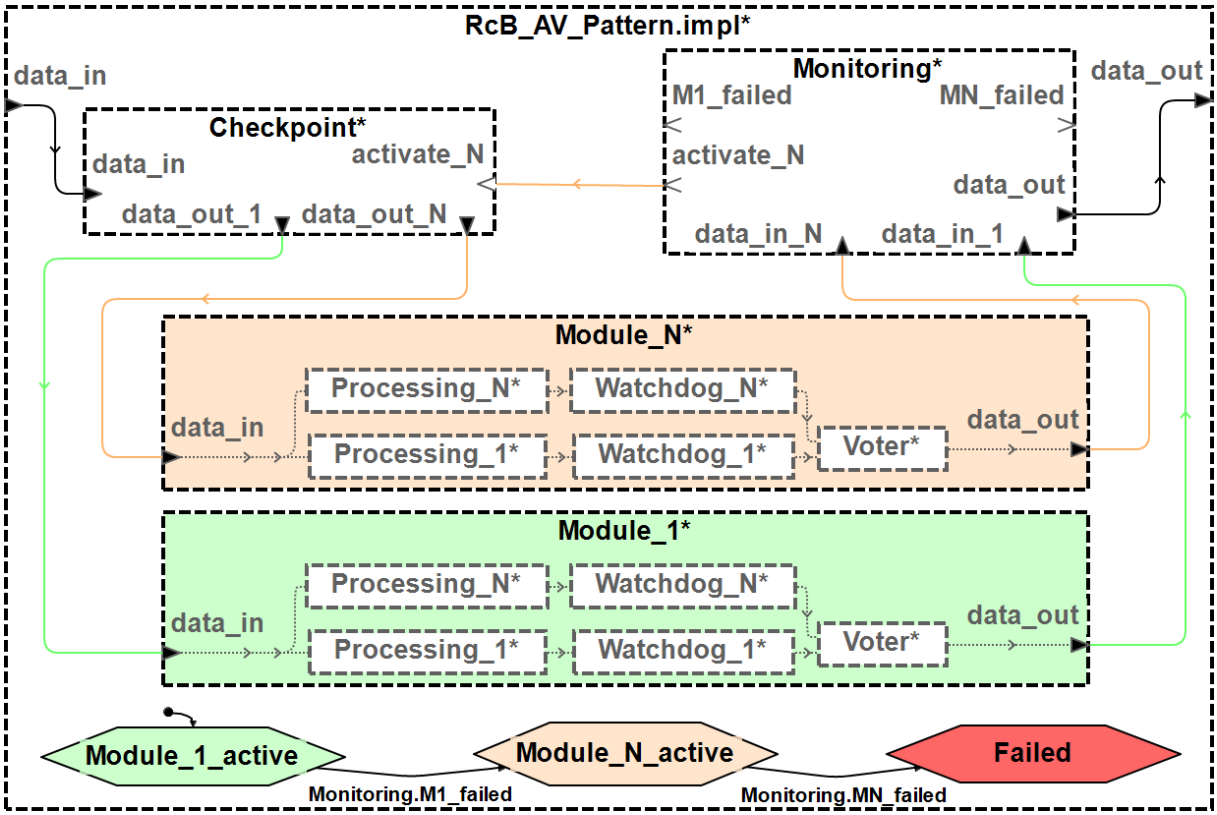


Abbildung 31 - Abstraktes AADL-Modell des RcB-AV-Muster zur Fehlerbehebung von Softwarefehlern

Die Komponenten des RcB-AV-Musters sind in Tabelle 6 dargestellt.

Tabelle 6 - Zusammensetzung und Konfiguration des RcB-AV-Fehlertoleranzmusters

Muster und Prinzipien	Zweck	Fehlererkennung und -behebung
N-Version Softwareredundanz: **RcB-Muster** [36]	Verschiede gleiche oder diversitäre Blöcke/Module einer Software stehen dem System zur Verfügung. Tritt ein Fehler im ersten Modul auf, werden sequenziell die zusätzlichen Back-up-Module aktiviert. Verbessert die Verfügbarkeit des Software-Moduls.	Absoluttest (Monitoring-Komponente) und Fehlerrückwärtsbehebung durch Checkpointing und Stand-by-Komponenten.
N-Version Softwareredundanz: **AV-Muster** [31]	In jedem RcB-Modul agiert das AV-Muster. Verschiedene Software-Versionen arbeiten simultan und werden separat durch einen Absoluttest (Watchdog-Timer) überwacht. Die Ausgaben der Akzeptanztests werden zusätzlich durch einen Mehrheitsentscheid geprüft. Verbessert die funktionale Korrektheit des Software-Moduls.	Absoluttest (Watchdog-Timer) und Relativtest (Mehrheitsentscheid) sowie Fehlermaskierung.

Der Trade-Off zwischen der funktionalen Korrektheit und Rechtzeitigkeit ist bei diesem Muster ausgeglichen. Die einzelnen RcB-Module sind stark durch den Mehrheitsentscheid des AV-Musters geprägt und dienen zur Verbesserung der Korrektheit. Die Monitoring-Komponente und die passiven Stand-by-Komponenten glätten die Entscheidung bezüglich der Rechtzeitigkeit, da erst dann, wenn alle zusätzlichen RcB-Module nicht rechtzeitig reagieren, das gesamte RcB-AV-Muster aus zeitlichen Gründen fehlschlägt.

4.2.3 LBH-Muster

Das LBH-Muster dient der Fehlerbehebung von Software- wie auch Hardwarefehlern. Abbildung 32 stellt den abstrakten Aufbau des LBH-Fehlertoleranzmusters dar. Das LBH-Fehlertoleranzmuster basiert auf dem Leaky-Bucket-Prinzip [36] und dem Riding-Over-Transients-Muster [36] zur Erkennung und Behandlung von transienten Fehlern im System. Treten zeitliche Fehler im System auf, werden diese kurzfristig toleriert und verändern den Systemzustand vorübergehend nicht. Reagiert das System nach einer gewissen Zeitspanne immer noch nicht zeitgerecht, so geht man von einem permanenten Fehler aus und übergeordnete Fehlertoleranzmuster müssen eingreifen.

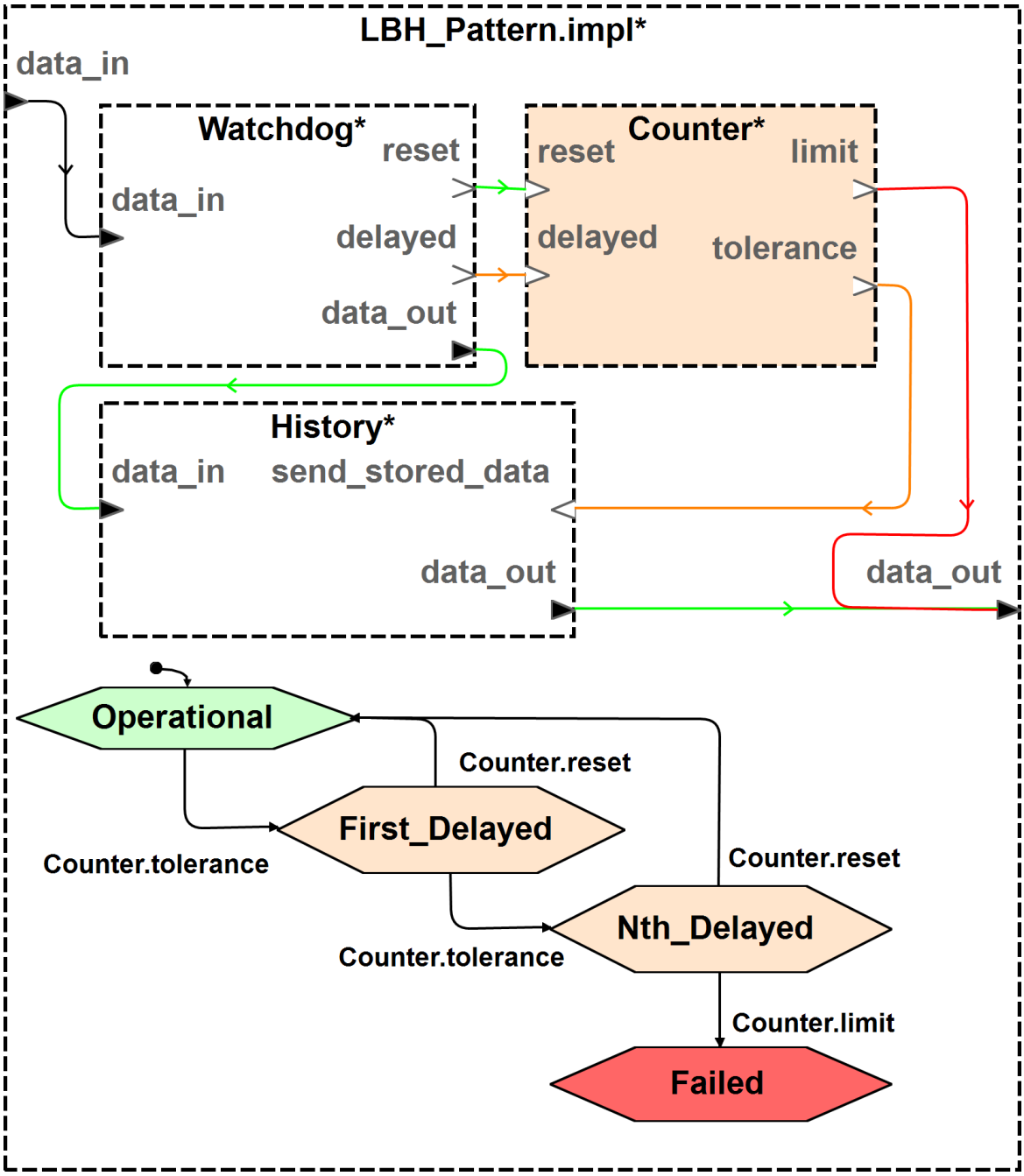


Abbildung 32 - Abstraktes AADL-Modell des LBH-Muster zur Fehlerbehebung von Software- und Hardwarefehlern

Ein Watchdog-Timer [36] überwacht den Dateneingang und speichert eintreffende Daten in einer History-Komponente [36] ab. Sollte der Watchdog-Timer eine Verspätung detektieren, speichert eine zusätzliche Zähler-Komponente [36] die Anzahl der nacheinander auftretenden Fehlnachrichten ab und verarbeitet einen in der Vergangenheit rechtzeitig abgespeicherten Datensatz. Erst wenn ein permanenter Fehler bzw. eine zu große Verzögerung auftritt, erkennt das LBH-Muster diese und schlägt fehl.

Die Komponenten des LBH-Musters sind in Tabelle 7 dargestellt.

Tabelle 7 - Zusammensetzung und Konfiguration des LBH-Fehlertoleranzmusters

Muster und Prinzipien	Zweck	Fehlererkennung und -behebung
Fehlerbehebung: **Riding-Over-Transients-Muster** [36]	Transiente Fehler werden im System toleriert. Wird ein temporärer Fehler erkannt, so verändert das Muster den Systemzustand nicht bzw. verwendet einen in der Vergangenheit fehlerfreien Zustand. Verbesserung der Verfügbarkeit.	Fehlervorwärtsbehebung durch die Verwendung eines abgespeicherten fehlerfreien Zustands (History).
Fehlererkennung: **Leaky-Bucket-Counter-Prinzip** [36]	Um transiente Fehler von permanenten Fehlern zu unterscheiden, wird im Riding-Over-Transients-Muster das Leaky Bucket Counter-Prinzip angewendet. Ein Zähler überwacht die aufeinanderfolgenden Fehler und speichert diese ab. Wird ein Schwellenwert überschritten, muss der Fehler als nicht mehr temporär behandelt werden.	Ein Absoluttest (Watchdog-Timer) erkennt zeitliche Verzögerungen und ein Zähler überwacht den zu tolerierenden Schwellenwert.

Das LBH-Muster ist stark durch die Rechtzeitigkeitsanforderungen geprägt, die an ein BDS gestellt werden. Der Trade-Off zwischen der Korrektheit und der Rechtzeitigkeit liegt in der Verwendung der abgespeicherten Datensätze und des zu tolerierenden Schwellenwertes.

4.2.4 NSCPAV-Muster

Das NSCPAV-Fehlertoleranzmuster dient der Fehlererkennung und -behebung von Softwarefehlern. Abbildung 33 stellt das NSCPAV-Muster abstrakt dar. Das NSCPAV-Muster basiert auf der Grundlage des NSCP-Musters mit Vergleichstests [30, 37]. Die Basis dieses Fehlertoleranzmusters bilden die in Abbildung 33 grün dargestellten Komponenten.

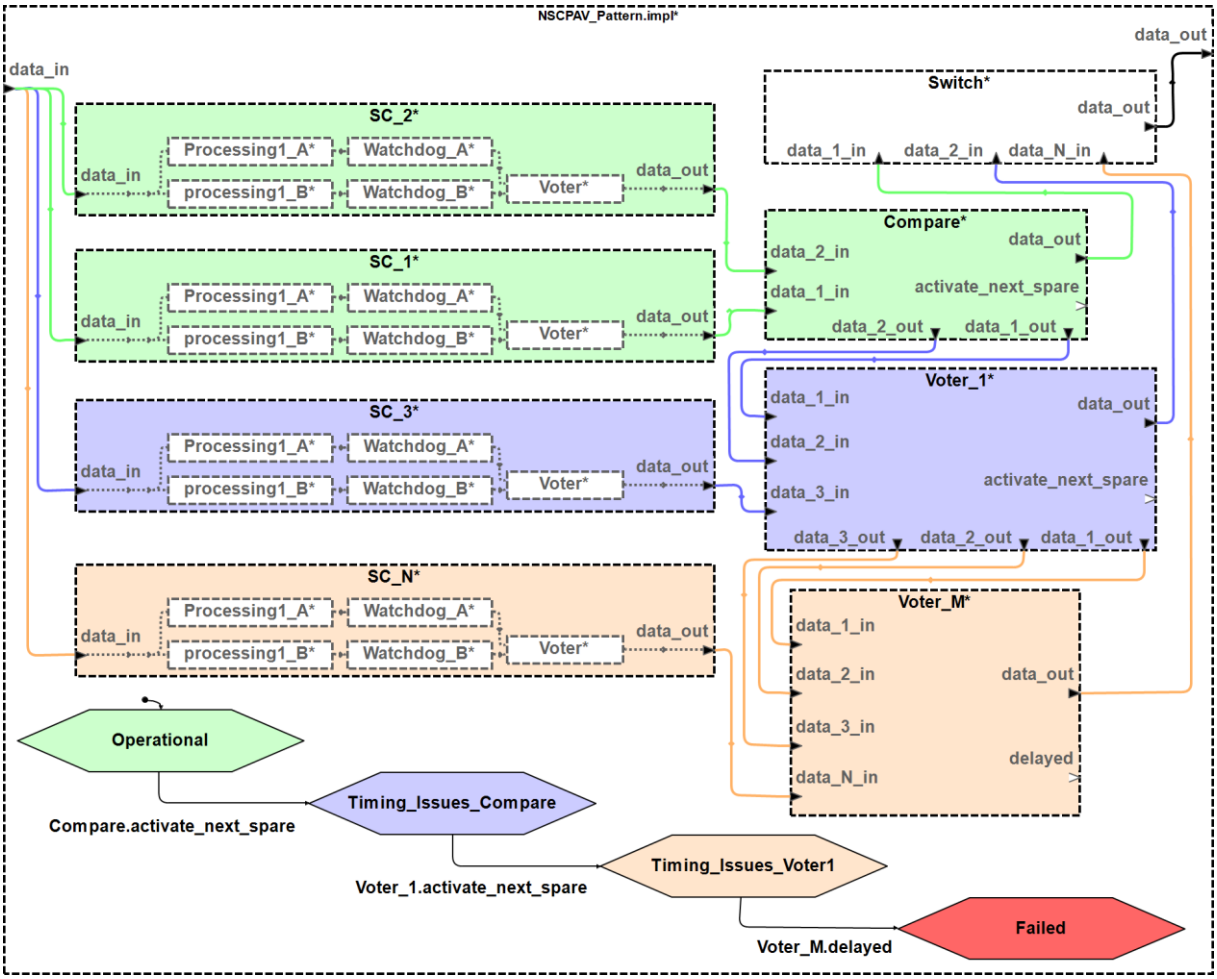


Abbildung 33 - Abstraktes AADL-Modell des NSCPAV-Muster zur Fehlerbehebung von Softwarefehlern

Die einzelnen Komponenten (*SC_1 und SC_2*) arbeiten nach dem AV-Muster [31]. Die Prozesse werden durch einen Watchdog-Timer (*Watchdog_A und Watchdog_B*) [36] zeitlich überwacht und prüfen sich zusätzlich durch einen Vergleichstest (*Voter*) selbst. Diese selbstüberwachenden Komponenten (*SC_1 und SC_2*) werden außerdem durch einen weiteren Vergleich geprüft. Dieser Vergleichstest (*Compare*) ist stark durch die Anforderungen der funktionalen Korrektheit geprägt und beeinflusst die Rechtzeitigkeitsanforderungen durch die harten Entscheidungen der Vergleichstests.

Um diesem Konflikt entgegenzuwirken, stehen dem Muster zusätzlich aktive Stand-by-Komponenten der selbstüberwachenden Komponenten (*SC_3 bis SC_N*) zur Verfügung. Kommt es im Vergleichstest zu einem Fehler, wird eine Zusatzkomponente aktiviert und ein Entscheidungsträger (*Voter_1 – Voter_M*) nach dem Prinzip M/N mit M = 2 angewandt [30]. Dieser Prozess wird solange durchgeführt, bis zwei der aktiven selbstüberwachenden Komponenten übereinstimmende und rechtzeitige Ergebnisse liefern oder der letzte Entscheidungsträger (*Voter_M*) fehlschlägt.

Die Komponenten des NSCPAV-Musters sind in Tabelle 8 dargestellt.

Tabelle 8 - Zusammensetzung und Konfiguration des NSCPAV-Fehlertoleranzmusters

Muster und Prinzipien	Zweck	Fehlererkennung und -behebung
N-Version Softwareredundanz: **NSCP-Muster mit Vergleich** [30, 37]	Zwei parallel arbeitende Softwareprozesse werden durch einen Vergleichstest überwacht. Sie bilden eine sich selbstüberwachende Softwarekomponente. Schlägt der Vergleichstest fehl, können zusätzliche selbstüberwachende Komponenten aktiviert werden. Verbesserung der funktionalen Korrektheit und Verfügbarkeit.	Relativtest (Vergleichstest) und Fehlervorwärtsbehebung durch redundante Back-up-Komponenten im Hot-Stand-by.
N-Version Softwareredundanz: **AV-Muster** [31]	In jeder selbstüberwachenden Softwarekomponente agiert das AV-Muster. Die Software-Versionen arbeiten simultan und werden separat durch einen Absoluttest (Watchdog-Timer) überwacht. Die Ausgaben der Absoluttests werden zusätzlich durch einen Vergleichstest geprüft. Verbessert die funktionale Korrektheit des Software-Moduls.	Absoluttests (Watchdog-Timer) und Relativtest (Vergleichstest) zur Fehlererkennung.
Hybride aktive und passive Redundanz: **Pair-and-a-Spare-Muster** [30]	Um die Verlässlichkeit zu verbessern, wird das Pair-and-a-Spare-Muster im Kontext des NSCP-Musters angewendet. Hierzu werden zwei selbstüberwachende Softwarekomponenten zusätzlich durch einen Vergleichstest überwacht. Es müssen zwei selbstüberwachende Softwarekomponenten korrekt und zeitgerecht arbeiten, sonst werden zusätzliche Back-up-Komponenten aktiviert.	Relativtest (Vergleichstest) zur Fehlererkennung sowie Relativtest mit 2/N zur Fehlervorwärtsbehebung durch zusätzliche selbstüberwachende Softwarekomponenten sowie Fehlermaskierung.

Das NSCPAV-Muster ist stark durch die aktiven selbstüberwachenden Komponenten mit Vergleichstests geprägt. Es dient in erster Linie der Einhaltung der funktionalen Korrektheit von Softwaremodulen. Der Trade-Off zur Rechtzeitigkeit wird dadurch gewährleistet, dass zusätzliche passive redundante Back-up-Komponenten zur Verfügung stehen und in den zuvor fehlgeschlagenen Entscheidungsprozess mit einbezogen werden. Dies ermöglicht es, dass das System trotz harter funktionaler

Korrektheitsanforderungen lebendig bleibt. Erst wenn gemäß der zur Verfügung stehenden Zeitredundanz die letzte Stand-by-Komponente keine Entscheidung nach 2/N treffen kann, schlägt das NSCPAV-Muster fehl und kann nicht mehr rechtzeitig reagieren.

4.3 Modellierung

Die zuvor vorgestellten Architekturkonzepte und Fehlertoleranzmuster können in Kombination zu einer deutlichen Verbesserung der Verlässlichkeit und insbesondere der Rechtzeitigkeit führen. Um die Dienstgüte des BDS-Kontrollsystems zu verbessern und die Fall-Back-Funktion des PKs so selten wie möglich zu aktivieren, müssen die Fehlertoleranzmuster in bestimmter Form miteinander interagieren. Dazu betrachtet man ein IoT-Gerät des BDS-Kontrollsystems und dessen Prozesse etwas genauer. Abbildung 34 stellt die einzelnen Kontrollprozesse der Datenverarbeitung in einem IoT-Gerät abstrakt dar.

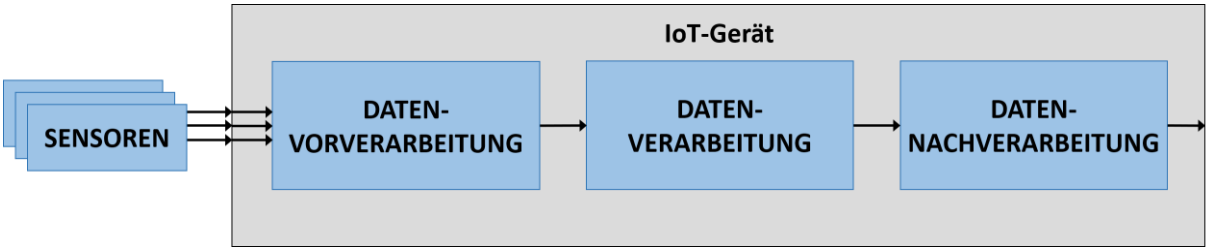


Abbildung 34 - Abstrakte Datenverarbeitung in einem IoT-Gerät des GBS

Dem IoT-Gerät stehen Sensordaten periodisch zur Verfügung und müssen durch geeignete Vorverarbeitung bereinigt bzw. aufbereitet werden (*Datenvorverarbeitung*). Die gewonnen Datensamples können durch Methoden und Algorithmen (*Datenverarbeitung*) gemäß den Anforderungen interpretiert und klassifiziert werden. Die berechnete Kontrollinformation kann schließlich weiterverarbeitet werden und eine Kontrollnachricht (*Datennachverarbeitung*) für die Aktuatorik erstellt werden.

Aus Sicht der Zuverlässigkeit und der Einhaltung der Rechtzeitigkeit und Dienstgüte stellen alle Komponenten (*Sensoren, Datenvorverarbeitung, Datenverarbeitung, Datennachverarbeitung und das IoT-Gerät*) einen zentralen Ausfallpunkt (*engl. Single Point of Failure*) im System dar. Demzufolge müssen geeignete Fehlertoleranzmuster um und innerhalb der IoT-Geräte eingeführt werden. Abbildung 35 zeigt den entwickelten abstrakten Architekturvorschlag für ein Duplex-BDS-Kontrollsystem mit den aus Kapitel 4.1 und Kapitel 4.2 angewandten Fehlertoleranzmechanismen zur verbesserten Verfügbarkeit und Einhaltung der Korrektheit und insbesondere der Rechtzeitigkeit und Dienstgüte.

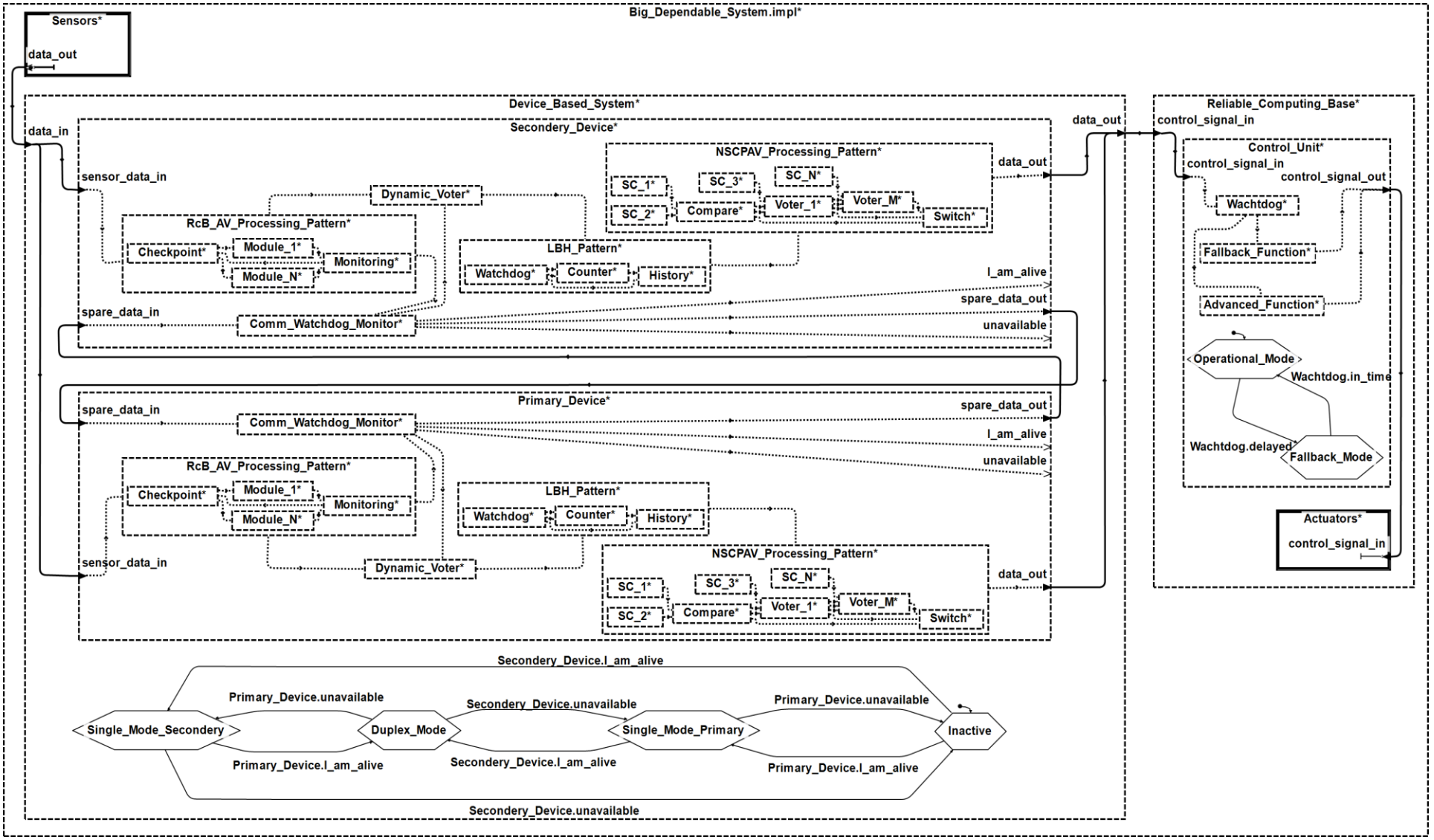


Abbildung 35 - Abstrakte AADL-Gesamtarchitektur eines Duplex-BDS-Kontrollsystems mit Fehlertoleranzmechanismen zur Einhaltung der Rechtzeitigkeit und Dienstgüte

Zwei IoT-Geräte verarbeiten die gemessenen Sensordaten und interagieren untereinander. Das Geräteensemble kann dynamisch im Duplex-Modus oder im Single-Modus einer der beiden IoT-Geräte agieren. Im Duplex-Betrieb agiert ein Gerät als Leitstation und das Back-up-Gerät als Folgestation. Dies setzt das Konzept des DRDV-Musters um. Der Betrieb im Duplex-Modus erhöht die Korrektheit durch den dynamischen Entscheider. Dies ermöglicht das Erkennen von Sensor-, Pre-Processing- und Processing-Fehlern. Sollte ein IoT-Gerät nicht zur Verfügung stehen oder ausfallen, kann das aktive Back-up-Gerät als Leitstation agieren und die Kontrollfunktion weiter gewährleisten. Dieser Graceful Degradation Ansatz erhöht einerseits die Verfügbarkeit des Kontrollsystems und reduziert andererseits die funktionale Korrektheit gegenüber dem Duplex-Betrieb. Dies setzt voraus, dass zusätzliche Fehlertoleranzmuster innerhalb eines IoT-Gerätes Anwendung finden, um die Datenverarbeitung und -nachverarbeitung fehlertoleranter zu gestalten.

Um die Korrektheit und die Verfügbarkeit der Datenverarbeitung zu verbessern, werden die Datensamples abgespeichert und stehen dem Prozess zur Verfügung. Die einzelnen Methoden der Datenverarbeitung arbeiten parallel nach dem RcB-AV-Muster. In jedem RcB-Block arbeiten N-Softwareversionen parallel. Diese werden separat zeitlich überwacht. Zusätzlich überprüft ein Mehrheitsentscheid die Ausgaben der Softwareversionen, erhöht die Verlässlichkeit der Datenverarbeitung und ermöglicht eine Fehlermaskierung. Im Falle eines Fehlers kann eine zusätzliche Monitoring-Komponente das System zurücksetzen und eine Wiederholung der Datenverarbeitung gemäß der zur Verfügung stehenden Zeitredundanz einleiten. Dieses Fehlertoleranzmuster erhöht die funktionale Korrektheit durch das AV-Muster und verbessert die Verfügbarkeit durch den RcB-Ansatz.

Um kurzfristige Hardware-, Software- oder Kommunikationsfehler zu erkennen und das System weiterhin lebendig zu halten, wird nach dem RcB-AV-Muster und dem dynamischen Entscheider des DRDV-Musters das LBH-Fehlertoleranzmuster angewendet. Das LBH-Muster speichert rechtzeitig eingetroffene Kontrollinformation ab. Dies ermöglicht das kurzfristige Tolerieren von Sensorausfällen, fehlerhaften Datenverarbeitungen eines der IoT-Geräte oder auch Kommunikationsfehler zwischen den beiden in Verbindung stehenden IoT-Geräten. Erkennt das LBH-Muster eine Überschreitung der zeitlichen Anforderungen, so verarbeitet das Muster einen in der Vergangenheit rechtzeitig eingetroffenen Datensatz.

Um eine Datennachverarbeitung fehlertolerant zu gestalten, wird das NSCPAV-Muster verwendet. Es ermöglicht die Erkennung von Softwarefehlern sowie die Behebung dieser Fehler. Die sich selbst überwachenden Softwarekomponenten mit zusätzlichem Vergleichstest erhöhen die Korrektheit. Im Falle eines nicht übereinstimmenden Ergebnisses oder einer zeitlichen Differenz können aktive Back-up-Komponenten zur Fehlertoleranz verwendet werden. Dies ermöglicht die Fehlermaskierung und erhöht die Verfügbarkeit durch einen 2/N

Entscheidungsprozess in der Nachverarbeitung sowie die rechtzeitige Datenverarbeitung im gesamten Softwaresystem.

Abschließend besitzt ein BDS einen kleinen PK mit Fall-Back-Funktion, der dem GBS vorgeschaltet ist. Dies ermöglicht das rechtzeitige Ansteuern der Aktuatoren.

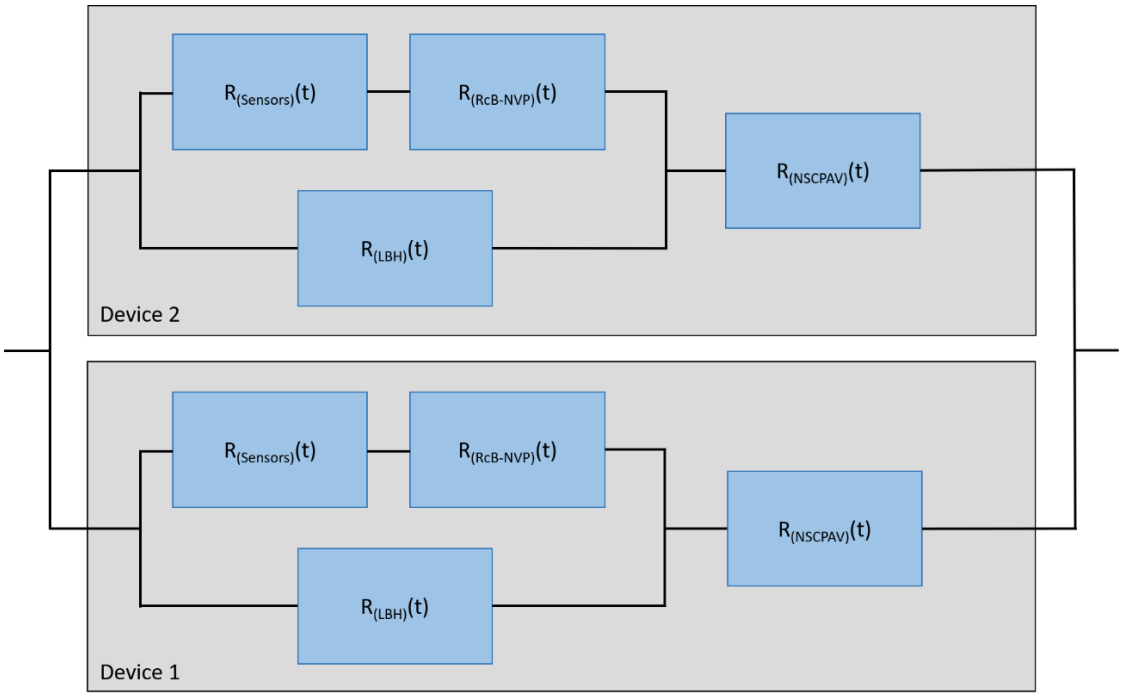


Abbildung 36 - Zuverlässigkeitsblockdiagram für ein fehlertolerantes Duplex-GBS-System

Diese Maßnahmen lösen das Problem der zentralen Ausfallpunkte und verbessern die Zuverlässigkeit und Verfügbarkeit in einem BDS. Abbildung 36 stellt das Zuverlässigkeitsblockdiagram des Architekturvorschlages für ein Duplex-GBS-Systems dar. Die zusätzlichen redundanten Hardware- und Softwarekomponenten erhöhen die Korrektheit und Verfügbarkeit. Des Weiteren verbessern die parallel arbeitenden Komponenten die Verfügbarkeit und Dienstgüte des GBS gegenüber einem GBS ohne Fehlertoleranzmechanismen.

Durch den Einsatz der MBE besteht die Möglichkeit frühzeitig im Entwicklungsprozess die Rechtzeitigkeits- und Dienstgüteanforderungen in BDS zu untersuchen und zu verbessern. Die zuvor vorgestellten abstrakten AADL-Modelle der Systemarchitektur können verfeinert werden. Durch geeignete Fehlerinformationen und Fehlermodelle können die AADL-Modelle toolgestützt analysiert werden. Das *Open Source AADL Tool Environment* (OSATE) ermöglicht es, die Fehlerentstehung und -auswirkung automatisiert zu analysieren. Insbesondere die Fehlerbaumanalyse dient der Untersuchung der Verlässlichkeit und der Verfügbarkeit von Systemkomponenten. Sie gibt Hinweise auf potenzielle Fehlerquellen. Die konkrete AADL-Modellierung und Fehlerbeschreibung eines BDS sowie die Analyse der Rechtzeitigkeit und Dienstgüte wird in Kapitel 6 an einem expliziten BDS-Beispiel verdeutlicht.

5 LÖSUNGSANSATZ AM BEISPIEL DES SPÜLZYKLUS DER AORTENKLAPPE

Das Herz stellt die zentrale Pumpstation in unserem Blutkreislauf dar. Durch rhythmische Kontraktionen des Herzmuskels pumpt das Herz Blut durch die Organe sowie das Gewebe und versorgt diese mit Sauerstoff und Nährstoffen. Ein gesundes Herz ist faustgroß und wiegt ca. 300g. Im Ruhezustand kontrahiert das Herz zwischen 60 bis 80 Mal pro Minute und pumpt ca. fünf Liter Blut pro Minute durch den Herzkreislauf [87, 88]. Abbildung 37 stellt die Anatomie des menschlichen Herzens dar.

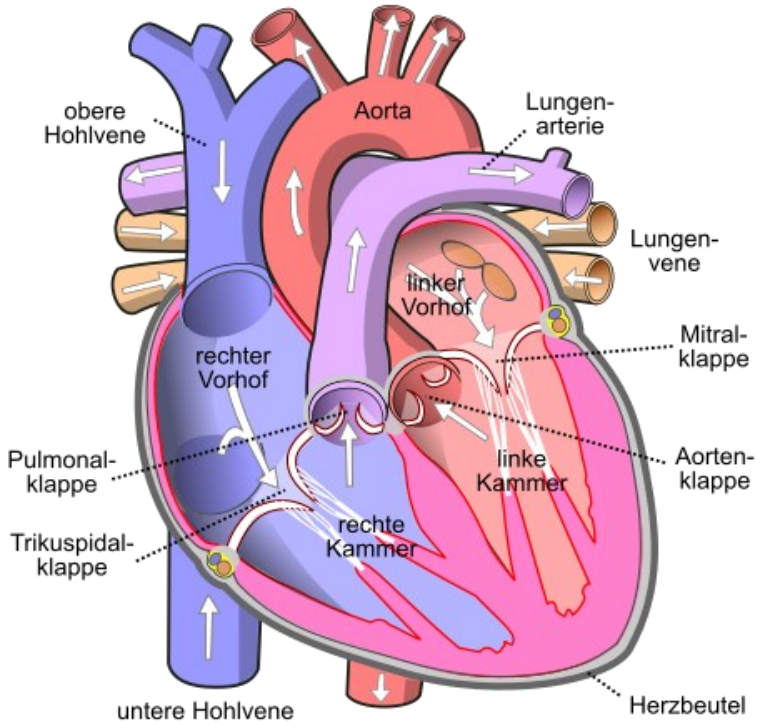


Abbildung 37 - Anatomie des Herzens [89]

Das Herz ist in eine rechte und eine linke Herzhälfte unterteilt. Jede Herzhälfte ist zusätzlich noch einmal in zwei Kammern dem Vorhof (Atrium) sowie einer Hauptkammer (Ventrikel) unterteilt. Die rechte Herzhälfte pumpt sauerstoffarmes Blut über die Lungenarterie in die Lunge. Dort wird das im Blut enthaltene Kohlendioxid abgegeben und das Blut mit Sauerstoff angereichert. Dieser

Blutkreislauf wird auch Lungenkreislauf genannt und führt über den linken Vorhof in die linke Herzkammer. Die linke Herzhälfte pumpt das aus der Lungenvene sauerstoffreiche Blut über die Aorta zu den Organen, dem Kopf und den Gliedmaßen [88]. Dieser Blutkreislauf wird auch Körperkreislauf genannt. Damit das Blut in die richtige Richtung fließen kann und es zu keinen Rückströmungen kommt, regulieren vier Herzklappen (Trikuspidalklappe, Mitralklappe, Pulmonalklappe, Aortenklappe) den Blutfluss im Herzen. Diese Herzklappen arbeiten wie ein Ventil und öffnen sich erst, wenn der Druck in den Ventrikeln ausreichend hoch ist[87].

Wie in der Einleitung dieser Arbeit schon dargestellt, bildet die Herzinsuffizienz die dritthäufigste Todesursache unter den Herzkreislauferkrankungen in Deutschland [9]. Die Herzinsuffizienz wird im Volksmund auch Herzschwäche genannt und ist nach Link und Böhm wie folgt definiert [90]:

> *„Das Syndrom Herzinsuffizienz beschreibt das Unvermögen des Herzens, eine adäquate Auswurfleistung (Herzminutenvolumen) zu erzeugen, um eine ausreichende Organdurchblutung zu gewährleisten."*

Die Störung des Herzmuskels ermöglicht es nicht, eine adäquate Durchblutung und Sauerstoffversorgung des Körpers über den linken Ventrikel und die Aorta zu gewährleisten. Als Therapie stehen neben der medikamentösen Behandlung von ACE-Hemmern, Aldosteronantagonisten und Betablockern auch operative Therapiemöglichkeiten zur Verfügung. Eine Bypass-Operation, ein Herzschrittmacher (HSM) oder auch ein Defibrillator (ICD) können die Herzinsuffizienz beeinflussen und die Auswurfleistung des linken Ventrikels erhöhen. Reichen diese Therapien nicht mehr aus, besteht die Möglichkeit einer Herztransplantation bzw. eines LVAD-Herzunterstützungssystems [91]. Diese Systeme unterstützten das Herz bei der Auswurfleistung in die Lungenarterie sowie Aorta und verbessern den Lungen- bzw. Körperkreislauf des Menschen deutlich.

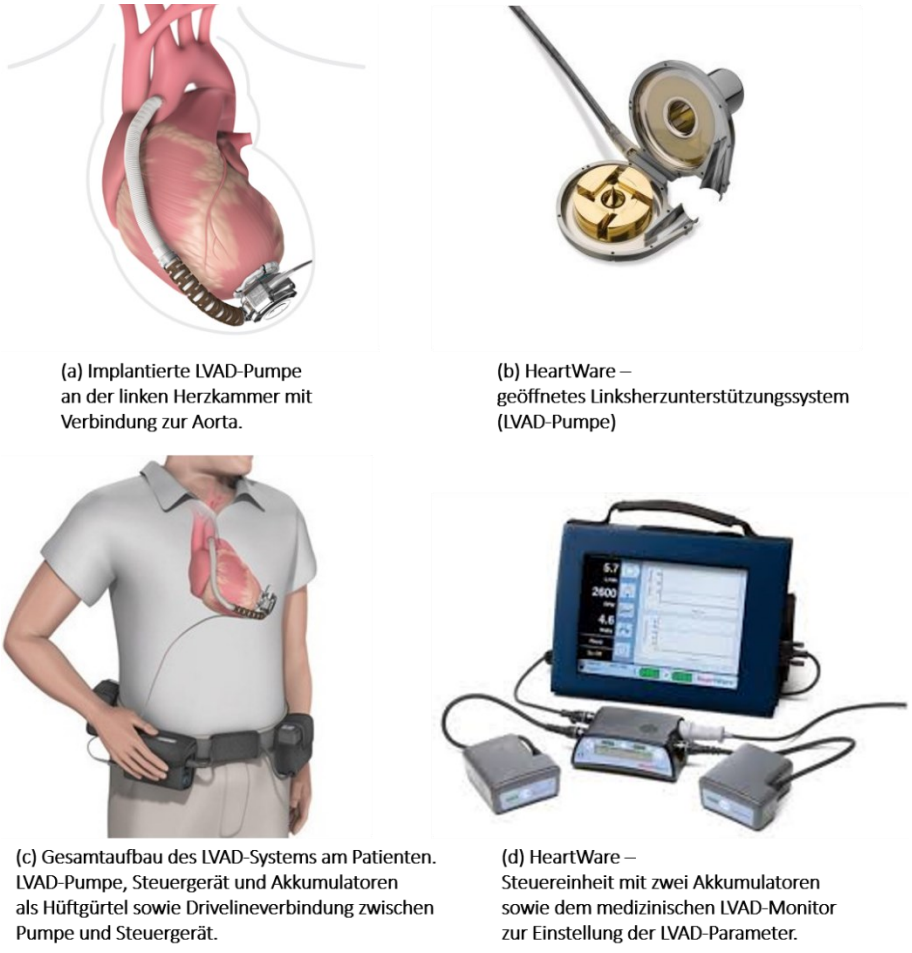
Im Folgenden wird der medizinische Anwendungsfall des Spülzyklus der Aortenklappe bei LVAD-Patienten näher betrachtet. In Kapitel 5.1 werden das LVAD-System und dessen Eigenschaften und Probleme kurz dargestellt. Kapitel 5.2 beschreibt den medizinischen Anwendungsfall für den Spülzyklus der Aortenklappe sowie die zu erfüllenden Rechtzeitigkeitsanforderungen. Abschließend wird in Kapitel 5.3 ein BDS-Lösungsansatz dargestellt, der das existierende LVAD-System funktional erweitert und die unzuverlässigen IoT-Geräte des GBS für eine adaptive Pumpensteuerung nutzen kann.

