

Architektur für verteilte, fehlertolerante Sensor-Aktor-Systeme

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke Universität Magdeburg



vorgelegt von: Dipl. Ing. Sebastian Ernst Zug
geboren am 18. Mai 1976 in Hoyerswerda

Erstgutachter: Prof. Dr. rer. nat. Jörg Kaiser
Zweitgutachter: Prof. Dr.-Ing. Ulrich Berger
Drittgutachter: Privatdoz. Dr. techn. Wilfried Elmenreich

Magdeburg, den 16.12.2011

Kurzfassung

Sensor-Aktor-Systeme bestehen in heutigen Anwendungen zumeist aus einer fest gefügten Hardwarestruktur: Die zur Designzeit definierte Anzahl von Sensoren liefert eine Abbildung der Umgebung, die von einem (Mikro-)Controller analysiert wird, um ausgehend vom Resultat dieser Verarbeitung einen oder mehrere Aktoren anzusteuern. In Anbetracht der wachsenden Zahl von eingebetteten Informationsquellen und Sensornetzen (Bewegungsmelder, Überwachungskameras, automatische Türen, andere mobile Systeme usw.) sollte die Wahrnehmung auf deren Messdaten ausgeweitet werden. Ausgehend davon untersucht diese Arbeit die Voraussetzungen einer adaptiven Konfiguration, die einen flexiblen Datenaustausch sicherstellt. Mit der Ausnutzung aller relevanten Informationen lassen sich für die Präzision und Sicherheit erhebliche Gewinne erzielen.

Für einen solchen nahtlosen Datenaustausch ergibt sich eine Reihe von Anforderungen. Zunächst bedarf es einer abstrakten Beschreibung der Sensoren und ihrer Messungen, die alle für die Verarbeitung entscheidenden Informationen bereitstellt. Dies können Angaben über die Sensorkeule, die physikalische Einheit, die Position des Sensors usw. sein. Bestehende Ansätze zur (Selbst-)Beschreibung decken den Umfang der für die adaptive Verarbeitung notwendigen Informationen nicht ab, sodass eine Erweiterung dieser Konzepte nötig ist.

Weiterhin wird ein Bewertungsschema für die Validierung der Messdaten vorgeschlagen, da die Güte der Ausgaben eines Sensors ausgehend vom Wirkprinzip, den Fehlerquellen und den nachgeordneten Detektionsmechanismen stark schwanken kann. Es kombiniert eine statische Validitätsaussage über den einzelnen Sensorknoten mit einer dynamischen Bewertung des einzelnen Datensatzes, wobei diese Aussage das Ergebnis der Fehlerdetektion widerspiegelt. Mit diesem mehrschichtigen Ansatz lassen sich unterschiedlichste Messwerte während der Verarbeitung objektiv beurteilen.

Nicht alle verfügbaren Informationen sind im Sinne einer Aufgabenstellung relevant, das heißt, sie sind möglicherweise zu alt, unpräzise, invalide oder betreffen einen Überwachungsbereich außerhalb des Fokus der Aufgabe. Entsprechend wurde eine Selektionsstrategie konzipiert, die der eigentlichen Verarbeitung vorangestellt ist und die Datenerfassung koordiniert. Im Weiteren diskutiert die Arbeit Fragen der adaptiven Fusion, die von einer veränderlichen Zahl von relevanten Sensoren und Datensätzen ausgeht.

Die genannten Konzepte werden in einer generischen Architektur integriert, die deren Integration bei der Programmierung von Sensor-Aktor-Netzen sicherstellt. In drei Testszenarien werden sowohl die Ansätze dieser Arbeit als auch die für verschiedene domänenspezifische Sprachen entwickelten Frameworks evaluiert.

Danksagung

Meinem Doktorvater Herrn Prof. Dr. Jörg Kaiser danke ich dafür, dass er diese Arbeit ermöglicht und mit interdisziplinären Anregungen begleitet hat. Neben den anregenden und vielfältigen Diskussionen gab er richtungsgebende Hinweise, die bei der Strukturierung dieser Ausarbeitung wichtige Hilfestellungen waren.

Herr Prof. Dr. Berger hat, insbesondere durch den mir ermöglichten Aufenthalt an der KU Leuven, mein Interesse an Fragen der Robotik mitgeprägt. Entsprechend freut es mich sehr, dass er bereit war, die Begutachtung dieser Promotionsschrift zu übernehmen. Der gleiche Dank gilt Herrn Privatdoz. Dr. Wilfried Elmenreich, der mit seinen Arbeiten im Bereich der intelligenten Sensoren wichtige Impulse für das Konzept setzte und ebenfalls als Gutachter gewonnen werden konnte.

Im Hinblick auf die Kollegen der Arbeitsgruppe für Betriebssysteme und Eingebettete Systeme gilt mein besonderer Dank Herrn André Dietrich, der gleichzeitig als engagierter Unterstützer und Kritiker wirkte. Die gemeinsame Arbeit an verschiedenen Projekten hat die Konzepte dieser Dissertation entscheidend geprägt. Aus der Zusammenarbeit mit Herrn Michael Schulze, der über den flexiblen Datenaustausch in verteilten Systemen forschte, sind viele Ansätze dieser Arbeit entstanden. Zudem hat er geduldig dazu beigetragen, mein Verständnis von der Informatik weiterzuentwickeln. Die Diskussionen mit Herrn Thomas Kiebel und die erkenntnisreichen Ratschläge von Herrn Christoph Steup haben die Arbeit insbesondere im Implementierungsteil wesentlich bereichert. Mein Dank gilt gleichermaßen den Kollegen der Arbeitsgruppe Kommunikation und Netze Herrn Timo Lindhorst, Herrn Felix Penzlin und Herrn Georg Lucas. Einem ehemaligen Kollegen, Herrn Stefan Sokoll, danke ich für sein kontinuierlich kritisches Hinterfragen meiner Ideen und Modelle. Ebenso hat Herr Christoph Walter vom Fraunhofer-Institut für Fabrikbetrieb und -automatisierung mit seinem industrie- und anwendungsorientierten Blick wichtige Hinweise gegeben.