5.1 Linksherzunterstützungssysteme

Ist eine Herzinsuffizienz nicht mehr medikamentös behandelbar, besteht die Möglichkeit, dass das geschwächte Herz durch ein Herzunterstützungssystem entlastet wird. Bei 90% aller Patienten, die ein Herzunterstützungssystem benötigen,

ist die Unterstützung des linken Ventrikels völlig ausreichend [92]. Diese Herzunterstützungssysteme, die die linke Herzkammer beeinflussen, nennt man Linksherzunterstützungssysteme (LVAD).

Abbildung 38 stellt das HeartWare-LVAD-System der Firma Medtronic dar. Die zentrale Komponente eines LVAD-Systems ist die LVAD-Pumpe. Diese Pumpe wird an die linke Herzkammer implantiert und fördert das Blut von dem linken Ventrikel direkt an die Aorta (Abbildung 38 (a)). Die LVAD-Pumpe verwendet eine Durchflusskreiselpumpe mit elektromagnetisch angetriebenem Impeller, welcher in Abbildung 38 (b) zu sehen ist. Um die LVAD-Pumpe zu steuern, trägt der Patient einen Hüftgürtel mit einer Steuereinheit sowie mindestens zwei Akkumulatoren. Die Steuereinheit hat eine direkte kabelgebundene Verbindung zur LVAD-Pumpe. Diese Verbindung zwischen LVAD-Pumpe und Steuereinheit heißt Driveline und verläuft durch die Bauchdecke des Patienten (Abbildung 38 (c)). Die HeartWare-LVAD-Pumpe arbeitet mit einer kontinuierlichen Geschwindigkeit und liefert einen konstanten Blutauswurf an der Aorta. Die Pumpengeschwindigkeit kann im Bereich von 1800 bis 4000 RPM durch den behandelnden Arzt mittels des LVAD-Monitors manuell eingestellt werden. (Abbildung 38 (d)) [93–95].



(a) Implantierte LVAD-Pumpe an der linken Herzkammer mit Verbindung zur Aorta.

(b) HeartWare – geöffnetes Linksherzunterstützungssystem (LVAD-Pumpe)

(c) Gesamtaufbau des LVAD-Systems am Patienten. LVAD-Pumpe, Steuergerät und Akkumulatoren als Hüftgürtel sowie Drivelineverbindung zwischen Pumpe und Steuergerät.

(d) HeartWare – Steuereinheit mit zwei Akkumulatoren sowie dem medizinischen LVAD-Monitor zur Einstellung der LVAD-Parameter.

Abbildung 38 - HeartWare LVAD-Systemkomponenten des Unternehmens Medtronic[93–95]

Die folgenden Komponenten sind zentraler Bestandteil eines LVAD-Systems und befindenden sich am und im Körper des Patienten:

- LVAD-Pumpe,
- Versorgungskabel (Driveline),
- Steuereinheit sowie
- Akkumulatoren.

Die Verwendung eines LVAD-Systems unterstützt den Patienten bei einer akuten Herzinsuffizienz und versorgt den Körperkreislauf weiterhin mit genügend sauerstoffreichem Blut. Doch die Verwendung kann auch Komplikationen und Probleme mit sich führen.

In der LVAD-Pumpe kann sich zu ca. 10% eine Thrombose bilden [96]. Dies kann jedoch bei frühzeitiger Erkennung durch eine veränderte Medikation positiv beeinflusst werden. Eine weitere Problematik bildet die offene Wunde, die durch die Driveline-Verbindung entsteht. Nach der Implantation eines LVADs kann sich bei ca. 23% der Patienten in den ersten Monaten eine Infektion der Austrittswunde bilden [97]. Durch eine geeignete Pflege sowie Beobachtung der Wunde wird diese Problematik unter Kontrolle gebracht. Bei LVADs mit kontinuierlichem Fluss kann sich eine Aortenklappeninsuffizienz (AI) bilden. Dieses Problem entsteht, indem die LVAD-Pumpe das vorhandene Blut aus dem linken Ventrikel direkt über die Aorta in den Körperkreislauf transportiert. Dadurch verringert sich der Druck in der linken Herzkammer dauerhaft und die Aortenklappe kann sich nicht mehr öffnen. Diese Inaktivität führt über die Dauer zu einer Verkalkung der Klappe und beeinträchtigt die Funktionalität des LVADs deutlich [73]. Des Weiteren weisen LVAD-Patienten durch den kontinuierlichen Blutfluss und die nicht adaptive Pumpengeschwindigkeit sehr häufig eine deutlich eingeschränkte körperliche Belastbarkeit auf [98].

Die Problematik der AI und die eingeschränkte Belastbarkeit von LVAD-Patienten kann durch eine dynamische LVAD-Steuerung auf Grundlage der aktuellen Belastung minimiert bzw. verbessert werden. Durch IoT-Sensorik können zusätzliche Patienten- und Umgebungsinformationen gemessen und eine kontinuierliche physische Belastung des Patienten berechnet werden. Dies ermöglicht eine Adaption der LVAD-Pumpe auf Basis der gemessenen Belastung und wirkt der Entstehung einer AI entgegen. Im Folgenden wird ein Lösungsansatz für die AI im Kontext der LVAD-Systeme durch die zusätzliche Verwendung von IoT-Geräten vorgestellt.

5.2 Medizinische Anforderungen

Der aktuelle Zustand bei LVADs mit kontinuierlicher Geschwindigkeit führt durch die unveränderlichen Druckverhältnisse im linken Ventrikel auf Dauer zu einer Inaktivität der Aortenklappe. Eine sich nicht regelmäßig öffnende bzw. schließende Aortenklappe verkalkt und wir über die Zeit porös. Studien belegen, dass sich bei ca.

20 - 40% der Patienten während der Unterstützung von kontinuierlich arbeitenden LVADs eine AI bildet und dass sich das Risiko einer AI pro Monat um ca. 4% erhöht [74, 99].

Abbildung 39 stellt eine Aortenklappe im geöffneten sowie geschlossenen Zustand dar. Auf der linken Seite ist eine gesunde Aortenklappe abgebildet, die sich vollständig öffnet bzw. schließt. Die Aortenklappe öffnet sich bei geeignetem Druck und transportiert das Blut über die Aorta in den Körperkreislauf. Nimmt der Druck ab, schließt sich die Klappe vollständig und verhindert einer Rückführung des Blutflusses zurück in die linke Herzkammer. Auf der rechten Seite von Abbildung 39 ist eine insuffiziente Aortenklappe abgebildet. Die Verkalkung ermöglicht es nicht mehr, dass die Klappe vollständig geöffnet bzw. geschlossen wird. In Verbindung mit einer LVAD Unterstützung bildet der Rückfluss in die linke Herzkammer einen permanenten Blutzyklus zwischen LVAD-Pumpe und linker Herzkammer. Dies beeinträchtigt die Funktionsweise sowie die Qualität des Auswurfvolumens des LVAD-Systems deutlich [100]. Der Entwicklung einer AI bei Verwendung eines LVAD-Systems kann man nur entgegenwirken, wenn die Druckverhältnisse im linken Ventrikel variabel sind und die Aortenklappe sich über die Zeit selbständig öffnen und schließen kann.

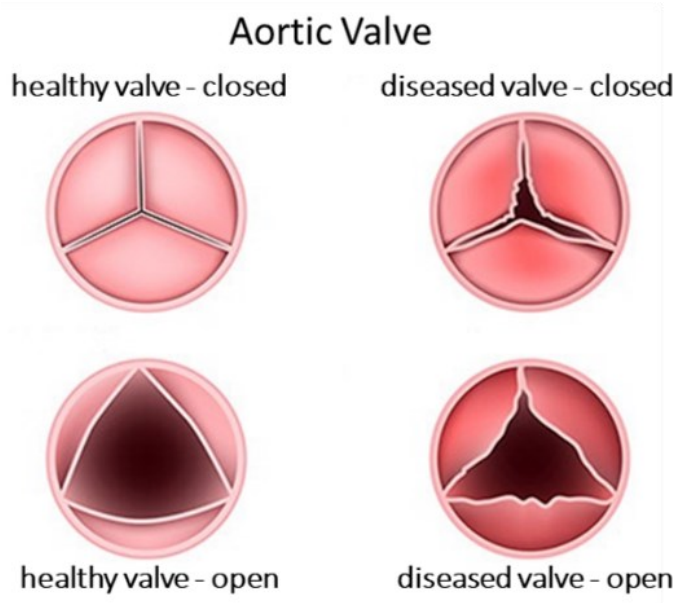


Abbildung 39 - Gesunde und insuffiziente Aortenklappe [101]

Eine Möglichkeit ist es, ein BDS im Umfeld eines LVAD-Systems zu verwenden. Ein solches BDS überwacht die Umgebungsparameter mittels drahtlos kommunizierender IoT-Geräte und bestimmt in regelmäßigen Abständen die körperliche Aktivität des Patienten. Aufgrund der erkannten physischen Aktivität des Patienten wird die Geschwindigkeit der LVAD-Pumpe in Ruhephasen abgesenkt und so ein Aortenklappenspülzyklus eingeleitet. Dies ermöglicht es, variable Druckverhältnisse im linken Ventrikel aufzubauen und so die Gefahr einer AI zu minimieren. Abbildung 40 stellt ein BDS-Kontrollsystem und dessen Systemkomponenten für einen

Spülzyklus der Aortenklappe im Umfeld eines LVAD-Systems dar. Accelerometer-, Pedometer- oder Gyroskopsensoren, die sich am Körper des Patienten befinden, messen die Umgebungsparameter. Die Sensoren können eigenständig oder aber auch in den IoT-Geräten des GBS eingebettet sein. Die gewonnen Sensordaten werden durch die IoT-Geräte (*Smartwatch oder Smartphone*) weiterverarbeitet und klassifizieren eine physische Aktivität. Für den Aortenklappenspülzyklus wird eine Klassifizierung zwischen Aktivität und Inaktivität verwendet. Ist der Patient aktiv, arbeitet das LVAD im Normalzyklus (NZ).

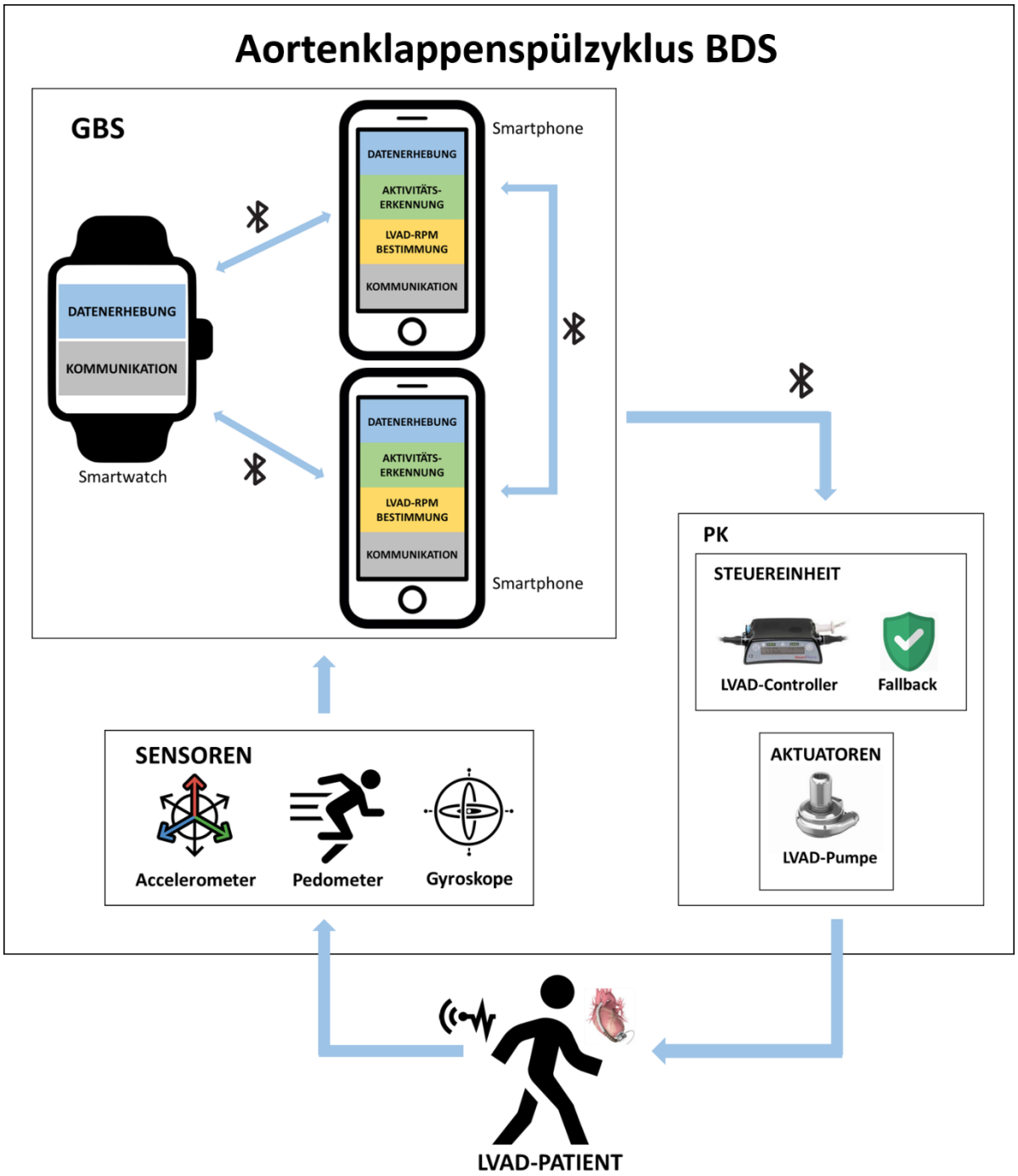


Abbildung 40 - BDS-Kontrollsystem für einen aktivitätsgesteuerten Aortenklappenspülzyklus für LVAD-Patienten

In diesem Zustand steuert das GBS die Geschwindigkeit der LVAD-Pumpe mittels eines durch das medizinische Fachpersonal voreingestellten funktionssicheren RPM-Wertes (*NG - Normalgeschwindigkeit*). Detektiert das GBS eine Inaktivität des Patienten, arbeitet das LVAD im Spülzyklus (SZ). Im SZ-Modus wird die Geschwindigkeit der LVAD-Pumpe alternierend für eine Zeitspanne t_{AO} mit einer geringeren Geschwindigkeit (*SG - Spülgeschwindigkeit*) und für eine Zeitspanne t_{AG} mittels der NG gesteuert. Durch die im SZ arbeitende geringe Geschwindigkeit der LVAD-Pumpe erhöht sich das Blutvolumen und der Druck im linken Ventrikel steigt an. Durch den Druckanstieg und die noch zur Verfügung stehende Muskelkraft des Herzens kann sich die Aortenklappe selbstständig durch die Kontraktion des Herzmuskels öffnen und schließen. Ein Teil des Blutauswurfs wird dadurch zur Aorta über die Aortenklappe transportiert. Liegt die NG an der LVAD-Pumpe an, verringert sich der Druck im linken Ventrikel wieder und die Aortenklappe schließt sich vollständig und dauerhaft. Unter diesem Druckverhältnis wird der gesamte Blutausfluss über die LVAD-Pumpe zur Aorta transportiert. Die Adaption der LVAD-Pumpe in einer Ruhephase des Patienten ermöglicht eine Aktivierung und Beanspruchung der Aortenklappe und verringert damit das Risiko einer AI über die Betriebsdauer der LVAD Unterstützung. Abbildung 41 stellt den Aortenklappenspülzyklus (SZ-Modus) grafisch dar.

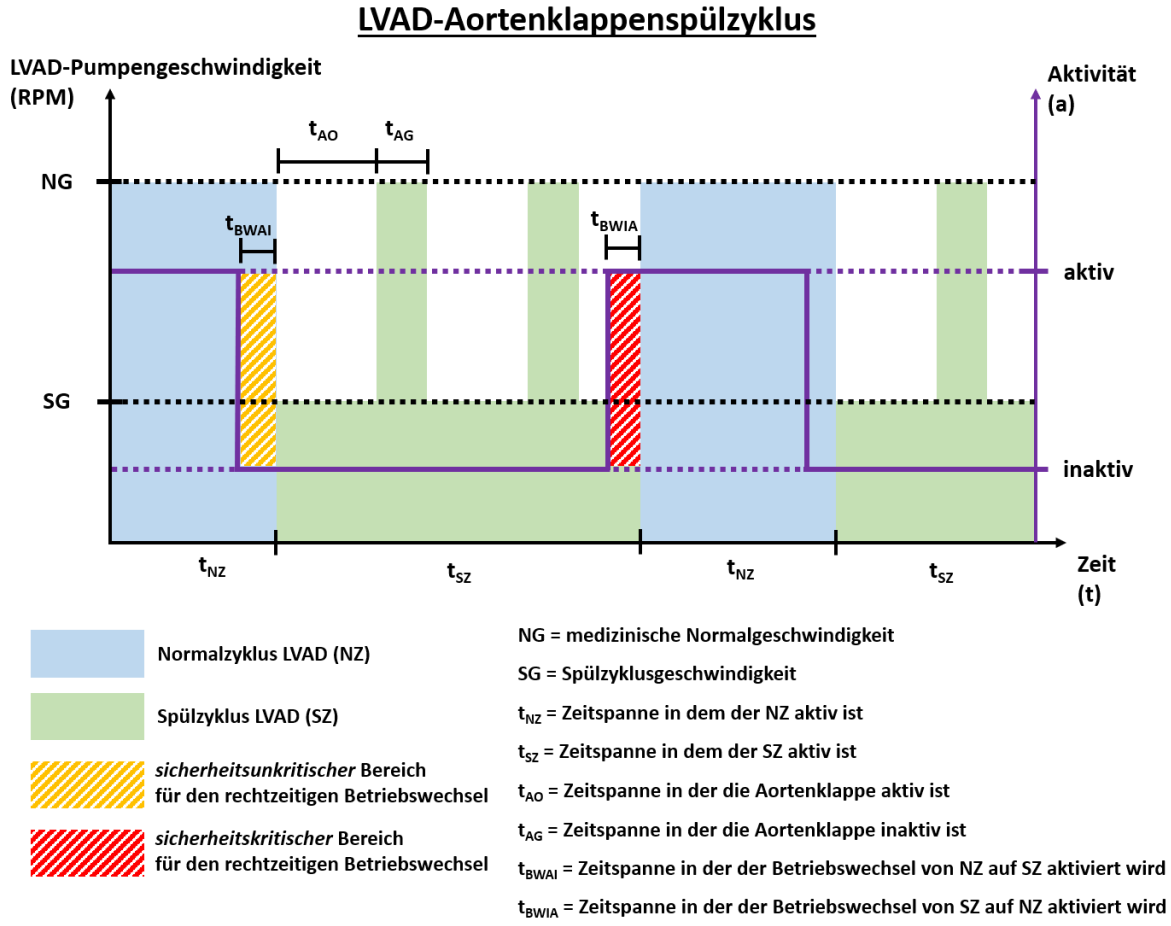


Abbildung 41 - Ablauf des Spülzyklus der Aortenklappe

Die Herausforderung zur Einhaltung der Funktionssicherheit sowie der rechtzeitigen Steuerung eines BDS-gesteuerten Aortenklappenspülzyklus sind die Betriebswechsel vom NZ-Modus in den SZ-Modus bzw. aus dem SZ-Modus in den NZ-Modus. Die zeitliche Spanne t_{BWAI} zwischen dem Betriebswechsel NZ-Modus in den SZ-Modus ist nicht sicherheitskritisch und gefährdet das Leben des LVAD-Patienten nicht. Abbildung 41 stellt diesen sicherheitsunkritischen Bereich durch den gelb schraffierten Bereich dar. Der Betriebswechsel in der Zeitspanne t_{BWIA} aus dem SZ-Modus in den NZ-Modus unterliegt dagegen einer erhöhten Funktionssicherheitsanforderung. Erkennt das BDS nicht rechtzeitig einen Aktivitätswechsel von inaktiv zu aktiv und steuert das LVAD im NZ-Modus, kann dies das Leben des Patienten gefährden. Diese ist in Abbildung 41 durch den rot schraffierten Bereich gekennzeichnet. Aus diesem Grund wurden im Medolution-Projekt funktionale und nicht-funktionale Anforderungen definiert, die einen BDS-gestützten Aortenklappenspülzyklus ermöglichen.

Tabelle 9 - Anforderungen an die Datenerhebung eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Datenerhebung
ID:	#F100
Eingabe:	/
Beschreibung:	IoT-Sensorik, die sich im Umfeld des Patienten befindet, muss ausgelesen werden. Beispiele: *Smartwatchsensorik, Fitness-Armbänder, Smartphonesensorik oder Informationen von intelligenten Waagen.*
Ausgabe:	Sensorwerte und Geräteinformationen Beispiel: *Accelerometerdaten, Gyroskopdaten, Schritte, Herzfrequenz, Gewicht.*

Tabelle 10 - Anforderungen an die Aktivitätserkennung eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Aktivitätserkennung
ID:	#F110
Eingabe:	Sensordaten aus #F100
Beschreibung:	Aus den zur Verfügung stehenden Sensordaten soll eine aktuelle physische Aktivität des Patienten bestimmt werden.
Ausgabe:	Physische Aktivität: Unterscheidung zwischen **Aktiv** *(Belastung) oder* **Inaktiv** *(Ruhe)*

Die Tabellen 9-14 stellen die wichtigen Anforderungen für die Datenerhebung, die Aktivitätserkennung, den Spülzyklus sowie Normalzyklus und die Kommunikation zwischen den Systemen dar. Neben den funktionalen Anforderungen werden zusätzlich zeitliche und qualitative Anforderungen dargestellt.

Tabelle 11 - Anforderungen an den Normalzyklus eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Normalzyklus (NZ) aktivieren
ID:	#F120
Eingabe:	Aktivitätserkennung hat Aktivität erkannt #F110< ***Belastung*** >
Beschreibung:	Erkennt die Aktivitätserkennung eine Belastung des Patienten, soll die LVAD-Pumpe mit einer Geschwindigkeit **NG** von **2500 RPM** gesteuert werden. Dieser Zustand liegt solange konstant an, bis eine andere Aktivität (Ruhe) erkannt wird.
Ausgabe:	LVAD-Steuernachricht mit Pumpengeschwindigkeit von **2500 RPM**.
Dienstgüte Anforderungen:	Ein Betriebswechsel in den NZ-Modus soll erst durchgeführt werden, wenn mindestens **5 Sekunden Belastung t_{BWIA}** erkannt wurde.
Safety Anforderungen:	Wenn der Patient aktiv wird und die Ruhephase beendet, muss spätestens nach **30 Sekunden t_{BWIA}** die LVAD-Pumpe mit der NG von **2500 RPM** gesteuert werden.

Tabelle 12 - Anforderungen an den Spülzyklus eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Spülzyklus (SZ) aktivieren
ID:	#F130
Eingabe:	Aktivitätserkennung hat Inaktivität erkannt #F110<**Ruhe**>
Beschreibung:	Erkennt die Aktivitätserkennung eine Ruhephase des Patienten, soll die LVAD-Pumpe alternierend zwischen einer Geschwindigkeit von **SG** mit **1800 RPM** für eine **Zeitspanne t_{AO}** und **NG** mit **2500 RPM** für eine **Zeitspanne t_{AG}** gesteuert werden. Dieser Ablauf wird konstant durchgeführt, solange sich der Patient in einer Ruhephase befindet.
Ausgabe:	LVAD-Steuernachricht für Pumpengeschwindigkeit von **1800 RPM** um die Aortenklappe zu aktivieren bzw. **2500 RPM** um die Aortenklappe zu schließen.
Zeitliche Anforderungen:	Die Zeitspanne **t_{AO} ,** in der die Aortenklappe aktiv ist und sich selbstständig öffnet und schließt, soll **10 Sekunden** anhalten. Nachdem die Aortenklappe aktiv war, soll eine Entlastungsphase von **t_{AG}** mit **5 Sekunden** folgen.
Dienstgüte Anforderungen:	Ein Betriebswechsel in den SZ-Modus soll erst durchgeführt werden, wenn mindestens **5 Sekunden Ruhe t_{BWAI}** erkannt wurde.

Tabelle 13 - Anforderungen an die Kommunikation eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Kommunikation
ID:	#F140
Eingabe:	Pumpengeschwindigkeiten **NG** bzw. **SG** aus #F120 oder #F130
Beschreibung:	Das GBS muss in der Lage sein, Steuersignale kabellos an den LVAD-Kontroller zu übermitteln.
Ausgabe:	Steuersignal mit der zu stellenden RPM Geschwindigkeit um SZ- bzw. NZ-Modus initiieren zu können.

Tabelle 14 - Anforderungen an den Fail-Safe-Modus eines BDS-gesteuerten Aortenklappenspülzyklus

Name:	Notfall (Fail-Safe)
ID:	#F150
Eingabe:	/
Beschreibung:	Wenn der LVAD-Kontroller über **10 Sekunden** keine Verbindung zu einem IoT-Gerät bzw. keine Stellsignale erhält, soll die LVAD-Pumpe im NZ-Modus mit einer Geschwindigkeit **NG** von **2500 RPM** gesteuert werden.
Ausgabe:	Fallback-Steuersignal an die LVAD-Pumpe mit NG von 2500 RPM
Dienstgüte Anforderungen:	Ein Betriebswechsel aus dem Fallbackmodus in den erweiterten Spülzyklusmodus soll erst durchgeführt werden, wenn eine aktive Verbindung zum GBS besteht und mindestens drei Stellsignale innerhalb von 30 Sekunden eingetroffen sind.

Die in den Anforderungen definierten konstanten Zeitspannen, Zeitgrenzen und Pumpengeschwindigkeiten sind im Medolution-Projekt mit den medizinischen Partnern ausgearbeitet worden und sollen für den BDS-gesteuerten Spülzyklus der Aortenklappe gelten. Im Folgenden werden ein Systementwurf sowie eine Softwarelösung des BDS gesteuerten LVAD-Aortenklappenspülzyklus vorgestellt.

5.3 Systementwurf und Softwarelösung

Wie zuvor dargestellt, kann eine AI durch ein phasenweises Absenken der LVAD-RPM gemindert werden. Dazu muss ein GBS im Umfeld des Patienten vorhanden sein und mittels IoT-Sensorik die körperliche Aktivität des Patienten gemessen und erkannt werden. Detektiert das GBS eine Ruhephase des Patienten, steuert dies das LVAD im SZ-Modus gemäß den medizinischen Anforderungen. Wird eine Belastung durch das GBS ermittelt, wird die LVAD-Pumpe in den NZ-Modus versetzt. Dies setzt voraus, dass das LVAD-System eine Schnittstelle bieten muss um Steuersignale eines GBS zu empfangen und zu verarbeiten. Zusätzliche müssen dem GBS IoT-Sensoren zur Verfügung stehen.

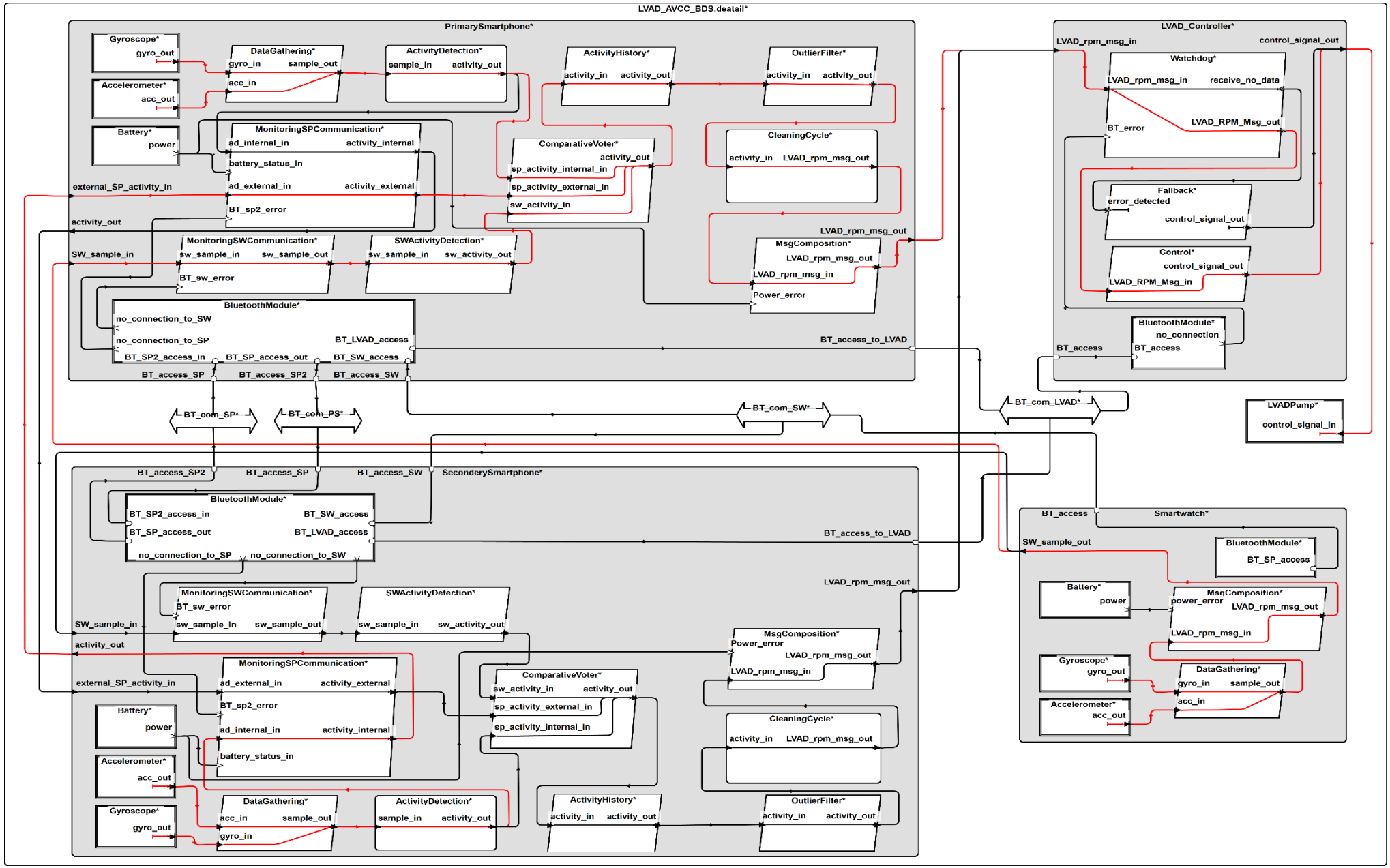


Abbildung 42 - Systementwurf eines BDS gesteuerten LVAD-Aortenklappenspülzyklus

Diese GBS ermöglichen es die Aktivität eines Patienten zu erkennen und LVAD-Pumpen-RPMs dynamisch zu bestimmen. Die einzelnen Hardware- und Softwarekomponenten des GBS, des LVAD-Systems sowie die Bluetooth-Schnittstellen für ein LVAD-BDS sind in Abbildung 42 dargestellt. Dieser Systementwurf für einen BDS-gesteuerten Aortenklappenspülzyklus zeigt ein Duplex-Smartphone mit Smartwatch, welches kabellos mit einem LVAD in Verbindung steht. Die Smartwatch (SW) dient der Datenerfassung (*Acceleromter und Gyroskop*) und erweitert die GBS-Sensorik. Sie kann mit einem Smartphone in Verbindung stehen und unidirektional Sensordaten austauschen. Die zwei Smartphones (*PrimarySmartphone (SP1) und SecondarySmartphone (SP2)*) führen die sicherheitskritische Spülzyklusfunktionalität für ein LVAD aus. Beschleunigungssensoren und Rotationssensoren (*Acceleromter und Gyroskop*), die in modernen Smartphones verbaut sind, dienen der Datenerfassung. Eine softwarebasierte Aktivitätserkennung (*ActivityDetection*) berechnet aus den erhobenen Sensorwerten die physische Aktivität. Beide Smartphones sind parallel aktiv und tauschen die intern berechneten Aktivitäten zwecks Entscheidungsprozess (*ComparativeVoter*) untereinander aus. Aufgrund der verringerten Leistungsfähigkeit einer Smartwatch gegenüber einem Smartphone werden die erhobenen Smartwatchdaten auf dem gekoppelten Smartphone weiterverarbeitet und eine Klassifikation der Aktivität (*SWActivityDetection*) dort vorgenommen. Diese berechnete Aktivität wird neben der intern erkannten Aktivität und der des SP2 dem Voter als Eingabe übergeben. Der Softwareprozess *ActivityHistory* dient der Toleranz von transienten Hardware- und Softwarefehlern. Damit das System nicht unnötig oft den Betriebswechsel vollzieht, wenn kurzzeitig Änderungen der Aktivität auftreten, dient der *OutlierFilter-Prozess* als Glättungsfunktion für die Aktivitätserkennung. Die Funktionalität des SZ-Prozesses (*CleaningCycle*) wird durch die anliegende Aktivität aktiviert bzw. deaktiviert. An dieser Stelle wird die LVAD-Solldrehzahl ermittelt und an den LVAD-Kommunikationsprozess (MsgComposition) weitergeleitet. Dieser erstellt eine LVAD-Nachricht mit der Solldrehzahl und sendet die Nachricht an die LVAD-Steuereinheit. Die Steuereinheit überwacht die Kommunikation (Watchdog) zum GBS und kann im Falle einer Störung die LVAD-Pumpe in einen medizinisch funktionssicheren Zustand (Fallback) überführen. Kommt es zu keiner Störung, wird die übermittelte Solldrehzahl verwendet. Neben diesen Systemkomponenten spielt die Stromzufuhr (Battery) sowie die kabellose Kommunikation (BluetoothModule, BT_com_SP, BT_com_PS, BT_com_SW sowie BT_com_LVAD) zwischen den einzelnen Systemen eine wichtige Rolle. Insbesondere für eine Beurteilung der Zuverlässigkeit und rechtzeitigen Steuerung des LVAD-PK. Eine detaillierte Betrachtung und Analyse der Systemzuverlässigkeit für die Rechtzeitigkeit sowie eine Beschreibung der Fehlertoleranzmuster wird in Kapitel 6 Modellbasierte Analyse vorgestellt.

Der rot dargestellte Datenfluss in Abbildung 42 stellt die Komponenten und Verbindungen des BDS-gesteuerten LVAD-Aortenklappenspülzyklus dar.

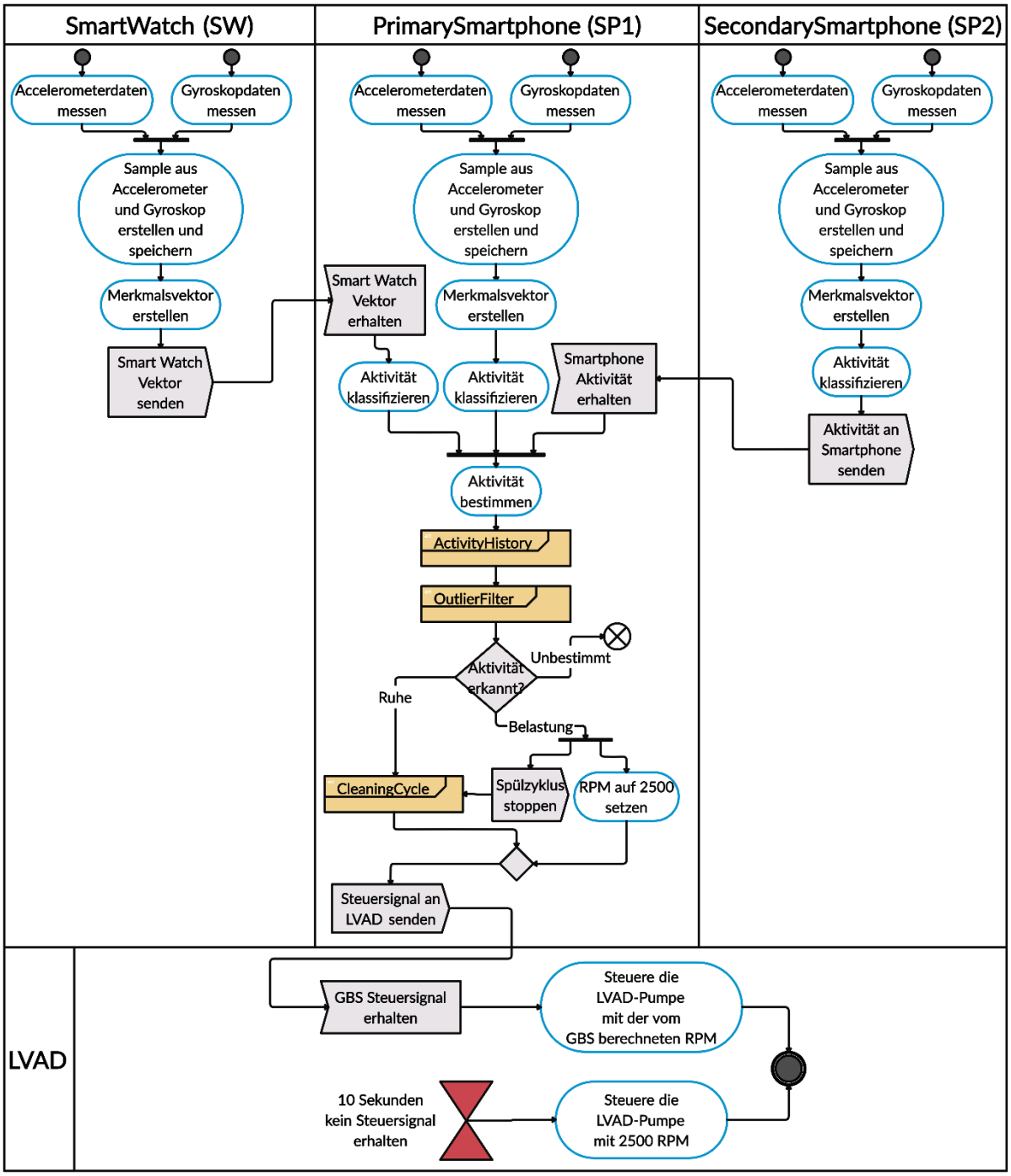


Abbildung 43 - Aktivitätsdiagramm für das Duplex-Smartphone mit Smartwatch BDS für den LVAD Aortklappenspülzyklus

Das SP1 dient als Leitstation und kommuniziert direkt mit dem LVAD und steuert die LVAD-Pumpe. Die SW dient als zusätzliche Datenerfassung sowie Sensorik und stellt dem SP1 diese Daten für eine verbesserte Aktivitätserkennung zur Verfügung. Das zweite Smartphone SP2 dient als Folgestation und arbeitet im Hot-Standby. Die berechnete Aktivität des SP2 wird zur verbesserten Aktivitätserkennung an das SP1 übermittelt. Dieser Datenfluss sowie die einzelnen Softwareaktivitäten sind zusätzlich im Aktivitätsdiagramm in Abbildung 43 dargestellt.

Die Sensoren (Accelerometer und Gyroskop) werden auf allen Geräten (Smartwatch und Smartphones) mit einer Abtastrate von 20Hz abgerufen. Dies bedeutet, dass den Geräten alle 50ms Beschleunigungs- und Rotationsdaten zur Verfügung stehen. Im nächsten Schritt werden die gemessenen Sensordaten aus einem Zeitfenster von 3,6 Sekunden zusammengefasst. Die einzelnen Zeitfenster überlappen sich zu 50% was dazu führt, dass alle 1,8 Sekunden ein zu verarbeitendes Datensample für die Aktivitätserkennung zur Verfügung steht. Aus diesen zusammengefassten Daten wird ein Merkmalsvektor erstellt. Dieser Merkmalsvektor besteht aus folgenden statistischen Werten: dem *Mittelwert*, der *Varianz* sowie dem *Maximum* für die Beschleunigungs- sowie die Drehgeschwindigkeitswerte. Da die SW keine interne softwarebasierte Aktivitätserkennung enthält, wird der erstellte Merkmalsvektor über eine Bluetooth-Schnittstelle an das gekoppelte SP1 versendet. In Abbildung 43 ist der Empfänger des Merkmalsvektors das als Leitstation agierende SP1. Dieses klassifiziert parallel aus den eigenen intern gemessenen Sensordaten und dem der SW mittels Neuronalem Netz (NN) eine physische Aktivität des Patienten. Als Klassifikation unterscheidet das NN zwischen „**Ruhe**“ und „**Belastung**“. Das als Folgestation agierende SP2 klassifiziert ebenfalls die aktuelle Aktivität intern mittels NN und versendet diese über eine Bluetooth-Schnittstelle zur Leitstation (SP1). Das SP1 synchronisiert die Aktivitäten der einzelnen IoT-Geräte (SW, SP2 und SP1) und entscheidet dynamisch auf Grundlage der zur Verfügung stehenden Eingaben, welche Belastung aktuell anliegt.

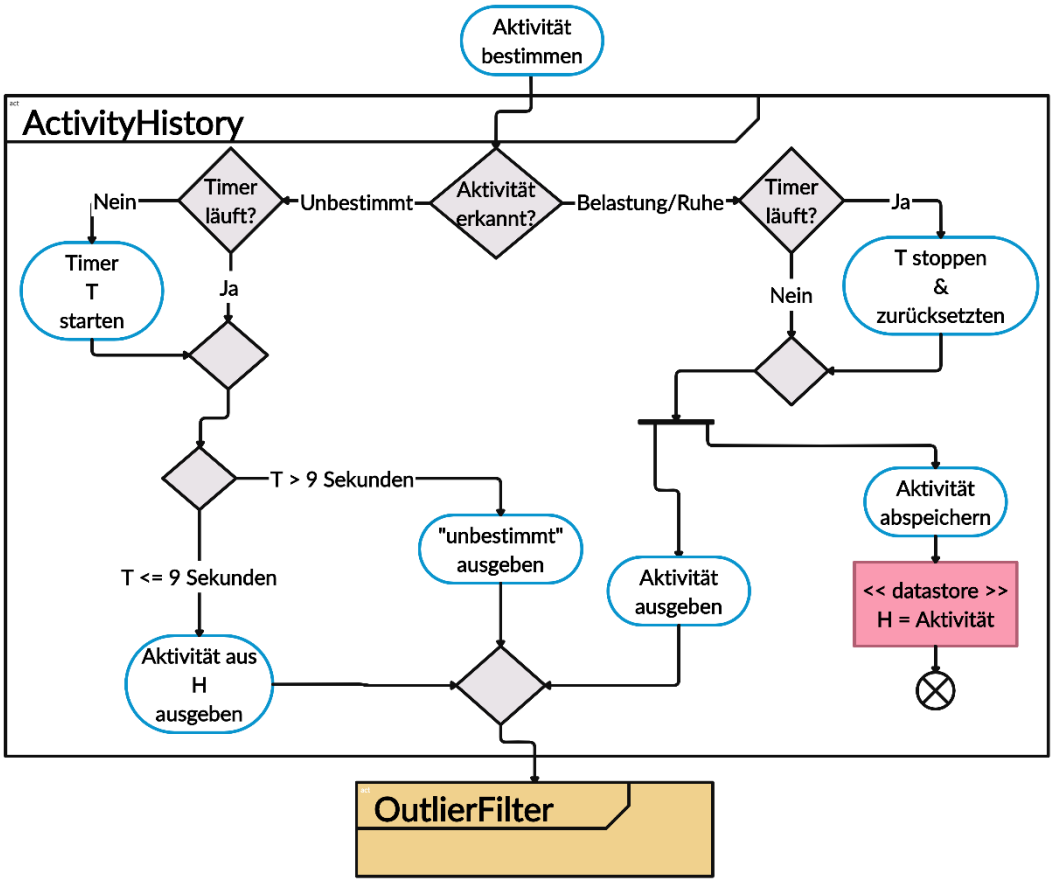


Abbildung 44 - Aktivitätsdiagramm für den AktivityHistory Prozess

Nachdem eine Aktivität detektiert wurde („Ruhe“, „Belastung“ oder „unbestimmt“), wird diese im SP1 durch den History-Prozess weiterverarbeitet. Abbildung 44 stellt das Aktivitätsdiagramm und den Informationsfluss des Softwareprozesses dar. Der Prozess dient dem Glätten und der kurzfristigen Toleranz von „unbestimmt“ erkannten Aktivitäten. Dies kann durch Hardwarefehler (Sensorausfälle, Sensorabweichungen, Smartphoneausfall, Kommunikationsstörungen) sowie Softwarefehler (Verzögerungen durch Überlastung, Synchronisationsfehler, Bitfehler) entstehen. Wurde eine Aktivität erkannt, wird diese persistent abgespeichert und im System weiterverarbeitet. Sollte keine Aktivität klassifiziert werden, erkennt die History dies, startet einen Timer und hält das System durch die Weiterleitung der zuletzt erkannten Aktivität aus dem Speicher am Leben. Wird keine Aktivität innerhalb der nächsten 9 Sekunden erkannt, so liegt ein permanenter Fehler im System an und „unbestimmt“ wird als Aktivität im System weiterverarbeitet. Dieser Prozess verbessert die Dienstgüte der BDS Funktionalität, verringert das zu häufige Schalten in den FB-Modus und hält das System trotz Fehler gemäß den Anforderungen reaktionsfähig.

Um ein zu häufiges Wechseln der Betriebsart zu verhindern, wird nach der Verarbeitung der ActivityHistory der OutlierFilter-Prozess angewandt. Abbildung 45 zeigt den Datenfluss des OutlierFilter-Prozesses im Detail. Dem Prozess stehen eine Queue (Q), eine zuletzt erkannte Aktivität (A) sowie ein Zähler (C) als Datenstruktur zur Verfügung. In Q werden die letzten drei Eingaben abgespeichert. Dies ermöglicht es zu prüfen, wie lange ein Aktivitätsstatus schon anliegt oder ob das System oft den Aktivitätsstatus in kurzer Zeit ändert bzw. ob keine Aktivität erkannt wurde. In A wird die Aktivität abgespeichert, die der Prozess zuvor ausgegeben hat. Der Zähler C dient zur Gewährleistung der Dienstgüteanforderung #120 und #130 aus Tabelle 11 und Tabelle 12, dass mindestens 5 Sekunden die gleiche Aktivität erkannt wurde, bevor ein Zyklus aktiviert wird. Im Gegensatz zum History-Prozess der im Fehlerfall („unbestimmt“ erkannte Aktivität) eingreift und das System vor transienten Fehlern schützt, verbessert der OutlierFilter die Qualität des GBS Service durch die Vermeidung von unnötigen Betriebswechseln.

Auf Grundlage der Eingabe unterscheidet der Prozess die folgenden drei Fälle:

1. Wurde eine „unbestimmte“ Aktivität detektiert, wird diese in Q und A abgespeichert und aus Sicherheitsgründen wird „unbestimmt“ im System weiterverarbeitet.
2. Liegt „Belastung“ als erkannte Aktivität an, wird die Aktivität in Q eingefügt und eine weitere Fallunterscheidung wird benötigt.
    I. Befindet sind in Q nur die Aktivität „Belastung“, wird die Aktivität in A gespeichert sowie ausgegeben und weiterverarbeitet.
    II. Enthält Q eine zuvor erkannte „unbestimmte“ Aktivität, wird „unbestimmt“ anstelle von „Belastung“ ausgegeben.

III. Ist neben „Belastung“ auch „Ruhe“ in Q enthalten, wird der alte Aktivitätszustand nicht verändert. Damit dieser Zustand nicht unendlich lange anliegt, wird ein Zähler C verwendet.

(1) In den ersten drei Fällen (C < 3) wird die Aktivität aus A ausgegeben und C um eins erhöht.

(2) Ist nach drei erkannten Aktivitäten (C >= 3) immer noch keine gleiche Aktivität in Q enthalten, wird „unbestimmt“ in A abgespeichert und ausgegeben.

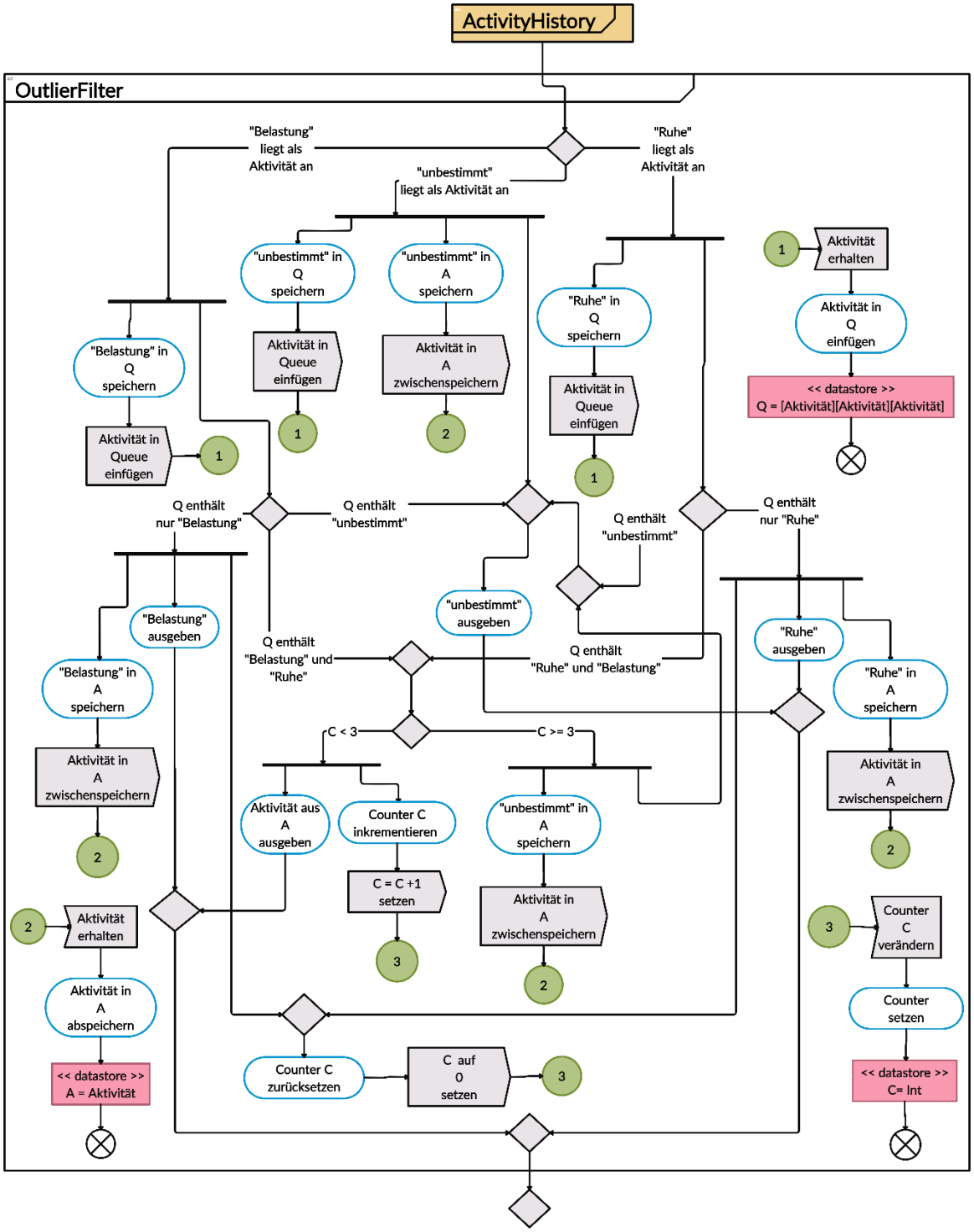


Abbildung 45 - Aktivitätsdiagramm für den OutlierFilter Prozess

3. Im dritten Fall liegt „Ruhe“ als erkannte Aktivität am Prozess an. In diesem Fall wird die Aktivität in Q eingefügt und wie im vorherigen 2. Fall fortgeführt.
    I. Befindet sind in Q nur die Aktivität „Ruhe“, wird die Aktivität in A gespeichert und „Ruhe“ wird ausgegeben und weiterverarbeitet.
    II. Enthält Q eine zuvor erkannte „unbestimmte“ Aktivität, wird „unbestimmt“ anstelle von „Ruhe“ ausgegeben.
    III. Ist neben „Ruhe“ auch „Belastung“ in Q enthalten, wird der alte Aktivitätszustand nicht verändert. Damit dieser Zustand nicht unendlich lange anliegt, wird ein Zähler C verwendet.
        (1) In den ersten drei Fällen (C < 3) wird die Aktivität aus A ausgegeben und C immer um eins erhöht.
        (2) Ist nach drei erkannten Aktivitäten (C >= 3) immer noch keine gleiche Aktivität in Q enthalten, wird „unbestimmt“ in A abgespeichert und ausgegeben.

Der Prozess des OutlierFilters ermöglicht eine verbesserte Dienstgüte für die Spülzyklusfunktionalität. Dadurch kann das BDS kurzfristige Bewegungen des Patienten tolerieren, und somit wird ein zu häufiges Umsteuern zwischen SZ-, NZ- und FB-Modus verringert.

Wurde eine „unbestimmte“ Aktivität erkannt, so endet der Datenfluss und es wird kein neues Steuersignal an die LVAD-Steuereinheit übermittelt. Liegt nach dem OutlierFilter-Prozess „Belastung“ als erkannte Aktivität an, wird ein möglicher laufender SZ-Modus gestoppt und eine LVAD-Steuernachricht mit der Solldrehzahl von 2500 wird an den LVAD versendet. Wurde „Ruhe“ detektiert, wird der CleaningCycle-Prozess ausgeführt. Abbildung 46 zeigt das Aktivitätsdiagramm für diesen Softwareprozess. Der CleaningCycle-Prozess prüft zu Beginn, ob der SZ-Modus schon aktiv und im Betrieb ist. Ist der SZ-Modus nicht aktiv, wird dieser Zyklus sowie ein SZ-Timer gestartet. Für eine Zeitspanne von 10 Sekunden wird nun die LVAD-Pumpe mit einer RPM von 1800 durch das GBS gesteuert. Liegt „Ruhe“ am Prozess an und der SZ-Modus ist aktiv, wird der SZ-Timer überprüft. Wurde die Aorta noch keine 10 Sekunden gespült, wird die Pumpe weiterhin mit 1800 RPM gesteuert. Sind die 10 Sekunden, in der die Aortenklappe gespült wurde, verstrichen, wird der SZ-Timer gestoppt und der NZ-Modus wird aktiviert. In dem NZ-Modus wird die LVAD-Pumpe mit einer RPM von 2500 gesteuert. Um den Zyklus zu überwachen und die zeitliche Anforderung von 5 Sekunden einzuhalten, wird ein NZ-Timer gestartet. Liegt auch nach den 5 Sekunden noch eine Ruhephase an und der NZ-Modus ist aktiv, wird dieser gestoppt und der SZ-Modus wird wieder aktiviert. Dieser periodische Ablauf von SZ-Modus (10 Sekunden) und NZ-Modus (5 Sekunden) wird solange durchgeführt, bis das GBS „Belastung“ als Aktivität erkennt und einen aktiven Spülzyklusablauf sofort beendet und die LVAD-Pumpe mit einer Drehzahl von 2500 RPM steuert.

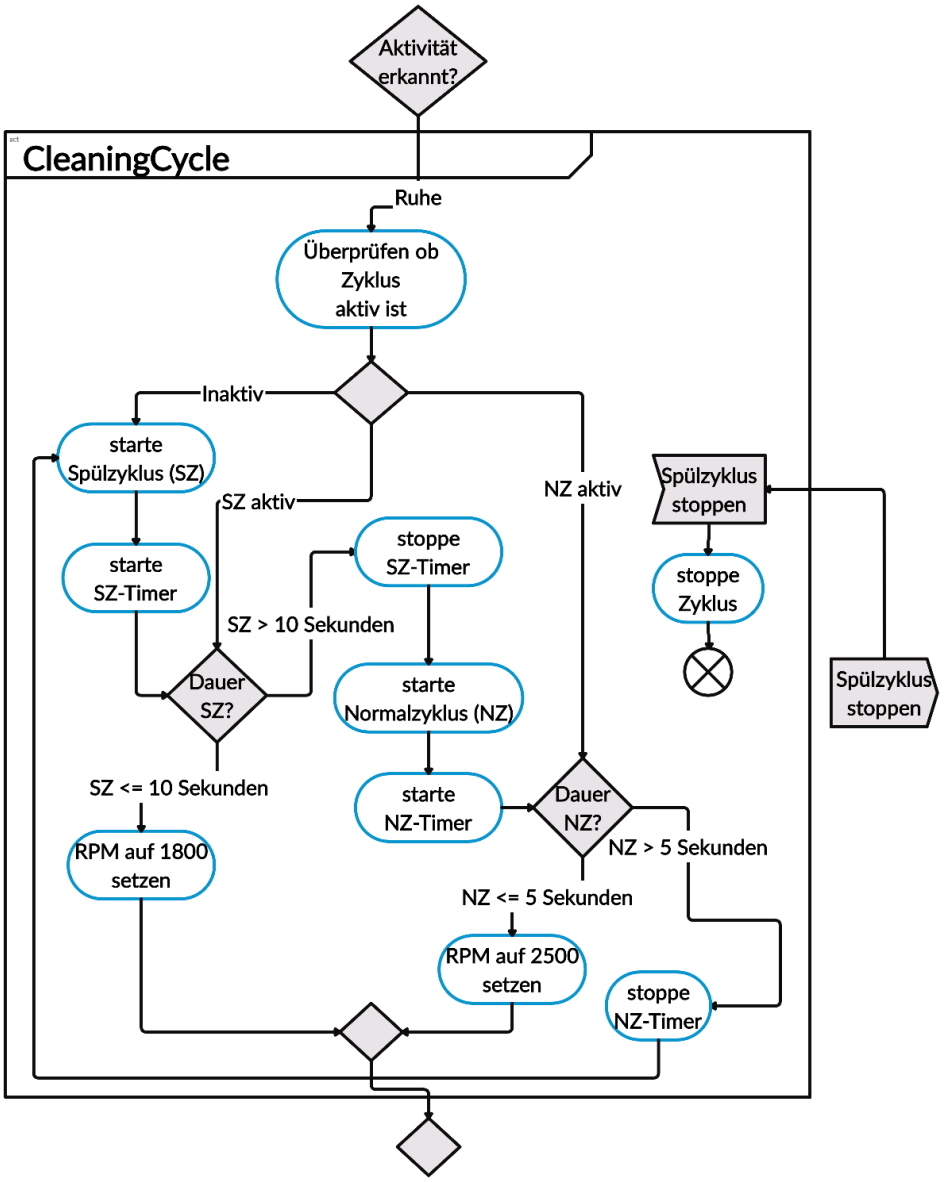


Abbildung 46 - Aktivitätsdiagramm für den CleaningCycle Prozess

Die LVAD-Steuereinheit erhält über die kabellose Bluetoothverbindung die zu steuernden Drehzahlen der LVAD-Pumpe. Abbildung 47 zeigt das Nachrichtenformat der LVAD-Bluetoothnachricht. Die interne Struktur der Nachricht enthält eine eindeutige ID, die Solldrehzahl (SZ- oder NZ-RPM), die sichere RPM-Geschwindigkeit des Fallback-Modus sowie eine digitale Signatur.

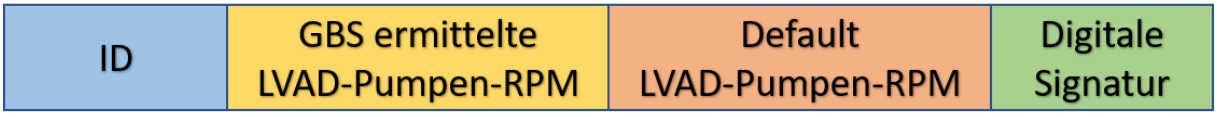


Abbildung 47 - Nachrichtenformat vom Smartphone zum LVAD

Verarbeitet die Steuereinheit periodisch mindestens alle 10 Sekunden ein Steuersignal durch das GBS, ist die erweiterte Funktionalität des Spülzyklus der Aorta aktiv. Dazu wird die GBS ermittelte LVAD-Pumpen-RPM-Geschwindigkeit aus dem Nachrichtenformat entnommen und an die Pumpe weitergeleitet. Kommt es zu

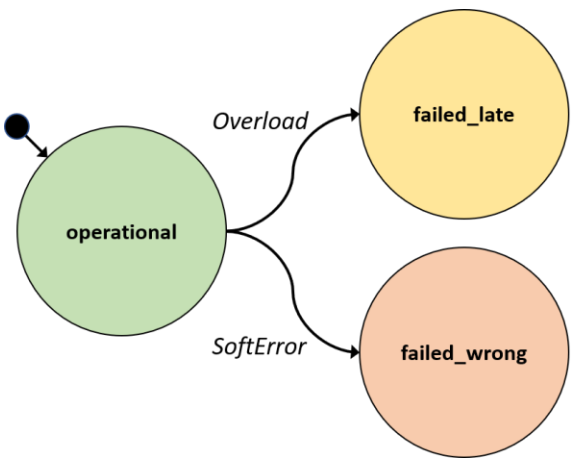
Fehlern bei der Übertragung oder das GBS reagiert über eine Dauer von mehr als 10 Sekunden nicht, wird der FB-Modus des LVAD-PK aktiviert. Dieser steuert die LVAD-Pumpe durch eine medizinisch funktionssichere Default-RPM. Diese RPM ist im LVAD-PK durch das betreuende medizinische Fachpersonal hinterlegt. Für den Anwendungsfall des Spülzyklus der Aortenklappe wurde im Medolution-Projekt eine Default-Geschwindigkeit von 2500 RPM ausgewählt, die an der LVAD-Pumpe im FB-Modus anliegen muss.

Im Folgenden wird die Entwicklung des Duplex-Smartphone-BDS näher betrachtet. Neben den einzelnen Entwicklungsstufen werden Fehlertoleranzmuster eingeführt und beschrieben. Die Erweiterungen der Systemkomponenten werden dazu mittels FT-Analyse analysiert und bewertet.

6 MODELLBASIERTE ANALYSE

Ein wichtiger Aspekt bei der Entwicklung von BDS ist der Einsatz der modellbasierten Entwicklung (MBE). Die Architektur-Beschreibungssprache AADL bietet im Zusammenhang mit dem OSATE-Werkzeug eine effektive Möglichkeit Sicherheitsanalysen auf abstrakter Systemebene durchzuführen. Um frühzeitig im Entwicklungsprozess Sicherheitsanalysen durchführen zu können, müssen Informationen über das Fehlerverhalten von Systemkomponenten sowie Fehler und deren Auftrittswahrscheinlichkeiten im AADL-Modell definiert sein.

Das Fehlerverhaltensmodell für die Hardware- und Softwarekomponenten im Anwendungsfall des LVAD-Spülzyklus ist in Abbildung 48 dargestellt. Eine Systemkomponente befindet sich zu Beginn im Initialzustand *operational* und ist fehlerfrei. Durch ein Fehlerereignis (*Overload* oder *SoftError*) kann eine Zustandsüberführung in einen fehlerhaften Zustand erfolgen. Kommt es zum Beispiel durch eine Systemüberlastung (*Overload-Event*) zu einer Verzögerung in der Verarbeitung, wechselt der Zustand zu *failed_late*. Neben einer Überlastung des Systems kann es zusätzlich zu einem *SoftError*-Ereignis kommen. Entsteht zum Beispiel eine temporäre Störung im Speicher (z.B. Bit-Flip), bewirkt dies einen Zustandswechsel von *operational* zu *failed_wrong*. Diese Fehlerereignisse führen im System zu einer Fehlermaskierung, -transformation oder -weiterleitung. Da der Fokus und das Interesse auf der Fehlerbaumanalyse für ein verspätetes Stellsignal an der LVAD-Steuereinheit bzw. das verspätete Wechseln in den Fallback-Modus liegt, werden keine Reparaturereignisse (Transitionen) zurück in den *operational* Zustand modelliert.




```
error behavior GBS_component
    events
        Overload: error event;
        SoftError: error event;
    states
        operational: initial state;
        failed_late: state;
        failed_wrong: state;
    transitions
        t1: operational -[Overload]-> failed_late;
        t2: operational -[SoftError]-> failed_wrong;
end behavior;
```

Abbildung 48 - Fehlerverhaltensmodell für eine Systemkomponente im LVAD-BDS

Neben dem Fehlerverhaltensmodell, das in Abbildung 48 dargestellt ist, ist ein zusätzliches abstraktes Modell definiert (*GBS_component_nt*), welches keine vordefinierten Transitionen mit Fehlerereignissen besitzt. Dieses Fehlermodell wird verwendet, um die Fehlerausbreitung und -maskierung durch Fehlertoleranzkomponenten abzubilden. Die Übergänge von einem Fehlerzustand in den anderen werden intern in den einzelnen AADL Komponenten durch zusätzliche interne Fehlerereignisse (*GYROFailure, ACCFailure, CalcError, PowerError, BTComError*) sowie den Fehlerfluss definiert. Wir nehmen weiterhin an, dass die zusätzlichen Fehlertoleranzkomponenten (*Watchdog, Voter, Checkpoint, Zähler*) perfekt sind und nicht von sich aus ausfallen können.

Für eine Abschätzung der Auftrittswahrscheinlichkeiten der BDS-Typischen Fehlerereignisse wird im ersten Schritt ein nicht fehlertolerantes LVAD-BDS modelliert und entwickelt. Die Ergebnisse erster experimenteller Bewertungen sowie Analysen des AADL-Modells liefern Auskunft über die Funktionssicherheit und Dienstgüte des BDS. In weiteren Schritten wird das Modell durch die erweiterten Fehlertoleranzmechanismen aus Kapitel 4 angepasst und durch zusätzliche Analysen bewertet. Die Analyseergebnisse der einzelnen Entwicklungsstufen geben erste Erkenntnisse über die Wirkung der Fehlertoleranzmuster auf die Rechtzeitigkeit und Dienstgüte in BDS.

Im Folgenden werden die AADL-Modelle der drei Entwicklungsstufen des LVAD-Spülzyklus-BDS und deren Fehlerbaumanalysen dargestellt. Die erste Entwicklungsstufe stellt das nicht fehlertolerante BDS (*Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen)* dar und dient als Ausgangspunkt für die weiteren Entwicklungsstufen. Die zweite Entwicklungsstufe beschreibt ein einfaches fehlertolerantes BDS (*Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen*). Die letzte Entwicklungsstufe repräsentiert das komplexe fehlertolerante LVAD-Spülzyklus-BDS (*Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranzmechanismen*), welches die Rechtzeitigkeit und Dienstgüteförderung von BDS gewährleistet.

6.1 Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen

Wie im Kapitel zuvor dargestellt, soll durch ein gerätebasiertes System anhand der aktuellen Aktivität des Patienten ein Spülzyklus der Aortenklappe durchgeführt werden. Das AADL-Modell des BDS für einen LVAD-Spülzyklus der Aortenklappe ohne Fehlertoleranzmechanismen ist in Abbildung 49 dargestellt. Das Modell besteht aus zwei Systemen -einem Smartphone und einer LVAD-Steuereinheit- und einer LVAD-Pumpe, die als Aktuator dient. Die Smartphone-Systemkomponente beinhaltet die folgenden Subkomponenten:

- Gerätekomponenten <*Gyroscope, Accelerometer, Battery, BluetoothModule*>,
- Software-Prozesse <*DataGathering, ActivtyDetection, CleaningCycle, MsgComposition*>,
- Software-Threads <*WDgyro, WDacc, DataGathering, AD_Unit, CC_Unit, MsgComposition*>.

Diese Gerätekomponenten sowie die Threadkomponenten dienen im AADL-Modell als Fehlerquellen und stellen potenzielle Gefahrenpunkte im Smartphone-Modell dar. Die System- und Prozesskomponenten dienen der Fehlerausbreitung sowie Fehlermaskierung im Modell.

Die LVAD-Steuereinheitskomponente besteht aus den folgenden Subkomponenten:

- Gerätekomponenten <*BluetoothModule*>
- Software-Prozesse <*Control*>
- Software-Threads <*Control_Unit*>

Diese Modellkomponente stellt einen kleinen Perfektionskern dar und besitzt deshalb keine Fehlerquelle. Der Control-Prozess verarbeitet die Fehlerereignisse aus dem Smartphone-Modell sowie dem Bluetooth-Kommunikationsbus. Die LVAD-Pumpe dient als Fehlersenke und steht für die Zuverlässigkeitsanalyse im Mittelpunkt.

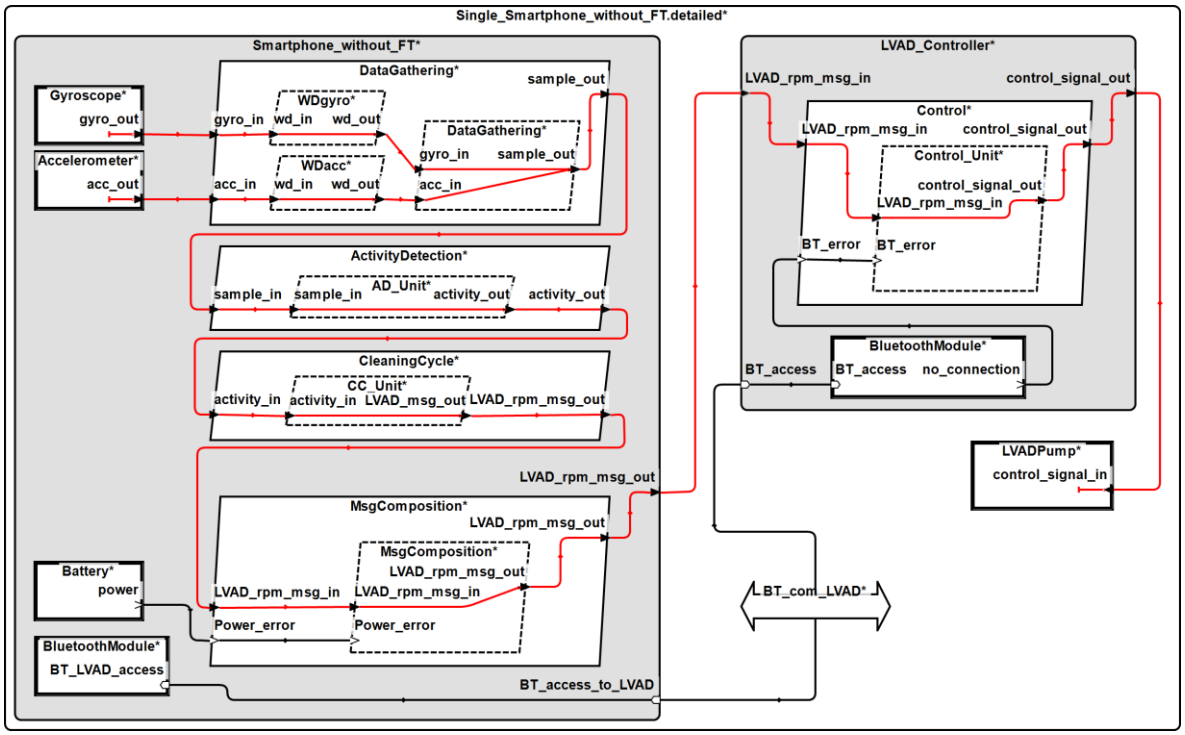


Abbildung 49 - LVAD-Spülzyklus-BDS ohne Fehlertoleranz

Der in Abbildung 49 dargestellte rote Informationsfluss stellt die seriell arbeitenden Systemkomponenten dar. Dieser Informationsfluss erstreckt sich von den Sensoren über die Softwarekomponenten hin zur LVAD-Pumpe.

Um eine quantitative Fehlerbaumanalyse mittels AADL und OSATE durchführen zu können, müssen die einzelnen Fehlerquellen der AADL-Komponenten (*Gyroscope, Accelerometer, ActivityDetection, CleaningCycle, Battery und das BluetoothModule*) mit Fehlern und den dazugehörigen Auftrittswahrscheinlichkeiten attributiert werden. Tabelle 15 listet die Fehlerquellen der einzelnen Modellkomponenten mit ihren Ereignissen und den Auftrittswahrscheinlichkeiten auf.

Im Folgenden werden die einzelnen Abschätzungen der Auftrittswahrscheinlichkeiten erläutert. Für die Wahrscheinlichkeiten wird eine zeitliche Abhängigkeit von 0.0005 Stunden gewählt, da im LVAD-BDS im Abstand von 0.0005 Stunden Sensordaten verarbeitet werden.

Tabelle 15 - Auftrittswahrscheinlichkeiten der einzelnen Komponenten im LVAD-BDS-Modell

AADL-Komponente	**Fehlerereignis**	**Auftritts-wahrscheinlichkeit pro 0.0005 Stunden**	**Fehlertyp**
Gyroscope	Overload	8.38E-2	LateSensorData
Gyroscope	GYROFailure	2.29E-8	LateSensorData
Gyroscope	SoftError	9.18E-7	WrongSensorData
Accelerometer	Overload	8.38E-2	LateSensorData
Accelerometer	ACCFailure	1.24E-8	LateSensorData
Accelerometer	SoftError	9.18E-7	WrongSensorData
ActivityDetection	Overload	8.38E-2	NoValue
ActivityDetection	SoftError	9.18E-7	WrongValue
ActivityDetection	CalcError	1.0E-3	WrongValue
CleaningCycle	Overload	8.38E-2	NoValue
CleaningCycle	SoftError	9.18E-7	WrongValue
Battery	PowerError	2.0E-9	NoPower
BluetoothModule	BTComError	1.36E-4	ServiceError

Wir nehmen an, dass die Wahrscheinlichkeit eines Fehlerereignisses (*Overload, SoftError, CalcError, GYROFailure, ACCFailure, PowerError, BTComError*) während der Laufzeit exponentialverteilt ist.

Für eine stetige Zufallsvariable X sowie die mittlere Anzahl an Fehlerereignissen pro Zeiteinheit (λ), ergibt sich die folgende Verteilungsfunktion:

$$F_X(t) = P(X \leq t) = \begin{cases} 1 - e^{-\lambda * t}, & t \geq 0 \\ 0, & t < 0 \end{cases}$$

Um die Auftrittswahrscheinlichkeiten der einzelnen Fehlerereignisse zu bestimmen, müssen die mittleren Raten der einzelnen Fehlerereignisse bekannt bzw. abgeschätzt werden. Um die nicht bekannten Fehlerraten im BDS-Umfeld abschätzen zu können, wurden verschiedene Experimente durchgeführt. Auf dieser Grundlage wurden für eine Zeiteinheit t = 0.0005/h (Lebensdauer pro Stunde) die entsprechenden Auftrittswahrscheinlichkeiten bestimmt. Eine weitere Eigenschaft der Exponentialverteilung ist der Erwartungswert E(X), der wie folgt definiert ist:

$$E(X) = \int_0^\infty t\lambda e^{-\lambda t} dt = \frac{1}{\lambda}$$

Der Erwartungswert E(X) entspricht der mittleren Betriebsdauer (MTBF) zwischen zwei Fehlerereignissen für eine Zeiteinheit.

6.1.1 Overload-Ereignis

Für eine Überlastung (*Overload*-Ereignis) des Systems und der daraus resultierenden Verzögerung wurde ein Belastungsexperiment mit dem nicht fehlertoleranten BDS durchgeführt. In einem 7-Stunden-Experiment wurde die CPU-Belastung eines Smartphones pro Stunde in 10% Schritten von 0% CPU-Belastung bis auf 60% CPU-Belastung angehoben und die Reaktionszeit der LVAD-Anwendung ausgewertet. Eine detaillierte Beschreibung des Experiments ist in Kapitel 7 aufgeführt. Im Zeitintervall von 7 Stunden kamen insgesamt 1224 Nachrichten nicht rechtzeitig am LVAD an. Dies führt zu einer mittleren Anzahl an *Overload*-Fehlerereignissen von $\lambda \approx$ 175/h. Daraus resultiert eine Auftrittswahrscheinlichkeit für ein *Overload*-Ereignis im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{Overload}}(0.0005) = 1 - e^{-175*0.0005} \approx 8.38E-2$$

Dies bedeutet, dass mit einer Wahrscheinlichkeit von 8.38E-2 eine Steuernachricht innerhalb von 0.0005 Stunden durch ein *Overload*-Ereignis nicht rechtzeitig am LVAD ankommt. Die MTBF zwischen zwei *Overload*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{Overload}) = \frac{1}{175} \approx 0.0057\ h$$

Dies bedeutet, dass im Mittel alle 20.52 Sekunden eine Verzögerung durch eine Überlastung im BDS auftritt.

6.1.2 GYROFailure-Ereignis

Der Gyroskopsensor kann zusätzlich ausfallen und damit zu einem zeitlichen Fehler im System führen. Nach [102] hat ein Gyroskop im Durchschnitt eine Fehlerrate von $\lambda \approx 0.000046/h$. Dies ergibt für ein *GYROFailure*-Ereignis eine Auftrittswahrscheinlichkeit im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{GYROFailure}}(0.0005) = 1 - e^{-0.000046*0.0005} \approx 2.29E-8$$

Was bedeutet, dass mit einer Wahrscheinlichkeit von 2.29E-8 ein *GYROFailure*-Ereignis innerhalb von 0.0005 Stunden durch ein *GYROFailure*-Ereignis im BDS auftritt. Die MTBF zwischen zwei *GYROFailure*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{GYROFailure}) = \frac{1}{0.000046} \approx 21739\ h$$

Das bedeutet, dass im Mittel alle 21739 Stunden eine Verzögerung durch einen Ausfall des Gyroskopsensors im BDS auftritt.

6.1.3 ACCFailure-Ereignis

Der Accelerometersensor hat eine mittlere Fehlerrate von $\lambda \approx 0.000025/h$ [102]. Dies ergibt für ein *ACCFailure*-Ereignis eine Auftrittswahrscheinlichkeit im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{ACCFailure}}(0.0005) = 1 - e^{-0.000025*0.0005} \approx 1.24E-8$$

Was bedeutet, dass mit einer Wahrscheinlichkeit von 1.24E-8 ein *ACCFailure*-Ereignis innerhalb von 0.0005 Stunden im BDS anliegt. Die MTBF zwischen zwei *ACCFailure* -Ereignissen im LVAD-BDS beträgt:

$$E(X_{ACCFailure}) = \frac{1}{0.000025} = 40000\ h$$

Im Mittel tritt alle 40000 Stunden eine Verzögerung durch einen Ausfall der Accelerometerkomponente im BDS auf.

6.1.4 SoftError-Ereignis

Neben den zeitlichen Fehlern können auch funktional falsche Werte, insbesondere durch Fehlertoleranzmechanismen, Einfluss auf die Rechtzeitigkeit nehmen. Speicherfehler (*SoftError*-Ereignisse) können in den einzelnen Systemkomponenten bzw. in den dazugehörigen Speichereinheiten auftreten. Nach [103] tritt bei modernen Speichergeräten ein Softerror im Durchschnitt mit 1000 FIT pro Mbit auf. Die Spülzyklus-App belegt max 1840 Mbit. Dies führt zu einer durchschnittlichen Fehlerrate von:

$$\lambda = (1000Mbit * 1840Mbit)/1000000000 = 0.00184/h$$

für ein *SoftError*-Ereignis pro Stunde. Daraus folgt, dass ein *SoftError*-Ereignis eine Auftrittswahrscheinlichkeit im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{SoftError}}(0.0005) = 1 - e^{-0.00184*0.0005} \approx 9.2E-7$$

besitzt. Dies bedeutet, dass mit einer Wahrscheinlichkeit von 9.2E-7 ein *SoftError*-Ereignis innerhalb von 0.0005 Stunden im BDS auftritt. Konservativ wird dieser Wert als Auftrittswahrscheinlichkeit für die einzelnen Systemkomponenten (*Gyroscope, Accelerometer, ActivityDetection und CleaningCycle*) im BDS-Kontext für ein *SoftError*-Ereignis angenommen. Die MTBF zwischen zwei *SoftError*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{SoftError}) = \frac{1}{0.00184} \approx 543\ h$$

Das bedeutet, dass im Durchschnitt alle 543 Stunden eine Verzögerung durch einen Speicherfehler im BDS auftreten kann.

6.1.5 CalcError-Ereignis

Gegenüber dem *SoftError*-Ereignis, welches einen Speicherfehler darstellt, stellen Berechnungsfehler der Funktionen (*CalcError*-Ereignis) eine zusätzliche Fehlerquelle im BDS dar. Die Aktivitätserkennung, die mittels Neuronalem-Netz entwickelt wurde, kann fehlerhafte Aktivitäten ausgeben und somit die Dienstgüte beeinflussen. Die experimentelle Erprobung der Aktivitätserkennung ergab eine durchschnittliche Fehlerrate für *CalcError*-Fehlerereignisse von λ = 2/h. Eine detaillierte Beschreibung des Versuchs ist in Kapitel 7 dargestellt. Daraus resultiert eine Auftrittswahrscheinlichkeit für ein *CalcError*-Ereignis im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{CalcError}}(0.0005) = 1 - e^{-2*0.0005} \approx 1.0E-3$$

Dies bedeutet, dass mit einer Wahrscheinlichkeit von 1.0E-3 eine Steuernachricht innerhalb von 0.0005 Stunden nicht rechtzeitig durch ein *CalcError*-Ereignis am LVAD ankommt. Die MTBF zwischen zwei *CalcError*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{CalcError}) = \frac{1}{2} = 0.5\ h$$

Was bedeutet, dass im Durchschnitt alle 2 Stunden eine Verzögerung durch einen Berechnungsfehler der Aktivitätserkennung im BDS auftritt.

6.1.6 PowerError-Ereignis

Weiterhin existieren typische BDS-Fehler wie Batterieausfälle oder Verzögerungen bei der drahtlosen Kommunikation der IoT-Geräte im Modell. Nach [102] treten pro 10^6h im Durchschnitt 3.95 Batterieausfälle auf. Dies führt zu einer durchschnittlichen Fehlerrate pro Stunde von λ = 3.95E-6/h für ein *PowerError*-Ereignis. Daraus folgt, dass ein *PowerError*-Ereignis eine Auftrittswahrscheinlichkeit im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{PowerError}}(0.0005) = 1 - e^{-0.00000395*0.0005} \approx 2.0E-9$$

besitzt. Dies bedeutet, dass mit einer Wahrscheinlichkeit von 2.0E-9 ein *PowerError*-Ereignis innerhalb von 0.0005 Stunden im BDS auftritt. Die MTBF zwischen zwei *PowerError*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{PowerError}) = \frac{1}{0.00000395} \approx 253164\ h$$

Dies bedeutet, dass im Mittel alle 253164 Stunden eine Verzögerung durch eine Batteriestörung im BDS auftritt.

6.1.7 BTComError-Ereignis

Um die Auftrittswahrscheinlichkeit für eine Verzögerung der Bluetooth-Kommunikation (*BTComError*-Ereignis) zu ermitteln, wurde ein Experiment über eine Zeitspanne von 44 Stunden durchgeführt. In dem Experiment tauschten zwei Smartphones (Sender/Empfänger) im Intervall von 0.0005 Stunden Steuernachrichten (siehe Abbildung 47) aus. Eine detaillierte Beschreibung des Versuchsaufbau ist in Kapitel 7 aufgeführt. Die Ergebnisse zeigen, dass in der Zeitspanne von 44 Stunden insgesamt 12 Nachrichten verzögert verarbeitet wurden. Dies führt zu einer durchschnittlichen Fehlerrate von λ = 0.273/h für ein *BTComError*-Ereignis. Das wiederum hat zur Folge, dass ein *BTComError*-Ereignis eine Auftrittswahrscheinlichkeit im Zeitintervall pro 0.0005 Stunden von:

$$F_{X_{BTComError}}(0.0005) = 1 - e^{-0.273*0.0005} \approx 1.36E-4$$

besitzt. Dies bedeutet, dass mit einer Wahrscheinlichkeit von 1.36E-4 ein *BTComError*-Ereignis innerhalb von 1.8 Sekunden im BDS auftritt. Die MTBF zwischen zwei *BTComError*-Ereignissen im LVAD-BDS beträgt:

$$E(X_{BTComError}) = \frac{1}{0.273} \approx 3.7\ h$$

Was bedeutet, dass im Mittel alle 3.7 Stunden eine Verzögerung durch die Bluetooth-Kommunikation im BDS auftritt.

6.1.8 Modellierung

Die Fehlerquellen der einzelnen Modellkomponenten mit den zugehörigen Auftrittswahrscheinlichkeiten aus Tabelle 15 müssen durch eine geeignete EMV2-Erweiterung (siehe Kapitel 2.4) den einzelnen Komponenten hinzugefügt werden. Abbildung 50 zeigt die Deklaration des Fehlermodells der Accelerometerkomponente unter AADL.

```
device Accelerometer
        Features
                acc_out: out data port;
        flows
                acc_flow_source: flow source acc_out;
        properties
                Dispatch_Protocol => Periodic;
                Period => 1800ms;
end Accelerometer;

device implementation Accelerometer.impl
        annex EMV2 {**
                use types ErrorLib;
                use behavior ErrorLib::GBS_component;

        error propagations
                acc_out: out propagation {LateSensorData, WrongSensorData};
        flows
                es_acc1: error source acc_out {LateSensorData} when failed_late;
                es_acc2: error source acc_out {WrongSensorData} when failed_wrong;
        end propagations;

        component error behavior
        events
                ACCFailure: error event;
        transitions
                t_acc_failed: operational -[ACCFailure]-> failed_late;
        propagations
                prop_acc_lv: failed_late -[]-> acc_out {LateSensorData};
                prop_acc_wv: failed_wrong -[]-> acc_out {WrongSensorData};
        end component;

        Properties
                EMV2::OccurrenceDistribution => [ProbabilityValue =>
                ProbabilityValue::Internal_OverloadError_Late_Sensor;Distribution => Exponential;]
                applies to Overload;

                EMV2::OccurrenceDistribution => [ProbabilityValue =>
                ProbabilityValue::Internal_ACCFailure_Late_Sensor;Distribution => Exponential;]
                applies to ACCFailure;

                EMV2::OccurrenceDistribution => [ProbabilityValue =>
                ProbabilityValue::Internal_SoftError_Wrong_Sensor;Distribution => Exponential;]
                applies to SoftError;
                **};
end Accelerometer.impl;
```

Abbildung 50 - Fehlermodell der Accelerometerkomponente in AADL

Das Accelerometer-Modell verwendet das generische Fehlerverhaltensmodell aus Abbildung 48. Es legt fest, dass die Komponente eine Fehlerquelle für die beiden Fehlertypen *LateSensorData* und *WrongSensorData* darstellt. Neben den Fehlerereignissen *Overload* und *SoftError* wird zusätzlich ein interner *ACCFailure* deklariert. Dieses Ereignis führt ebenfalls zu einem Zustandsübergang in den *failed_late* Zustand.

```
...
error propagations
        WD_in: in propagation {LateSensorData, WrongSensorData};
        WD_out: out propagation {NoSensorData, WrongSensorData};
flows
        ep_wd1: error path WD_in {LateSensorData} -> WD_out {NoSensorData};
        ep_wd2: error path WD_in {WrongSensorData} -> WD_out {WrongSensorData};
end propagations;

component error behavior
propagations
        prop_wd_lv: operational -[WD_in {LateSensorData}]-> WD_out {NoSensorData};
        prop_wd_wv: operational -[WD_in {WrongSensorData}]-> WD_out {WrongSensorData};
end component;
...
```

(a) Fehlerverhalten eines Watchdog-Threads des DataGathering-Prozesses

```
...
error propagations
        ACC_in: in propagation {NoSensorData, WrongSensorData};
        Gyro_in: in propagation {NoSensorData, WrongSensorData};

        sample_out: out propagation {NoValue, WrongValue};
flows
        ...
end propagations;

component error behavior
transitions
        t_nv: operational -[
                1ormore((Gyro_in {NoSensorData} and ACC_in {NoError}),
                (Gyro_in {NoError} and ACC_in {NoSensorData}),
                (Gyro_in {NoSensorData} and ACC_in {NoSensorData}),
                (Gyro_in {WrongSensorData} and ACC_in {NoSensorData}),
                (Gyro_in {NoSensorData} and ACC_in {WrongSensorData}))
        ]-> failed_late;
        ...
propagations
        prop_nv: failed_late -[]-> sample_out {NoValue};
        prop_wv: failed_wrong -[]-> sample_out {WrongValue};
end component;
...
```

(b) Fehlerverhalten des DataGathering-Threads des DataGathering-Prozesses

Abbildung 51 - Fehlermodell des DataGathering-Prozesses in AADL

Ergänzend wird die Fehlerausbreitung über die einzelnen Schnittstellen definiert. Für die Accelerometerkomponente, die einen Ausgangsport (*acc_out*) besitzt, wird im Zustand *failed_late* der Fehler *LateSensorData* und im Zustand *failed_wrong* der Fehler *WrongSensorData* über den Ausgangsport weitergeleitet. Des Weiteren müssen für eine quantitative Fehlerbaumanalyse die Auftrittswahrscheinlichkeiten den Fehlerereignissen zugeordnet werden. Für eine strukturiertere Möglichkeit die Werte im Entwicklungsprozess anzupassen, werden die Auftrittswahrscheinlichkeiten in einer separaten Datei (*ProbabilityValue*) hinterlegt.