Ein besonderer Dank gilt den Studenten, die mit ihren Laborpraktika, Studien- und Diplomarbeiten einen wichtigen Grundstock für diese Arbeit gelegt haben. Insbesondere die Zusammenarbeit mit Herrn Tino Brade war durch einen gemeinsamen, kontinuierlichen Erkenntnisgewinn geprägt, der in der Umsetzung der Ansätze dieser Arbeit in Matlab/Simulink mündete. Des Weiteren trug Herr Marc Schappeit durch aufopferungsvolle Nachtschichten dazu bei, dass das praktische Evaluationsszenario umgesetzt werden konnte.

Im Hinblick auf administrative Aufgaben haben mich Frau Petra Duckstein und Herr Jürgen Lehmann stets freundlich unterstützt. Den vielen Korrekturlesern und dabei

insbesondere Frau Nicole Zug-Schmitz und Frau Doreen Kuntze bin ich für ihre Geduld dankbar.

Ein wenig rückblickend ist der Dank an Herrn Thomas Stepputat. Er hat mit unseren studentischen Roboterentwicklungen meine Begeisterung für die wunderbare Welt der autonomen Systeme geweckt und damit einen Grundstein für diese Arbeit gelegt.

Meinen Eltern Liane und Gerhard Zug sowie meiner Großmutter Ursula Zerna danke ich dafür, dass sie mich auf dem Weg, der zu dieser Arbeit führte, begleitet, kontinuierlich ermutigt und in allen Fragen unterstützt haben. Die vielen Diskussion mit meinem Vater und meinem Bruder Constantin Zug halfen die zentralen ingenieurwissenschaftlichen Fragestellungen dieser Arbeit herauszuarbeiten.

Mein letzter und wichtigster Dank gilt meiner Frau Kirsten Jandt und meiner Tochter Anna-Dorothea für ihr Verständnis, ihre Geduld und ihre Liebe.

Inhaltsverzeichnis

1	Einleitung	1
1.1	Motivation	1
1.2	Struktur der Arbeit	10
2	Klassifikation der Anforderungen	11
2.1	Architekturbezogene Erfordernisse	11
2.2	Funktionsbezogene Erfordernisse	14
3	Stand der Forschung	19
3.1	Begriffsbildung	19
3.1.1	Sensor	19
3.1.2	Transducer	21
3.1.3	Integrierter Sensor	22
3.1.4	Smart Sensor	24
3.1.5	Smart Transducer	26
3.1.6	Einordnung des <i>Smart-X</i>	27
3.2	Programmierung von Smart Devices	28
3.2.1	Standards und Spezifikationen	28
	IEEE-1451	29
	OMG STIS	31
3.2.2	Programmiermodelle und ihre Implementierungen	33
	Abstract Sensor	33
	Virtual Sensor	34
	Sentient Object	36
	Logical Sensor	40
	Fusion Channel	42
3.2.3	Zusammenfassung	44
3.3	Beschreibungskonzepte für Sensordaten	46
3.3.1	SensorML	46
3.3.2	OpenRave	49
3.3.3	AutomationML	50
3.3.4	Netzwerkspezifische Beschreibungskonzepte	51
3.3.5	TinyDB	51

3.3.6	ROS	52
3.3.7	Zusammenfassung	53
3.4	Fehlermodelle und Detektionsverfahren	54
3.4.1	Methoden zur Fehlerdetektion	54
3.4.2	Fehlermodelle	58
3.4.3	Fehlersemantik	59
3.4.4	Zusammenfassung	59
3.5	Datenselektion und adaptive Verarbeitung	60
3.5.1	Selektion	60
	Sensorselektion zur Compilezeit	61
	Sensorselektion zur Laufzeit	62
3.5.2	Fusion	64
	Fusionsarchitekturen	65
	Adaptive Fusionsansätze	66
3.5.3	Zusammenfassung	67
4	Konzepte des <i>MOSAIC</i>-Frameworks	69
4.1	Basisabstraktionen	69
4.1.1	Smart-X	69
4.1.2	Progammiermodell	74
4.2	Fehlerdetektion und Datenvalidierung	75
4.2.1	Herkunft von Fehlern	76
4.2.2	Fehlerabstraktionen	80
4.2.3	Fehlerdetektion	82
4.2.4	Sensor- und Messdatenvalidierung	83
	Statische Systemvalidität	85
	Dynamische Datensatzvalidität	90
4.3	Datenbeschreibung	91
4.3.1	XML-Darstellung	93
	Definition der physikalischen Einheiten	95
	Abbildungsbeschreibung	96
4.3.2	Beschreibung der Attribute im Datenblatt	97
	Fehlermodelle	98
	Überwachungsbereiche	98
	Interaktionsmodelle und Zeitverhalten	100
	Datenherkunft	102
	Detektionsmethoden	104
	Datenformat	105
	Lokalisationsbezüge	105
4.3.3	Einbindung der Datenblätter in <i>MOSAIC</i>	106

4.4	Adaptive Verarbeitung	108
4.4.1	Selektion	108
	Kriterienkatalog	108
	Ausführungsumgebung	112
	Mechanismen	114
	Selektionsfolge in <i>MOSAIC</i>	115
	Parameterdefinition	116
4.4.2	Adaptive Fusion	118
4.5	<i>Smart-X</i> -Architektur	120
5	Implementation	125
5.1	Übergreifende Vorgaben	125
5.1.1	Werkzeugkette	125
5.1.2	Kommunikationsmiddleware	128
5.2	Umsetzung in Simulink	129
5.2.1	Matlab/Simulink	129
5.2.2	Werkzeugkette in Simulink	130
	Spezifikationsphase	130
	Implementierungsphase	133
5.3	Python	136
5.3.1	Basisstruktur	136
5.3.2	Eingangsschnittstellen	139
5.3.3	Verarbeitung	143
5.3.4	Beispiel	145
5.4	Zusammenfassung	148
6	Evaluation	151
6.1	Einleitung	151
6.1.1	Einordnung	151
6.1.2	Verwendete Sensorik	153
6.2	Simulation der Fehlerdetektion	155
6.2.1	Szenariobeschreibung	155
6.2.2	Ergebnisse	159
6.2.3	Zusammenfassung	161
6.3	Entwurf eines roboterzentrierten <i>Smart-X</i> -Lokalisationssystems	161
6.3.1	Ansatz und Konfiguration	161
6.3.2	Evaluation des Distanzsensors	162
6.3.3	Implementierung	164
6.3.4	Diskussion	166
6.4	Adaptive Lokalisation	167
6.4.1	Aufbau	167