Um eine verbesserte Darstellung der Fehlerbäume zu gewährleisten, werden im folgenden einzelne Komponenten zusammengefasst oder schon analysierte Modelle als Blackbox-Komponente weiterverwendet.

Betrachtet man die gesamte Sensoreinheit in Abbildung 49, so besteht diese aus den beiden Sensoren *Gyroscope* und *Accelerometer* sowie dem *DataGathering*-Softwareprozess. Auf Fehlerebene transformieren die Watchdog-Komponenten einen *LateSensorData*-Fehler in einen *NoSensorData*-Fehler. Diese Umwandlung des Fehlertyps ist in Abbildung 51(a) dargestellt. Der *DataGathering*-Thread verarbeitet die gesamten Fehlerereignisse der Sensoren. Das Fehlerverhalten ist durch die Transition *t_nv* abgebildet. Auf abstrakter Fehlerebene wird eine Verzögerung erkannt, wenn mindesten einer der beiden Sensoren keine Werte liefert.

Ab diesem Punkt breitet sich der Fehlertyp *NoValue* im Modell aus und beeinflusst die Zuverlässigkeit des LVAD-BDS.

6.1.9 Fehlerbaumanalyse

Um die rechtzeitige Reaktion des Systems bewerten zu können, werden die einzelnen Modellkomponenten auf den *failed_late* Zustand analysiert. Abbildung 52 zeigt die vier minimalen Schnittmengen die zu einem *NoValue*-Ereignis durch die Sensoreinheit führen. Die erste Schnittmenge beschreibt das Fehlerereignis eines technischen Ausfalls (*ACCFailure*) der Accelerometerkomponente und führt zu einem *NoValue*-Fehler. Die zweite Schnittmenge beschreibt eine Verzögerung des Accelerometers durch eine Überlastung (*Overload*) der Smartphone-CPU und führt ebenfalls zu einem *NoValue*-Fehler. Die Schnittmengen drei und vier sind parallel zum Accelerometer technische Fehler bzw. Performancefehler der Gyroskopeinheit. Mit einer Wahrscheinlichkeit von 1.6E-1 pro 0.0005 Stunden tritt ein zeitlicher Fehler durch die Sensoreinheit auf. Dies entspricht einer abgeschätzten mittleren Fehlerrate von ca. 320 *NoValue*-Fehlern pro Stunde, die im Modell durch die Sensoreinheit entstehen können.

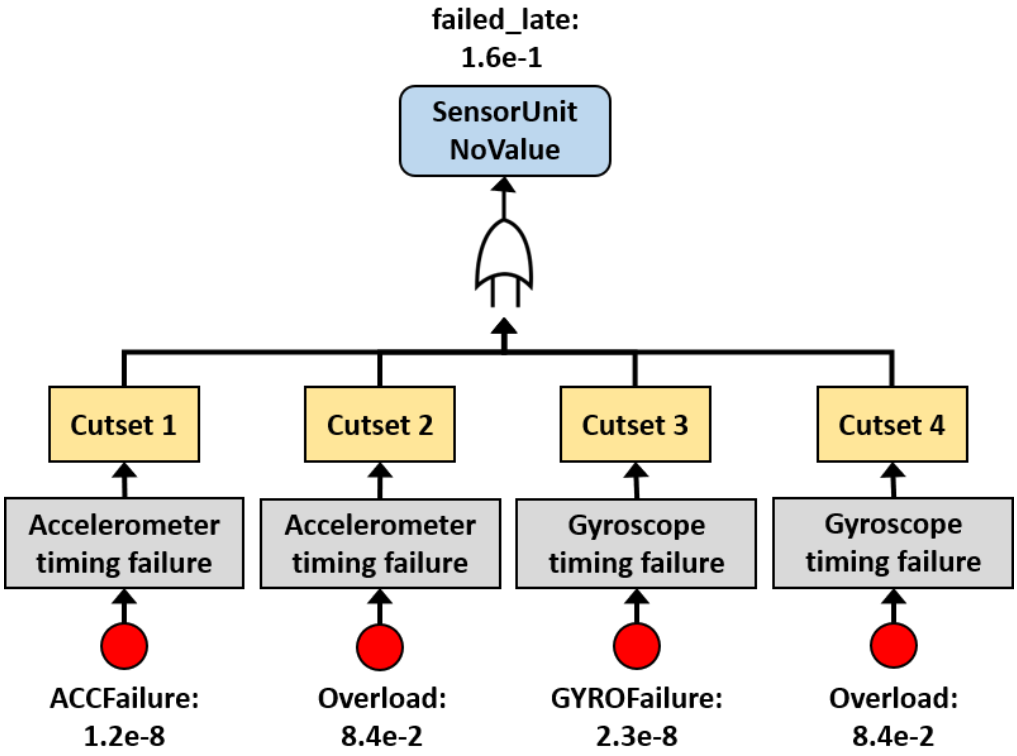


Abbildung 52 - Fehlerbaum für den failed_late-Zustand der Sensoreinheit

Neben der Sensoreinheit fallen die Aktivitätserkennung sowie der Spülzyklusprozess gemäß der Auftrittswahrscheinlichkeiten aus Tabelle 15 aus. Um auf Modellebene einen Batterieausfall in der Fehlerausbreitung darstellen zu können, vereint die *MsgComposition*-Komponente den Batterieausfall mit den einzelnen Fehlern der Sensoreinheit, Aktivitätserkennung und des Spülzyklus.

```
...
error propagations
        LVAD_RPM_Msg_in: in propagation {NoValue, WrongValue};
        Power_error: in propagation {NoPower};

        LVAD_RPM_Msg_out: out propagation {NoValue, WrongValue};
flows
        ep_1: error path LVAD_RPM_Msg_in {NoValue} -> LVAD_RPM_Msg_out {NoValue};
        ep_2: error path LVAD_RPM_Msg_in {WrongValue} -> LVAD_RPM_Msg_out {WrongValue};
        ep_3: error path Power_error {NoPower} -> LVAD_RPM_Msg_out {NoValue};
end propagations;

component error behavior
transitions
        t_nv: operational -[1ormore(LVAD_RPM_Msg_in {NoValue}, Power_error {NoPower})]-> failed_late;
        t_wv: operational -[LVAD_RPM_Msg_in {WrongValue}]-> failed_wrong;
propagations
        prop_wd_lv: failed_late -[]-> LVAD_RPM_Msg_out {NoValue};
        prop_wd_wv: failed_wrong -[]-> LVAD_RPM_Msg_out {WrongValue};
end component;
...
```

Abbildung 53 - Fehlermodell der MsgComposition-Komponente in AADL

Abbildung 53 zeigt das Fehlermodell der *MsgComposition*-Komponente unter AADL. Dies ermöglicht es, das Fehlerverhalten einer Smartphonekomponente für ein zeitliches Fehlerereignis zu modellieren (siehe Transition *t_nv*). Neben den zeitlichen Fehlerereignissen der einzelnen Softwarekomponenten führt ein *NoPower*-Ereignis zu einer zeitlichen Verzögerung (*NoValue*-Ereignis).

Die qualitative und quantitative Fehlerbaumanalyse des gesamten nicht fehlertoleranten LVAD-BDS ist in Abbildung 54 dargestellt. Der minimale Fehlerbaum besteht aus fünf unabhängigen Schnittmengen, die zu einem zeitlichen Fehler an der LVAD-Pumpe führen. Jede einzelne Fehlerquelle führt zu einer Verzögerung an der LVAD-Steuereinheit. Die erste Schnittmenge stellt einen Batterieausfall des Smartphones dar. Die zweite Menge beschreibt eine Verzögerung bei der Bluetooth-Kommunikation zwischen Smartphone und LVAD-Steuereinheit. Die Schnittmengen drei, vier und fünf stellen die Fehlerereignisse der Sensoren, der Aktivitätserkennung sowie des Spülzyklus eines Smartphones dar. Mit einer Wahrscheinlichkeit von 2.9E-1 pro 0.0005 Stunden tritt ein zeitlicher Fehler (*NoValue*) an der LVAD-Steuereinheit auf. Dies entspricht einer abgeschätzten mittleren Fehlerrate von 580 nicht rechtzeitig eintreffenden Steuersignalen pro Stunde an der LVAD-Pumpe.

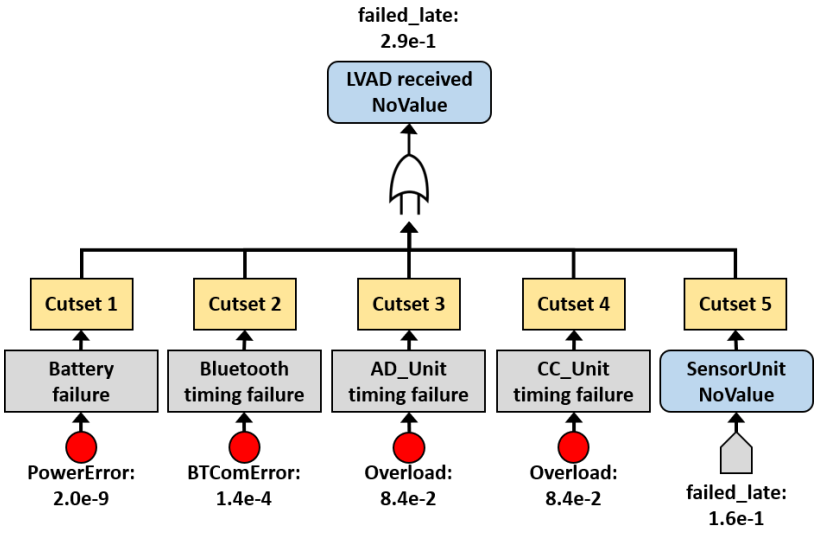


Abbildung 54 - Fehlerbaum für den failed_late-Zustand der LVAD-Steuereinheit durch ein Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen

Das Ergebnis zeigt, dass die einzelnen Hardware- und Softwarekomponenten jeweils als einzelner Ausfallpunkt agieren. Das System erfüllt die Anforderungen bezüglich Zuverlässigkeit und Rechtzeitigkeit des LVAD-Spülzyklus durch die unzuverlässigen IoT-Geräte nicht. Um eine Verbesserung zu ermöglichen, werden im Folgenden die einzelnen Komponenten durch die in Kapitel 4 vorgestellten Fehlertoleranzmechanismen erweitert und analysiert.

6.2 Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen

Als erste Maßnahme wird die LVAD-Steuereinheit um einen Fallback-Modus erweitert. Der Perfektionskern überwacht die IoT-Aktivitäten und schaltet im Ausnahmefall nach 10 Sekunden in einen funktionssicheren Zustand. Abbildung 55 stellt das Modell der erweiterten AADL-LVAD-Steuereinheit-Komponente dar.

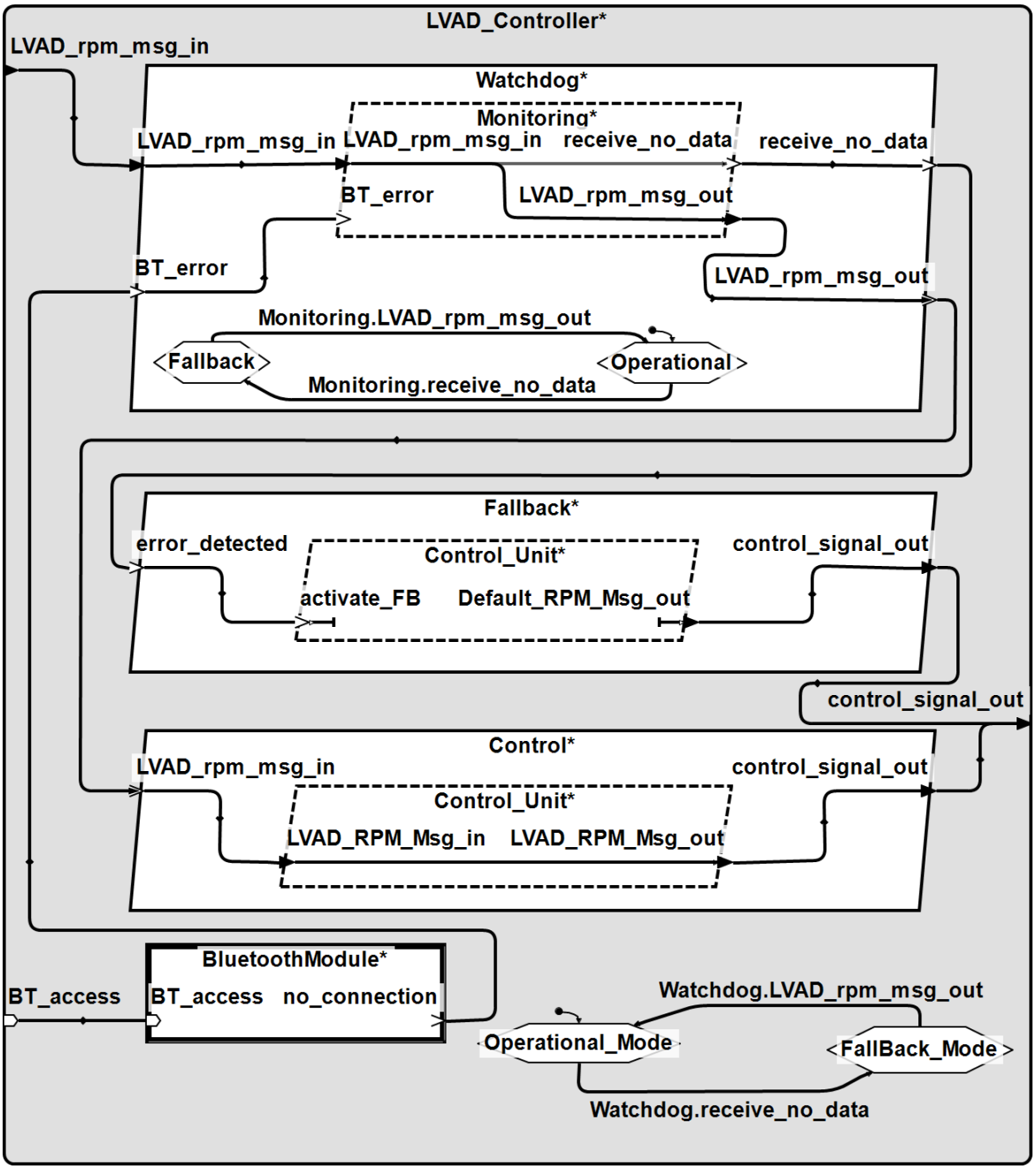


Abbildung 55 - Erweiterte LVAD-Steuereinheit mit Fallback-Modus

Der *Watchdog*-Prozess mit seiner Monitoring-Funktionalität überwacht die *NoValue*-Fehlerereignisse des gerätebasierten Systems sowie der Bluetooth-Kommunikation. Die *Watchdog*-Komponente aktiviert zur Steuerung der LVAD-Pumpe einen der

beiden Prozesse *Fallback* und *Control*. Abbildung 56 stellt das Fehlermodell des Monitoring-Threads unter AADL dar. Dem Modell stehen zusätzlich sechs Fehlerereignisse zur Verfügung, um kurzfristige transiente Fehler am LVAD zu tolerieren.

```
...
error propagations
        LVAD_rpm_msg_in: in propagation {NoValue, WrongValue};
        BT_error: in propagation {NoValue};

        LVAD_rpm_msg_out: out propagation {WrongValue};
        receive_no_data: out propagation {NoValue};
flows
        ep_wd_nv: error path LVAD_rpm_msg_in {NoValue} -> receive_no_data {NoValue};
        ep_wd_nv2: error path LVAD_rpm_msg_in {NoValue} -> LVAD_rpm_msg_out {WrongValue};
        es_wd_nv3: error sink LVAD_rpm_msg_in {NoValue};
        ep_wd_wv: error path LVAD_rpm_msg_in {WrongValue} -> LVAD_rpm_msg_out {WrongValue};
        ep_bt_nv: error path BT_error {NoValue} -> receive_no_data {NoValue};
        ep_bt_nv2: error path BT_error {NoValue} -> LVAD_rpm_msg_out {WrongValue};
        es_bt_nv3: error sink BT_error {NoValue};
end propagations;

component error behavior
events
        StoredWrongValue: error event;
        OneMissingSignal: error event;
        TwoMissingSignal: error event;
        ThreeMissingSignal: error event;
        FourMissingSignal: error event;
        OneSignalTolerance: error event;
transitions
        t_fb: operational -[
                1ormore((LVAD_rpm_msg_in {NoValue} and OneSignalTolerance),
                (BT_error {NoValue} and OneSignalTolerance))
        ]-> fallback;

        t_ef: operational -[
                1ormore((LVAD_rpm_msg_in {NoError}),
                (LVAD_rpm_msg_in {NoValue}),
                (LVAD_rpm_msg_in {NoValue} and OneMissingSignal),
                (LVAD_rpm_msg_in {NoValue} and TwoMissingSignal),
                (LVAD_rpm_msg_in {NoValue} and ThreeMissingSignal),
                (LVAD_rpm_msg_in {NoValue} and FourMissingSignal),
                (BT_error {NoValue}),
                (BT_error {NoValue} and OneMissingSignal),
                (BT_error {NoValue} and TwoMissingSignal),
                (BT_error {NoValue} and ThreeMissingSignal),
                (BT_error {NoValue} and FourMissingSignal))
        ]-> operational;
        ...
propagations
        prop_wd_lv: fallback -[]-> receive_no_data {NoValue};
        prop_wd_ef: operational -[]-> LVAD_rpm_msg_out {NoError};
        ...
detections
        fallback -[]-> receive_no_data !;
end component;
...
```

Abbildung 56 - Fehlermodell des Monitoring-Threads der LVAD-Watchdog-Komponente in AADL

Das Fehlerereignis *StoredWrongValue* dient zur Modellierung einer zuvor falsch ermittelten Pumpengeschwindigkeit des gerätebasierten Systems, welches im

zeitlichen Fehlerfall weiterhin an der LVAD-Pumpe anliegt. Die anderen fünf Fehlerereignisse dienen der Modellierung des Fallback-Modus. Ein *NoValue*-Fehler wird in einem Zeitfenster von 10 Sekunden maskiert (Transition *t_ef*) und stellt eine Senke für die Fehlerausbreitung dar (Fluss *es_wd_nv3* und *es_bt_nv3*). Zusätzlich stellt die modellierte Fehlerausbreitung *ep_wd_nv2* und *ep_bt_nv2* die potenziell nicht korrekten Geschwindigkeitsparameter an der LVAD-Pumpe dar. Transition *t_fb* dient der Modellierung für ein Überschreiten der zeitlichen Anforderungen. In diesem Fall, wird der Fallback-Modus aktiviert und ein *NoValue*-Fehler über die Schnittstelle *receive_no_data* weitergeleitet.

Durch das nicht fehlertolerante Smartphone liegt ein *NoValue*-Fehler mit einer Wahrscheinlichkeit von 2.9E-1 am LVAD an (siehe Abbildung 54). Für einen Betriebswechsel in den Fallback-Modus müssen neben dem *NoValue*-Fehler noch zusätzlich 8.2 Sekunden lang keine Signale (*OneSignaleTolerance*-Fehlerereignis) empfangen worden sein. Mit einer Wahrscheinlichkeit von $2.9E\text{-}1^{4.555} \approx 3.6E\text{-}3$ liegen 8.2 Sekunden lang keine Signale am LVAD an. Abbildung 57 stellt den Fehlerbaum für das Wechseln in den Fallback-Modus durch das nicht fehlertolerante Smartphone dar. Die einzelnen Schnittmengen – Cutset1 bis Cutset5- werden durch das zusätzliche Ereignis *OneSignaleTolerance* erweitert. Das System wechselt mit einer Wahrscheinlichkeit von 1.2E-3 pro 0.0005 Stunden in den Fallback-Modus. Dies entspricht einer abgeschätzten mittleren Fehlerrate von 2.4 Betriebswechseln pro Stunden.

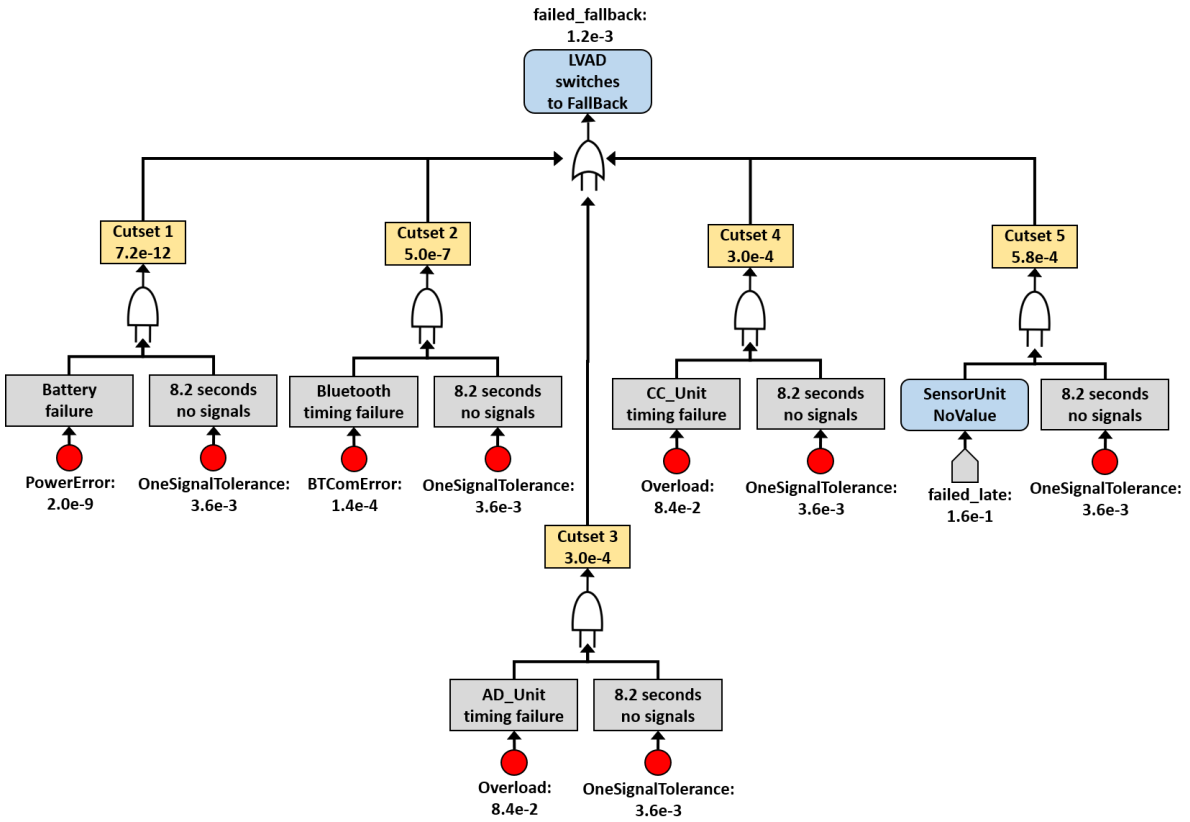


Abbildung 57 - Fehlerbaum für den failed_fallback-Zustand der LVAD-Steuereinheit durch ein Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen

Durch den erweiterten Perfektionskern mit Fallback-Modus wird die Funktionssicherheit zwar gewährleistet, doch aus Sicht der Dienstgüte müssen die auftretenden Fehler sowie Betriebswechsel deutlich minimiert werden. Dazu werden im Folgenden die Softwarekomponenten (*ActivityDetection*-Prozess und *CleaningCycle*-Prozess) durch geeignete Fehlertoleranzmechanismen angepasst.

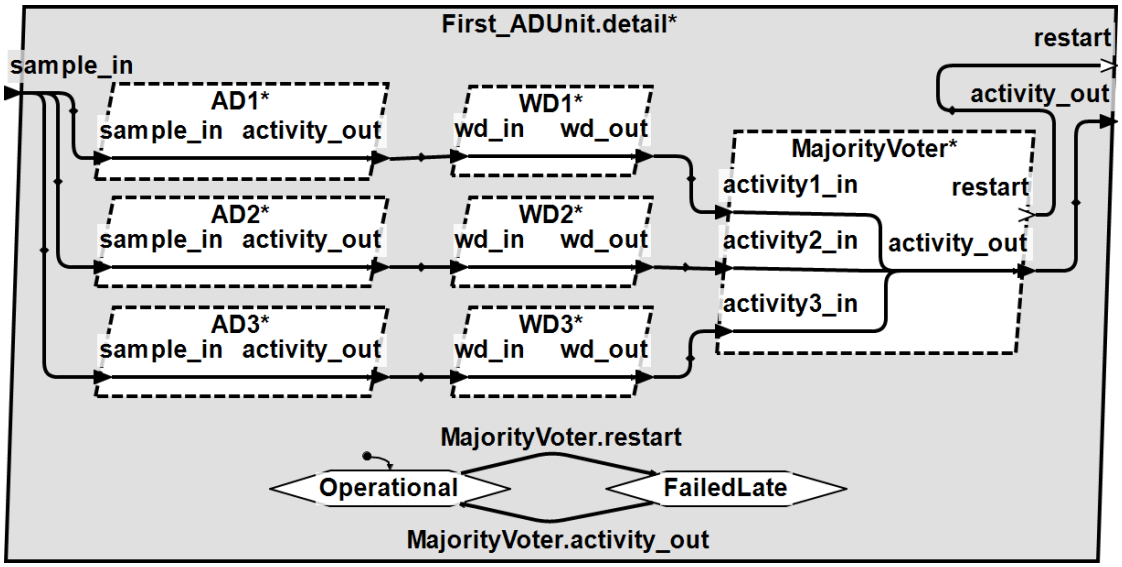


Abbildung 58 - Ein Aktivitätserkennungsprozess des RcB-AV-Musters

Die Aktivitätserkennung wurde nach dem RcB-AV-Muster modelliert und entwickelt. Die Komponente besteht aus drei Recovery-Blöcken. Jeder dieser Blöcke arbeitet nach dem NVP-Muster mit drei parallel arbeitenden Softwarethreads zur Bestimmung der Aktivität. Abbildung 58 stellt das AADL-Modell eines Recovery-Blocks dar. Die *Watchdog*-Threads (*WD1, WD2* und *WD3*) erkennen interne zeitliche Fehler (*InternalLateValue*) der Aktivitäts-Threads (*AD1, AD2* und *AD3*) und lösen ein Ereignis (Maskierung, Transformation oder Weiterleitung) im Mehrheitsentscheider (*MajorityVoter*-Thread) aus. Das Fehlermodell des Entscheidungsprozesses (*MajorityVoter*-Komponente) ist in Abbildung 59 dargestellt.

Das Muster ist stark durch die funktionale Korrektheit geprägt. Speicher- sowie Berechnungsfehler lösen in Zusammenhang mit einem internen zeitlichen Fehler (Transition *t_nv*) einen Zustandswechsel in den Fehlerzustand *failed_late* aus. Das Modell kann *InternalNoValue*-, *InternalWrongValue*- sowie *WrongValue*-Fehlerereignisse maskieren (Fehlerfluss *esk_mv1-3*). Des Weiteren wird definiert, dass ein zeitlicher Fehler zu einer Wiederholung der Aktivitätsberechnung (*ep_mv_4-6*) führt und den *NoValue*-Fehler über die Schnittstelle *restart* weiterleitet.

```
...
error propagations
        Activity1_in: in propagation {InternalNoValue, InternalWrongValue, WrongValue};
        Activity2_in: in propagation {InternalNoValue, InternalWrongValue, WrongValue};
        Activity3_in: in propagation {InternalNoValue, InternalWrongValue, WrongValue};

        Activity_out: out propagation {WrongValue};
        restart: out propagation {NoValue};
flows
        ep_mv1: error path Activity1_in -> Activity_out {WrongValue};
        ep_mv2: error path Activity2_in -> Activity_out {WrongValue};
        ep_mv3: error path Activity3_in -> Activity_out {WrongValue};
        ep_mv4: error path Activity1_in -> restart {NoValue};
        ep_mv5: error path Activity2_in -> restart {NoValue};
        ep_mv6: error path Activity3_in -> restart {NoValue};

        esk_mv1: error sink Activity1_in {InternalNoValue, WrongValue, InternalWrongValue};
        esk_mv2: error sink Activity2_in {InternalNoValue, WrongValue, InternalWrongValue};
        esk_mv3: error sink Activity3_in {InternalNoValue, WrongValue, InternalWrongValue};
end propagations;

component error behavior
        transitions
        ...
        t_nv: operational -[
                1ormore(
                (Activity1_in {InternalNoValue} and Activity2_in {InternalNoValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {NoError} and Activity2_in {InternalNoValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {NoError} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalNoValue} and Activity3_in {NoError}),
                (Activity1_in {InternalWrongValue} and Activity2_in {InternalNoValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalWrongValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalNoValue} and Activity3_in {InternalWrongValue}),
                (Activity1_in {WrongValue} and Activity2_in {InternalNoValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {WrongValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalNoValue} and Activity3_in {WrongValue}),
                (Activity1_in {NoError} and Activity2_in {InternalWrongValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {NoError} and Activity2_in {InternalNoValue} and Activity3_in {InternalWrongValue}),
                (Activity1_in {InternalWrongValue} and Activity2_in {NoError} and Activity3_in {InternalNoValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {NoError} and Activity3_in {InternalWrongValue}),
                (Activity1_in {InternalWrongValue} and Activity2_in {InternalNoValue} and Activity3_in {NoError}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalWrongValue} and Activity3_in {NoError}),
                (Activity1_in {InternalNoValue} and Activity2_in {InternalWrongValue} and Activity3_in {WrongValue}),
                (Activity1_in {InternalNoValue} and Activity2_in {WrongValue} and Activity3_in {InternalWrongValue}),
                (Activity1_in {InternalWrongValue} and Activity2_in {InternalNoValue} and Activity3_in {WrongValue}),
                (Activity1_in {InternalWrongValue} and Activity2_in {WrongValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {WrongValue} and Activity2_in {InternalWrongValue} and Activity3_in {InternalNoValue}),
                (Activity1_in {WrongValue} and Activity2_in {InternalNoValue} and Activity3_in {InternalWrongValue})
                )
        ]-> failed_late;
propagations
        ...
        prop_mv_nv: failed_late -[]-> restart {NoValue};
end component;
...
```

Abbildung 59 - Fehlermodell des MajorityVoter-Threads einer AD-Komponente in AADL

Aus allen Transitionen (*t_nv*) stellt Abbildung 60 den Fehlerbaum mit den minimalen Schnittmengen für das Weiterleiten eines *NoValue*-Ereignisses dar. Ein Recovery-Block der Aktivitätserkennung fällt pro 0.0005 Stunden mit einer Wahrscheinlichkeit von 2.1E-2 durch zeitliche Anomalien aus. Die Ausfallwahrscheinlichkeit für ein *NoValue*-Ereignis sind durch die Cutsets 1-3 geprägt.

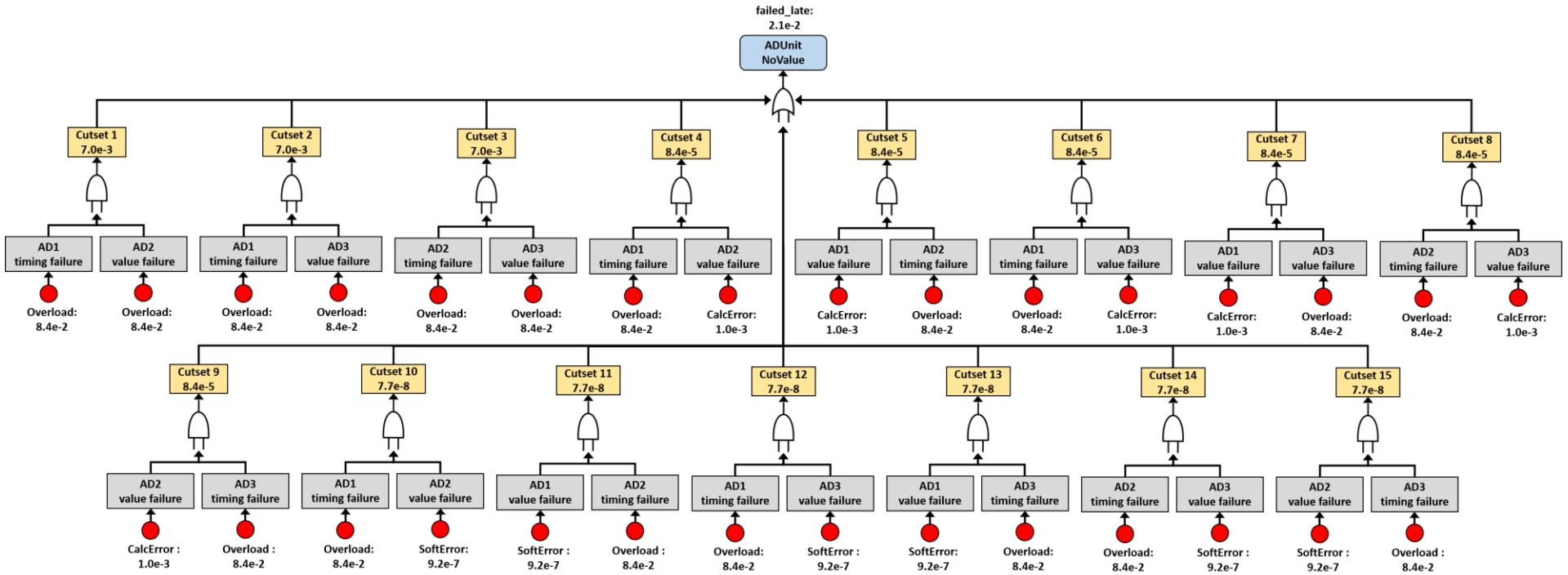


Abbildung 60 - Fehlerbaum für den failed_late-Zustand einer Aktivitätserkennungskomponente des RcB-AV-Musters

Sollten zwei der drei Aktivitätserkennungs-Threads nicht rechtzeitig regieren, wird eine Standby-Komponente aktiviert. Die restlichen Cutsets 4-15 sind die Permutationen aus dem 2-aus-3 Entscheidungsprozess mit einem *NoValue*- und einem *InternalWrongValue*-Ereignis. Der zeitliche Ausfall wird zwar um ein Vierfaches gesenkt, es treten aber pro Stunde noch im Mittel 42 Fehler durch den Aktivitätserkennungsprozess auf.

Um eine weitere Verbesserung der Rechtzeitigkeit und Dienstgüte zu gewährleisten, werden zwei zusätzliche Aktivitätserkennungen als Standby-Komponenten hinzugefügt. Abbildung 61 zeigt die Aktivitätserkennung wie sie im LVAD-BDS verwendet wird.

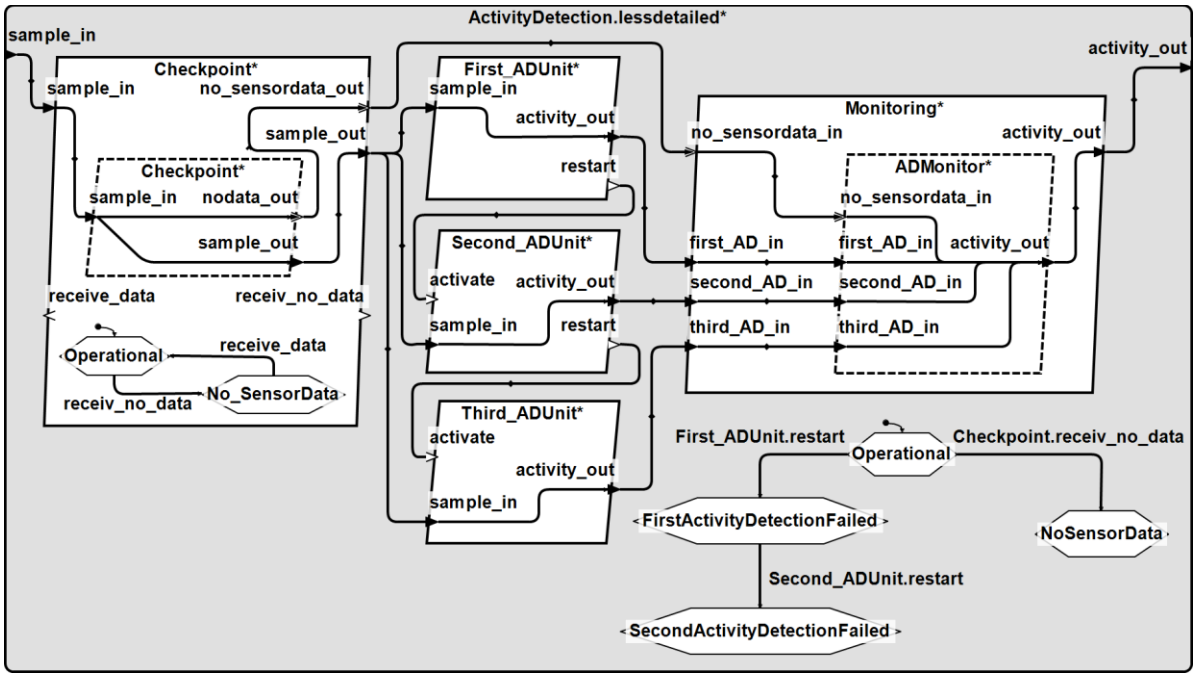


Abbildung 61 - Fehlertolerante Aktivitätserkennung nach dem RcB-AV-Muster

Durch zusätzliche Zeitredundanz wird nach einem Ausfall der ersten Aktivitätserkennung die Berechnung durch eine Standby-Komponente wiederholt. Erst wenn der dritte Recovery-Block zeitlich fehlschlägt, wird keine Aktivität (*NoValue*-Fehlerereignis) im System über die Schnittstelle *activty_out* weitergeleitet. Abbildung 62 stellt die minimale Schnittmenge der Ereignisse dar, die zu einem internen zeitlichen Ausfall der gesamten fehlertoleranten Aktivitätserkennung führen. Durch die sequenzielle Abarbeitung aller Recovery-Blöcke besteht die Schnittmenge aus einer Vereinigung aller drei *failed_late*-Ereignisse. Mit einer Wahrscheinlichkeit von 9.3E-6 pro 0.0005 Stunden tritt ein *NoValue*-Fehler durch die Aktivitätserkennung im System auf. Dies entspricht einer abgeschätzten mittleren Fehlerrate von ca. 0,0185 fehlenden Aktivitäten pro Stunde. Gegenüber den im Mittel auftretenden 168 fehlenden Aktivitäten pro Stunde durch die nicht fehlertolerante Softwarekomponente ist dies eine deutliche Verbesserung.

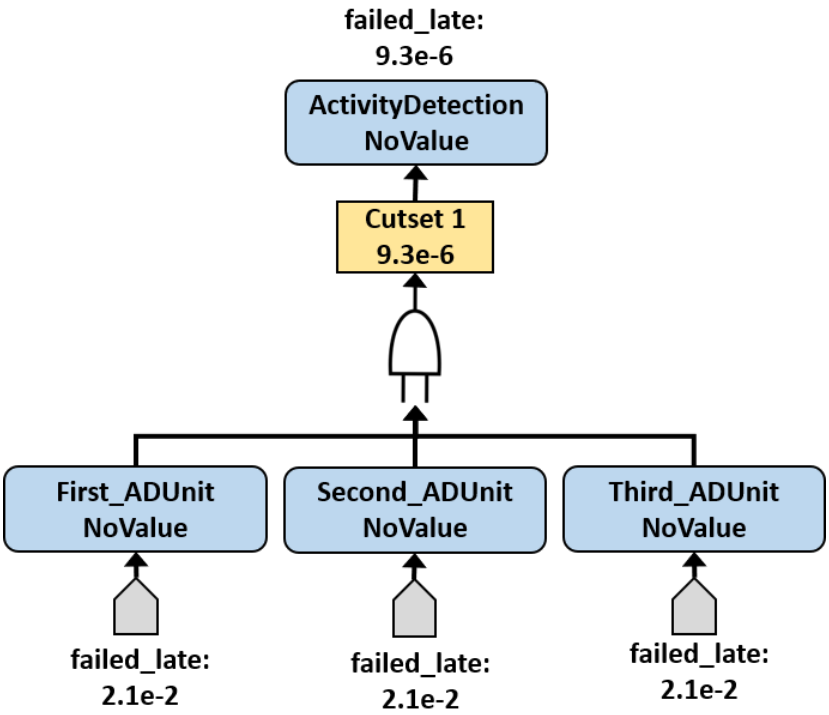


Abbildung 62 - Fehlerbaum für den failed_late-Zustand der Aktivitätserkennung

Neben der Aktivitätserkennung müssen zusätzliche Maßnahmen am Spülzyklusprozess vorgenommen werden. Der Prozess wurde nach dem NSCPAV-Muster modelliert und entwickelt.

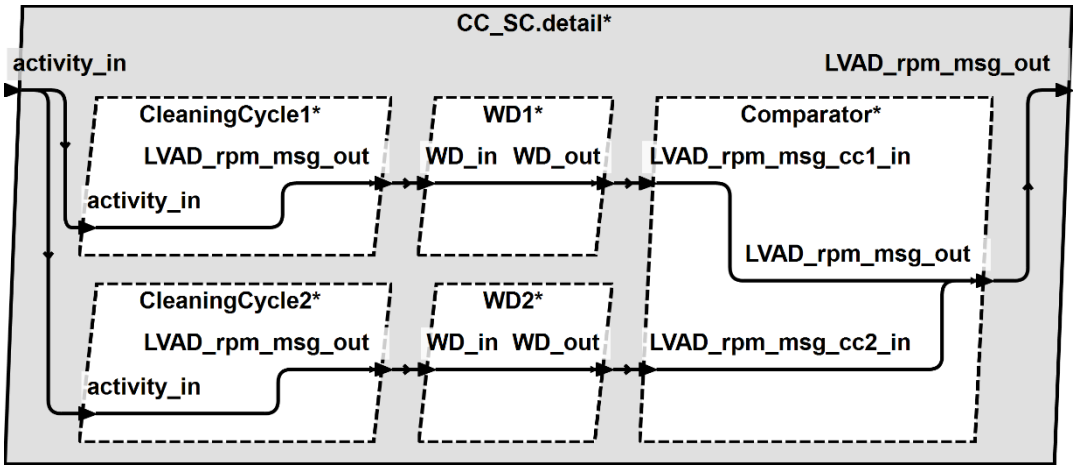


Abbildung 63 - Ein sich selbst überwachende Spülzykluskomponente

Der zentrale Mechanismus ist bestimmt durch die sich selber überwachenden Spülzyklen. Abbildung 63 zeigt das AADL-Modell einer solchen Spülzykluskomponente. Der Aufbau sowie die Logik sind stark durch die Anforderungen nach funktionaler Korrektheit geprägt. Der Entscheidungsprozess wird durch einen Vergleichstest (*Comparator*-Threads) realisiert. Das Fehlermodell des *Comparator*-Threads ist in Abbildung 64 abgebildet. Sowohl zeitliche Fehler als auch Datenfehler führen zu einem Zustandswechsel in den *failed_late* Zustand (Transition *t_nv*).

```
...
error propagations
        LVAD_rpm_msg_cc1_in: in propagation {InternalNoValue, InternalWrongValue, WrongValue};
        LVAD_rpm_msg_cc2_in: in propagation {InternalNoValue, InternalWrongValue, WrongValue};

        LVAD_rpm_msg_out: out propagation {NoValue, InternalWrongValue, WrongValue};
end propagations;

component error behavior
        transitions
...
        t_nv: operational -[
                (LVAD_rpm_msg_cc1_in {InternalNoValue} and LVAD_rpm_msg_cc2_in {InternalNoValue}) or

                (LVAD_rpm_msg_cc1_in {InternalNoValue} and LVAD_rpm_msg_cc2_in {InternalWrongValue}) or
                (LVAD_rpm_msg_cc1_in {InternalNoValue} and LVAD_rpm_msg_cc2_in {WrongValue}) or
                (LVAD_rpm_msg_cc1_in {InternalNoValue} and LVAD_rpm_msg_cc2_in {NoError}) or

                (LVAD_rpm_msg_cc1_in {InternalWrongValue} and LVAD_rpm_msg_cc2_in {InternalNoValue}) or
                (LVAD_rpm_msg_cc1_in {WrongValue} and LVAD_rpm_msg_cc2_in {InternalNoValue}) or
                (LVAD_rpm_msg_cc1_in {NoError} and LVAD_rpm_msg_cc2_in {InternalNoValue}) or

                (LVAD_rpm_msg_cc1_in {InternalWrongValue} and LVAD_rpm_msg_cc2_in {WrongValue}) or
                (LVAD_rpm_msg_cc1_in {InternalWrongValue} and LVAD_rpm_msg_cc2_in {NoError}) or

                (LVAD_rpm_msg_cc1_in {WrongValue} and LVAD_rpm_msg_cc2_in {InternalWrongValue}) or
                (LVAD_rpm_msg_cc1_in {NoError} and LVAD_rpm_msg_cc2_in {InternalWrongValue})
        ]-> failed_late;
propagations
...
        prop_comparator_nv: failed_late -[]-> LVAD_rpm_msg_out {NoValue};
end component;
...
```

Abbildung 64 - Fehlermodell des Comparator-Threads eines SelfChecking_CC-Prozesses

Abbildung 65 stellt die Fehlerbäume mit den minimalen Schnittmengen für ein *NoValue*-Fehler sowie die Schnittmenge für den *WrongValue*-Fehler dar. Ein *NoValue*-Fehler tritt mit einer Wahrscheinlichkeit von 1.6E-1 pro 0.0005 Stunden durch den Spülzyklusprozess im BDS auf.

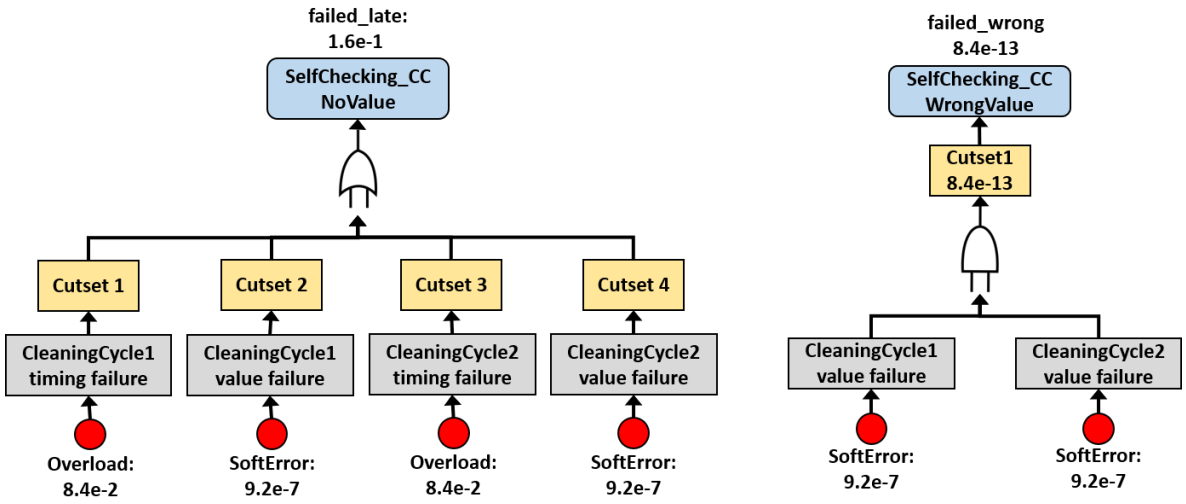


Abbildung 65 - Fehlerbäume für den failed_late- und failed_wrong-Zustand einer SelfChecking_CC-Komponente

Eine *SelfChecking_CC*-Komponente befindet sich im *failed_late*-Zustand, wenn mindestens eine der beiden *CleaningCycle*-Threads durch eine CPU-Überlastung verzögert ausgeführt wird (Cutset1 und Cutset3) oder wenn eine der beiden Threads einen Soft-Error enthält (Cutset2 und Cutset4). Ein *WrongValue*-Ereignis tritt mit einer Wahrscheinlichkeit von 8.4E-13 pro 0.0005 Stunden durch den Spülzyklusprozess im BDS auf.

Der zweite Schritt im NSCPAV-Muster ist ein zusätzlicher Vergleich von zwei selbstüberwachenden Spülzyklen. Dies verschlechtert die Rechtzeitigkeit und Dienstgüte massiv. Abbildung 66 stellt die vier Schnittmengen dar, die zu einem *NoValue*-Fehler durch den *Compare*-Entscheidungsprozess im Spülzyklusprozess durch zwei *SelfChecking_CC*-Komponente führen. Mit einer Wahrscheinlichkeit von 2.9E-1 pro 0.0005 Stunden tritt eine Verzögerung im Spülzyklusprozess auf. Dies ist eine Steigerung der Auftrittswahrscheinlichkeit um das 3,5-Fache zu einem einzelnen Spülzyklusprozess.

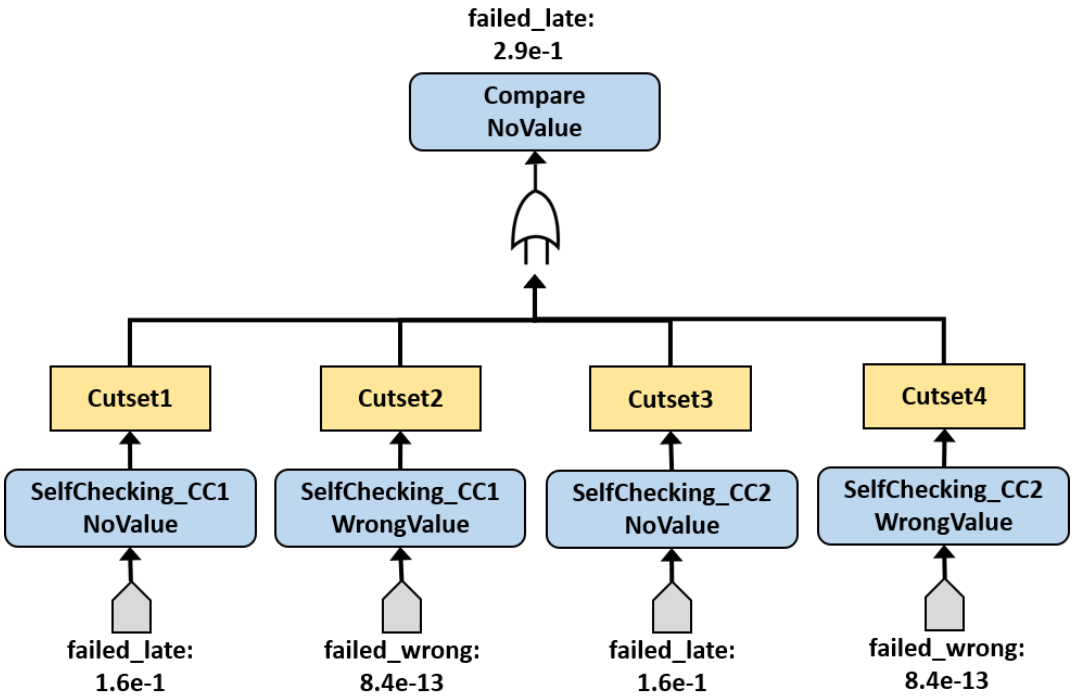


Abbildung 66 - Fehlerbaum für den failed_late-Zustand des Compare-Entscheidungsprozess

Das wiederum führt zum dritten Schritt des NSCPAV-Musters. Zusätzlich zu den zwei selbstüberwachenden Spülzyklen mit Vergleichs-Entscheider werden weitere Spülzyklusprozesse mit zusätzlichen Relativtests und einer 2/N Logik angewendet. Abbildung 67 zeigt den Aufbau der gesamten Spülzyklusanwendung, wie er im LVAD-BDS verwendet wird. Insgesamt arbeiten neben den zwei parallel arbeitenden Spülzyklen drei weitere selbstüberwachende Spülzyklusprozesse im Hot-Standby. Schlägt der Vergleichstest fehl, wird ein zusätzlicher Spülzyklusprozess aktiviert. Die Ausgaben aller aktiven Spülzyklen werden durch einen Relativtest (*Voter1-3*) verarbeitet.

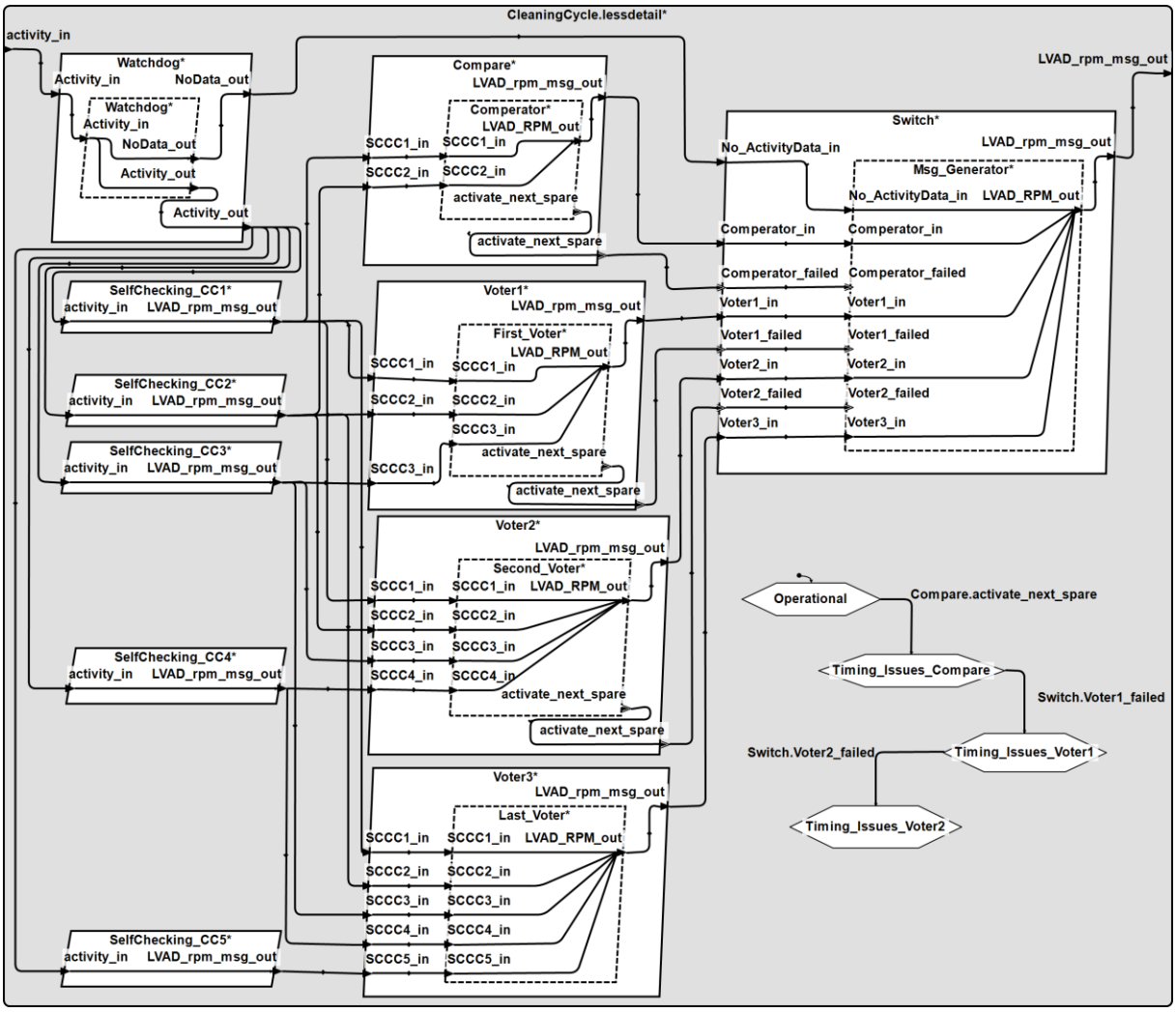


Abbildung 67 - Fehlertoleranter Spülzyklusprozess nach dem NSCPAV-Muster

Eine zusätzliche *Switch*-Komponente verarbeitet die gesamten Eingaben der einzelnen Entscheidungsprozesse und schaltet gemäß den Anforderungen genau den Eingang durch, der akzeptiert wurde. Erst wenn der letzte Entscheidungsprozess (*Voter3*) keine Übereinstimmung bzw. kein Ergebnis liefert, wechselt der gesamte Spülzyklusprozess in den *failed_late*-Zustand und schlägt zeitlich fehl. Abbildung 68 stellt den Fehlerbaum für den *failed_late* Zustand dar.

Mit einer Wahrscheinlichkeit von 3.3E-3 pro 0.0005 Stunden tritt ein interner *NoValue*-Fehler durch die Spülzykluskomponente auf. Dies entspricht einer abgeschätzten mittleren Fehlerrate von 6.6 fehlenden Nachrichten pro Stunde und ist gegenüber der nicht fehlertoleranten Version -mit 168 fehlende Nachrichten pro Stunde- eine deutliche Verbesserung der Dienstgüte. Die in Abbildung 68 dargestellten Mengen 1-5 bilden für den failed_late-Zustand die dominierenden Fehlerereignisse. Die Ereignisse bestehen aus einem Vierer-Tupel von *NoValue*-Fehlern. Die restlichen Mengen 6-25 setzen sich aus allen Permutationen der ersten fünf Mengen mit je einem *WrongValue*-Fehler zusammen.

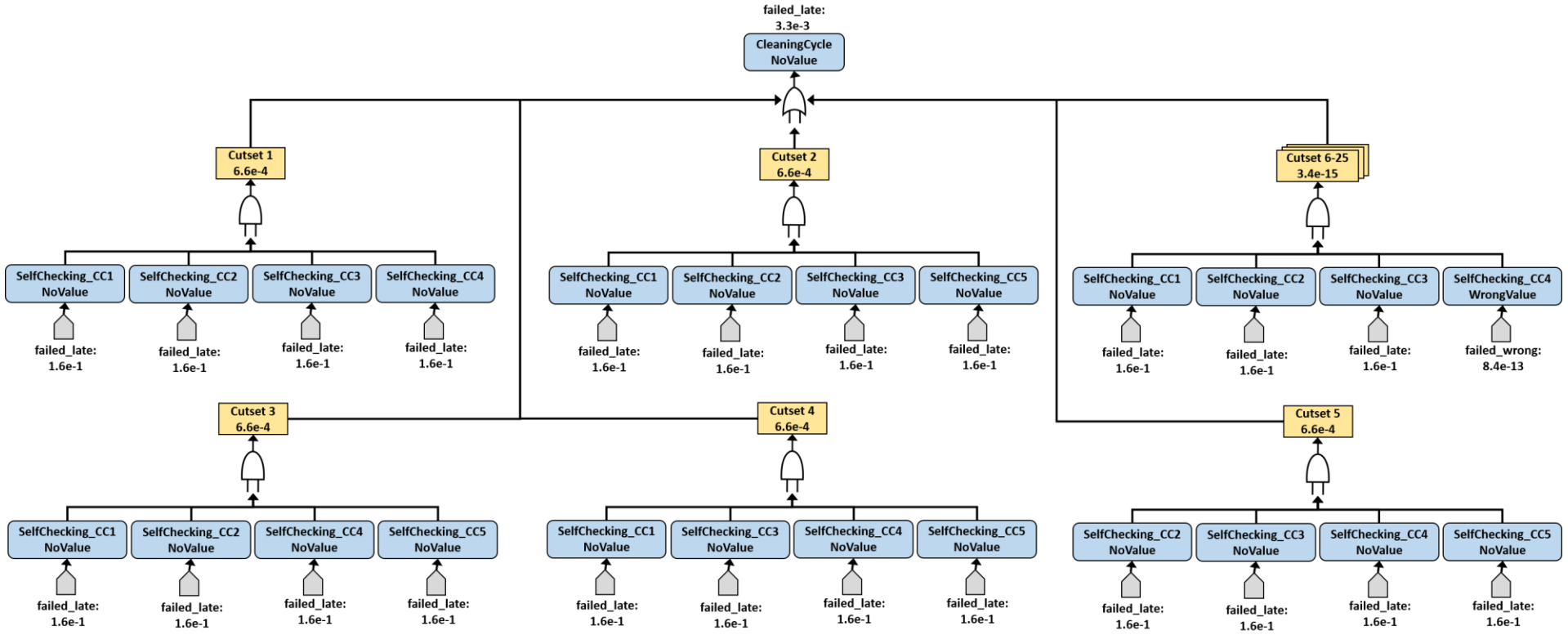


Abbildung 68 - Minimale Schnittmenge für den failed_late-Zustand der NSCPAV-Spülzykluskomponente

Trotz der harten Bedingungen der funktionalen Korrektheit und des Einflusses von Speicherfehlern auf die Rechtzeitigkeit verbessert das NSCPAV-Muster die Verfügbarkeit und damit auch die Dienstgüte der Softwarekomponente deutlich.

Nachdem die Aktivitätserkennung sowie der Spülzyklusprozess durch geeignete Fehlertoleranzmaßnahmen (RcB-AV- und NSCPAV-Muster) angepasst wurden, zeigt Abbildung 69 den Fehlerbaum für einen *NoValue*-Fehler an der LVAD-Steuereinheit. Am LVAD liegt mit einer Wahrscheinlichkeit von 1.6E-1 pro 0.0005 Stunden ein fehlendes Steuersignal an.

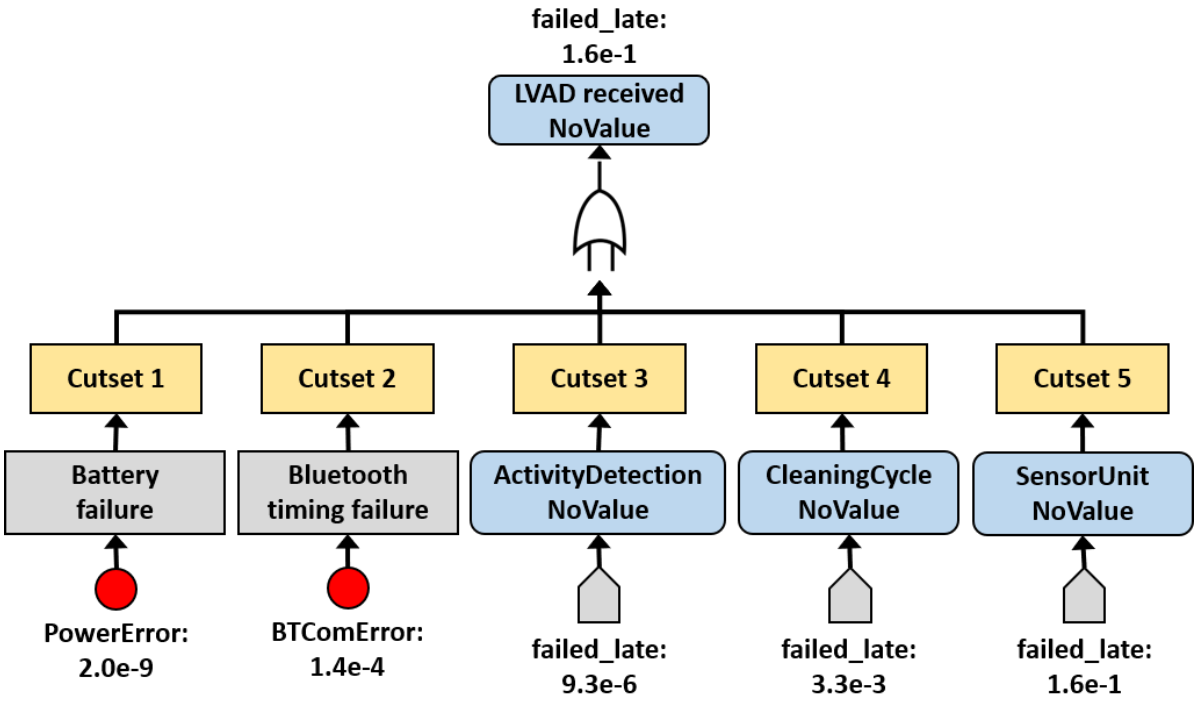


Abbildung 69 - Fehlerbaum für den failed_late-Zustand der LVAD-Steuereinheit durch ein Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen

Dies entspricht einer abgeschätzten mittleren Fehlerrate von bis zu 320 fehlenden Signalen pro Stunde. Gegenüber der Fehlerrate von 580 durch das Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen verbessern die angewendeten Erweiterungen und Anpassungen die Dienstgüte. Weiterhin bildet die Sensoreinheit (Cutset5) einen zentralen Ausfallpunkt im BDS. Ein Betriebswechsel in den Fallback-Modus des LVADs geschieht pro 0.0005 Stunden mit einer Wahrscheinlichkeit von 3.9E-5. Abbildung 70 stellt den Fehlerbaum für das wechseln in den Fallback-Modus dar. Dies bedeutet, dass pro Stunde im Mittel bis zu 7.8E-2 Betriebswechsel in den Fallback-Modus auftreten können.

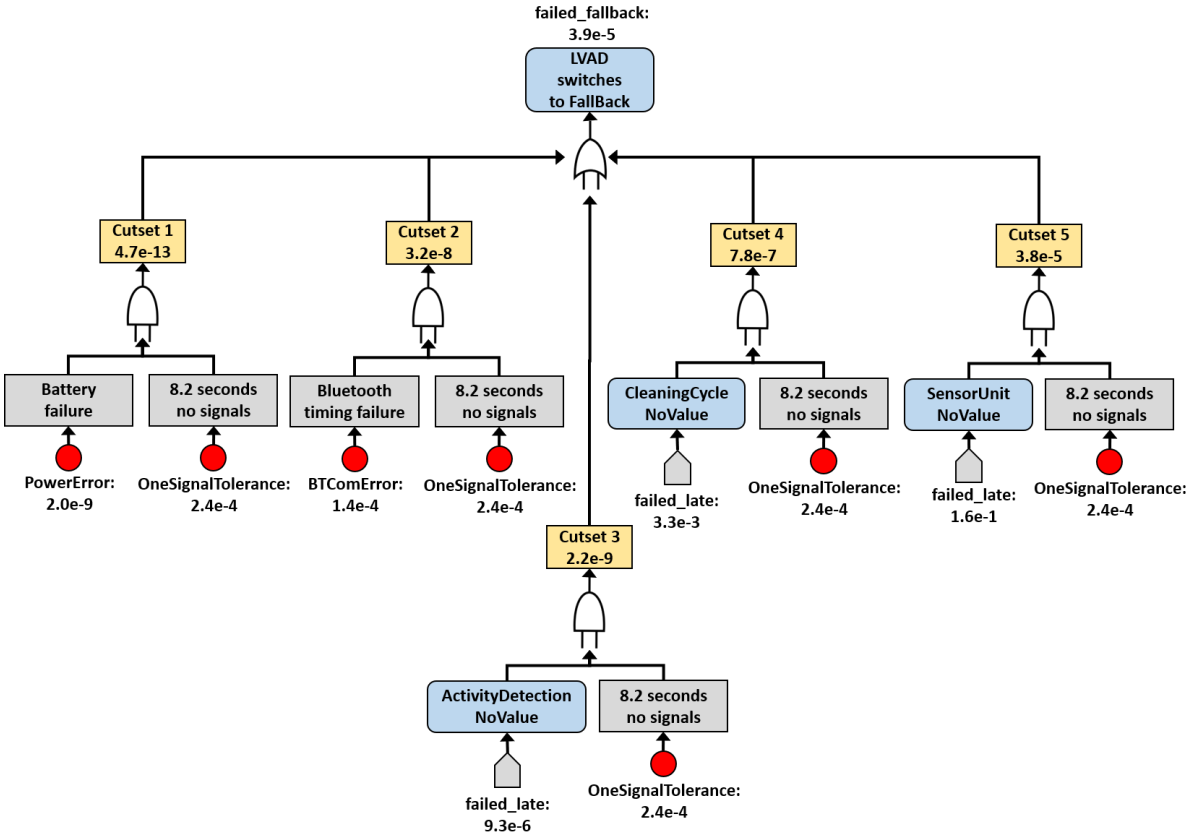


Abbildung 70 - Minimale Schnittmenge für den failed_fallback-Zustand der LVAD-Steuereinheit durch ein Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen

Die Maßnahmen, die ergriffen wurden (*RcB-AV-, NSCPAV-Muster sowie Fallback-Modus*) um das LVAD-BDS fehlertolerant zu gestalten, haben Wirkung gezeigt. Damit die Sensoreinheit, die noch immer einen zentralen Ausfallpunkt darstellt und die Dienstgüte des gesamten BDS beeinflusst, eine verbesserte Ausfallrate aufweist, müssen auf gerätebasierter Seite weitere Fehlertoleranzmechanismen im Umfeld des LVAD-BDS eingeführt werden.

6.3 Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranzmechanismen

Um Verzögerungen der Sensoreinheit durch Ausfälle oder Störungen zu mindern, müssen redundante Sensorinformationen im gerätebasierten System bereitgestellt werden. Damit zusätzliche *Accelerometer-* sowie *Gyroskopesensoren* im LVAD-BDS zur Verfügung stehen, werden weitere IoT-Geräte in der näheren Umgebung als Informationsquellen genutzt. Abbildung 71 stellt das erweiterte LVAD-BDS, bestehend aus zwei Smartphones und einer Smartwatch, dar. Das dargestellte Duplex-System agiert nach dem DRDV-Muster und besitzt eine Leitstation (*PrimarySmartphone*), welche zusätzliche Sensorinformationen von einer Smartwatch sowie eine Aktivität eines zweiten Smartphones (*SecondarySmartphone*), das als Folgestation agiert, erhält. Die einzelnen Fehlermodelle der Hardware- und Softwarekomponenten (*Battery, SensorUnit, ActivityDetection und BluetoothModule*) der

Standby-Geräte sind identisch mit denen des Single-Smartphones (*PrimarySmartphone*) und weisen die gleichen Auftrittswahrscheinlichkeiten für Fehler auf.

Das *PrimarySmartphone* überwacht die Kommunikation zwischen den Geräten über zwei Monitoring-Komponenten (*MonitoringSPCommunication*, *MonitoringSWCommunication*) und nutzt die gesamten Fehlerinformationen für einen fehlertoleranten Entscheidungsprozess (*ComparativeVoter*).

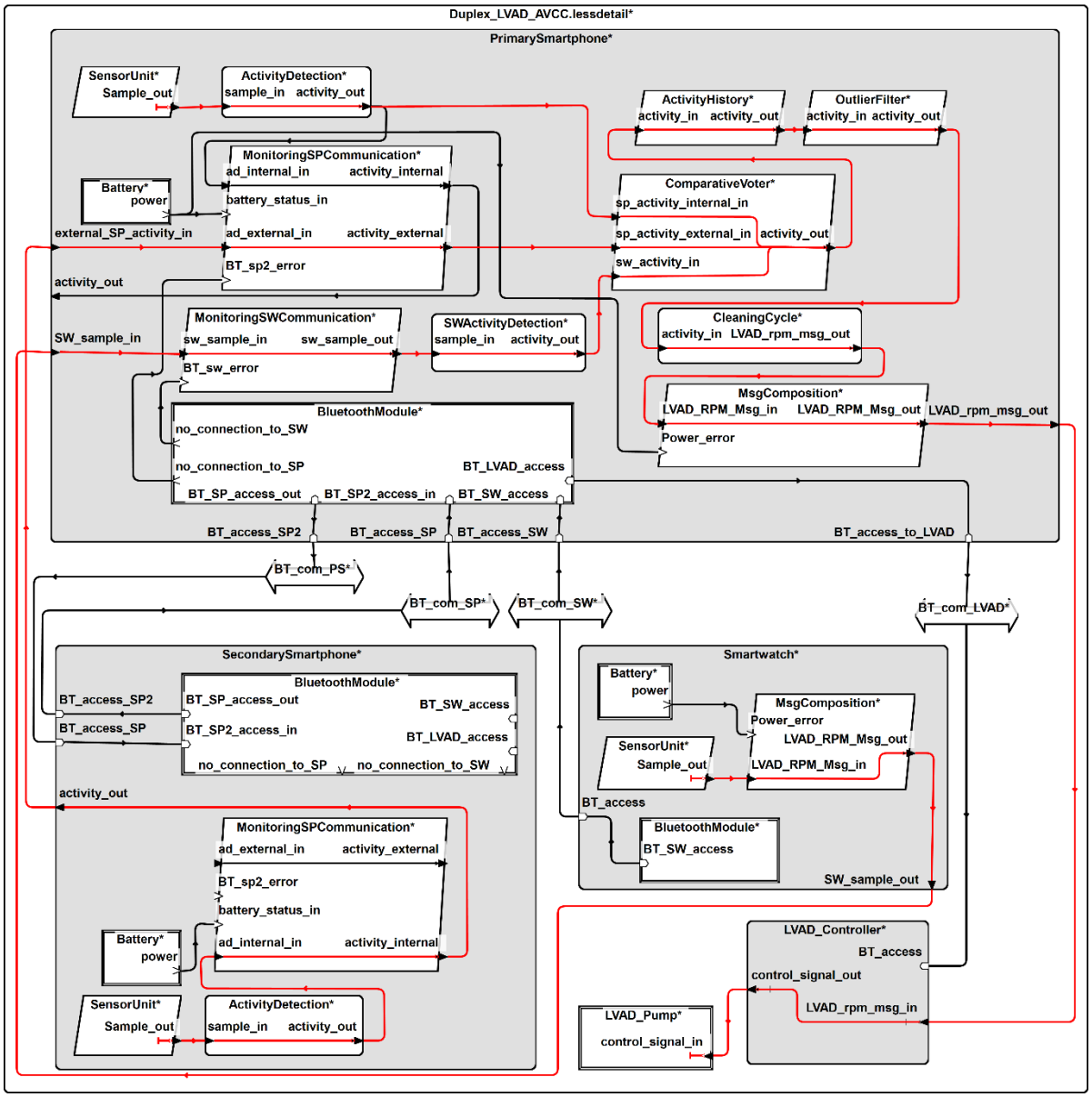


Abbildung 71 - Duplex-Smartphone-Smartwatch-LVAD-Spülzyklus-BDS mit Fehlertoleranzmechanismen

Die *MonitoringSPCommunication*-Komponente dient zur Überwachung der Kommunikation zwischen den beiden Smartphones. Das Fehlermodell dieser Komponente vereint die zeitlichen Fehler der internen Sensoren, der Batterie sowie der Aktivitätserkennung und leitet den potenziellen Fehler über die Schnittstelle (*activity_out*) zum parallel arbeiteten Smartphone weiter. Des Weiteren verarbeitet die Komponente die eingehenden zeitlichen Fehler des parallel agierenden Smartphones

(*external_SP_activity_in* und *BT_access_SP*). Abbildung 72 zeigt die beiden Fehlerbäume mit den minimalen Schnittmengen für das Anliegen eines *NoValue*- bzw. *WrongValue*-Fehlers am *ComparativeVoter* (*sp_activity_external_in*) durch das parallel agierende *SecondarySmartphone*.

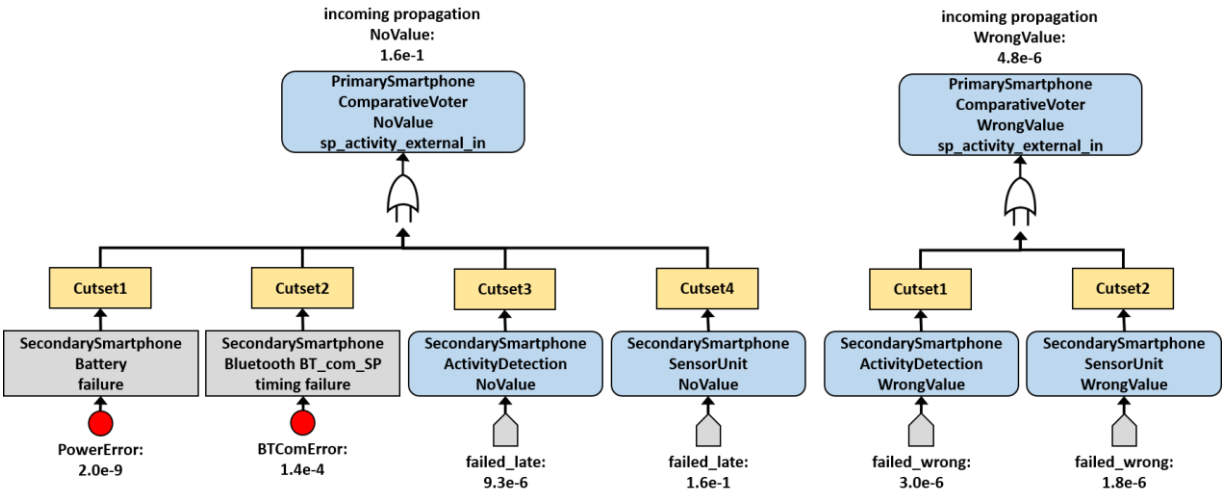


Abbildung 72 - Fehlerbäume für den NoValue- und WrongValue-Fehlertyp des zweiten Smartphones am ComparativeVoter

Ein *NoValue*-Fehlerereignis durch das Standby-Smartphone (*SecondarySmartphone*) liegt am *PrimarySmartphone* an, wenn:

- die Batterie ausfällt (Cutset1),
- eine Verzögerung durch die Bluetooth-Kommunikation (Cutset2) entsteht,
- ein *NoValue*-Ereignis durch die Aktivitätserkennung (Cutset3) entsteht,
- die Sensoreinheit zeitlich fehlschlägt (Cutset4).

Ein *WrongValue*-Fehlerereignis liegt am *PrimarySmartphone* an, wenn ein *SoftError*- oder *CalcError*-Fehler durch die Sensoreinheit oder die Aktivitätserkennung (Cutset1-2) im *SecondarySmartphone* auftreten. Mit einer Auftrittswahrscheinlichkeit von 1.6E-1 liegt ein zeitlicher *NoValue*-Fehler und mit 4.8E-6 ein funktionaler *WrongValue*-Fehler pro 0.0005 Stunden am Entscheidungsträger an.

Neben der Überwachung der bidirektionalen Smartphone-Kommunikation überwacht das *PrimarySmartphone* auch die unidirektionale Verbindung mit der Smartwatch über die *MonitoringSWCommunication*-Komponente. Da die Smartwatch als zusätzliche redundante Sensoreinheit dient, werden alle 0.0005 Stunden Sensorsamples zur Weiterverarbeitung an das *PrimarySmartphone* versendet. Die *MonitoringSWCommunication*-Komponente verarbeitet und vereint die Fehler, die intern in der Smartwatch auftreten und im System weitergeleitet werden sowie eine mögliche Kommunikationsverzögerungen bei der Datenübertragung.

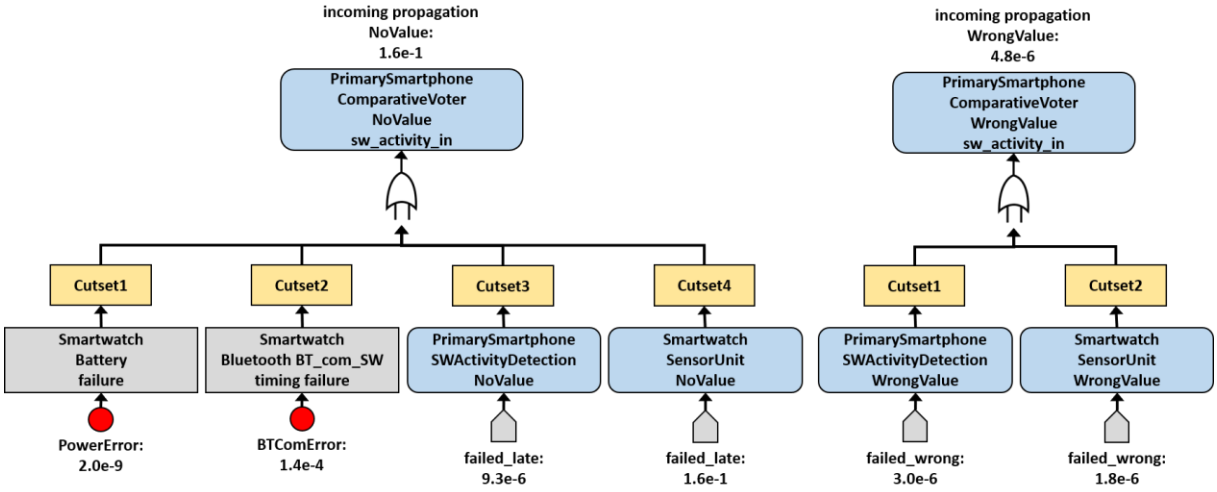


Abbildung 73 - Fehlerbäume für den NoValue- und WrongValue-Fehlertyp der Smartwatch am ComparativeVoter

Abbildung 73 zeigt die beiden Fehlerbäume mit den minimalen Schnittmengen für das Anliegen eines *NoValue*- bzw. *WrongValue*-Fehlers am *ComparativeVoter* (*sw_activity_in*) durch die *Smartwach*. Nachdem die Sensorsamples der Smartwatch anliegen, wird anhand der Daten eine Aktivitätserkennung, die ebenfalls nach dem RcV-AV-Muster aufgebaut ist, durchgeführt (Cutset3). Wie bei der Fehlerausbreitung des *SecondarySmartphones* liegt mit einer Auftrittswahrscheinlichkeit von 1.6E-1 ein zeitlicher *NoValue*-Fehler und mit 4.8E-6 ein funktionaler *WrongValue*-Fehler pro 0.0005 Stunden am Entscheidungsträger an.

Der *ComparativeVoter* bekommt als dritte Eingabe die internen *NoValue*- und *WrongValue*-Fehlerereignisse der internen Sensoren sowie der Aktivitätserkennung.

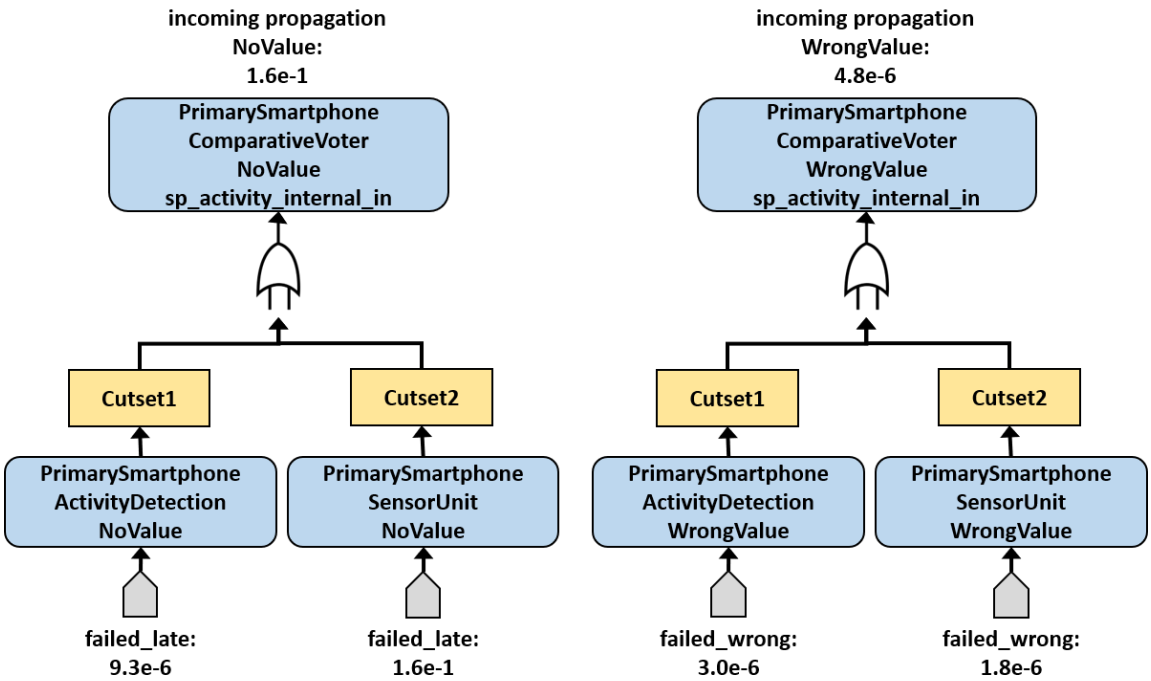


Abbildung 74 - Fehlerbäume für den NoValue- und WrongValue-Fehlertyp des internen Smartphones am ComparativeVoter

Abbildung 74 zeigt die beiden Fehlerbäume mit den minimalen Schnittmengen für das Auftreten eines *NoValue*- bzw. *WrongValue*-Fehlers am *ComparativeVoter* (*sp_activity_internal_in*) durch die internen Prozesse. Mit einer Auftrittswahrscheinlichkeit von 1.6E-1 liegt ein zeitlicher *NoValue*-Fehler und mit 4.8E-6 ein funktionaler *WrongValue*-Fehler pro 0.0005 Stunden am Entscheidungsträger an.

```
...
error propagations
	sw_activity_in: in propagation {NoValue, WrongValue};
	sp_activity_internal_in: in propagation {NoValue, WrongValue};
	sp_activity_external_in: in propagation {NoValue, WrongValue};

	activity_out: out propagation {NoValue, WrongValue};
flows
	ep_cv1: error path sp_activity_internal_in -> activity_out;
	ep_cv2: error path sp_activity_external_in -> activity_out;
	ep_cv3: error path sw_activity_in -> activity_out;

	es_cv1: error sink sp_activity_internal_in {NoValue};
	es_cv2: error sink sp_activity_internal_in {WrongValue};
	es_cv3: error sink sp_activity_external_in {NoValue};
	es_cv4: error sink sp_activity_external_in {WrongValue};
	es_cv5: error sink sw_activity_in {NoValue};
	es_cv6: error sink sw_activity_in {WrongValue};
end propagations;

component error behavior
Transitions

	t_op: operational -[
		1ormore(
		...
		(sp_activity_internal_in{NoError} and sp_activity_external_in{NoValue} and sw_activity_in{NoValue}),
		(sp_activity_internal_in{WrongValue} and sp_activity_external_in{NoValue} and sw_activity_in{NoValue}))
	]-> operational;

	t_nv: operational -[
		1ormore((sp_activity_internal_in{NoValue} and sp_activity_external_in{NoValue} and sw_activity_in{NoValue}),

		(sp_activity_internal_in{NoValue} and sp_activity_external_in{NoValue} and sw_activity_in{NoError}),
		(sp_activity_internal_in{NoValue} and sp_activity_external_in{NoError} and sw_activity_in{NoValue}),

		(sp_activity_internal_in{NoValue} and sp_activity_external_in{NoValue} and sw_activity_in{WrongValue}),
		(sp_activity_internal_in{NoValue} and sp_activity_external_in{WrongValue} and sw_activity_in{NoValue}),

		(sp_activity_internal_in{NoError} and sp_activity_external_in{NoValue} and sw_activity_in{WrongValue}),
		(sp_activity_internal_in{NoError} and sp_activity_external_in{WrongValue} and sw_activity_in{NoValue}),

		(sp_activity_internal_in{NoValue} and sp_activity_external_in{NoError} and sw_activity_in{WrongValue}),
		(sp_activity_internal_in{NoValue} and sp_activity_external_in{WrongValue} and sw_activity_in{NoError}),

		(sp_activity_internal_in{WrongValue} and sp_activity_external_in{NoError} and sw_activity_in{NoValue}),
		(sp_activity_internal_in{WrongValue} and sp_activity_external_in{NoValue} and sw_activity_in{NoError}))
	]-> failed_late;
	...
propagations
	prop_lv: failed_late -[]-> activity_out {NoValue};
	...
end component;
...
```

Abbildung 75 - Fehlermodell des ComparativeVoter in AADL

Der *ComparativeVoter* arbeitet nach einer angepassten Mehrheitsentscheids-Logik. Abbildung 75 stellt das Fehlermodell der AADL-Komponente dar. Liegen nicht mindestens zwei gleiche Eingaben am Voter an (*NoError* oder *WrongValue*), wird keine Aktivität (*NoValue*) weitergeleitet (Transition *t_nv*). Dies ermöglicht eine Fehlermaskierung (Fluss *es_cv1-6*) eines Fehlers der Sensor- oder Aktivitätserkennungs-Komponente. Um das System zusätzlich mit degradierter Funktionalität am Leben zu erhalten, leitet die Komponente im Falle eines zeitlichen Fehlers durch das *SecondarySmartphone* und der *Smartwatch* den funktionsunkritischen Aktivitätszustand „Belastung“ im System weiter (*NoError*). Das *PrimarySmartphone* kann in diesem Fall nicht nur auf der Grundlage der eigenen erkannten Aktivität die Korrektheit verifizieren. Statt dem Spülzyklusprozess kein Signal zu übermitteln, wird die Aktivität „Belastung“ an den Spülzyklus weitergeleitet. Dies führt dazu, dass erstens kein Spülzyklus aktiviert wird bzw. ein aktiver gestoppt wird und zweitens die LVAD-Steuereinheit seltener in den Fallback-Modus wechselt (Transition *t_op*). Der Fehlerbaum in Abbildung 77 zeigt die minimalen Schnittmengen des zeitlichen *NoValue*-Fehlers, die durch den *ComparativeVoter* weitergeleitet werden. Mit einer Auftrittswahrscheinlichkeit von 5.1E-2 pro 1.8 Sekunden wird ein *NoValue* im System weitergeleitet. Der dominierende zeitliche Fehler der Sensoreinheit (1.6E-1) wird mittels Redundanz und angepasstem Entscheidungsprozess (Cutset1-2) um den Faktor 3 verbessert.