6.4.2	Ergebnisse	169
6.4.3	Zusammenfassung	172
7	Zusammenfassung und Ausblick	173
7.1	Beiträge der Arbeit	173
7.2	Zukünftige Arbeiten	176

Abbildungsverzeichnis

1.1	Traditioneller Regelkreis	2
1.2	Motivationsszenario mit verteilter Umgebungserfassung	5
1.3	Implementierung eines Regelkreises mit SANs	9
3.1	Formen integrierter Sensoren	23
3.2	Was bedeutet „Smart Sensor“?	24
3.3	Abstraktionsgrad der verschiedenen Sensorbegriffe	27
3.4	Interfaces eines Servies nach [EPS04]	32
3.5	Marzullo's Fault Tolerant Sensor Fusion nach [Mar90]	34
3.6	Virtual Sensors nach [Bos+07]	36
3.7	Grundstruktur eines Sentient Objects nach [Sch11b]	37
3.8	Datenfluss im Sentient Object (nach [Bie04])	38
3.9	Erweiterter Logischer Sensor [HD98]	41
3.10	Fusion Channel (nach [Aga+02])	43
3.11	Fehlerdetektionsansätze	55
3.12	Veränderlicher Einfluss redundanter Senoren nach [NL80]	60
3.13	JDL Ebenen der Datenfusion nach [XS02]	65
4.1	Erweiterung des statischen Sensorkonzeptes	70
4.2	Abstrakte Darstellung eines <i>Smart-X</i>	72
4.3	Beispielhaftes Szenario aus mehreren SANs	73
4.4	Kategorien von Sensorstörungen	76
4.5	Fehlertypen im Hinblick auf die notwendigen Signalverarbeitungsschritte	78
4.6	Detektierbarkeit von Ausreißern	87
4.7	Beispielhafte Umsetzung einer fehlertoleranten Kollisionsvermeidung	89
4.8	XML-Schema der Einheitencodierung	96
4.9	XML-Schema zur Darstellung von Input/Output Relationen	97
4.10	XML-Schema der Sensorfehler	98
4.11	Sensorkategorien in Bezug auf Lokalisierung	100
4.12	XML-Schema der Darstellung von Sensorrohdaten	101
4.13	Pyramidenstumpfmodell für Überwachungsbereiche	102
4.14	Dateninzeptszenarios in adaptiven Fusionsanwendungen	103
4.15	XML-Schema zur Zuordnung von Fehlerdetektionsmethoden	104
4.16	XML-Schema der Darstellung von Sensorrohdaten	106

4.17	Selektionssystematik für Einheiten	109
4.18	Relevanz von Datensätzen anhand der physikalischen Einheit	113
4.19	XML-Schema der Kriteriendefinition für die individuelle Selektion	117
4.20	Architektur eines <i>Smart-X</i>	122
5.1	Phasen der Anwendungsentwicklung in SardaS	126
5.2	Struktur eines Spezifikationsmodells in Simulink	131
5.3	Ablauf der Spezifikationsphase in Simulink	132
5.4	Struktur eines <i>Smart-X</i> -Modells in Simulink	134
5.5	Ablauf der Implementierungsphase in Simulink	135
5.6	Klassendiagramm eines <i>Smart-X</i> -Komponente	137
5.7	Klassenhierarchie der Eingangsschnittstellen	140
6.1	RoboCup Junior Spielsituation	152
6.2	Illustration des Simulationsszenarios	156
6.3	Ultraschall-Distanzmessungen mit stochastischen Störungen	156
6.4	<i>Smart-X</i> -Ultraschall Sensor mit rückgekoppelter Validitätsprüfung	157
6.5	Vermessung der Position des Roboters mit fixen Umgebungssensoren	159
6.6	Auswirkung der Rückkopplung von Fusionsdaten	160
6.7	Konfiguration eines roboterlokalen Lokalisationssystems	162
6.8	Rauschverhalten des GP2Y0A02YK	163
6.9	Winkelabhängigkeit der Distanzmessungen des GP2Y0A02YK	163
6.10	MEDUSA Lokalisationssystem	164
6.11	MEDUSA Implementierung in Simulink	165
6.12	Verteilung der Standardabweichung der Positionsschätzung	166
6.13	Roboter und Versuchsaufbau der adaptiven Positionsbestimmung	167
6.14	Sensorverteilung in der Umgebung	169
6.15	Statischer Roboter in instrumentierter Umgebung	170
6.16	Adaptiver Roboter in instrumentierter Umgebung	170

Tabellenverzeichnis

3.1	Einheitenkodierung nach IEEE 1451	31
3.2	Merkmale vergleichbarer Programmiermodelle	45
3.3	Modelle in SensorML	47
3.4	Mögliche Sensorselektionskriterien nach [GZ04, DS04]	63
4.1	Fehlerkategorisierung für Sensoren	79
4.2	Fehlerdetektionsmethoden für die Fehlerkategorien	84
4.3	Adaptierter VDA 4.2 Standard zur Klassifikation von Sensorfehlern	86
4.4	Fehlerklassifikation des GP2D120-Distanzsensors	88
4.5	Herkunft der Attribute im Datenblattsystem von <i>MOSAIC</i>	94
4.6	Selektionsstufen innerhalb einer <i>Smart-X</i> -Komponente	115
5.1	Unterstützte Hardware der <i>MOSAIC</i> Implementierungen	149
6.1	Sensoren des Evaluationsszenarios	153
6.2	Statische Fehlerbewertung der Sensoren des Evaluationsszenarios	154

Quellcodeverzeichnis

3.1	Definition eines Spannungswertes in CODES	39
3.2	Beschreibung eines Laserscanners in OpenRave	50
3.3	Definitionsdatei der Standard-Laserscanner-Message in ROS	53
4.1	Datenblattauszug GP2D120-Sensor - Sensorfehler	99
4.2	Datenblattauszug GP2D120-Sensor - Erfassungsbereiches	101
4.3	Datenblattauszug GP2D120-Sensor - Detektionsmethoden	105
5.1	Definition einer passiven Netzwerkschnittstelle	141
5.2	Definition eines Kriteriums für die individuelle Selektion	141
5.3	Definition eines passiven Netzwerkinterfaces	141
5.4	Konfiguration eines aktiven Netzwerkinterfaces	142
5.5	Beispielhafte Definition eines physischen Interfaces	142
5.6	Implementierung der Verarbeitungsfunktionalität als Dekoratorokette . . .	144
5.7	Implementierung eines adaptiven <i>Smart-X</i> zur Erfassung der Position . .	147