Trotz der verbesserten Auftrittswahrscheinlichkeit gegenüber dem Single-Smartphone-LVAD-System mit Fehlertoleranz sind ca. 5% Sensorausfälle aus Sicht der Rechtzeitigkeit und Dienstgüte noch nicht akzeptabel. Um transiente *NoValue*-Fehler der Sensoreinheit sowie der Aktivitätserkennung im System zu kompensieren, wird zusätzlich, nach dem *ComparativeVoter*, das LBH-Fehlertoleranzmuster angewendet. Abbildung 76 zeigt das *ActivityHistory*-Modell, das kurzfristige Nachrichtenausfälle toleriert und in der Vergangenheit anliegende Aktivitäten an den Spülzyklusprozess weiterleitet.

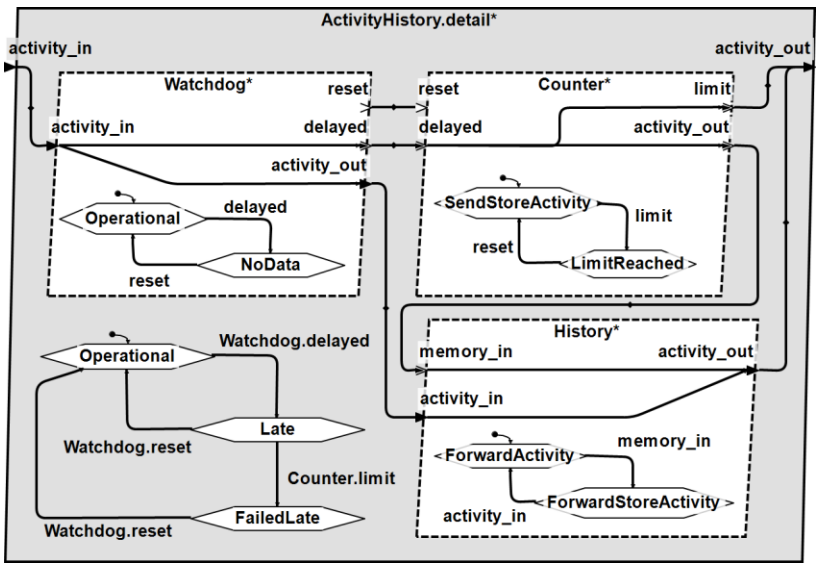


Abbildung 76 - ActivityHistory-Prozess nach LBH-Muster

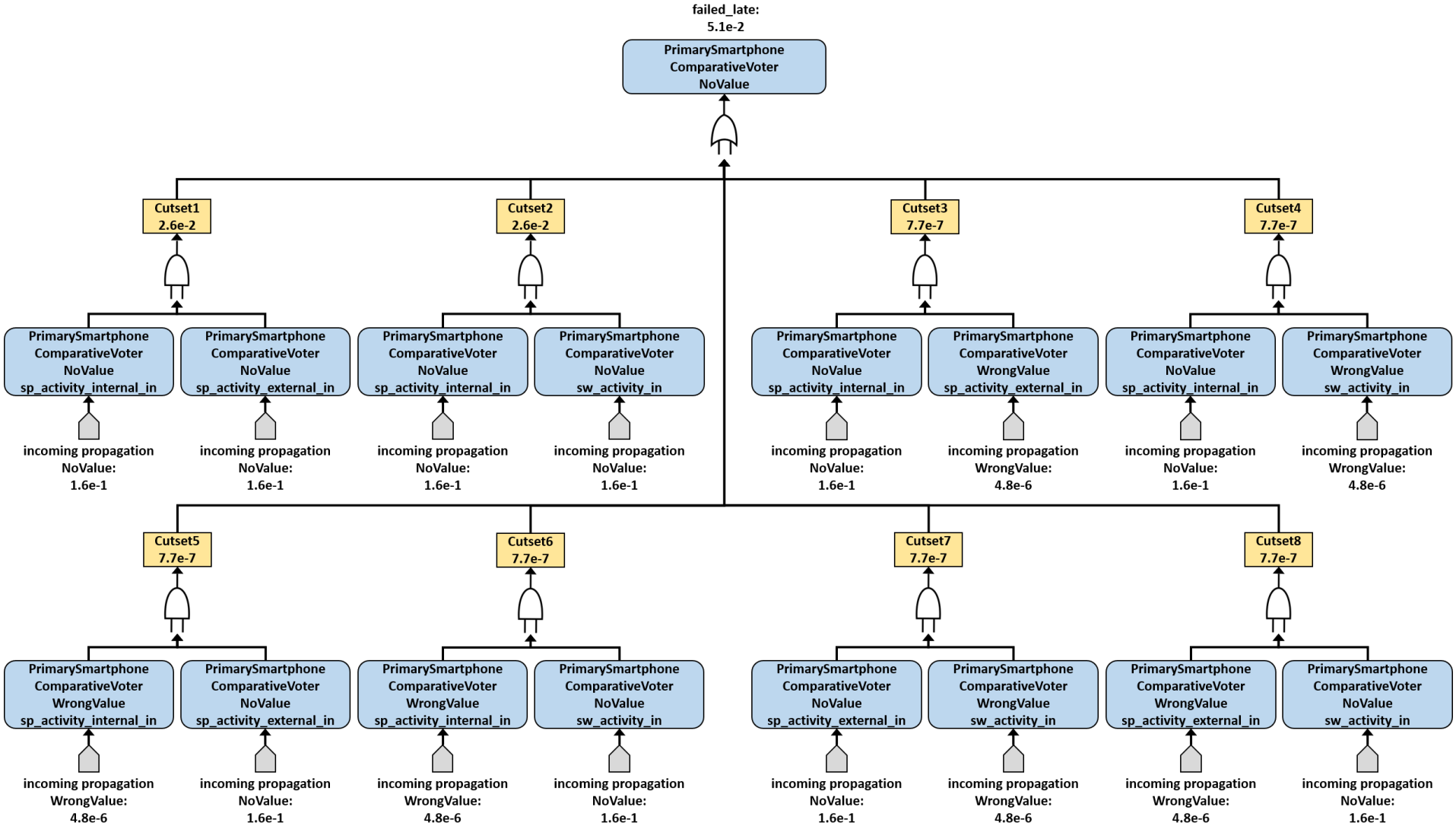


Abbildung 77 - Fehlerbaum für den failed_late-Zustand der ComparativeVoter-Komponente

Der Prozess besteht aus drei Software-Threads. Der *Watchdog*-Thread dient der Überwachung des Datentransfers. Kommt eine Aktivität an der Komponente rechtzeitig an (*WrongValue* oder *NoError*), wird diese in dem *History*-Thread abgespeichert und im System weitergeleitet. Liegt ein *NoValue*-Ereignis am Watchdog an, wird ein *Counter*-Thread (Zähler) aktiviert.

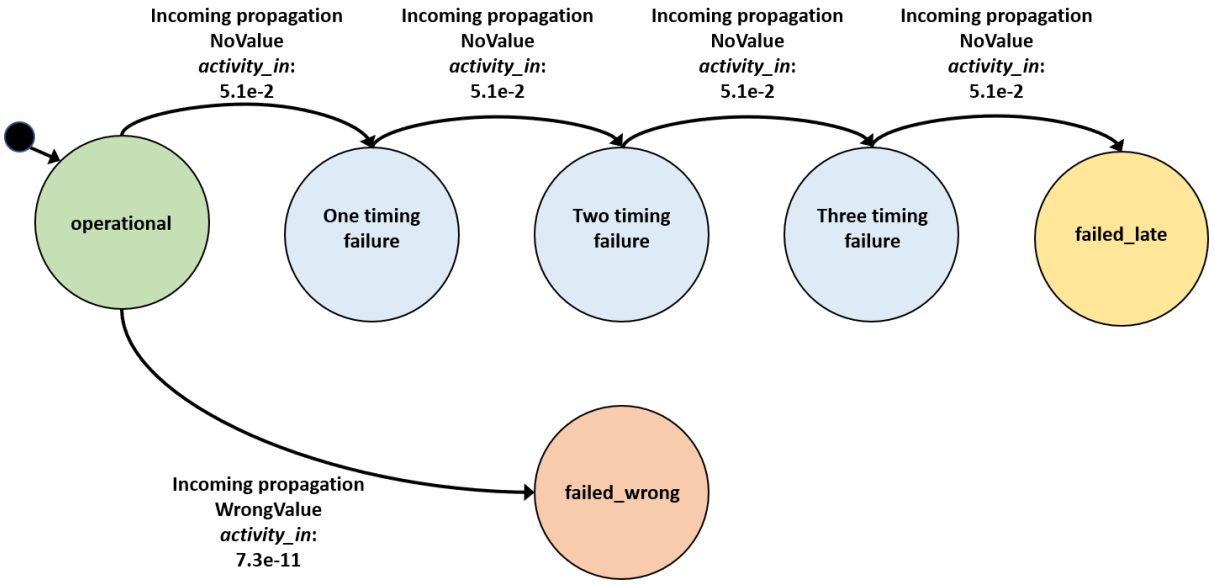


Abbildung 78 - Fehlerverhaltensmodell für die ActivityHistory-Komponente

Nachdem der *Zähler* aktiviert wurde, überwacht dieser dann die Häufigkeit der nacheinander auftretenden zeitlichen *NoValue*-Fehler. Das System kann eine Sequenz von drei fehlenden Aktivitäten tolerieren und leitet die in der *History* abgespeicherte Aktivität weiter. Erst wenn vier Nachrichten in Folge fehlen, schlägt die Komponente fehl und leitet einen *NoValue*-Fehler weiter. Abbildung 78 zeigt das Fehlerverhaltensmodell der *ActivityHistory*.

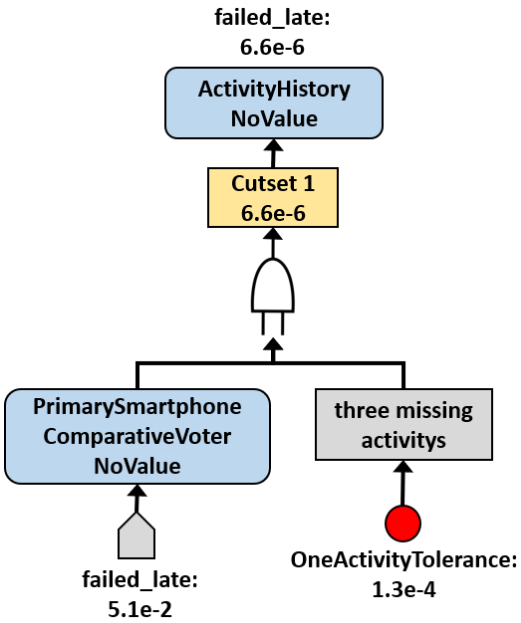


Abbildung 79 - Fehlerbaum für den failed_late-Zustand der ActivityHistory-Komponente

Unter diesem Aspekt, dass vier *NoValue*-Fehler in Folge an der *ActivityHistory* anliegen, ergibt sich eine Auftrittswahrscheinlichkeit für einen *NoValue*-Fehler durch den *ActivityHistory*-Prozess von 6.6E-6 pro 0.0005 Stunden. Abbildung 79 zeigt den Fehlerbaum der *AktivityHistory*. Die Schnittmenge (Cutset1) vereint ein *NoValue*-Fehlerereignis durch den *ComparativeVoter* mit dem Fehlerereignis *OneActivityTolerance* (drei unabhängige *NoValue*-Fehler in Folge). Die Sensoreinheit, die den schwächsten zentralen Ausfallpunkt darstellt, hat eine abgeschätzte mittlere Fehlerrate von 1.32E-2 *NoValue*-Fehlern pro Stunde. Im Vergleich zum nicht fehlertoleranten System, in dem die Sensoreinheit eine mittlere Fehlerrate von 320 *NoValue*-Fehlern pro Stunde aufweist, liefert das Duplex-System eine deutliche Verbesserung der Verfügbarkeit und Dienstgüte im gesamten LVAD-BDS.

Für das gesamte LVAD-Spülzyklus-BDS aus Abbildung 71 mit den vorgestellten und analysierten Fehlertoleranzmustern ergibt sich der in Abbildung 80 dargestellte Fehlerbaum für ein fehlendes Stellsignal an der LVAD-Steuereinheit. Mit einer Wahrscheinlichkeit von 3.4E-3 pro 0.0005 Stunden tritt ein *NoValue*-Fehler am LVAD auf.

Dies entspricht einer abgeschätzten mittleren Fehlerrate von 6.8 fehlenden Stellsignalen pro Stunde. Gegenüber der mittleren Fehlerrate von 320 fehlenden Stellsignalen durch das Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen verbessern die angewandten Fehlertoleranzmechanismen die Verfügbarkeit und Dienstgüte des BDS deutlich.

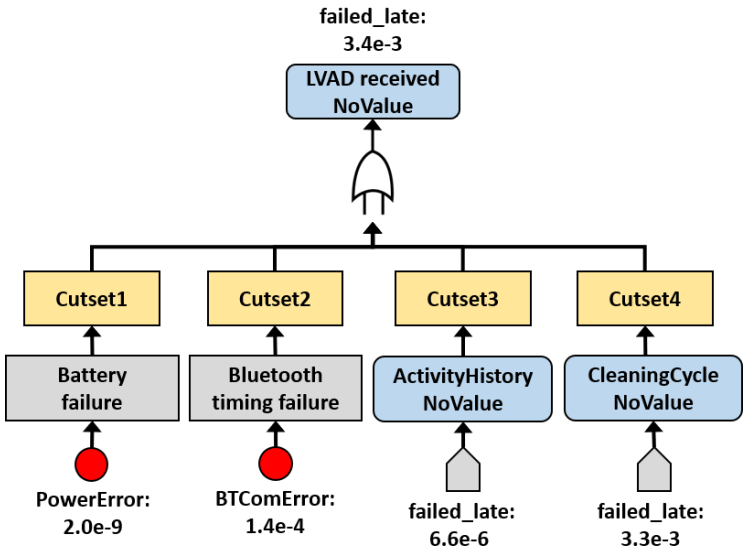


Abbildung 80 - Fehlerbaum für den failed_late-Zustand der LVAD-Steuereinheit durch ein Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranzmechanismen

Ein Betriebswechsel in den Fallback-Modus des LVADs geschieht pro 0.0005 Stunden mit einer Wahrscheinlichkeit von 2.0E-14. Abbildung 81 stellt den minimalen Fehlerbaum für den Wechsel in den Fallback-Modus dar. Dies entspricht einer

abgeschätzten mittleren Fehlerrate von 4.0E-11 Betriebswechseln pro Stunde in den Fallback-Modus.

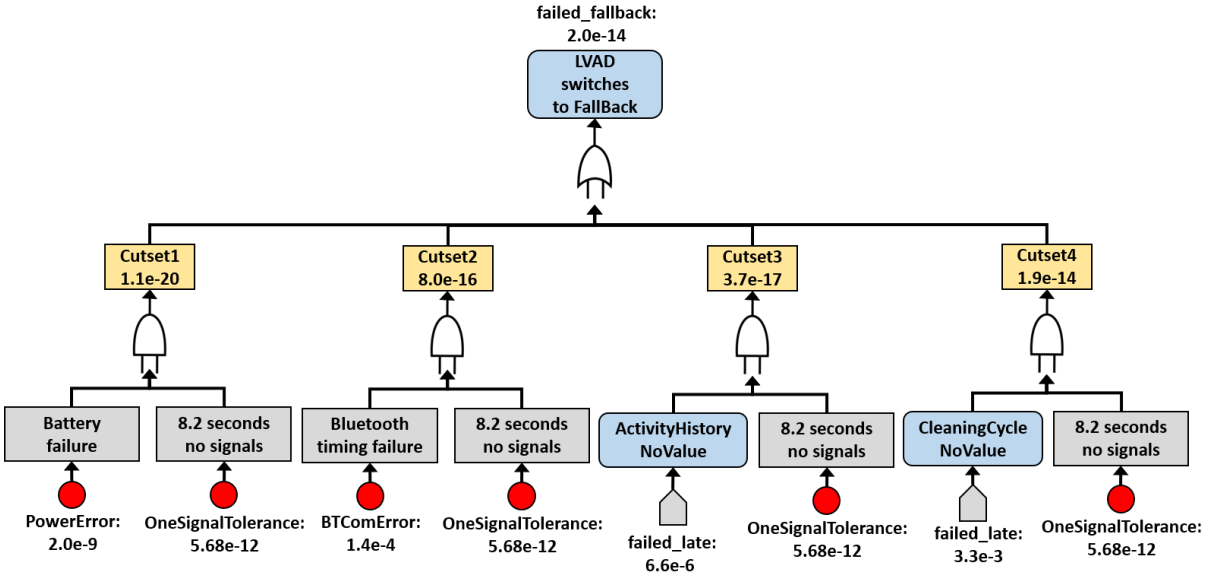


Abbildung 81 - Fehlerbaum für den failed_fallback-Zustand der LVAD-Steuereinheit durch ein Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranzmechanismen

Eine Übersicht der einzelnen Entwicklungsstufen des LVAD-BDS und der gewonnenen quantitativen Fehlerbaumanalysen ist in Tabelle 16 aufgelistet. Die Ergebnisse zeigen, dass die vorgestellten Fehlertoleranzmuster (*DRDV-, RcB-AV-, LBH-, NSCPAV-Muster sowie Fallback-Modus*) in Kombination geeignete Mechanismen darstellen, um ein BDS zu entwickeln, und dass die Zuverlässigkeitsanforderungen bezüglich der Rechtzeitigkeit und Dienstgüte gewährleistet werden können.

Tabelle 16 - Vergleich der einzelnen Evolutionsstufen des LVAD-BDS bezüglich der Rechtzeitigkeit und Dienstgüte

Evolutionsstufe	**LVAD empfängt ein *NoValue*-Fehler**		**LVAD wechselt in den Betriebsmodus *Fallback***	
	Auftritts-wahrscheinlichkeit pro 0.0005 Stunden	**Fehlerrate pro Stunde**	**Auftritts-wahrscheinlichkeit pro 0.0005 Stunden**	**Fehlerrate pro Stunde**
Single-Smartphone-LVAD-System ohne Fehlertoleranz	2.9E-1	580	/	/

Single-Smartphone-LVAD-System mit Fehlertoleranz	1.6E-1	320	3.9E-5	0.078
Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranz	3.4E-3	6.8	2.0E-14	4.0E-11

Im nicht fehlertoleranten BDS (Single-Smartphone-LVAD-System ohne Fehlertoleranz) entstehen im Mittel 580 fehlende Nachrichten in der Stunde. Dies entspricht einer mittleren Betriebsdauer des BDS zwischen zwei *NoValue*-Fehlern (MTBF) von ca. 6.2 Sekunden. Das verbesserte fehlertolerante Duplex-System (Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranz) weist im Mittel 6.8 fehlende Nachrichten pro Stunde auf. Dies entspricht einer MTBF zwischen zwei *NoValue*-Fehlern von ca. 8.8 Minuten. Zusätzlich zur Rechtzeitigkeit hat sich auch die Dienstgüte des BDS deutlich verbessert. Das fehlertolerante BDS (Single-Smartphone-LVAD-System mit Fehlertoleranz) wechselt im Mittel 0.078 mal in den Fallback-Modus. Dies weist eine MTBF zwischen zwei Fallback-Modus Wechsel von ca. 12.8 Stunden auf. Das erweiterte fehlertolerante Duplex-BDS (Duplex-Smartphone-Smartwatch-LVAD-System mit Fehlertoleranz) Wechselt im Mittel nur 4.0E-11 mal in den Fallback-Modus des Perfektionskerns. Dies entspricht einer MTBF zwischen zwei Fallback-Modus Wechsel von ca. 25000000000 Stunden.

Die modellbasierte Entwicklung ist ein gutes und effizientes Mittel um Systemklassen wie BDS zuverlässig zu entwerfen und zu entwickeln. Des Weiteren ermöglicht der Ansatz frühzeitig im Entwicklungsprozess Informationen und Kenntnisse über Zuverlässigkeits-Eigenschaften zu erhalten. Im Falle der LVAD-BDS Entwicklung lieferte die AADL-Modellierung und toolgestützte Analyse durch OSATE frühzeitig Erkenntnisse über die zu entwickelnden Fehlertoleranzmuster. Fragestellungen über strukturelle Anpassungen der Modelle wie zum Beispiel die Anzahl an Standby-Komponenten einer Softwarekomponente, die Reihenfolge von Fehlertoleranzmustern oder das Einhalten von Rechtzeitigkeits- und Dienstgüteanforderungen, die an das LVAD-BDS gestellt wurden, konnten frühzeitig im Entwicklungsprozess beantwortet werden.

7 EXPERIMENTELLE BEWERTUNG

Neben der modellbasierten Entwicklung und Analyse mittels AADL wurden ein LVAD-Demonstrator sowie eine prototypische Spülzyklus-Anwendung für einen Proof-of-Concept entwickelt. Dieser Aufbau ermöglicht eine experimentelle Bewertung der Rechtzeitigkeit und Dienstgüte des LVAD-BDS.

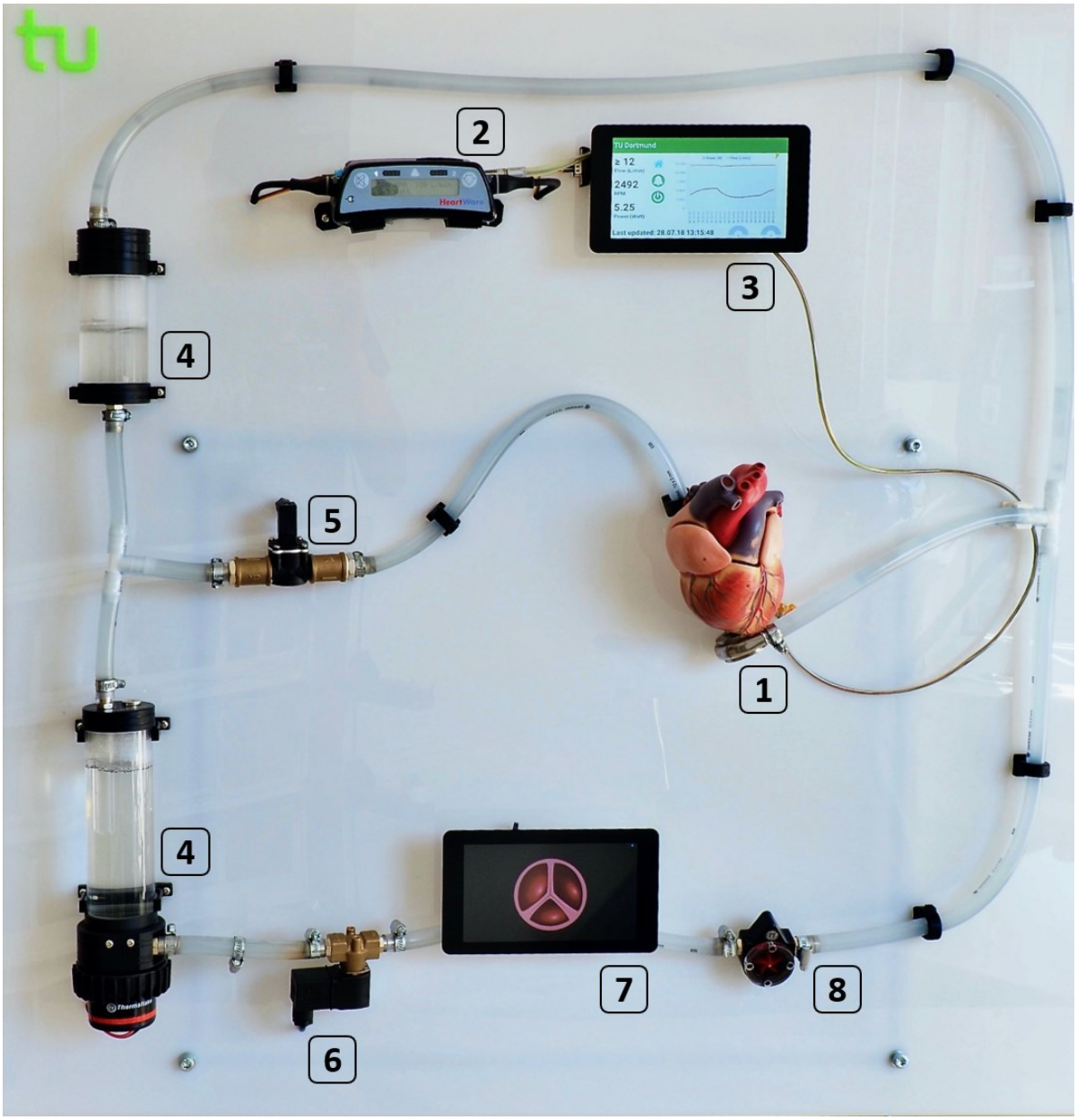


Abbildung 82 - LVAD-Mockup zur Erprobung der Spülzyklusanwendung

Abbildung 82 zeigt das entwickelte LVAD-Mockup, das einen einfachen funktionalen Körperkreislauf simuliert. Das Mockup besteht aus zwei Kreisläufen, einem Kreislauf über die LVAD-Pumpe (**1**) und einem Zweiten über die Aortenklappe (**6**).

Der gesamte LVAD-Demonstrator umfasst die folgenden Hardware-Komponenten:

1. LVAD-Pumpe,
2. LVAD-Steuereinheit,
3. LVAD-RaspberryPi mit Monitor,
4. Wasserreservoir,
5. Flusssensor,
6. Magnetventil,
7. Aortenklappen-RaspberryPi mit Monitor,
8. Flussanzeige.

Die LVAD-Pumpe (**1**) ist über eine Driveline mit der LVAD-Steuereinheit (**2**) verbunden. Durch Reverse-Engineering wurde die Struktur der Nachrichtenpakete des LVADs für das Monitoring sowie die Geschwindigkeitsanpassung extrahiert. Dies ermöglicht die Fernüberwachung sowie -steuerung des LVADs durch passende IoT-Geräte. Hierzu wurde ein RaspberryPi (**3**) verwendet, der eine serielle Schnittstelle zur LVAD-Steuereinheit sowie eine verbindungslose Bluetooth-Schnittstelle zu den IoT-Geräten zur Verfügung stellt. Zur visuellen Darstellung der aktuellen LVAD-Parameter wurde der RaspberryPi um einen Monitor (**3**) erweitert. Damit der Fluss des Mockups durch die variablen Pumpengeschwindigkeiten nicht leerläuft bzw. es zu keinen Ansaugeffekten in der LVAD-Pumpe kommt, simulieren zwei Wasserreservoirs (**4**) die linke Herzkammer. Da die Druckverhältnisse sowie die Viskosität der Flüssigkeit kein physiologisch korrektes Modell des Menschen abbilden, besitzt das untere Wasserreservoir zusätzlich eine kleine Pumpe, um die Druckverhältnisse im Aorten-Strang aufrecht zu erhalten. Diese Pumpe simuliert die noch zur Verfügung stehende restliche Muskelkraft des geschwächten Herzens. Ein Flusssensor (**5**) misst den Durchfluss vor der LVAD-Pumpe und dient dem Nachweis, dass bei geöffneter Aortenklappe der Durchfluss an der Pumpe reduziert ist. Um das Verhalten einer Aortenklappe im Aufbau zu simulieren, wurde ein Magnetventil (**6**) verwendet. Die Steuerung des Ventils findet über ein Relais statt, welches wiederum durch den LVAD-RaspberryPi gesteuert wird. Liegt eine reduzierte Pumpengeschwindigkeit (Spülereignis) am LVAD an, wird das Ventil geöffnet. Damit der Zustand der Aortenklappe (geöffnet oder geschlossen) visuell am Mockup sichtbar ist, wurde ein zweiter Aorten-RaspberryPi (**7**) mit zugehörigem Monitor verbaut. Dieser erhält durch den LVAD-RaspberryPi die zusätzlichen Informationen des Ventilzustandes und versorgt das Relais mit der benötigten Eingangsspannung. Eine zusätzliche Flussanzeige (**8**) zeigt den vorhandenen Fluss im Aorten-Strang (geöffnete oder geschlossene Klappe).

Zusätzlich zum Mockup-Aufbau, der den Blutkreislauf über die LVAD-Pumpe bzw. die Aortenklappe abbildet, wurde eine prototypische Android-Anwendung für den

Spülzyklusanwendungsfall entwickelt. Abbildung 83 zeigt die Spülzyklus-Applikation sowie den Roboterarm zur Bewegungssimulation.

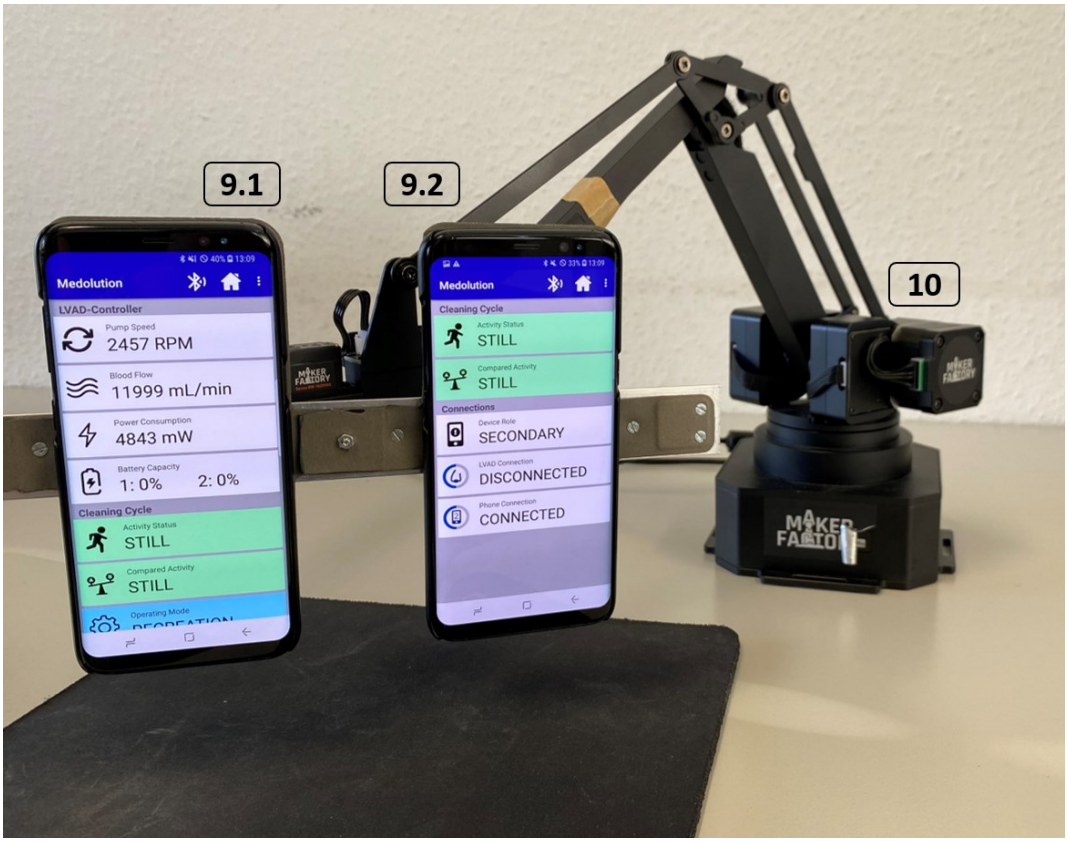


Abbildung 83 - Spülzyklus-APP mit Roboterarm zur Bewegungssimulation

Das Duplex-Smartphone-System besteht aus einem *PrimarySmartphone* (**9.1**) und einem *SecondarySmartphone* (**9.2**). Um Ruhe- bzw. Bewegungsphasen über einen längeren Zeitraum zu simulieren, wurden die beiden Smartphones an den Roboterarm CCR-45 (**10**) des Herstellers MakerFactory befestigt. Über eine Programmierschnittstelle können Bewegungsprofile entworfen werden, die eine experimentelle Bewertung der Aktivitätserkennung sowie der Dienstgüte des LVAD-BDS ermöglichen.

Im Folgenden wird das entwickelte LVAD-BDS durch Funktionstests, Fehlerinjektionen und durch geeignete Belastungstests untersucht und bewertet.

7.1 Funktionstests

Durch die Funktionstests wird die Spülzyklus-Anwendung, insbesondere der Aktivitätserkennungsprozess und der funktionale Spülzyklusablauf, untersucht. Abbildung 85 stellt die graphische Oberfläche der entwickelten Anwendung dar. Auf der linken Seite ist die Anzeige des *PrimarySmartphones* dargestellt. Dieses stellt die Leitstation des Duplex-LVAD-BDS dar und hat eine direkte Verbindung zur LVAD-Steuereinheit. Die Geräterolle (**1**) sowie die aktiven Verbindungen zum LVAD (**2**) als auch zum Backup-Smartphone (**3**) werden am unteren Ende angezeigt. Das

PrimarySmartphone hat neben der aktiven Verbindung zum LVAD noch eine Verbindung zum *SecondarySmartphone*. Um eine Fernüberwachung sowie eine verbesserte Statusanzeige zu ermöglichen, sendet die LVAD-Steuereinheit kontinuierlich Statusinformationen der LVAD-Pumpe mit dem aus Abbildung 84 dargestellten Nachrichtenformat zum verbundenen Smartphone.

TimeStamp	MotorSpeed RPM	MotorPower mW	Flow ml/min	Batter1 %	Batter2 %	Digitale Signatur

Abbildung 84 - Nachrichtenformat vom LVAD zum Smartphone

Die Monitoring-Informationen (**4**) des LVADs werden in der App am oberen Bildschirmrand graphisch dargestellt und stehen im gesamten BDS zur Weiterverarbeitung zur Verfügung. Neben den Verbindungsinformationen und den LVAD-Statusinformationen werden zusätzlich die berechneten Aktivitäten (**5**), der Entscheidungsprozess des *ComparativeVoters* (**6**) sowie das aktuelle Spülereignis (**7**) angezeigt.

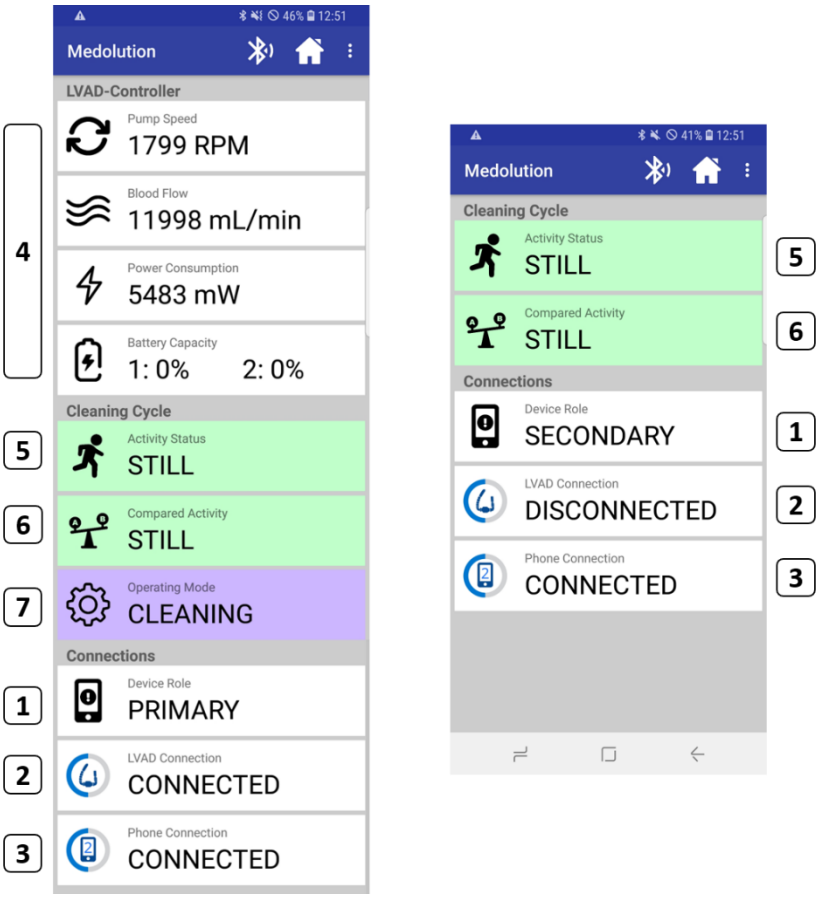


Abbildung 85 - UI der Spülzyklus-APP - links PrimarySmartphone rechts SecondarySmartphone

Um die Funktionalität des Monitorings sowie die Wechsel der Geräterollen zu testen, wurde die Bluetooth-Verbindung der einzelnen Smartphones kontinuierlich deaktiviert und aktiviert. Die Tests wiesen keine funktionalen Probleme auf. Wurde im Duplex-Betrieb das *PrimarySmartphone* durch Deaktivierung der Bluetooth-

Verbindung aus dem Systemkontext entfernt, so übernahm das *SecondarySmartphone* in der neuen Betriebsrolle als „*SingleSmartphone*“ die Kontrolle des LVAD-BDS. Zusätzlich wurden die LVAD-Informationen bezüglich der Ansteuerung (2500RPM oder 1800RPM) korrekt dargestellt. Wurde die Bluetooth-Verbindung des *PrimarySmartphones* wieder aktiviert, wechselte das System aus dem Single-Smartphone-Betrieb zurück in den Duplex-Betrieb. Die maximal gemessene Zeitspanne, bis die zwei Smartphones ihre Rollen im Duplex-Modus einnahmen, betrug ca. 9 Sekunden.

Ein weiterer Funktionstest wurde für die Aktivitätserkennung durch das Neuronale-Netz durchgeführt. Für den Versuchsaufbau wurde ein Smartphone, welches als *SingleSmartphone* agiert, an einen Roboterarm zur Bewegungsanalyse (vergleiche Abbildung 83) befestigt. Ein Bewegungsprofil über 1 Stunde wurde für die Bewegungsanalyse entworfen. Dieses Profil simulierte periodisch in Abständen von 5 Minuten die Aktivität zwischen Belastung (Bewegung) und Ruhe (Stillstand). In einem Zyklus von 1.8 Sekunden wurde eine Aktivität durch das Neuronale-Netz berechnet. Dies führte zu einem Gesamtumfang von 2000 gemessenen Aktivitäten, die in dem Funktionstest untersucht wurden. Die Ergebnisse des Funktionstests der Aktivitätserkennung sind in Tabelle 17 dargestellt.

Tabelle 17 - Ergebnis des Funktionstests der Aktivitätserkennung

Ergebnis des Funktionstests der Aktivitätserkennung durch das Neuronale-Netz	
Korrekte Ruhephase:	998
Inkorrekte Ruhephase:	2
korrekte Belastungsphase:	1000
Inkorrekte Belastungsphase:	0

Von den 2000 gemessenen Aktivitäten wurden nur zwei inkorrekt berechnet. In zwei unterschiedlichen Ruhephasen wurde je eine Aktivität fälschlicherweise als Belastung erkannt. Von den in der Belastungsphase erkannten Aktivitäten wurden alle korrekt berechnet.

Um die Funktionalität der gesamten Prozesskette und deren Latenz zu bewerten, werden alle Prozessinformationen, die im Smartphone auftreten, in einer Logdatei abgespeichert. Abbildung 86 zeigt eine Logdatei, in der ein Ablauf der erweiterten Spülzyklusfunktionalität eines Smartphones dargestellt ist. Die grün markierten Loginformationen (siehe Zeile *1*, *7*, *9*, *10*, *11* und *19*) stellen die Ausgaben der einzelnen

Softwarekomponenten (*SensorUnit, ActivityDetection, ComparativeVoter, ActivityHistory, OutlierFilter* und *CleaningCycle*) für einen BDS-gestützten LVAD-Spülzyklus dar.

Msg. No.	Timestamp	Error INFO	Device_Component	Information	Input	Result
1	2020-12-03 16:17:56.751	INFO	SensorDataCheckpoint	Receive Sensor Data Sample		DataSample
2	2020-12-03 16:17:56.777	INFO	First_ADUnit	Activity Watchdog Timer start: 16:17:56.776		
3	2020-12-03 16:17:56.782	INFO	AD1	ActivityDetection1 Calculation Result by neural network	DataSample	ACTIVE
4	2020-12-03 16:17:56.786	INFO	AD2	ActivityDetection2 Calculation Result by neural network	DataSample	ACTIVE
5	2020-12-03 16:17:56.792	INFO	AD3	ActivityDetection3 Calculation Result by neural network	DataSample	ACTIVE
6	2020-12-03 16:17:56.798	INFO	MajorityVoter	Compare TMR of ActivityDetection1, ActivityDetection2 and ActivityDetection3	[ACTIVE, ACTIVE, ACTIVE]	ACTIVE
7	2020-12-03 16:17:56.802	INFO	ActivityDetection Monitoring	Activity Detection Result	ACTIVE	ACTIVE
8	2020-12-03 16:17:57.508	INFO	Monitoring SPCommunication	Received Remote Device Activity at 16:17:57.507	ACTIVE	ACTIVE
9	2020-12-03 16:17:57.531	INFO	ComparativeVoter	Compare Activity Detection Results	[ACTIVE, ACTIVE]	ACTIVE
10	2020-12-03 16:17:57.537	INFO	ActivityHistory	Save to History	ACTIVE	ACTIVE
11	2020-12-03 16:17:57.548	INFO	OutlierFilter	Computation of [ACTIVE, ACTIVE, ACTIVE]	ACTIVE	ACTIVE
12	2020-12-03 16:17:57.557	INFO	SelfChecking_CC	Cleaning Watchdog Timer start: 16:17:57.554		
13	2020-12-03 16:17:57.569	INFO	CleaningCycle1	CleaningCycle Calculation Result	ACTIVE	STANDBY
14	2020-12-03 16:17:57.583	INFO	CleaningCycle2	CleaningCycle Calculation Result	ACTIVE	STANDBY
15	2020-12-03 16:17:57.598	INFO	CleaningCycle3	CleaningCycle Calculation Result	ACTIVE	STANDBY
16	2020-12-03 16:17:57.604	INFO	CleaningCycle4	CleaningCycle Calculation Result	ACTIVE	STANDBY
17	2020-12-03 16:17:57.611	INFO	SelfChecking_CC Comparator	Compare CleaningCycle1 and CleaningCycle2	[STANDBY, STANDBY]	STANDBY
18	2020-12-03 16:17:57.623	INFO	SelfChecking_CC Comparator	Compare CleaningCycle3 and CleaningCycle4	[STANDBY, STANDBY]	STANDBY
19	2020-12-03 16:17:57.631	INFO	CleaningCycleSwitch	CleaningCycle Compare of 2 SelfCheckingComponents	[STANDBY, STANDBY, null, null, null]	STANDBY

Abbildung 86 - Logdatei eines fehlerfreien Funktionsablaufs im Smartphone

Stehen im GBS neue Sensordaten zur Verfügung (siehe Zeile 1 in Abbildung 86), werden diese durch den Aktivitätserkennungsservice weiterverbreitet (siehe Zeile 2-6

in Abbildung 86). Drei Neuronale-Netze verarbeiten parallel die Sensordaten und klassifizieren eine Aktivität. Die Ergebnisse der einzelnen Services werden durch einen Mehrheitsentscheid geprüft (siehe Zeile 6 in Abbildung 86). Die erkannte Aktivität „*Belastung*“ wird an den *ComparativeVoter* weitergleitet (siehe Zeile 7 in Abbildung 86). Ein zweites im Backup arbeitendes Smartphone sendet die Information der klassifizierten Aktivität an das aktive Smartphone. Diese zusätzliche Information über die aktuelle Belastung steht dem Entscheidungsprozess zusätzlich zur Verfügung (siehe Zeile 8-9 in Abbildung 86). Die übereinstimmenden Aktivitäten werden in der *History* für eine Fehlerbehandlung eines in der Zukunft auftretenden transienten Fehlers abgespeichert (siehe Zeile 10 in Abbildung 86). Der *OutlierFilter* überprüft die aktuell erkannte Aktivität mit den zwei zuvor detektierten Aktivitäten aus den vergangenen Spülzyklusabläufen (siehe Zeile 11 in Abbildung 86). Der *OutlierFilter* leitet die in Abbildung 86 erkannte Aktivität „*Belastung*“ an den Spülzyklusservice weiter. Der Spülzyklusservice aktiviert zu Beginn vier Spülzyklen (siehe Zeile 13-16 in Abbildung 86), die in zwei selbstüberwachende Module zusammengefasst werden (siehe Zeile 17-18 in Abbildung 86). Die erkannte Aktivität „*Belastung*“ führt zu keinem Spülzyklus der Aortenklappe und die Steuerinformation „*Standby*“ (Pumpengeschwindigkeit von 2500 RPM) wird an die LVAD-Steuereinheit weitergeleitet (siehe Zeile 19 in Abbildung 86).

Der fehlerfreie Ablauf in Abbildung 86 zeigt die Ausgaben der zwei *SelfChecking_CC* Module und übermittelt ein Steuersignal für die Geschwindigkeit von 2500 RPM an die LVAD-Steuereinheit. Die Ergebnisse der verschiedenen im Zusammenspiel miteinander abhängigen Services des LVAD-BDS ergaben im Integrationstest keine fehlerhaften Ereignisse. In diesem Zusammenhang wurden zusätzlich die Latenzzeiten des LVAD-Spülzykluses untersucht. Die Analyse ergab, dass die minimale Laufzeit (Best-Case-Execution-Time) eine Ende-zu-Ende-Verzögerung von 0,790 Sekunden in einem Smartphone aufwies. Dies bedeutet, dass ein Betriebswechsel vom Spülzyklus-Modus in den Standard-Modus minimal 6.19 Sekunden benötigt. Dies ist dem *OutlierFilter*-Service geschuldet, welcher durch einen Aktivitätswechsel drei Sensorzyklen (zu je 1.8 Sekunden) sowie die minimale Latenz (0,790 Sekunden) benötigt, um einen neuen Betriebsstatus anzunehmen. Die maximale Laufzeit (Worst-Case-Execution-Time), welche einer erhöhten Zuverlässigkeitsanforderung unterliegt (vergleiche Kapitel 5.2), wird im folgenden Abschnitt durch geeignete Fehlerinjektionen analysiert und bewertet.

7.2 Fehlerinjektion

Um zeitliche Anomalien im System zu simulieren sowie die Funktionalität und Korrektheit der Fehlertoleranzmechanismen bewerten zu können, wurden Fehlerinjektionsmaßnamen in der Anwendung durchgeführt.

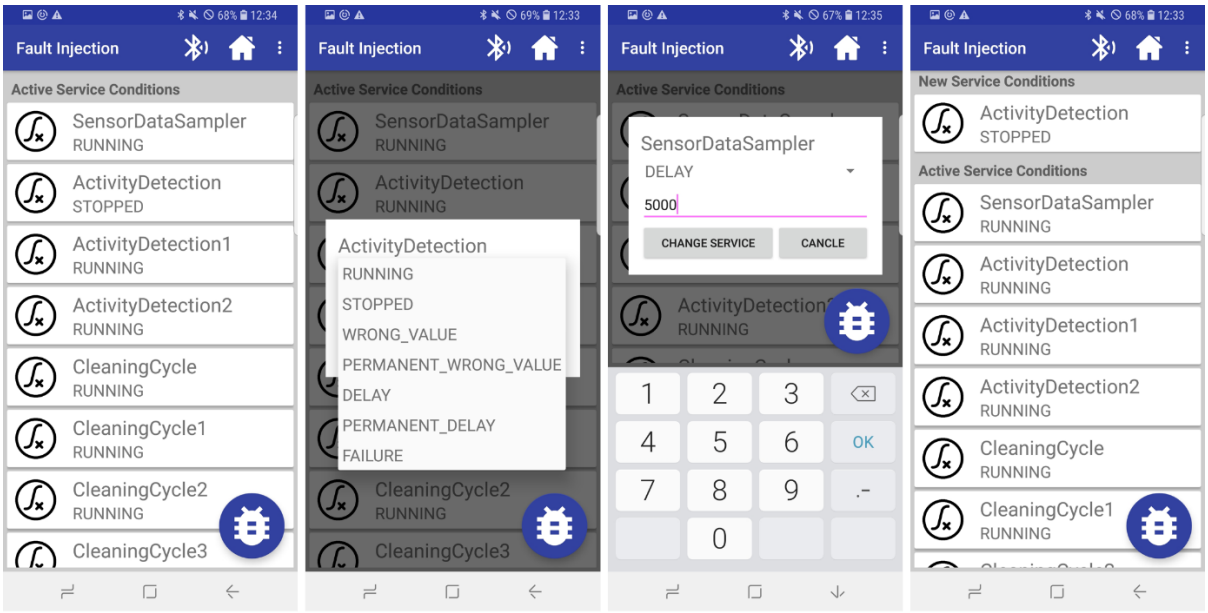


Abbildung 87 - UI der Fehlerinjektionsfunktionalität in der Spülzyklus-APP

Abbildung 87 stellt die graphische Benutzeroberfläche der LVAD-Anwendung zur Fehlerinjektion dar. Abbildung 87 (a) zeigt eine Auflistung aller aktiven LVAD-Services (*SensorDataSample*, *ActivityDetection*, *CleaningCycle*), die mittels Fehlerinjektion modifiziert werden können. Die zu injizierenden Fehler können zeitliche oder funktionale Anomalien darstellen (siehe Abbildung 87 (b)). Zeitliche Unregelmäßigkeiten können zusätzlich mit einer Zeitspanne erweitert werden (Abbildung 87 (c)). Des Weiteren kann eine Übersicht der aktuellen Informationen zu den einzelnen Services inklusive aller Fehlerinjektionen angezeigt werden (siehe Abbildung 87 (d)).

Dies ermöglicht es, die verwendeten Fehlertoleranzmaßnahmen der *Aktivitätserkennung*, der *History* sowie des *Spülzyklus* zu testen. Als Erstes wird die Aktivitätserkennung und damit das RcB-AV-Muster untersucht. Abbildung 88 stellt die Logdatei eines Smartphones dar, in dem zwei der jeweils drei aktiven Aktivitätserkennungs-Threads mit einer Verzögerung durchgeführt werden (siehe Zeile 2-3 in Abbildung 88 blau gekennzeichnete Fehlerinjektions-Ereignisse). Als Fehlerinjektion werden die Aktivitätserkennungs-Threads 1 und 2 jeweils unter Verzögerung von 500ms ausgeführt.

Ein Watchdog-Timer überwacht die Berechnung des Neuronalen-Netzes und toleriert eine Berechnungszeit von bis zu 200ms (siehe Zeile 5-6 in Abbildung 88 rot gekennzeichnete TIME ERROR). Dies führt in den ersten beiden Ausführungsblöcken AD1 und AD2 zu einer Fehlererkennung am Mehrheitsentscheider (siehe Zeile 7 und 16 in Abbildung 88 rot gekennzeichnete FUNCTIONAL ERROR). Stehen dem Prozess noch zusätzliche Recoveryblöcke zur Verfügung, werden diese aktiviert (siehe Zeile 8 und 17 in Abbildung 88 gelb gekennzeichnete TIME WARNING) und wiederholen den Aktivitätserkennungsservice. Der dritte Durchlauf zur Bestimmung der Aktivität

(siehe Zeile 18-24 in Abbildung 88) des gesamten RcB-AV-Musters liefert nach ca. 500ms durch den Mehrheitsentscheid (siehe Zeile 24 in Abbildung 88) ein akzeptables Ergebnis. Die Auswertungen der Fehlerinjektionsdaten zeigen, dass die Maßnahmen des RcB-AV-Musters zur Fehlererkennung, -maskierung, -eindämmung und -behandlung wirken. Die maximal zu tolerierende Verzögerung für die Erkennung einer Aktivität beträgt ca. 629ms.

Msg. No.	Timestamp	Error INFO	Device_Component	Information	Input	Result
1	2020-11-25 13:49:46.103	INFO	First_ADUnit	Activity Watchdog Timer start: 13:49:46.102		
2	2020-11-25 13:49:46.106	FAULT INJECTION	AD1	Injecting: DELAYof 500 ms		
3	2020-11-25 13:49:46.109	FAULT INJECTION	AD2	Injecting: DELAYof 500 ms		
4	2020-11-25 13:49:46.115	INFO	AD3	ActivityDetection3 Calculation Result by neural network	DataSample	STILL
5	2020-11-25 13:49:46.307	TIME_ERROR	AD2	Missing ActivityDetection2 after 200ms		
6	2020-11-25 13:49:46.332	TIME_ERROR	AD1	Missing ActivityDetection1 after 200ms		
7	2020-11-25 13:49:46.335	FUNCTIONAL ERROR	MajorityVoter	Compare TMR of ActivityDetection1, ActivityDetection2 and ActivityDetection3	[NONE, NONE, STILL]	NONE
8	2020-11-25 13:49:46.339	TIME WARNING	ActivityDetection Monitoring	1. Repeat of ActivityDetection	NONE	
9	2020-11-25 13:49:46.343	INFO	Second_ADUnit	Activity Watchdog Timer start: 13:49:46.342		
10	2020-11-25 13:49:46.346	FAULT INJECTION	AD1	Injecting: DELAYof 500 ms		
11	2020-11-25 13:49:46.351	FAULT INJECTION	AD2	Injecting: DELAYof 500 ms		
12	2020-11-25 13:49:46.357	INFO	AD3	ActivityDetection3 Calculation Result by neural network	DataSample	STILL
13	2020-11-25 13:49:46.368	INFO	Monitoring SPCommunication	Received Remote Device Activity at 13:49:46.368	STILL	STILL
14	2020-11-25 13:49:46.547	TIME_ERROR	AD1	Missing ActivityDetection1 after 200ms		
15	2020-11-25 13:49:46.549	TIME_ERROR	AD2	Missing ActivityDetection2 after 200ms		
16	2020-11-25 13:49:46.552	FUNCTIONAL ERROR	MajorityVoter	Compare TMR of ActivityDetection1, ActivityDetection2 and ActivityDetection3	[NONE, NONE, STILL]	NONE
17	2020-11-25 13:49:46.556	TIME WARNING	ActivityDetection Monitoring	2. Repeat of Activity Detection	NONE	

18	2020-11-25 13:49:46.558	INFO	Third_ADUnit	Activity Watchdog Timer start: 13:49:46.558		
19	2020-11-25 13:49:46.561	FAULT INJECTION	AD1	Injecting: DELAYof 500 ms		
20	2020-11-25 13:49:46.565	FAULT INJECTION	AD2	Injecting: DELAYof 500 ms		
21	2020-11-25 13:49:46.569	INFO	AD3	ActivityDetection3 Calculation Result by neural network	DataSample	STILL
22	2020-11-25 13:49:46.610	INFO	AD2	ActivityDetection2 Calculation Result by neural network	DataSample	STILL
23	2020-11-25 13:49:46.625	INFO	AD1	ActivityDetection1 Calculation Result by neural network	DataSample	STILL
24	2020-11-25 13:49:46.628	INFO	MajorityVoter	Compare TMR of ActivityDetection1, ActivityDetection2 and ActivityDetection3	[STILL, STILL, STILL]	STILL
25	2020-11-25 13:49:46.636	INFO	ActivityDetection Monitoring	Activity Detection Result	STILL	STILL
26	2020-11-25 13:49:46.642	INFO	ComparativeVoter	Compare Activity Detection Results	[STILL, STILL]	STILL

Abbildung 88 - Fehlerinjektion einer Verzögerung des Aktivitätserkennungs-Service

Ein weiterer wichtiger Mechanismus zur Einhaltung der Rechtzeitigkeit und Dienstgüte stellt die *AktivityHistory* dar. Abbildung 89 zeigt Ausschnitte der Logdatei, in der die Abtastrate der Sensoreinheit für 8.9 Sekunden verzögert agiert (siehe Zeile 1, 10 und 17 in Abbildung 89 blau gekennzeichnete FAULT INJECTION). Der Sensor-Watchdog toleriert eine zeitliche Verzögerung von 3 Sekunden und leitet nach Ablauf der Zeit ein *NoValue*-Ereignis im System weiter (siehe Zeile 3, 11 und 19 in Abbildung 89 rot gekennzeichnete TIME_ERROR). Der Kontrollpunkt (engl. checkpoint) der Aktivitätserkennung bemerkt die fehlenden Sensordaten und leitet das *NoValue*-Ereignis im System an den *ComparativeVoter* weiter (siehe Zeile 4, 12 und 20 in Abbildung 89). Der *ComparativeVoter* (siehe Zeile 5, 13 und 21 in Abbildung 89) vergleicht die berechnete Aktivität mit der berechneten Aktivität des externen Smartphones.

Der *History*-Prozess reagiert auf das *NoValue*-Ereignis und maskiert die fehlende Aktivität (siehe Zeile 6, 14 und 21 in Abbildung 89 gelb gekennzeichnete TIME WARNING), indem eine zuvor anliegende Aktivität (Ruhe) im System weiterleitet wird. In der simulierten Verzögerung der Sensoreinheit von 8.9 Sekunden wird der *History*-Prozess drei Mal aktiviert und hält die Spülzyklusfunktionalität am Leben. Die Mechanismen des LBH-Musters greifen und transiente *NoValue*-Fehler werden erfolgreich maskiert. Die Auswertungen der gesamten Logdaten zeigen, dass Verzögerungen an der Sensoreinheit sowie der Aktivitätserkennung von bis zu 12.82 Sekunden durch den *History*-Prozess toleriert werden können.

Msg. No.	Timestamp	Error INFO	Device_Component	Information	Input	Result
1	2020-12-03 16:00:27.084	FAULT INJECTION	SensorDataCheckpoint	Injecting: DELAYof 8900 ms		
2	2020-12-03 16:00:27.124	INFO	Monitoring SPCommunication	Received Remote Device Activity at 16:00:27.124	STILL	STILL
3	2020-12-03 16:00:28.304	TIME_ERROR	SensorDataCheckpoint	No Sensor Data Sample Received after: 3000 ms		
4	2020-12-03 16:00:28.332	TIME_ERROR	ActivityDetection Checkpoint	Missing Sensor Data		NONE
5	2020-12-03 16:00:28.337	FUNCTIONAL ERROR	ComparativeVoter	Compare Activity Detection Results	[NONE, STILL]	NONE
6	2020-12-03 16:00:28.342	TIME WARNING	ActivityHistory	1. History Repeat	NONE	STILL
7	2020-12-03 16:00:28.346	INFO	OutlierFilter	Computation of [STILL, STILL, STILL]	STILL	STILL
8	...	...	...	...	...	...
9	2020-12-03 16:00:29.183	INFO	Monitoring SPCommunication	Received Remote Device Activity at 16:00:29.183	ACTIVE	ACTIVE
10	2020-12-03 16:00:30.677	FAULT INJECTION	SensorDataCheckpoint	Injecting: DELAYof 8900 ms		
11	2020-12-03 16:00:31.371	TIME_ERROR	SensorDataCheckpoint	No Sensor Data Sample Received after: 3000 ms		
12	2020-12-03 16:00:31.393	TIME_ERROR	ActivityDetection Checkpoint	Missing Sensor Data		NONE
13	2020-12-03 16:00:31.397	FUNCTIONAL ERROR	ComparativeVoter	Compare Activity Detection Results	[NONE, ACTIVE]	NONE
14	2020-12-03 16:00:31.400	TIME WARNING	ActivityHistory	2. History Repeat	NONE	STILL
15	2020-12-03 16:00:31.404	INFO	OutlierFilter	Computation of [STILL, STILL, STILL]	STILL	STILL
16	...	...	...	...	...	...
17	2020-12-03 16:00:34.277	FAULT INJECTION	SensorDataCheckpoint	Injecting: DELAYof 8900 ms		
18	2020-12-03 16:00:34.321	INFO	Monitoring SPCommunication	Received Remote Device Activity at 16:00:34.320	ACTIVE	ACTIVE
19	2020-12-03 16:00:34.425	TIME_ERROR	SensorDataCheckpoint	No Sensor Data Sample Received after: 3000 ms		
20	2020-12-03 16:00:34.446	TIME_ERROR	ActivityDetection Checkpoint	Missing Sensor Data		NONE
21	2020-12-03 16:00:34.449	FUNCTIONAL ERROR	ComparativeVoter	Compare Activity Detection Results	[NONE, ACTIVE]	NONE
22	2020-12-03 16:00:34.456	TIME WARNING	ActivityHistory	3. History Repeat	NONE	STILL
23	2020-12-03 16:00:34.461	INFO	OutlierFilter	Computation of [STILL, STILL, STILL]	STILL	STILL
24	...	...	...	...	...	...
25	2020-12-03 16:00:36.081	INFO	SensorDataCheckpoint	Receive Sensor Data Sample		
26	...	...	...	...	...	...
27	2020-12-03 16:00:36.121	INFO	ActivityDetection Monitoring	Activity Detection Result	ACTIVE	ACTIVE
28	2020-12-03 16:00:36.142	INFO	Monitoring SPCommunication	Received Remote Device Activity at 16:00:36.141	ACTIVE	ACTIVE
29	2020-12-03 16:00:36.147	INFO	ComparativeVoter	Compare Activity Detection Results	[ACTIVE, ACTIVE]	ACTIVE
30	2020-12-03 16:00:36.151	INFO	ActivityHistory	Save to History	ACTIVE	ACTIVE
31	2020-12-03 16:00:36.156	INFO	OutlierFilter	Computation of [STILL, STILL, ACTIVE]	ACTIVE	STILL
32	...	...	...	...	...	...

Abbildung 89 - Fehlerinjektion einer Verzögerung der Sensoreinheit

Weiterhin wurde der Spülzyklusprozess bzw. das NSCPAV-Muster durch die Fehlerinjektion untersucht. Für eine maximale Auslastung des Fehlertoleranzmusters wurden fünf selbstüberwachende Spülzyklusprozesse (*CleaningCycle1-10*) aktiviert. Abbildung 90 zeigt eine Logdatei der Fehlerinjektionen für den gesamten Spülzyklusprozess. Die blau gekennzeichneten Fehlerereignisse (siehe Zeile 2, 3, 13, 20 und 27 in Abbildung 90 FAULT INJECTION) stellen die zeitlichen Fehlerinjektionen (*Verzögerungen sowie Stoppen*) dar. Hierzu werden eine Verzögerung der Spülzyklusberechnung von 150ms für die Spülzyklen 1 und 9 injiziert (siehe Zeile 2 und 27) sowie die Berechnungen der Spülzyklen 3, 5 und 7 gestoppt (siehe Zeile 3, 13, und 20). Die gelb gekennzeichneten Ereignisse (TIME WARNING) stellen die Watchdog-Timer dar, die eine Fehlererkennung für die Spülzyklen 3, 5 und 7 nach 200ms detektieren (siehe Zeile 8, 15 und 22 in Abbildung 90).

Durch den Ausfall der dritten Spülzykluskomponente erkennt der Comparative-Voter der SC_CC2-Komponente (siehe Zeile 9) einen funktionalen Fehler (rot gekennzeichnete Fehlerereignisse FUNCTIONAL ERROR). Daraufhin wird eine neue Spare-Backup-Komponente (SC_CC3-Komponente) aktiviert (siehe Zeile 10). Im unteren Beispiel liefert erst der letzte selbstüberwachende Spülzyklusprozess (siehe Zeile 30 SC_CC5-Komponente) ein akzeptables Ergebnis (RECREATION). Die Auswertungen der Fehlerinjektionsdaten zeigen, dass die Maßnahmen zur Fehlererkennung und -behandlung des NSCPAV-Musters greifen. Die maximale Verzögerung die gemessen und noch toleriert wird, beträgt ca. 896ms für eine Berechnung der Spülzyklusaktivität.

Msg. No.	Timestamp	Error INFO	Device_Component	Information	Input	Result
1	2020-12-03 16:00:36.167	INFO	SelfChecking_CC	Cleaning Watchdog Timer start: 16:00:36.166		
2	2020-12-03 16:00:36.173	FAULT INJECTION	CleaningCycle1	Injecting: PERMANENT_DELAYof 150 ms		
3	2020-12-03 16:00:36.182	FAULT INJECTION	CleaningCycle3	Injecting: STOPPED		
4	2020-12-03 16:00:36.186	INFO	CleaningCycle4	CleaningCycle Calculation Result	STILL	RECREATION
5	2020-12-03 16:00:36.194	INFO	CleaningCycle2	CleaningCycle Calculation Result	STILL	RECREATION
6	2020-12-03 16:00:36.332	INFO	CleaningCycle1	CleaningCycle Calculation Result	STILL	RECREATION
7	2020-12-03 16:00:36.336	INFO	SelfChecking_CC1 Comparator	Compare CleaningCycle1 and CleaningCycle2	[RECREATION, RECREATION]	RECREATION
8	2020-12-03 16:00:36.367	TIME WARNING	CleaningCycle3 Watchdog	Missing CleaningCycle3 after 200ms		
9	2020-12-03 16:00:36.370	FUNCTIONAL ERROR	SelfChecking_CC2 Comparator	Compare CleaningCycle3 and CleaningCycle4	[NONE, RECREATION]	NONE
10	2020-12-03 16:00:36.375	FUNCTIONAL ERROR	CleaningVoter	Start next spare: SelfChecking_CC3	[RECREATION, NONE, null, null, null]	
11	2020-12-03 16:00:36.380	INFO	CleaningCycle5 Watchdog	Start CleaningCycle5 Watchdog Timer start: 16:00:36.379		
12	2020-12-03 16:00:36.385	INFO	CleaningCycle6 Watchdog	Start CleaningCycle6 Watchdog Timer start: 16:00:36.379		

13	2020-12-03 16:00:36.389	FAULT INJECTION	CleaningCycle5	Injecting: STOPPED		
14	2020-12-03 16:00:36.398	INFO	CleaningCycle6	CleaningCycle Calculation Result	STILL	RECREATION
15	2020-12-03 16:00:36.581	TIME WARNING	CleaningCycle5 Watchdog	Missing CleaningCycle5 after 200ms		
16	2020-12-03 16:00:36.585	FUNCTIONAL ERROR	SelfChecking_CC3 Comparator	Compare CleaningCycle5 and CleaningCycle6	[NONE, RECREATION]	NONE
17	2020-12-03 16:00:36.590	FUNCTIONAL ERROR	CleaningVoter	Start next spare: SelfChecking_CC4	[RECREATION, NONE, NONE, null, null]	
18	2020-12-03 16:00:36.600	INFO	CleaningCycle7 Watchdog	Start CleaningCycle7 Watchdog Timer start: 16:00:36.599		
19	2020-12-03 16:00:36.605	INFO	CleaningCycle8 Watchdog	Start CleaningCycle8 Watchdog Timer start: 16:00:36.599		
20	2020-12-03 16:00:36.611	FAULT INJECTION	CleaningCycle7	Injecting: STOPPED		
21	2020-12-03 16:00:36.616	INFO	CleaningCycle8	CleaningCycle Calculation Result	STILL	RECREATION
22	2020-12-03 16:00:36.801	TIME WARNING	CleaningCycle7 Watchdog	Missing CleaningCycle7 after 200ms		
23	2020-12-03 16:00:36.804	FUNCTIONAL ERROR	SelfChecking_CC4 Comparator	Compare CleaningCycle7 and CleaningCycle8	[NONE, RECREATION]	NONE
24	2020-12-03 16:00:36.810	FUNCTIONAL ERROR	CleaningVoter	Start next spare: SelfChecking_CC5	[RECREATION, NONE, NONE, NONE, null]	
25	2020-12-03 16:00:36.815	INFO	CleaningCycle9 Watchdog	Start CleaningCycle9 Watchdog Timer start: 16:00:36.814		
26	2020-12-03 16:00:36.819	INFO	CleaningCycle10 Watchdog	Start CleaningCycle10 Watchdog Timer start: 16:00:36.814		
27	2020-12-03 16:00:36.822	FAULT INJECTION	CleaningCycle9	Injecting: PERMANENT_DELAYof 150 ms		
28	2020-12-03 16:00:36.826	INFO	CleaningCycle10	CleaningCycle Calculation Result	STILL	RECREATION
29	2020-12-03 16:00:36.977	INFO	CleaningCycle9	CleaningCycle Calculation Result	STILL	RECREATION
30	2020-12-03 16:00:36.980	INFO	SelfChecking_CC5 Comparator	Compare CleaningCycle9 and CleaningCycle10	[RECREATION, RECREATION]	RECREATION
31	2020-12-03 16:00:36.986	INFO	CleaningVoter	Result CleaningCycle Calculation of 5 SelfChecking_CC	[RECREATION, NONE, NONE, NONE, RECREATION]	RECREATION

Abbildung 90 - Fehlerinjektion einer Verzögerung des Spülzyklus-Service

Die Auswertung der einzelnen Fehlerinjektionsabläufe ermöglicht eine Berechnung der maximalen Ende-zu-Ende-Latenz für die gesamte LVAD-BDS-Anwendung. Aus Kapitel 5.2 unterliegt das LVAD-BDS einer sicherheitskritischen Anforderung (Anforderung #120) bezüglich der zeitlichen Grenzen für einen Betriebswechsel aus dem Spülzyklus in den normalen LVAD-Betrieb. Befindet sich ein Patient in einer Ruhephase mit aktivem Spülzyklus und eine Belastungsphase tritt ein, muss nach spätestens 30 Sekunden eine konstante RPM von 2500 am LVAD anliegen. Betrachtet man die einzelnen Ereignisse aus allen Systemkomponenten und Fehlertoleranzmustern, ergibt sich im schlechtesten Fall eine maximale Latenz von 29.79 Sekunden um in den sicheren Normalzustand des LVADs zu wechseln. Die

Fehlerinjektion und Analyse demonstrieren, dass die Rechtzeitigkeitsanforderungen für eine erweiterte LVAD Steuerung durch unzuverlässige IoT-Geräte eingehalten werden können.

7.3 Kommunikationsverzögerungen

Um Fehlerinformationen über die Bluetooth-Kommunikation in einem BDS zu erhalten, wurden drei Experimente zur Bestimmung der Übertragungsverzögerung durchgeführt. Für den Versuchsaufbau wurden zwei Android-Anwendungen entwickelt, die einen Bluetooth-Sender bzw. -Empfänger repräsentieren. Das Smartphone, welches als Senderprozess agiert, verschickt in einem Intervall von 1.8 Sekunden ein Nachrichtenformat wie in Abbildung 47 dargestellt an den Server-Socket des Empfängerprozesses. Das Smartphone, welches als Empfängerstation dient, speichert die empfangene Nachricht ab und fügt einen Zeitstempel hinzu. Dies ermöglicht eine Bewertung der aufgetretenen Verzögerungen sowie der Fehlerrate der Bluetooth-Kommunikation. Als Grenzwert für eine Verzögerung wurde 10% der Abtastrate angenommen. Dieser Grenzwert stellt eine Verzögerung von 180ms dar.

In einem ersten Experiment wurde der Sender am Körper getragen und der Empfänger stationär betrieben. Die Distanz zwischen den beiden Geräten variierte zwischen wenigen Zentimetern und 4 Metern. Zusätzlich wurden auf dem Sendergerät darüber hinaus Soziale-Medien, Browseraktivitäten sowie Multimedia-Anwendungen über den Testzeitraum normal genutzt.

Tabelle 18 - Bluetoothexperiment 1 zur Ermittlung der Übertragungsverzögerung

Bluetooth-Experiment 1:	
Datenumfang:	20814 Datenpakete
Maximale Verzögerung:	402ms
Durchschnittliche Verzögerung:	9.7ms
Anzahl an Datenpakete, die eine Verzögerung von >= 180ms aufwiesen:	11

Tabelle 18 stellt die Messergebnisse des ersten Bluetooth-Experiments dar. Im Testzeitraum wurden insgesamt 20814 Nachrichten untersucht. Die gemessene maximale Verzögerung (*engl. delay*) der Kommunikation betrug 402ms. Im Durchschnitt betrug die Verzögerung der Bluetooth-Kommunikation 9.7ms. Insgesamt wiesen im Experiment 11 Nachrichten eine erhöhte Latenz von mehr als 180ms auf.

In einem weiteren Experiment wurden die beiden Smartphones (Sender und Empfänger) für eine Bewegungsanalyse an einen Roboterarm befestigt (vergleiche Abbildung 83). Ein Bewegungsprofil, welches durchgängig 10 Minuten Ruhe gefolgt von 10 Minuten Bewegung simuliert, wurde entworfen und angewandt. Tabelle 19 stellt die Messergebnisse des zweiten Experiments dar. Im Testzeitraum wurden insgesamt 29494 Nachrichten untersucht. Die gemessene Verzögerung der Kommunikation betrug maximal 275ms und im Durchschnitt 9.0ms. In diesem Experiment zeigte nur eine Nachricht eine erhöhte Verzögerung von mehr als 180ms.

Tabelle 19 - Bluetoothexperiment 2 zur Ermittlung der Übertragungsverzögerung

Bluetooth-Experiment 2:	
Datenumfang:	29494 Datenpakete
Maximale Verzögerung:	275ms
Durchschnittliche Verzögerung:	9.0ms
Anzahl an Datenpakete, die eine Verzögerung von >= 180ms aufwiesen:	1

Im dritten Kommunikations-Experiment wurden die beiden Smartphones mit einem Abstand von ca. 30cm stationär positioniert und verweilten über den gesamten Zeitraum in dieser Position. Tabelle 20 stellt die Messergebnisse des dritten Experiments dar. Im Testzeitraum wurden insgesamt 37692 Nachrichten untersucht. Die gemessene Verzögerung betrug maximal 154ms und im Durchschnitt betrug die Verzögerung 10.4ms. In dem Experiment zeigte keine Nachricht eine höhere Verzögerung als der Grenzwert von 180ms.

Die gesamte Testreihe umfasst 88000 Datenpakete über 44 Stunden. Abbildung 91 stellt die gesamten Messdaten der drei Experimente grafisch dar. Die durchschnittliche Verzögerung betrug 9.7ms und insgesamt 12 Nachrichten überschritten den Grenzwert der Bluetooth-Kommunikation von 180ms.

Tabelle 20 - Bluetoothexperiment 3 zur Ermittlung der Übertragungsverzögerung

Bluetooth-Experiment 3:	
Datenumfang:	37692 Datenpakete
Maximale Verzögerung:	154ms
Durchschnittliche Verzögerung:	10.4ms
Anzahl an Datenpakete, die eine Verzögerung von >= 180ms aufwiesen:	0

Dies bedeutet, dass die Bluetooth-Kommunikation eine Fehlerrate von 0.27 Verzögerungen pro Stunde aufweist, was einer mittleren Betriebsdauer zwischen zwei verspäteten Nachrichten von 3.6 Stunden entspricht.

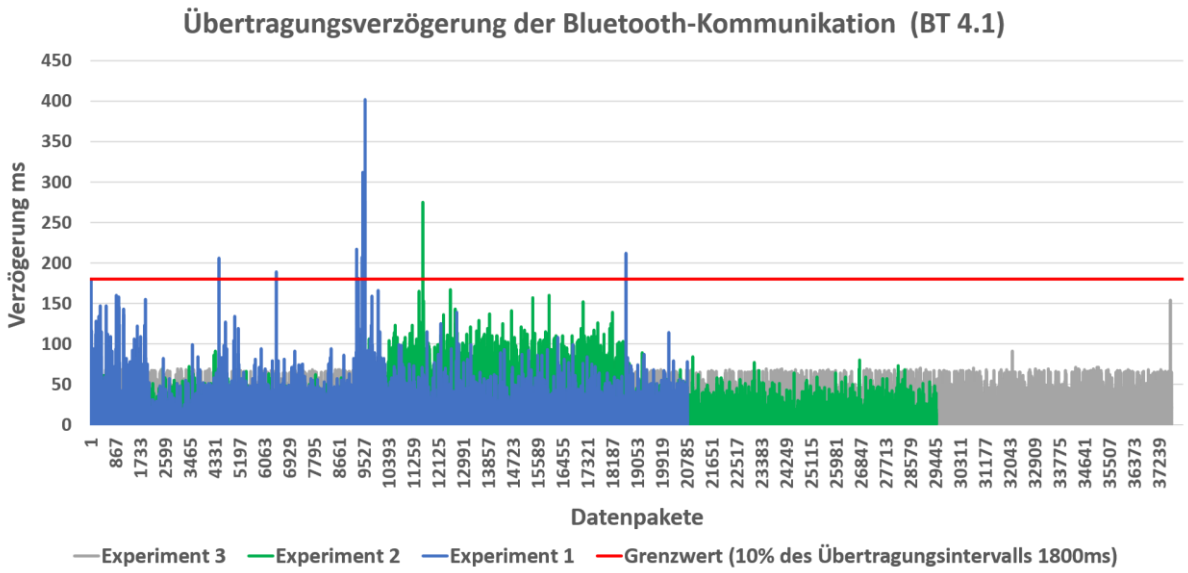


Abbildung 91 - Messergebnisse der Übertragungsverzögerung von Bluetooth 4.1

Im Folgenden werden Belastungstests des BDS durchgeführt, um die Dienstgüte zu bewerten. Dazu werden das Single- sowie Duplex-LVAD-BDS durch eine dynamische CPU-Anpassung einem Stresstest unterzogen und die Reaktionszeit sowie das Aktivieren des Fallback-Modus an der LVAD-Steuereinheit untersucht.

7.4 Belastungstests

Für eine zusätzliche Beurteilung der Dienstgüte wurde das LVAD-BDS mittels verschiedener CPU-Belastungsphasen analysiert. Um zur Laufzeit eine variable CPU-Auslastung der Smartphones zu ermöglichen, wurde darüber hinaus ein Service entwickelt, der durch das Ausführen von zusätzlichen parallelen Threads die CPU-

Belastung beeinflusst. Die Threads rufen die aktuelle Systemzeit ab, welche eine aufwendige Berechnung auf Basis von Fließkommazahlen darstellt. Durch das Aktivieren bzw. Beenden von Belastungs-Threads kann so eine variable CPU-Belastung simuliert werden.

In einem ersten Experiment wurden das Single-Smartphone-LVAD-System ohne Fehlertoleranzmechanismen sowie das Single-Smartphone-LVAD-System mit Fehlertoleranzmechanismen unter verschiedenen CPU-Belastungsstufen auf deren Reaktionszeit untersucht. Für die Testreihe wurde pro Stunde die CPU-Belastung von 0% bis 70% in 10% Schritten gesteigert. Weiterhin wurde ab 70% CPU-Belastung eine Anpassung der Belastung in 5% Schritten pro Stunde auf bis zu 95% CPU-Belastung vorgenommen. Abbildung 92 stellt die Ergebnisse des Experiments dar. Das nicht fehlertolerante System zeigt erste Verzögerungen ab einer CPU-Belastung von 40%. Ab einer Belastungsstufe von 40% kommt es zu einer Nachrichtenverzögerung und dadurch zu einem geringeren Nachrichtenaustausch von ca. 15% pro Stunde. Mit höheren Belastungsstufen wird das System durchgehend langsamer und es kommen immer weniger Stellsignale am LVAD an. Bei einer konstanten CPU-Belastung von ca. 80% kam es über die gemessene Zeitspanne zu einem verringerten Datentransfer von ca. 40%.

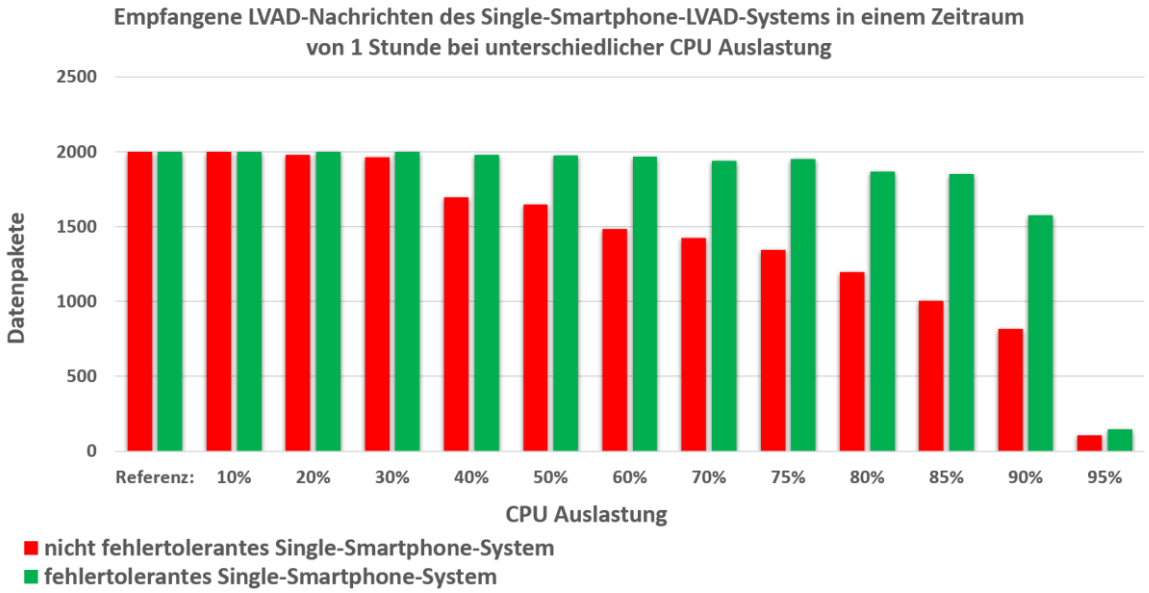


Abbildung 92 - Abweichungen der empfangenen LVAD-Nachrichten bei verschiedenen CPU-Belastungsphasen

Das fehlertolerante System weist hingegen auch bei erhöhter CPU-Auslastung keine erheblichen Nachrichtenverzögerungen auf. Bei einer CPU-Belastung von ca. 80% wurde eine Abweichung der Datenpakete von ca. 6.6% gemessen. Die gewonnenen Informationen bezüglich der Dienstgüte wurden zusätzlich durch die gemessenen Reaktionszeiten der beiden Systeme gefestigt. Abbildung 93 zeigt die gemessenen durchschnittlichen Reaktionszeiten der beiden LVAD-BDS bei unterschiedlichen CPU-Belastungen. Die Daten belegen, dass das nicht fehlertolerante System ab ca. 40% CPU-Auslastung verzögert reagiert. Hingegen gewährleisten die

Fehlertoleranzmechanismen des fehlertoleranten Systems erfolgreich die Qualität der Services auch unter erhöhter CPU-Auslastung. Erst ab einer CPU-Auslastung von ca. 85% kommt es zu einer geringen Verzögerung der Reaktionszeit von ca. 140 ms gegenüber den 1.8 Sekunden im Normalbetrieb.

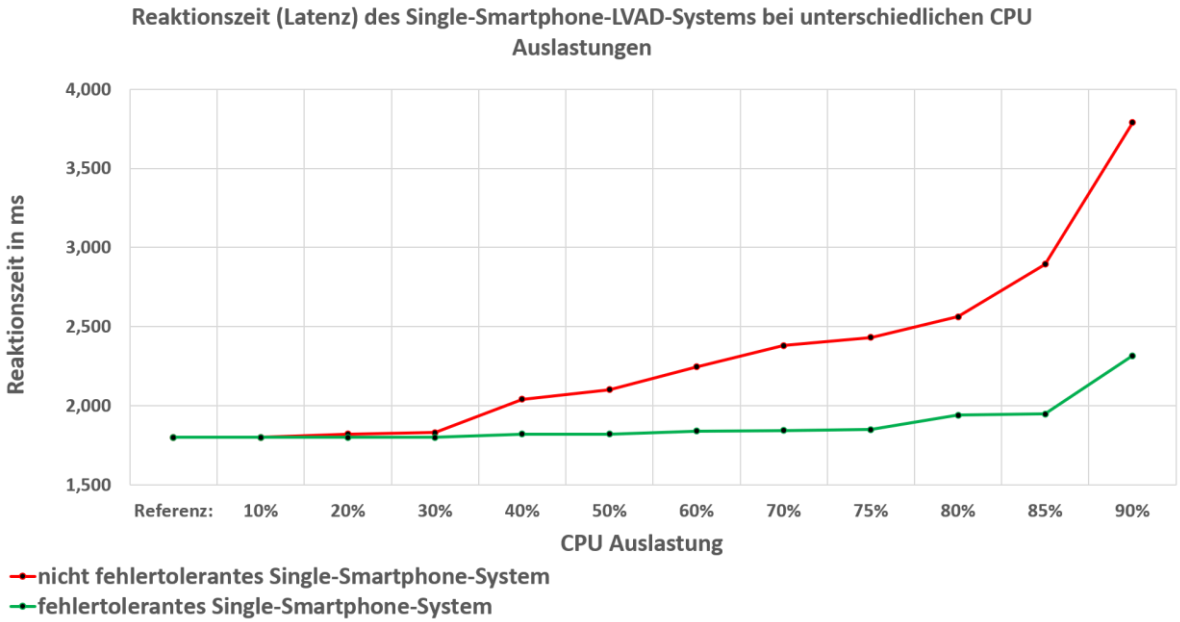


Abbildung 93 - Reaktionszeit der LVAD-Anwendung bei verschiedenen CPU-Belastungsphasen

Neben der Bewertung des Nachrichtenverlusts und der Reaktionszeit liegt das Interesse in der Dienstgüte und Qualität der Steuerung des LVADs durch die erweiterte BDS-Funktionalität der IoT-Geräte. Abbildung 94 zeigt die Ergebnisse der Untersuchung für die Verweilzeit des LVAD im FB-Modus in verschiedenen Belastungsphasen durch das nicht fehlertolerante bzw. fehlertolerante Single-Smartphone-System. Die LVAD-Steuereinheit wechselt durch die Ansteuerung des nicht fehlertoleranten Smartphone-Systems zum ersten Mal bei einer CPU-Auslastung von ca. 70% in den FB-Modus. In der gemessenen Stunde existierte nur ein Betriebswechsel und das LVAD-BDS-System befand sich für ca. 20.8 Sekunden im FB-Modus. Ein weiterer Betriebswechsel wurde bei ca. 80% Auslastung des nicht fehlertoleranten Smartphone-Systems detektiert, und die Verweilzeit im abgesicherten FB-Modus betrug ca. 13.5 Sekunden. Bei 85% CPU-Auslastung des nicht fehlertoleranten Smartphone-Systems wurden insgesamt 6 Betriebsmodiwechsel gemessen und das LVAD befand sich insgesamt für ca. 4 Minuten im FB-Modus. Durch eine CPU-Auslastung von ca. 90% des nicht fehlertoleranten Smartphone-Systems verweilte das LVAD für ca. 17 Minuten im FB-Modus und insgesamt 9 Betriebsmodiwechsel wurden durchgeführt. Bei einer CPU-Auslastung von ca. 95% des nicht fehlertoleranten Smartphone-Systems verblieb das LVAD nach dem 3. Betriebsmodiwechsel durchgängig im FB-Modus, wobei die Verweilzeit ca. 53 Minuten betrug. Das fehlertolerante Smartphone-Systems hingegen hielt das LVAD und die erweiterte BDS-Funktionalität auch unter erhöhter Belastung am Leben und

erst ab einer CPU-Auslastung von ca. 90% wurde am LVAD ein Betriebswechsel über die Dauer von 6.09 Minuten beobachtet. Ähnlich wie beim nicht fehlertoleranten Smartphone-System ist auch das fehlertolerante System ab einer CPU-Auslastung von ca. 95% nach zwei Betriebsmodiwechseln durchgängig im FB-Modus und dies mit einer gemessenen Gesamtdauer von ca. 51 Minuten.