1 Einleitung

1.1 Motivation

Autonome oder teil-autonome eingebettete Systeme verbreiten sich in unserem täglichen Leben in immer stärkerem Maße. Die Applikationen reichen von immer ausgefeilteren Komfortfunktionen wie automatischen Beleuchtungsanlagen oder Aufzugsteuerungen über den gesamten Bereich der modernen Verfahrens- und Automatisierungstechnik bis zu hochkomplexen verteilten Anwendungen wie Fabrik- oder Verkehrssteuerungen. Derartige Systeme kombinieren Komponenten zur Überwachung eines Umgebungsprozesses mittels geeigneter Erfassungsmechanismen, zur Verarbeitung dieser Werte sowie den darauf aufbauenden Eingriff in bestimmte Zustände des observierten Systems über eine Aktorik. Die komplette Abfolge aus Wahrnehmung, Verarbeitung und Interaktion mit einer entsprechenden Rückkopplung repräsentiert einen klassischen Regelkreis. Für den Fall einer fehlenden Rückkopplung bezeichnet man ein solches System als Steuerung. In beiden Fällen, der Regelung und Steuerung, ist erst mit dem Zusammenspiel mechanischer und elektrischer/elektronischer Komponenten, die großer Bandbreite der unser tägliches Leben berührenden Anwendungen möglich. Seit mehreren Jahrzehnten werden solche Anwendungen flächendeckend nicht mehr über analoge Technik, sondern zeitgemäß mittels Mikrocontrollern in unterschiedlicher Ausprägung umgesetzt. Diese Kombination aus Elektrotechnik, Maschinenbau, Regelungstechnik und Informatik macht Sensor-Aktor-Systeme zu einer interdisziplinären Herausforderung.

In gegenwärtigen Implementierungen werden die Sensor-Aktor-Systeme, wie in Abbildung 1.1 dargestellt, als statische Kompositionen der genannten Bestandteile entworfen. Aufgabenspezifische Werte des Systems y werden durch verschiedene Sensoren erfasst und als Messwerte y_n fusioniert, an den Regler übergeben, der daraus ein Steuersignal u für den Aktor bestimmt. Dieser bewirkt eine Veränderung des Systems in der Weise, dass ein Sollwert für eine oder mehrere Systemgrößen erreicht wird. Die Pfeile in der Darstellung illustrieren entsprechende Kommunikations- und Interaktionsmechanismen zwischen den einzelnen Elementen, die zwischen Sensoren und Controller zum Beispiel als Feldbus, zwischen Aktor, System und Sensoren als physikalische Wechselwirkung zu interpretieren sind. Für den Entwurf einer solchen Applikation wird in einem zumeist iterativen Verfahren zunächst eine passende Zahl und die Art der Sensoren/Aktoren in Abhängigkeit von der Dynamik des Systems festgelegt. Davon ausgehend wird geprüft, ob mit dieser Konfiguration bestimmte Anforderungen zur Regelgüte, Geschwindigkeit, Abweichung zum Sollwert usw. erfüllt werden können. Für diese Auslegung existieren

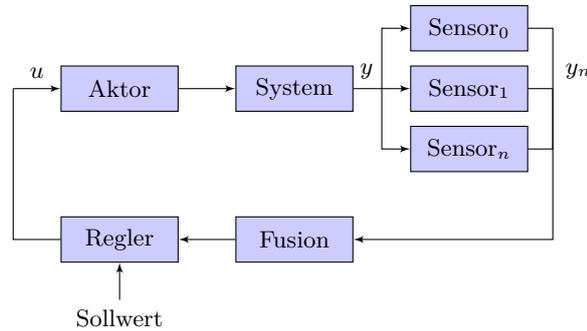


Abbildung 1.1: Übliches Sensor-Aktor-System in einem geschlossenen Regelkreis

etablierte Verfahren der Automatisierungs- und Regelungstechnik, die auf einer theoretischen oder aber praktischen Untersuchung der Applikation basieren. Der Vorteil dieser fest konfigurierten Systeme liegt in der Möglichkeit einer sehr zielgenauen Optimierung der entsprechenden Algorithmen, Kommunikationsparameter, Regelverstärkungen etc. im Hinblick auf eine oder mehrere Zielgrößen, unter anderem die Gesamtkosten.

Sofern eine Applikation nicht auf einem zentralen Knoten umgesetzt wird, spricht man von einem verteilten Sensor-Aktor-System. In diesem Fall sind die für zeitkritische Anwendungen relevanten (nicht-)deterministischen Verzögerungen beim Austausch der Nachrichten zu berücksichtigen. Im schlimmsten Fall kann eine Blockade des Mediums oder ein Fehler das vollständige Ausbleiben von Messdaten oder Steuersignalen bewirken. Der Einfluss eines solchen verzögerten Austausches auf eine konkrete Applikation hängt dabei von deren Dynamik ab. In Anwendungen mit einer hohen Änderungsgeschwindigkeit sind die Störungen der Kommunikation entsprechend zu berücksichtigen.

Die angesprochene enge Abstimmung der einzelnen Komponenten aufeinander bringt Nachteile mit sich, die bei einer Rekonfiguration oder Erweiterung des Systems deutlich werden. In einem solchen Fall ist in der Regel eine komplette Überarbeitung nötig. Soll beispielsweise ein Sensor durch einen anderen mit einer unterschiedlichen Taktrate ersetzt werden, muss das Zeitverhalten des Regelkreises und damit dessen Parametrisierung neu entwickelt und implementiert werden. Die Schwierigkeit, einzelne Elemente modular in anderen Projekten einzusetzen, liegt also in der impliziten Integration von spezifischem Wissen über Sensoreigenschaften, Berechnungsparameter und Konstanten u. ä., die einen flexiblen Austausch von Komponenten unmöglich macht. Die Konzeption des Datenaustausches liegt beispielsweise häufig in der Hand des individuellen Entwicklers, sodass dessen Wissen über die zugrunde liegenden Protokolle und Kommunikationsmechanismen nötig ist, um einen Zugriff auf die Informationen zu gewährleisten. Im Fall einer Neukomposition ist eine manuelle Anpassung des Codes nötig.

In einer statischen Konfiguration entwirft der Entwickler die Anwendung mit mehr oder minder ausgeprägten Erwartungen an die Umgebung. Für den Einsatz eines Ultra-

schallentfernungssensors zum Beispiel sind die Umgebungstemperatur und der Luftdruck maßgebliche Einflussgrößen, während ein Kamerasystem auf ein konstantes Helligkeitsniveau angewiesen ist. Die Stabilität dieser externen physikalischen Größen kann in einem statischen Szenario, beispielsweise einer Produktionshalle mit einem Industrieroboter, angenommen bzw. durch besondere Vorkehrungen wie Scheinwerfer und eine konstante Temperatur sichergestellt werden. Im Unterschied dazu können für ein mobiles System, das selbstständig in einer möglicherweise unbekanntem Umgebung navigiert, solche fixen Annahmen nicht getroffen werden. Die kamera- und ultraschallbasierte Umgebungserfassung eines häuslichen Serviceroboters kann schon durch die geänderten Temperaturen und Lichtverhältnisse, zum Beispiel beim Öffnen einer Haustür, stark beeinflusst werden.

Der Entwickler statischer Systeme antizipiert somit alle relevanten und eventuell auftretenden Zustände und entwirft dafür ein entsprechendes Überwachungskonzept. Für einen Industrieroboter mit der heute typischen Einhausung seines Arbeitsbereiches kann also eine Kollision mit einem Bediener dadurch vermieden werden, dass ein Türschalter die Anlage beim Betreten durch eine Person stoppt. Die Zahl der Zustände „Arbeiter im Arbeitsbereich“, „kein Arbeiter im Arbeitsbereich“ erlaubt die Erfassung dieser Situation mit einem reduzierten Sensoraufbau. Im Unterschied dazu ist für frei navigierende Transportplattformen eine ungleich größere Zahl an Konstellationen auf unterschiedlichem Abstraktionsniveau – „Person im Frontbereich“, „Objekt zwei Meter entfernt“, „anderer Roboter nähert sich auf Kollisionskurs“ – zu berücksichtigen. In einem solchen System kann die Sensorik nicht mehr auf einen Einzelfall zugeschnitten werden, sondern zielt darauf, eine homogene „Rundumerfassung“ zu gewährleisten. Diese hat auch im ungünstigsten Fall zum Beispiel ein glänzendes, schlankes Stuhlbein korrekt zu erkennen. Um diese omnidirektionale Erfassung umzusetzen, ist ein hoher technischer Aufwand nötig, obwohl über die gesamte Einsatzdauer gesehen nur eine geringe Zahl der Sensoren wirklich genutzt wird. Für die Abdeckung selten auftretender Phänomene und Situationen ist mit einem statischen System ein erheblicher Aufwand nötig.

Erschwerend kommt hinzu, dass die Heterogenität der Umweltbedingungen die Messgüte der Sensoren verringert. Mit dem Fehlen der statischen Kontextvorgabe für die Umgebungsparameter steigt auch die Wahrscheinlichkeit von gestörten Messungen, was wiederum ein Fehlverhalten des Gesamtsystems bewirken kann. Ein Sensor allein zeigt in seinem Verhalten in der Regel eine Überlagerung verschiedener Fehlermodelle, zum Beispiel ein stochastisches Messrauschen und einen Offset, wobei die Ausprägung (Rauschparameter, Drift etc.) von den Umgebungsbedingungen abhängt. Dieses individuelle Verhalten wird, soweit es Berücksichtigung findet, analog zum Kommunikationsverhalten in den Anwendungscode fest integriert. Beim Zusammenspiel mehrerer Sensoren ergibt sich eine unübersichtliche Verschmelzung verschiedenster Fehlerdetektionsansätze und unterschiedlicher Kompensationsmethoden. Auch hier ist eine tiefe Kenntnis der Applikationsimplementierung nötig, um zum Beispiel die Frage zu beantworten, ob ein bestimmter Messwert bereits auf seine Gültigkeit hin geprüft wurde.

Statisch konfigurierte Sensor-Aktor-Systeme weisen also neben unstrittigen Vorteilen entscheidende Nachteile auf. Um einen optimalen Zuschnitt der Sensorik und der nachgeordneten Verarbeitung auf eine Anwendung zu gewährleisten,

- ist in der Regel eine Einschränkung von Umgebungsbedingungen nötig,
- muss die Sensorausstattung zwingend auf den (unter Umständen gar nicht korrekt identifizierten) ungünstigsten Fall zugeschnitten sein und
- verhindert eine enge Verzahnung der einzelnen Elemente deren Wiederverwendung und Wartbarkeit.

Damit stehen technische Applikationen im Kontrast zur menschlichen Wahrnehmung, die in hohem Maße adaptiv wirkt. Zum einen lassen sich die verschiedenen Sinne aufgaben- und kontextbezogen dynamisch komponieren, sodass die Wahrnehmung zum Beispiel beim Betreten eines dunklen Raumes auf die Arme als Tastorgan erweitert wird. Zum anderen sind Menschen in der Lage, ihre Wahrnehmung durch externe Geräte oder aber andere Personen zu unterstützen oder zu erweitern. Für eine ungefähre Einschätzung der Temperatur genügt innerhalb gewisser Grenzen die Haut des Menschen. Für einen präzisen Wert müssen zunächst entsprechende Messmittel herangezogen, abgelesen und deren Resultate dann interpretiert werden. Analog werden Informationen akquiriert, wenn diese von anderen Menschen erlangt werden sollen, vorausgesetzt der Gegenüberstehende kennt die Antwort, spricht eine gemeinsame Sprache und benutzt ein einheitliches Abbildungskonzept für Größen und Maße. Anhand von Erfahrungen oder der Einschätzung der Kompetenz des Auskunftgebenden erfolgt zudem eine Validierung der Informationen. Mit dieser Anpassungsfähigkeit und der Möglichkeit zusätzliche Informationsquellen einzubeziehen, ist der Mensch den beschriebenen, statischen Implementierungen eines Sensor-Aktor-Systems im Hinblick auf wechselnde Umgebungsbedingungen, fehlerhafte Wahrnehmungen oder unerwartete Situationen überlegen.