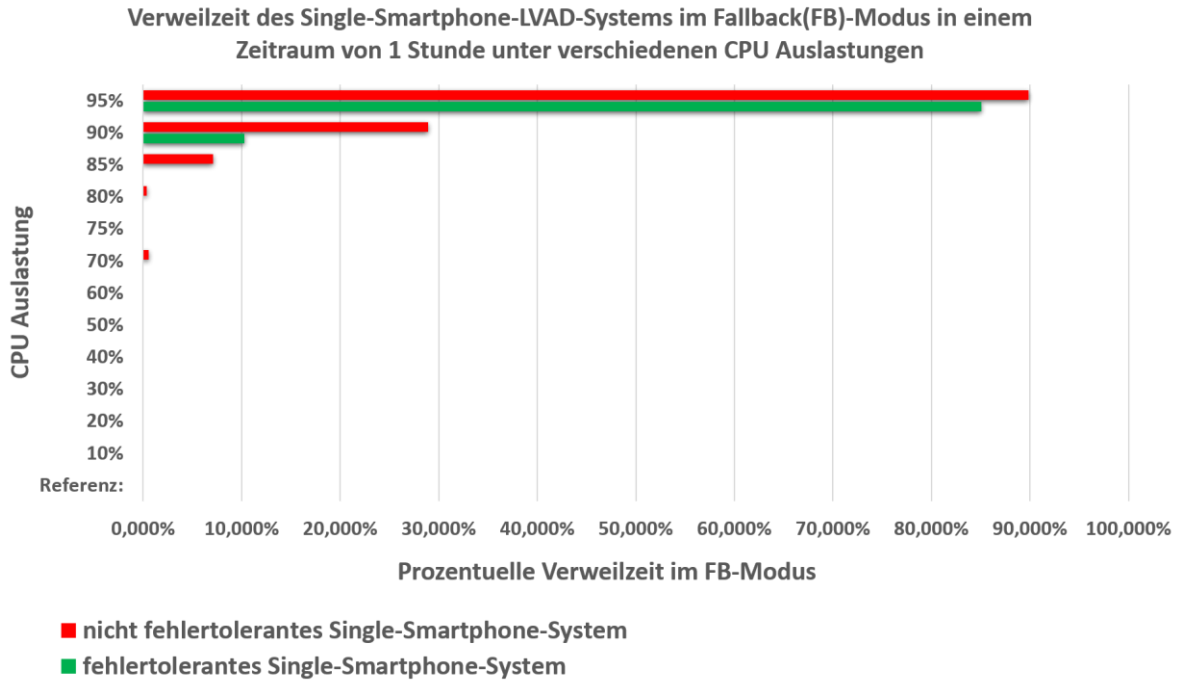


Abbildung 94 - Verweilzeit im FB-Modus unter verschiedenen CPU-Belastungsphasen

Alle Testreihen haben gezeigt, dass das Smartphone sowie das Betriebssystem ab einer CPU-Belastung von 90% nicht mehr ordnungsgemäß reagieren. Darüber hinaus zeigen die Ergebnisse deutlich, dass die entworfenen und entwickelten Fehlertoleranzmuster des fehlertoleranten Smartphone-System ihre Wirkung zeigen und die Dienstgüte trotz harter Sicherheitsanforderungen deutlich verbessern.

Für eine Beurteilung des entwickelten Duplex-Smartphone-Systems wurden die beiden Smartphones (*Primary- und Secondary-Smartphone*) unabhängig voneinander mittels zufallsgesteuerter CPU-Belastung betrieben und die LVAD Aktivität untersucht.

Silva et al. [104] untersuchen die durchschnittliche zeitliche Nutzung von verschiedenen Anwendungen auf mobilen Endgeräten in Südamerika. Abbildung 95 zeigt die Anwendungen, die von Benutzern am häufigsten ausgeführt wurden. Kurznachrichtendienste, Soziale-Netzwerke, Internetbrowser sowie E-Mail-Anwendungen wurden am häufigsten genutzt.

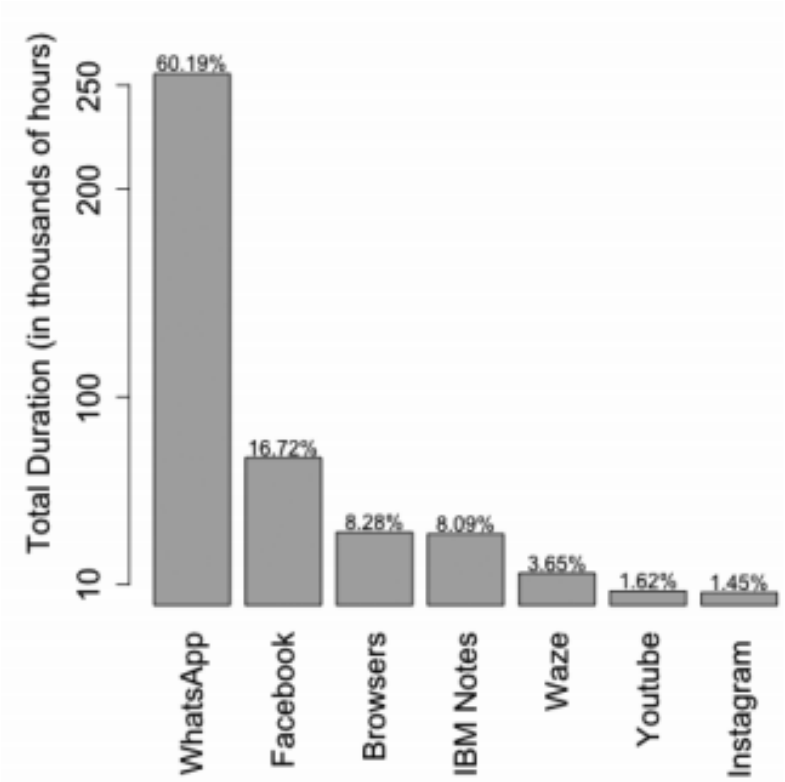


Abbildung 95 - Gesamtdauernutzung von Smartphoneanwendungen [104]

In einer weiteren Arbeit von F. Smolinski [105], wurde in einem achtstündigen Experiment die CPU-Auslastung der aus [104] ermittelten Anwendungen ermittelt. Die gemessenen CPU-Belastungen sind in Abbildung 96 dargestellt. Die CPU-Auslastung durch die gängigen Anwendungen lag im Schnitt zwischen 8% und 43%. Sporadisch traten bis zu 67% CPU-Auslastungen im Experiment auf.

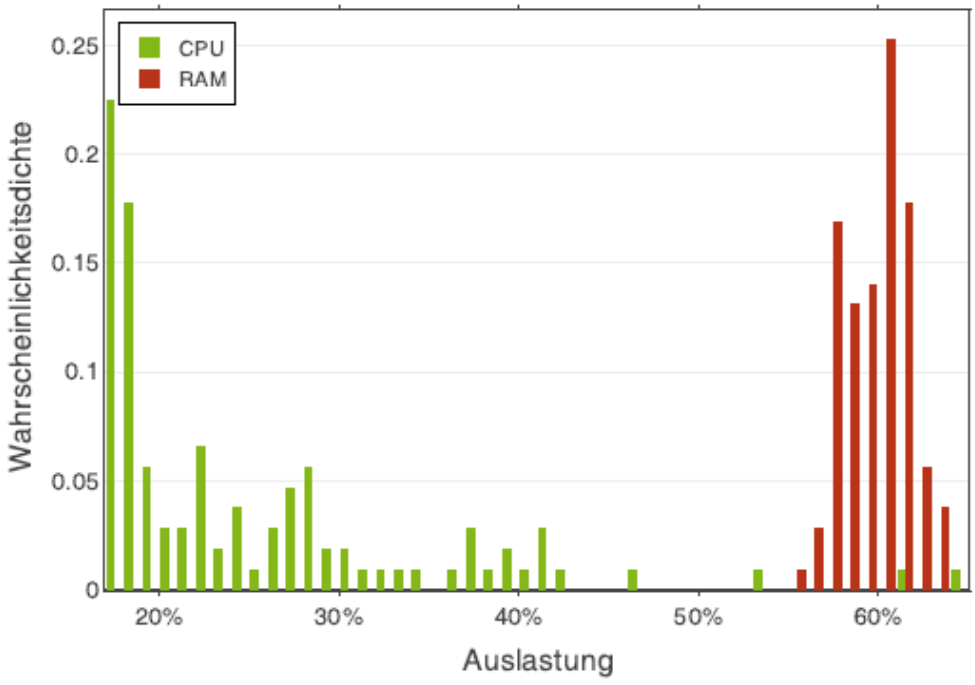


Abbildung 96 - CPU- und RAM-Auslastung eines Smartphones bei normaler Nutzung [105]

Diese Erkenntnisse bezüglich der CPU-Auslastung für beliebte Anwendungen wurden genutzt, um eine Belastungsprofil zu entwerfen, welches die Bewertung des Duplex-Smartphone-Systems im alltäglichen Betrieb ermöglicht. Tabelle 21 stellt das im Experiment verwendete zufallsgesteuerte Belastungsprofil des Duplex-Smartphone-Systems dar. Konservativ wurde eine erhöhte CPU-Belastung von ca. 20% gegenüber den Ergebnissen aus [105] angenommen, was eine Auslastung zwischen 40% bis 90% darstellt.

Tabelle 21 - Zufallsgesteuertes Belastungsprofil für das Duplex-LVAD-BDS

Zufallsgesteuerter Belastungsprofil des Duplex-Smartphone-Systems	
Zeitspanne innerhalb 1 Stunde:	**CPU-Auslastung:**
35%	40%
20%	50%
15%	60%
10%	75%
10%	80%
5%	85%
5%	90%

In einem einstündigen Experiment lagen unabhängig voneinander für 42 Minuten eine CPU-Belastung zwischen 40% und 60% an den beiden Smartphones an. Zusätzlich wurde zum Zweck eines Stresstests des Duplex-Smartphone-Systems die CPU-Auslastung zwischen 75% bis 95% in einem Zeitraum von 18 Minuten angelegt. Des Weiteren wurden die beiden Smartphones im Versuchsaufbau durch einen Roboterarm (vergleiche Abbildung 83) in Ruhe- bzw. Bewegungsphasen versetzt. Alternierend wechselte das Duplex-Smartphone-Systems alle 6 Minuten die Bewegungsphase. Die Ergebnisse der Latenzmessungen sowie der Wechsel des Betriebsmodus sind in Abbildung 97 dargestellt.

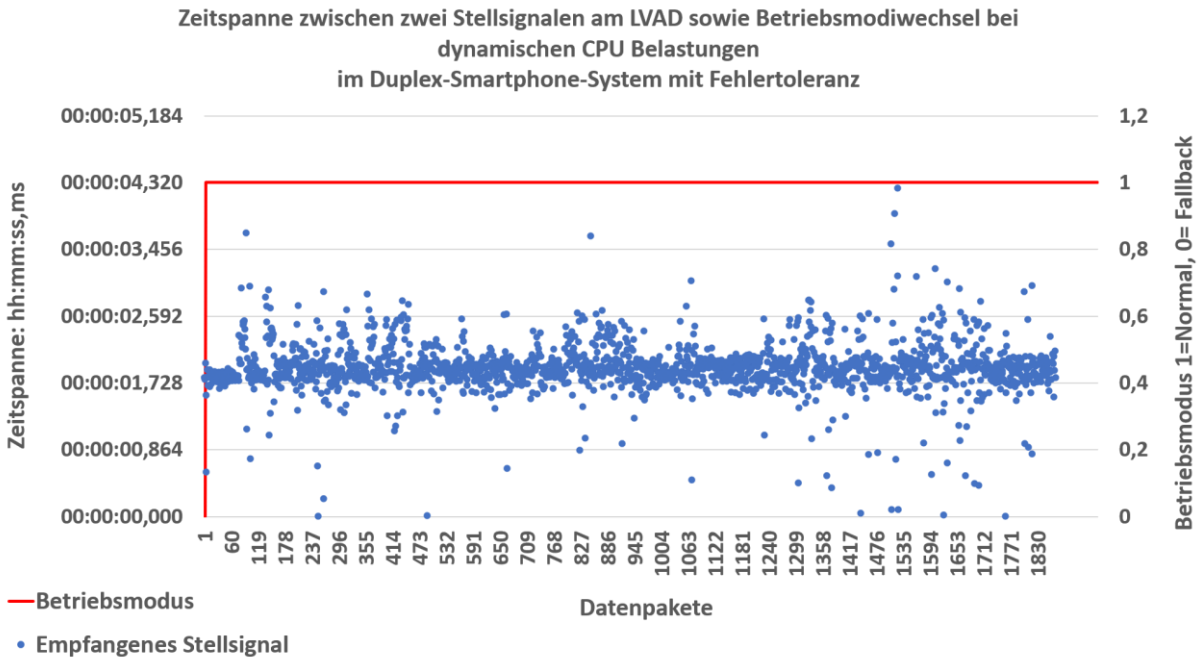


Abbildung 97 - Ergebnisse des Duplex-Smartphone-Systems unter dynamischer CPU-Auslastung

Die Auswertung des durchgeführten Experiments ergibt eine maximale Reaktionszeit von 4.25 Sekunden und eine durchschnittliche Reaktionszeit von 1.97 Sekunden. Insgesamt wurden in dem einstündigem Experiment 1869 Datenpakete am LVAD durch das gerätebasierte Duplex-Smartphone-Systems verarbeitet. Ein wichtiger Faktor für die Dienstgüte stellt der Wechsel in den FB-Modus dar. In dem gesamten Zeitraum wurde kein Wechsel in den FB-Modus am LVAD durchgeführt. Die Fehlertoleranzmuster verbessern die Dienstgüte und halten das LVAD-BDS trotz erhöhter CPU-Belastungsphasen reaktionsfähig.

8 ZUSAMMENFASSUNG

In dieser Arbeit wurde der Nachweis erbracht, dass BDS kostengünstig und qualitätsgesichert hinsichtlich der Rechtzeitigkeit entwickelt werden können. Dies wurde erfolgreich im Medolution-Projekt am medizinischen Anwendungsfall des Aortenklappenspülzyklusses in einer LVAD-Umgebung nachgewiesen.

Die folgenden Abschnitte präsentieren eine Zusammenfassung der Ergebnisse des entwickelten LVAD-BDS, die praktische Anwendung des LVAD-BDS sowie einen Ausblick für zukünftige Forschungsaktivitäten im Bereich von BDS.

8.1 Ergebnisse der Arbeit

Die vorliegende Arbeit stellt geeignete Fehlertoleranzverfahren und Sicherheitsmechanismen sowie daran angepasste Architekturkonzepte, Entwurfs- und Analyseverfahren vor. Zum Nachweis der Tragfähigkeit des Ansatzes ist ein BDS zur Steuerung und Überwachung von Linksherzunterstützungssystemen (LVAD) durch mobile Endgeräte modelliert, entwickelt und untersucht worden. Die im Rahmen dieser Arbeit gewonnenen Ergebnisse zeigen, dass die Rechtzeitigkeit und Dienstgüte durch die Einbindung von unzuverlässigen IoT-Geräten im Systemumfeld von sicherheitskritischen Systemen (SkS) weiterhin gewährleistet werden kann.

Als Grundlage für die Entwicklung von BDS dienen die in Kapitel 4 entworfenen Maßnahmen. Ein Geräteensemble aus unzuverlässigen und zuverlässigen Systemkomponenten bildet die Grundstruktur eines BDS und erweitert den Funktionsumfang der zuverlässigen Geräte. Die zuverlässigen Systemkomponenten werden um einen Fallback-Modus erweitert und fungieren als Perfektionskern im BDS-Kontext. Die unzuverlässigen Systemkomponenten stellen ein verteiltes dynamisches IoT-Geräteensemble im BDS dar. Diese IoT-Geräte arbeiten in multimodalem Betriebswechsel dynamisch nach dem Graceful Degradation Ansatz zusammen. Des Weiteren zeigen die Ergebnisse, dass die ausgewählten und angepassten Fehlertoleranzmechanismen (DRDV-, RcB-AV-, LBH- und NSCPAV-Muster), welche auf den IoT-Geräten Anwendung finden, maßgeblich dazu beitragen, dass die Rechtzeitigkeitsanforderungen sowie Dienstgüte der gerätebasierten Systeme und damit des gesamten BDS eingehalten werden können.

Die Modellanalysen aus Kapitel 6 zeigen, dass die MTBF durch die angewendeten Fehlertoleranzverfahren und Sicherheitsmechanismen von 6.2 Sekunden auf 8.8

Minuten verbessert wurde. Zusätzlich bekräftigen die aus Kapitel 7 gewonnen Ergebnisse der experimentellen Bewertung, dass aus Sicht der Rechtzeitigkeit und Dienstgüte ein BDS qualitätsgesichert entwickelt werden kann. Das fehlertolerante Duplex-Smartphone-LVAD-BDS weist im Belastungstest erst ab einer CPU-Belastung von 80% zeitliche Anomalien auf. Zusätzlich wurde in einem realistischen dynamischen Stresstest gezeigt, dass das fehlertolerante BDS auch unter erhöhter Belastung kaum Betriebswechsel in den Fallback-Sicherheitsmechanismus durchführt und gleichzeitig eine maximale Reaktionszeit von nur ca. 4 Sekunden aufweist.

Die technischen und medizinischen Anforderungen, die einem LVAD-BDS unterliegen und im Medolution Projekt definiert wurden, können trotz der Einbindung von unzuverlässigen IoT-Geräten eingehalten werden.

8.2 Praktische Anwendung der Ergebnisse

Das entwickelte LVAD-BDS fand in dem vom Bundesministerium für Bildung und Forschung (BMBF) geförderten Medolution Projekt Verwendung. Das BDS wurde im Projekt-Demonstrator eingesetzt, um eine gerätebasierte Datenerfassung, Aktivitätserkennung sowie Steuerungsfunktionen zur optimierten Gerätekonfiguration und zur Kontrolle von Vorbeugungs-, Versorgungs- und Notfallmaßnahmen von LVAD-Systemen zu gewährleisten. Zusätzlich diente das BDS zur Verifikation der funktionalen Korrektheit, der Rechtzeitigkeit sowie der Dienstgüte für den Anwendungsfall des Aortenklappenspülzyklus. Des Weiteren fungierte das entwickelte LVAD-BDS als Schnittstelle zur Evaluation des technischen Managements in medizinischen Umgebungen und der frühzeitigen Erkennung von LVAD-Pumpenthrombosen für Projektpartner.

8.3 Ausblick

Das in dieser Arbeit entwickelte LVAD-BDS zum Spülen der Aortenklappe steuert die LVAD-Pumpe auf Basis der Klassifikation aus zwei verschiedenen Aktivitäten (Ruhe und Belastung). Aktuelle Forschungsergebnisse zeigen, dass durch eine erweiterte Klassifikation der Aktivität (Treppe heraufsteigen, Treppe heruntersteigen, Gehen und schnelles Laufen) und die Verwendung zusätzlicher Informationen des gerätebasierten IoT-Systems eine dynamische Lastanpassung durch ein BDS durchgeführt werden kann [106–108]. Die zusätzlichen Informationen wie Herzfrequenz, Schrittgeschwindigkeit, Steigung, Patientengewicht und Patientenalter stehen durch die unzuverlässigen IoT-Geräte im BDS zur Verfügung und ermöglichen eine kontinuierliche Anpassung der Pumpengeschwindigkeit anhand der aktuell anliegenden Belastung. Solch eine dynamische Lastanpassung durch ein erweitertes LVAD-BDS kann nicht nur die Lebenszeit verlängern, sondern auch eine bessere Lebensqualität für LVAD-Patienten ermöglichen.

Ein weiterer Aspekt für die Verwendung sowie Weiterentwicklung von BDS im medizinischen Umfeld ist die Fernüberwachung der einzelnen LVAD-Geräte. Die medizinischen Partner im Medolution Projekt zeigen starkes Interesse an einem Telemedizin-Portal für LVAD-Patienten, welches eine geeignete Infrastruktur/Schnittstelle bei den Patienten voraussetzt. Das in dieser Arbeit entwickelte LVAD-BDS bietet eine solche Schnittstelle zwischen der LVAD-Pumpe und einem solchen Telemedizin-Portal der Kliniken. Das gerätebasierte System ermöglicht das Teilen von LVAD-Parametern zu externen Partnern. Diese frühzeitig zur Verfügung stehenden Informationen können die Kosten von Operationen vermeiden und den Patienten eine erhöhte Sicherheit durch die kontinuierliche Überwachung der Pumpenparameter geben. Zusätzlich können weitere Funktionen im BDS bereitgestellt werden, wie zum Beispiel die aktuelle Medikation oder Nahrungsaufnahme der Patienten. Diese würden dann dem medizinischen Fachpersonal über das Telemedizin-Portal für eine verbesserte Behandlung zur Verfügung gestellt.

Die neue Systemklasse der BDS, in der verschiedene vernetzte unzuverlässige IoT-Komponenten einen zusätzlichen Beitrag für kritische Funktionen liefern, um neuartige Optimierungen und Funktionserweiterungen der Gesamtsysteme zu ermöglichen, ist noch wenig erforscht. Dies hat zur Folge, dass aktuell kaum Informationen zu BDS-typischen Fehlern bzw. quantitative Informationen über diese Fehler vorliegen. Zusätzliche Forschungsereignisse über das Fehlerverhalten von Konsumergeräten sowie IoT-Sensorik mit Sicht auf die Zuverlässigkeit kann die modellbasierte Entwicklung und Analyse von BDS deutlich verbessern.

BIBLIOGRAPHIE

[1] V. P. Andelfinger und T. Hänisch, Hrsg., *Industrie 4.0*, 1. Aufl. Wiesbaden: Gabler Verlag, 2017.

[2] „Global Mobile Consumer Survey 2017 - Mobile Evolution“, Deloitte, 2017. Zugegriffen: Mai 01, 2020. [Online]. Verfügbar unter: https://www2.deloitte.com/de/de/pages/technology-media-and-telecommunications/articles/global-mobile-consumer-survey-2017.html.

[3] A. Taivalsaari und T. Mikkonen, „A Roadmap to the Programmable World: Software Challenges in the IoT Era“, *IEEE Softw.*, Bd. 34, Nr. 1, S. 72–80, Jan. 2017.

[4] „Prognose zur Anzahl der vernetzten Geräte im Internet der Dinge (IoT) weltweit in den Jahren 2016 bis 2020“, Gartner, Weltweit, 2017. Zugegriffen: Juni 07, 2020. [Online]. Verfügbar unter: https://de.statista.com/statistik/daten/studie/537093/umfrage/anzahl-der-vernetzten-geraete-im-internet-der-dinge-iot-weltweit/.

[5] P. Marwedel, *Embedded System Design*, 3. Aufl. Basel: Springer International Publishing, 2018.

[6] W. R. Dunn, „Designing safety-critical computer systems“, *IEEE - Comput.*, Bd. 36, Nr. 11, S. 40–46, Nov. 2003.

[7] T. A. Henzinger und J. Sifakis, „The Discipline of Embedded Systems Design“, *IEEE - Comput.*, Bd. 40, Nr. 10, S. 32–40, Okt. 2007.

[8] „Zahl der Todesfälle im Jahr 2019“, Statistisches Bundesamt, Deutschland, 2019. Zugegriffen: Mai 01, 2021. [Online]. Verfügbar unter: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Todesursachen/todesfaelle.html.

[9] „Die 10 häufigsten Todesfälle durch Herz-Kreislauf-Erkrankungen“, Statistisches Bundesamt, Deutschland, 2019. Zugegriffen: Mai 01, 2020. [Online]. Verfügbar unter: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Todesursachen/Tabellen/sterbefaelle-herz-kreislauf-erkrankungen-insgesamt.html.

[10] „Diagnosedaten der Patienten und Patientinnen in Krankenhäusern (einschl. Sterbe- und Stundenfälle)“, Statistisches Bundesamt, Deutschland, Fachserie 12 Reihe 6.2.1, 2016. Zugegriffen: Mai 01, 2020. [Online]. Verfügbar unter: https://www.destatis.de/DE/Themen/Gesellschaft-Umwelt/Gesundheit/Krankenhaeuser/Publikationen/Downloads-Krankenhaeuser/diagnosedaten-krankenhaus-2120621167004.html.

[11] „Eurotransplant - Aktive Warteliste für eine Herztransplantation für das Jahr 2020“, Eurotransplant - Statistics Report Library, Deutschland, 2020. Zugegriffen: Mai 01, 2020. [Online]. Verfügbar unter: https://statistics.eurotransplant.org/index.php?search_type=waiting+list&search_organ=&search_region=All+ET&search_period=2020&search_characteristic=&search_text=&search_collection=.

[12] „Eurotransplant - Durchgeführte Herztransplantationen für das Jahr 2020“, Eurotransplant - Statistics Report Library, Deutschland, 2020. Zugegriffen: Mai 01, 2020. [Online]. Verfügbar unter: https://statistics.eurotransplant.org/index.php?search_type=transplants&search_organ=&search_region=All+ET&search_period=2020&search_characteristic=&search_text=&search_collection=.

[13] D. Willemsen, C. Cordes, B. Bjarnason-Wehrens, *et al.*, „Rehabilitationsstandards für die Anschlussheilbehandlung und allgemeine Rehabilitation von Patienten mit einem Herzunterstützungssystem (VAD – ventricular assist device)“, *Clin. Res. Cardiol. Suppl.*, Bd. 11, Nr. 1, S. 2–49, 2016.

[14] M. Strüber, A. Haverich, A. L. Meyer, *et al.*, „Situation der Herztransplantation und Weiterentwicklung von Kunstherzen“, *Dtsch. Ärztebl.*, Bd. 106, Nr. 28–29, S. 471–477, 2009.

[15] L. Compostella, T. Setzu, T. Bottio, *et al.*, „A practical review for cardiac rehabilitation professionals of continuous-flow left ventricular assist devices: historical and current perspectives.“, *J. Cardiopulm. Rehabil. Prev.*, Bd. 35, Nr. 5, S. 301–311, 2015.

[16] ISO/IEC, „ISO/IEC 9126. Software engineering -- Product quality“, ISO/IEC, 2001.

[17] „Projektbeschreibung Medical care evolution“, ITEA3. Zugegriffen: Juni 06, 2020. [Online]. Verfügbar unter: https://itea3.org/project/medolution.html.

[18] A. Litvina, „Policy-based Management of Medical Devices and Applications“, PhD Thesis, Technische Universität Dortmund, Dortmund, 2018.

[19] A. S. Tanenbaum und M. van Steen, *Verteilte Systeme*, 2. Aufl. München: Pearson, 2007.

[20] G. F. Coulouris, J. Dollimore, T. Kindberg, *et al.*, *Distributed Systems*, 5. Aufl. Boston: Pearson Education, 2012.

[21] I. Foster und C. Kesselman, *The Grid 2*, 2. Aufl. Burlington: Morgan Kaufmann, 2003.

[22] A. Gustavo, F. Casati, H. Kuno, *et al.*, *Web Services*, 1. Aufl. Berlin/Heidelberg: Springer-Verlag, 2004.

[23] M. Chen, S. Gonzalez, A. Vasilakos, *et al.*, „Body Area Networks: A Survey“, *Mob. Netw. Appl.*, Bd. 16, Nr. 2, S. 171–193.

[24] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, *et al.*, „Wireless sensor networks: a survey“, *IEEE Comput. Netw.*, Bd. 38, Nr. 4, S. 393–422.

[25] M. Satyanarayanan, „Pervasive computing: vision and challenges“, *IEEE Pers. Commun.*, Bd. 8, Nr. 4, S. 10–17, 2001.

[26] J.-C. Laprie, „DEPENDABLE COMPUTING AND FAULT TOLERANCE : CONCEPTS AND TERMINOLOGY“, in *Twenty-Fifth International Symposium on Fault-Tolerant Computing, 1995, ' Highlights from Twenty-Five Years'.*, Juni 1995, S. 2–11.

[27] A. Avizienis, J.-C. Laprie, B. Randell, *et al.*, „Basic concepts and taxonomy of dependable and secure computing“, *IEEE Trans. Dependable Secure Comput.*, Bd. 1, Nr. 1, S. 11–33, Jan. 2004.

[28] „DIN EN 50126-2; VDE 0115-103-2:2018-10 Bahnanwendungen - Spezifikation und Nachweis von Zuverlässigkeit, Verfügbarkeit, Instandhaltbarkeit und Sicherheit (RAMS)“, Beuth Verlag, Berlin, 2018.
[29] I. Sommerville, *Software Engineering*, 9. Aufl. Boston: Pearson Education, 2011.
[30] E. Dubrova, *Fault-Tolerant Design*, 1. Aufl. New York: Springer, 2013.
[31] L. L. Pullum, *Software Fault Tolerance Techniques and Implementation*, 1. Aufl. Norwood: Artech House, 2001.
[32] A. Avizienis, „Toward systematic design of fault-tolerant systems“, *IEEE - Comput.*, Bd. 30, Nr. 4, S. 51–58, 1997.
[33] A. Avizienis, „Design diversity: An approach to fault tolerance of design faults“, *Natl. Comput. Conf. Expo. AFIPS 84*, Bd. 1, S. 163–171.
[34] K. Echtle, *Fehlertoleranzverfahren*, 1. Aufl. Berlin/Heidelberg: Springer, 1990.
[35] A. Avizienis, „Fault-Tolerant Systems“, *IEEE Transations Comput.*, Bd. C–25, Nr. 12, S. 1304–1312, 1976.
[36] R. S. Hanmer, *Patterns for Fault Tolerant Software*, 1. Aufl. Hoboken: John Wiley & Sons, 2007.
[37] A. Armoush, „Design Patterns for Safety Critical Embedded Systems“, PhD Thesis, RWTH Aachen Universität, Aachen, 2010.
[38] R. Höhn und S. Höppner, *Das V-Modell XT*, 1. Aufl. Berlin/Heidelberg: Springer, 2008.
[39] „IEEE Standard Glossary of Software Engineering Terminology“, IEEE Std 610.12-1990, 1990.
[40] „INCOSE MBSE“, INCOSE. Zugegriffen: Aug. 12, 2020. [Online]. Verfügbar unter: https://www.incose.org/incose-member-resources/working-groups/transformational/mbse-initiative.
[41] „SYSTEMS ENGINEERING VISION 2020“, International Council on Systems Engineering (INCOSE), 2007.
[42] O. Alt, *Modellbasierte Systementwicklung mit SysML*, 1. Aufl. München: Carl Hanser Verlag GmbH Co KG, 2012.
[43] P. H. Feiler und D. P. Gluch, *Model-Based Engineering with AADL*, 1. Aufl. Reading: Addison-Wesley, 2012.
[44] „ISO/IEC/IEEE 42010 - Systems and software engineering - Architecture description“, ISO/IEC/IEEE, 2011.
[45] The Open Group, *TOGAF® Version 9.1*, 10. Aufl. ’s-Hertogenbosch: Van Haren, 2011.
[46] P. B. Kruchten, „The 4+1 View Model of architecture“, *IEEE Softw.*, Bd. 12, Nr. 6, S. 42–50, 1995.
[47] C. Hofmeister, R. Nord, und D. Soni, *Applied Software Architecture*, 1. Aufl. Reading: Addison-Wesley Professional, 1999.
[48] „About the OMG System Modeling Language Specification Version 1.6“, Object Management Group. Zugegriffen: Aug. 20, 2020. [Online]. Verfügbar unter: https://www.omg.org/spec/SysML/About-SysML/.

[49] „About the Unified Modeling Language Specification Version 2.5.1“, Object Management Group. Zugegriffen: Aug. 20, 2020. [Online]. Verfügbar unter: https://www.omg.org/spec/UML/About-UML/.

[50] F. Kordon, J. Hugues, A. Canals, *et al.*, *Embedded Systems - Analysis and Modeling with SysML, UML and AADL*, 1. Aufl. Hoboken: John Wiley & Sons, 2013.

[51] S. Friedenthal, A. Moore, und R. Steiner, *A Practical Guide to SysML*. Boston: Morgan Kaufmann, 2011.

[52] „Standard AS5506C - Architecture Analysis & Design Language (AADL) AS5506C“, SAE International, Warrendale, 2017. [Online]. Verfügbar unter: https://www.sae.org/standards/content/as5506c/.

[53] „AS5506C Standard: Architecture Analysis & Design Language (AADL)“, SAE International. Zugegriffen: Okt. 09, 2020. [Online]. Verfügbar unter: https://www.sae.org/standards/content/as5506c/.

[54] J. Delange, *AADL in practice - design and validate the architecture of critical systems*, 1. Aufl. United States of America: Reblochon Development Company, 2017.

[55] „Standard AS5506/3 - Architecture Analysis and Design Language (AADL) Annex D: Behavior Model Annex“, SAE International, Warrendale, 2017. [Online]. Verfügbar unter: https://www.sae.org/standards/content/as5506/3/.

[56] „Welcome to OSATE — OSATE 2.7.1 documentation“, Carnegie Mellon University. Zugegriffen: Mai 05, 2020. [Online]. Verfügbar unter: https://osate.org/.

[57] „Standard AS5506/1A - SAE Architecture Analysis and Design Language (AADL) Annex Volume 1: Annex A: ARINC653 Annex, Annex C: Code Generation Annex, Annex E: Error Model Annex“, SAE International, Warrendale, 2015. [Online]. Verfügbar unter: https://www.sae.org/standards/content/as5506/1a/.

[58] S. Procter und P. Feiler, „The AADL Error Library: An Operationalized Taxonomy of System Errors“, *ACM SIGAda Ada Lett.*, Bd. 39, Nr. 1, S. 63–70, 2020.

[59] J. Delange, P. Feiler, J. Hudak, *et al.*, „Architecture Fault Modeling with the AADL Error-Model Annex, Version 2“, Software Engineering Institute (SEI) Carnegie Mellon University, Pittsburgh, 2016.

[60] „Standard - MIL-STD-882E System Safety“, Military Departments and Defense Agencies within the Department of Defense (DoD), Washington, D.C., 2012. [Online]. Verfügbar unter: https://www.dau.edu/cop/armyesoh/DAU%20Sponsored%20Documents/MIL-STD-882E.pdf.

[61] „Standard ARP4761 - Guidelines and Methods for Conducting the Safety Assessment Process on Civil Airborne Systems and Equipment“, SAE International, Warrendale, 1996. [Online]. Verfügbar unter: https://www.sae.org/standards/content/arp4761/.

[62] C. A. Ericson, *Hazard Analysis Techniques for System Safety*, 2. Aufl. Hoboken: John Wiley & Sons, 2005.

[63] M. Rausand, *Reliability of Safety-Critical Systems*, 1. Aufl. Hoboken: John Wiley & Sons, 2014.

[64] J. Delange, P. Feiler, D. P. Gluch, *et al.*, „AADL Fault Modeling and Analysis Within an ARP4761 Safety Assessment“, Software Engineering Institute (SEI) Carnegie Mellon University, Pittsburgh, 2014.

[65] P. Feiler und J. Delange, „Automated Fault Tree Analysis from AADL Models“, *ACM SIGAda Ada Lett.*, Bd. 36, Nr. 2, S. 39–46, 2017.

[66] P. Feiler und J. Hansson, „Flow Latency Analysis with the Architecture Analysis and Design Language (AADL)“, Software Engineering Institute (SEI) Carnegie Mellon University, Pittsburgh, 2007.

[67] W. E. Vesely, N. H. Roberts, F. F. Goldberg, *et al.*, *Fault Tree Handbook*. Washington, D.C.: U.S. Government Printing Office, 1981.

[68] J. Delange und P. H. Feiler, „Incremental latency analysis of heterogeneous cyber-physical systems.“, *React. 2014 - 3rd IEEE Int. Workshop Real-Time Distrib. Comput. Emerg. Appl.*, S. 21–27, 2014.

[69] K.-H. Chen, „Optimization and analysis for dependable application software on unreliable hardware platforms“, PhD Thesis, Technische Universität Dortmund, Dortmund, 2019.

[70] C. A. Boano, K. Römer, R. Bloem, *et al.*, „Dependability for the Internet of Things—from dependable networking in harsh environments to a holistic view on dependability“, *E Elektrotechnik Informationstechnik*, Bd. 133, Nr. 7, S. 304–309, 2016.

[71] Y. Jasemian und L. Arendt-Nielsen, „Evaluation of a realtime, remote monitoring telemedicine system using the Bluetooth protocol and a mobile phone network“, *J. Telemed. Telecare*, Bd. 11, Nr. 5, S. 256–260, 2005.

[72] D. Macedo, L. A. Guedes, und I. Silva, „A dependability evaluation for Internet of Things incorporating redundancy aspects“, in *Proceedings of the 11th IEEE International Conference on Networking, Sensing and Control*, Apr. 2014, S. 417–422.

[73] R. John, K. Mantz, P. Eckman, *et al.*, „Aortic valve pathophysiology during left ventricular assist device support“, *J. Heart Lung Transplant. Off. Publ. Int. Soc. Heart Transplant.*, Bd. 29, Nr. 12, S. 1321–1329, Dez. 2010.

[74] S. V. Deo, V. Sharma, Y. H. Cho, *et al.*, „De novo aortic insufficiency during long-term support on a left ventricular assist device: a systematic review and meta-analysis“, *ASAIO J.*, Bd. 60, Nr. 2, S. 183–188, Apr. 2014.

[75] E. Tuzun, M. Rutten, M. Dat, *et al.*, „Continuous-flow cardiac assistance: effects on aortic valve function in a mock loop“, *J. Surg. Res.*, Bd. 171, Nr. 2, S. 443–447, Dez. 2011.

[76] N. Dranishnikov, „Linksventrikuläre Kreislaufunterstützungssysteme und ihre Wechselwirkungen mit dem menschlichen Körper“, PhD Thesis, FU Berlin, Berlin, 2014.

[77] E. Tuzun, I. D. Gregoric, J. L. Conger, *et al.*, „The effect of intermittent low speed mode upon aortic valve opening in calves supported with a Jarvik 2000 axial flow device“, *ASAIO J.*, Bd. 51, Nr. 2, S. 139–143, Apr. 2005.

[78] O. D. Lara und M. A. Labrador, „A Survey on Human Activity Recognition using Wearable Sensors“, *IEEE Commun. Surv. Tutor.*, Bd. 15, Nr. 3, S. 1192–1209, 2012.

[79] K. Chen, D. Zhang, L. Yao, *et al.*, „Deep Learning for Sensor-based Human Activity Recognition: Overview, Challenges and Opportunities“, *J ACM*, Bd. 37, Nr. 4, 2018.

[80] M. A. Ayu, S. A. Ismail, T. Mantoro, *et al.*, „Real-time activity recognition in mobile phones based on its accelerometer data“, in *2016 International Conference on Informatics and Computing (ICIC)*, Okt. 2016, S. 292–297.

[81] R.-A. Voicu, C. Dobre, L. Bajenaru, *et al.*, „Human Physical Activity Recognition Using Smartphone Sensors“, *Sensors*, Bd. 19, Nr. 3, S. 1–18, 2019.

[82] B. Larson, J. Hatcliff, K. Fowler, *et al.*, „Illustrating the AADL error modeling annex (v.2) using a simple safety-critical medical device“, in *Proceedings of the 2013 ACM SIGAda annual conference on High integrity language technology*, Nov. 2013, S. 65–84.

[83] J. Delange, „Architecture Analysis with AADL - The Speed Regulation Case-Study“, Software Engineering Institute (SEI) Carnegie Mellon University, Pittsburgh, 2014.

[84] M. Muñoz Fernández, „Model-based systems engineering with the Architecture Analysis and Design Language (AADL) applied to NASA mission operations“, gehalten auf der SpaceOps 2014 13th International Conference on Space Operations, Mai 2014.

[85] P. H. Feiler, D. P. Gluch, und K. Woodham, „Case Study: Model-Based Analysis of the Mission Data System Reference Architecture“, Software Engineering Institute (SEI) Carnegie Mellon University, Pittsburgh, 2010.

[86] S. H. Hashemi, „Fault tolerant digital computer system having two processors which periodically alternate as master and slave“, US5491787A, Feb. 13, 1996.

[87] A. Faller und M. Schünke, *Der Körper des Menschen*, 14. Aufl. Stuttgart: Thieme, 2004.

[88] H. Lippert, *Lehrbuch Anatomie*, 8. Aufl. München: Elsevier, Urban & Fischer, 2011.

[89] Prof. Dr. med. Andreas van de Loo, „Cardio-Guid“, *Herzinsuffizienz | Ursachen, Symptome und Behandlung*. https://www.cardio-guide.com/erkrankung/herzinsuffizienz/ (zugegriffen Sep. 16, 2020).

[90] A. Link und M. Böhm, „Akute Herzinsuffizienz“, in *DGIM Innere Medizin*, H. Lehnert, S. M. Schellong, J. Mössner, C. C. Sieber, W. Swoboda, A. Neubauer, B. Kemkes-Matthes, M. P. Manns, J. Rupp, G. Hasenfuß, J. Floege, M. Hallek, T. Welte, M. Lerch, E. Märker-Hermann, und L. S. Weilemann, Hrsg. Berlin, Heidelberg: Springer, 2015, S. 1–11.

[91] G. Hasenfuß und S. D. Anker, „Systolische Herzinsuffizienz“, in *DGIM Innere Medizin*, H. Lehnert, S. M. Schellong, J. Mössner, C. C. Sieber, W. Swoboda, A. Neubauer, B. Kemkes-Matthes, M. P. Manns, J. Rupp, G. Hasenfuß, J. Floege, M. Hallek, T. Welte, M. Lerch, E. Märker-Hermann, und L. S. Weilemann, Hrsg. Berlin, Heidelberg: Springer, 2015, S. 1–13.

[92] Klinik für Thorax- und Kardiovaskularchirurgie, „Herz- und Diabeteszentrum NRW“, *Herzunterstützungssysteme und Kunstherzen*. https://www.hdz-nrw.de/kliniken-institute/kliniken/klinik-fuer-thorax-und-kardiovaskularchirurgie/behandlung/transplantationschirurgie-und-

herzunterstuetzung/vad-systeme/herzersatzsysteme.html (zugegriffen Okt. 13, 2020).

[93] Medtronic, „Ventricular Assist Devices“, *Product information HVAD™ System Advanced Heart Failure Management*. https://europe.medtronic.com/xd-en/healthcare-professionals/products/cardiac-rhythm/ventricular-assist-devices/heartware-hvad-system.html (zugegriffen Aug. 23, 2020).

[94] Heartware.com, „MyLVAD“, *Medtronic - HVAD*. https://www.mylvad.com/patients-caregivers/learn-about-lvads/lvad-technology/medtronic-hvadr (zugegriffen Aug. 24, 2020).

[95] J. S. Hanke, S. V. Rojas, M. Avsar, *et al.*, „HeartWare left ventricular assist device for the treatment of advanced heart failure“, *Future Cardiol.*, Bd. 12, Nr. 1, S. 17–26, Nov. 2015.

[96] A. L. Meyer, J. Fischer, und J. Garbade, „Pumpenthrombosen“, *Z. Für Herz-Thorax- Gefäßchirurgie*, Bd. 31, Nr. 4, S. 276–282, Aug. 2017.

[97] Z. Andreas, M. Spencer J., V. Rochus K., *et al.*, „Late-onset driveline infections: the Achilles' heel of prolonged left ventricular assist device support“, *Ann. Thorac. Surg.*, Bd. 84, Nr. 2, S. 515–520, Aug. 2007.

[98] N. Reiss, M. Altesellmeier, S. Mommertz, *et al.*, „Hämodynamik und körperliche Belastbarkeit bei Patienten mit Linksherzunterstützungssystem“, *Herz*, Bd. 41, Nr. 6, S. 507–513, Sep. 2016.

[99] M. L. Goodwin, C. M. Bobba, N. A. Mokadam, *et al.*, „Continuous-Flow Left Ventricular Assist Devices and the Aortic Valve: Interactions, Issues, and Surgical Therapy“, *Curr. Heart Fail. Rep.*, Bd. 17, Nr. 4, S. 97–105, Aug. 2020.

[100] R. J. Kalathiya, J. Grinstein, N. Uriel, *et al.*, „Percutaneous Transcatheter Therapies for the Management of Left Ventricular Assist Device Complications“, *J. Invasive Cardiol.*, Bd. 29, Nr. 5, S. 151–162, Mai 2017.

[101] A. Pick, „Aortic Valve Disease“, *Aortic Stenosis: Symptoms, Diagnosis & Treatment*. https://www.heart-valve-surgery.com/aortic-stenosis-valve-heart-narrowing.php (zugegriffen Aug. 23, 2020).

[102] W. Denson, G. Chandler, W. Crowell, *et al.*, „Nonelectronic Parts Reliability Data 1991“, RELIABILITY ANALYSIS CENTER, Mai 1991.

[103] J. Catania, „Soft errors in electronic memory-a white paper“, Tezzaron Semiconductor, Naperville, 2004.

[104] F. A. Silva, A. C. S. A. Domingues, und T. R. M. B. Silva, „Discovering Mobile Application Usage Patterns from a Large-Scale Dataset“, *ACM Trans. Knowl. Discov. Data*, Bd. 12, Nr. 5, S. 59:1-59:36, Juni 2018.

[105] F. Smolinski, „Experimentelle Zuverlässigkeitsbewertung einer Smartphone-basierten Kunstherzsteuerung zur Herzklappen-Spülung“, Masterarbeit, Technische Universität Dortmund, Dortmund, 2019.

[106] A. Mezzani, M. Pistono, U. Corrà, *et al.*, „Systemic perfusion at peak incremental exercise in left ventricular assist device recipients: Partitioning pump and native left ventricle relative contribution“, *IJC Heart Vessels*, Bd. 4, S. 40–45, Sep. 2014.

[107] D. R. Bassett, J. A. Vachon, A. O. Kirkland, *et al.*, „Energy cost of stair climbing and descending on the college alumnus questionnaire“, *Med. Sci. Sports Exerc.*, Bd. 29, Nr. 9, S. 1250–1254, Sep. 1997.

[108] K. C. Teh und A. R. Aziz, „Heart rate, oxygen uptake, and energy cost of ascending and descending the stairs“, *Med. Sci. Sports Exerc.*, Bd. 34, Nr. 4, S. 695–699, Apr. 2002.