Vor diesem Hintergrund muss die Frage gestellt werden, ob sich dieses am Menschen beobachtbare Prinzip einer dynamischen Messdatenaggregation und Verarbeitung auf technische Probleme übertragen lässt. Mit der fortschreitenden Miniaturisierung und Performancesteigerung elektronischer Komponenten steigt die Zahl der eingebetteten Geräte von Jahr zu Jahr, wobei die immer preiswertere Bereitstellung von Rechenleistung und drahtloser Kommunikation insbesondere die Verbreitung von Sensornetzen stark beschleunigt hat. In der Gebäudeautomatisierung ist beispielsweise eine Vielzahl von Sensoren und Aktoren vorhanden, die in verschiedensten Regelkreisen eingebunden sind und ihre Daten über eine zentrale Datenbank oder aber über entsprechende lokale Schnittstellen bereitstellen. Mit einem solchen Sensorerfassungssystem ausgestattet, entstehen intelligente Umgebungen, die, da nicht zwingend von einer gleich verteilten und gleichartigen Sensorausstattung ausgegangen werden kann, an unterschiedlichen Positionen verschiedene Informationen bereithalten. Gleichermaßen werden in zukünftigen

Umgebungen neben stationären Sensorelementen auch mobile Geräte (Sensoren am Menschen, andere Robotersysteme, Smartphones) vertreten sein, für die ebenso wegen einer veränderlichen Erreichbarkeit eine statische Integration in ein Sensorsystem ausgeschlossen ist.



Abbildung 1.2: Erweitertes Wahrnehmungspotenzial eines mobilen Roboters (rot) in intelligenten Umgebungen mit mobiler externer Sensorik (gelb) in Form anderer autonomer Roboter und stationärer externer Sensorik (grün) wie Überwachungskameras, Sensornetze, Bewegungsmelder und Türsteuerungen

Abbildung 1.2 illustriert den Ansatz. Die autonome Transportplattform (rot markiert) ist selbst mit Abstandssensoren, einer Kamera und einem Laserscanner ausgestattet. Sie befindet sich in einer instrumentierten Umgebung, in der mobile externe (gelb umrandet) und stationäre externe Sensoren (grün erfasst) eine Vielzahl von Informationen über die lokalen Bedingungen bereithalten. Um beispielsweise in den Raum hinter der Tür zu gelangen, muss deren Zustand (offen/geschlossen) erkannt werden, um sicher hindurch zu fahren oder gegebenenfalls eine neue Trajektorie mit einem anderen Weg zu berechnen. Natürlich ließe sich diese Information mit einer Kamera oder einem Laserscanner erlangen, aber der Aufwand wäre ungleich höher. Sollte also die über der Tür in Abbildung 1.2 sichtbare Steuerung einen entsprechenden Status verbreiten, wäre diese Information vor allem für die autonom manövrierenden Systeme hilfreich. Gleichmaßen können natürlich auch die Informationen der benachbarten mobilen Roboter (gelb markiert) oder der Bewegungsmelder über der Tür die Annäherung zum Beispiel eines Menschen ankündigen, was im Sinne einer vorausschauenden Trajektorienplanung und Lageregelung in die Planungen einbezogen werden sollte. Die Verbindungen zwischen dem Roboter und den anderen möglichen Informationsquellen in Abbildung 1.2 zeigen eine Momentaufnahme.

Mit dem Durchfahren der Tür und dem damit verbundenen Verlassen des Observationsbereiches der genannten Sensoren werden die Messdaten von anderen Sensoren relevant, die sich im Raum hinter der Tür befinden. Damit wird eine Neukonfiguration der zu erwarteten Messgrößen notwendig. Der mobile Roboter hat also eine situationsabhängige Umgebungserfassung umzusetzen und nutzt hierfür möglichst effektiv externe Informationsquellen zur Erweiterung der eigenen, möglicherweise limitierten oder fehleranfälligen Sensorik. Mobile Transportplattformen zum Beispiel können damit in kollisionsgefährdeten Bereichen wie Kreuzungen von externer Sensorik unterstützt werden, während für unkritische Abschnitte die kostengünstige eigene Sensorausstattung zum sicheren Betrieb ausreicht. Unter anderem sehen Guibas und Zhao in eben diesem Ansatz ein *Future Research Area*, das die Arbeiten im Bereich der Sensornetze mit den der mobilen Robotik kombiniert:

„Robotic systems may utilize stationary sensors as extra antennas for navigational purposes.“ [GZ04, S. 305].

Die Interaktion mit den intelligenten Umgebungen aus dem beschriebenen Szenario bedeutet zugleich eine Abkehr von der Idee des „fest verdrahteten“ Systems. Das verteilte Sensor-Aktor-System wird damit adaptiv im Hinblick auf das Zusammenspiel der verfügbaren Sensoren, aber auch der Filter und Fusionsknoten, die zur Laufzeit konfiguriert werden. Die Verarbeitungskette hängt dabei zum Beispiel vom Erfüllungsgrad einer Aufgabe, neuen (Sensor-)Knoten, bestimmten Umgebungsbedingungen, der Position usw. ab. Die konzeptionelle Entwicklung dieser Idee ist Gegenstand der vorliegenden Arbeit. Entsprechend dem integrativen Grundansatz wird das sich daraus entwickelnde prototypische Framework mit *MOSAIC* bezeichnet.

Als grundsätzliche Voraussetzungen für die Umsetzung einer adaptiven, verteilten Sensordatenverarbeitung lassen sich vier Kernaufgabenstellungen für diese Arbeit differenzieren:

1. Einschätzung der Datenvalidität Die zu erwartende Heterogenität der Applikation im Hinblick auf Sensoren, Knotenhardware usw. bei gleichzeitiger Unwägbarkeit der die Messungen begleitenden Umgebungsinformationen führt zu Messdaten, deren Güte höchst unterschiedlich zu bewerten ist. Vor dem Hintergrund der Verschiedenartigkeit der zugehörigen Fehlermodelle muss die Aussage zur Gültigkeit in einer einheitliche Fehlersemantik eingebettet werden, um erstens die individuellen Sensoreigenschaften soweit wie möglich zu kapseln und zweitens eine Vergleichbarkeit zwischen verschiedenen Systemen zu gewährleisten. Erst damit kann eine adaptive Verarbeitung eine optimale Selektion und Gewichtung der Datensätze ausführen. Folglich bedarf es einer entsprechenden Prüfung der Messdaten und Fusionsresultate, wobei es im Sinne einer konsistenten Bewertung wünschenswert wäre, dies unmittelbar auf dem den Datensatz generierenden Knoten vorzunehmen. Allerdings fehlte eingebetteten Knoten dazu häufig die für die Analyse der Daten

notwendige Rechenleistung und zum anderen der Überblick über möglicherweise redundante Messungen, sodass ein verteilter Fehlerdetektions- und Bewertungsansatz nötig ist.

- 2. Beschreibung der Daten** Neben der Validität muss auch für andere Aspekte eines Datensatzes sichergestellt sein, dass die Informationen beim Empfänger in ihrem korrekten Kontext eingeordnet und richtig interpretiert werden können. Dies sind auf einfachster Ebene die physikalische Einheit eines Messwertes sowie der Zeitpunkt und Ort der Messung. Neben diesen offensichtlichen Angaben sind, wie im Kapitel 4 zu zeigen sein wird, aber auch komplexere Angaben zum Beispiel über die Messunsicherheit oder den Überwachungsbereich eines Sensors erforderlich, um überhaupt die Relevanz von dessen Messungen hinterfragen zu können. In einem statischen Szenario würden derartige Informationen direkt in den Applikationscode eingewoben werden. Da, wie in der Szenarienbeschreibung argumentiert, aber von unbekanntem Knoten ausgegangen wird, sind diese Informationen zur Entwicklungszeit nicht bekannt. Voraussetzung für die angestrebte Adaptivität ist also eine (Selbst-)Beschreibung aller auszutauschenden Informationen.
- 3. Datenselektion und adaptive Fusion** Mit der adaptiven Allokation der Daten werden mit jedem Zyklus, anders als in klassischen Fusionsansätzen, eine unterschiedliche Art und Anzahl von Messdatensätzen zur weiteren Verarbeitung bereitstehen. Vorbereitend ist also eine Selektion nötig, die auf zwei Ebenen vollzogen werden muss. Zum einen ist für jeden Datensatz zu hinterfragen, inwieweit dieser überhaupt relevant für die spezifische Fragestellung ist. Zum anderen müssen die verbliebenen Messdaten sowohl der Rechenkapazität als auch dem Zeitfenster eines Fusionszyklus angepasst werden, wobei die größtmögliche Validität des Resultats angestrebt wird. Wie im Kapitel zum Stand der Technik noch ausführlich diskutiert wird, lassen sich einfache Fusionsverfahren wie zum Beispiel eine (gewichtete) Mittelwertbildung von n Entfernungsmessungen rasch für eine variable Zahl von Datensätzen implementieren. Abgesehen davon, dass wie unter Punkt 2 – Datenvalidität – beschrieben, in diesem Fall die Güte des Resultats schwankt und somit auch permanent bestimmt werden muss, bedarf der eigentliche Algorithmus nur geringer Anpassungen. Konsequenterweise soll diese Anpassungsfähigkeit auch auf Verfahren ausgedehnt werden, die multimedialen Daten, wie sie entsprechend dem Motivationsszenario zu erwarten sind, fusionieren.
- 4. Umsetzung** Die Heterogenität der in die verteilte Applikation einzubeziehende Hardware erfordert eine Architektur und ein Framework, die den Anwendungsentwickler bei elementaren Aufgaben wie der Schnittstellenkonfiguration, der korrekten Codierung der Daten usw. unterstützt. Die Idee ist dabei, eine generelle Architektur zu entwerfen, die einen Rahmen für die Anwendungsentwicklung definiert

und gleichzeitig die Integrierbarkeit einer Applikation in ein verteiltes Szenario sicherstellt.

Die ersten beiden Punkte der Aufzählung – Datenbeschreibung und Datenvalidität – betreffen zwei Kernfunktionalitäten eines sogenannten intelligenten Sensors, der in der Literatur häufig als Smart Sensor bezeichnet wird. Dabei genügen die bestehenden unterschiedlichen Ausprägungen dieses Gedankens aber den beiden erstgenannten Punkten der Anforderungen des *MOSAIC*-Konzeptes nicht, da Selbstbeschreibungen von Sensoren und Fehlertoleranz auf statische Applikationen zugeschnitten sind. Erst mit den erweiterten Beschreibungen und dem die Verteiltheit der Anwendung berücksichtigenden Fehlerkonzept lassen sich die variablen Interaktionen zwischen Robotern und externer Sensorik im Sinne des Motivationsbeispiels umsetzen. Zudem erweitert diese Arbeit mit dem vorletzten Punkt der Aufzählung die Ansätze des Smart Sensors auf die Verarbeitung der Daten. Die daraus resultierende Smart Fusion steht nicht mehr übergeordnet und losgelöst von der intelligenten Datenerfassung an der Spitze einer zentralistisch aufgebauten Hierarchie, sondern wird als gleichrangige Komponente integriert. Erst mit dieser Angleichung lassen sich die Elemente dynamisch komponieren und veränderliche Applikationen, wie sie vom *MOSAIC*-Konzept vorgesehen sind, umsetzen. Konsequenterweise wird diese Idee auch auf die Aktorschnittstellen ausgeweitet. Der Smart Sensor wird damit zum *Smart-X*. Anders als der häufig auf die physische Hardwarekomponente zielende Begriff des intelligenten Sensors definiert *MOSAIC* mit diesem Begriff ein Programmierkonzept, das verschiedene Abstraktionen für Sensorschnittstellen, Messdaten, Fehler usw. flexibel kombiniert.

Um diese Kombinierbarkeit der *Smart-X* zu gewährleisten, die die angesprochene Modularität und Austauschbarkeit garantiert, bedarf es einer geeigneten Programmierabstraktion. Diese definiert die Schnittstellen für den Datenaustausch, das Zusammenspiel von Anwendung und Validitätsprüfung, die Messdatenselektion usw. Erst mit einer solchen Vorgabe wird die Anwendung von der zugrundeliegenden Basisstruktur, die für die Selektion, Datenvalidierung, Kommunikation usw. verantwortlich ist, entkoppelt. Damit wird sichergestellt, dass der Anwendungsentwickler nicht durch individuelle Festlegungen die Austauschbarkeit und Integrationsfähigkeit einer *Smart-X*-Komponente gefährdet. Im Gegenteil: Dadurch dass eine übergreifende Architektur besteht, sind automatisierte Prüfungen wie zum Beispiel der konsistente Gebrauch der physikalischen Einheiten während des Entwicklungsprozesses leicht prüfbar.

Als Basisprogrammierabstraktion für *Smart-X* wird im Rahmen dieser Arbeit das Konzept eines Sentient Objects, das unter anderem von Fitzpatrick u. a. [Fit+02], Biegel [Bie04], Casimiro, Kaiser und Verissimo [CKV04] beschrieben wurde, aufgebaut. Eine Diskussion dieser Entscheidung folgt in Kapitel 3. Ein Sentient Object definiert ein Element einer verteilten Anwendung, das abstrahierte Daten konsumiert, verarbeitet und publiziert. Eine Anwendung wird aus dem Zusammenspiel unabhängiger Sentient Objects gebildet, wobei ein Sentient Object selbst aus hierarchischen Strukturen Sentient

Objects aufgebaut sein kann. Dieser Ansatz wurde konsequent genutzt und zu einer Programmierabstraktion im Hinblick auf die Anforderungen der *Smart-X*-Komponenten weiterentwickelt, mit der die Selbstbeschreibung, die Validierung der ausgetauschten Daten und die adaptive Fusion umsetzbar sind.

Mit diesem Konzept lässt sich der klassische Regelkreis in statischer Ausprägung aus Abbildung 1.1 auflösen und in eine dynamische Form überführen. Die Sensoren, der Fusionsknoten und der Regler sind, wie Abbildung 1.3 im Kontrast dazu illustriert, als Sentient Objects in einem Netzwerk organisiert, wobei die Art und Anzahl der verfügbaren Sensoren schwanken bzw., wie mit dem $Sensor_x$ angedeutet, auch irrelevant für die Applikation sein können. In der Grafik werden die Sentient Objects mit einer elliptischen Grundform markiert, sie können wie die Sensoren und der Regler eine physische Schnittstelle zur Umgebung (blau) besitzen. Andererseits verbreiten sie ihre Informationen im Netzwerk, sodass diese für alle anderen Komponenten verfügbar sind. Der in ein Sentient Object eingebundene Fusionsalgorithmus konsumiert, wie durch die geschlängelten Pfeile angedeutet, Messdaten und gibt ein entsprechendes Resultat aus. Dazu analysiert die Selektionskomponente im Fusions Sentient Object die verfügbaren Datensätze und generiert ein mit einer Validitätseinschätzung versehenes Resultat. Stehen nur Daten geringer Güter bereit, können dem Regler nur noch Schätzungen geringer Validität übermittelt werden, der daraufhin die Ausführung stoppt. Grundsätzlich ließe sich natürlich auch der Datenaustausch zwischen Regler und Aktor adaptiv umsetzen. Da für diese Komponenten aber von einer 1:1-Beziehung ausgegangen wird, macht die Umsetzung des adaptiven Konzeptes für diese Verbindung wenig Sinn.

Zusammenfassend kann gesagt werden, dass in dieser Arbeit eine Programmierabstraktion für verteilte Systeme entwickelt wird, die eine adaptive und fehlertolerante Fusion von Messdaten erlaubt, die bei der Interaktion von mobilen Systemen mit externer intelligenter Umgebungssensorik erzeugt wird.

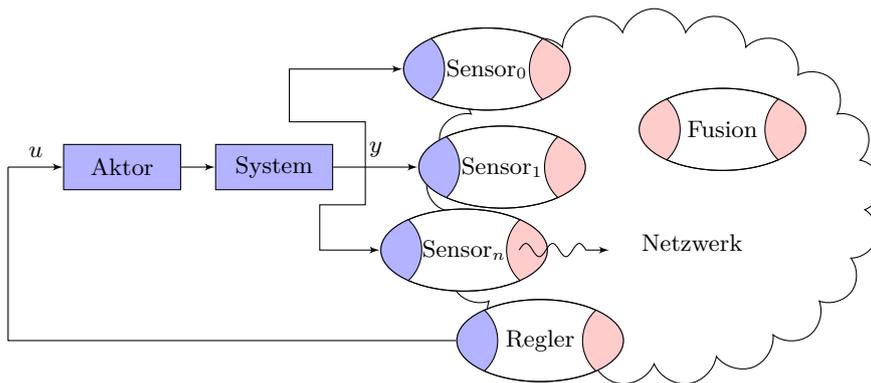


Abbildung 1.3: Regelungssystem mit veränderlichen Komponenten im Kontrast zum konventionellen Entwurf entsprechend Abbildung 1.1

1.2 Struktur der Arbeit

Diese Arbeit beschreibt den Entwurf und die Umsetzung einer Architektur für die fehlertolerante Datenerfassung und Verarbeitung in verteilten Systemen sowie deren Einbettung in das *MOSAIC*-Framework. Dazu wird folgender Weg gegangen:

- | | |
|-----------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Kapitel 2
Definition der
Anforderungen | Das folgende Kapitel bildet mit der systematischen Ableitung der Anforderungen die Grundlage der weiteren Arbeit, indem diese Aufstellung die Erfassung des Standes der Forschung kanalisiert. Zu diesem Zweck wird das Szenario aus Kapitel 1 aufgegriffen, um daraus Kernanforderungen zu extrahieren und zu kategorisieren. |
| Kapitel 3
Stand der
Forschung | Für die korrekte Einordnung der in dieser Arbeit diskutierten Programmierabstraktion werden im ersten Absatz dieses Kapitels die relevanten Begriffe aus dem Bereich der intelligenten Sensoren eingeführt. Dabei erfolgt die Vorstellung der gängigsten Sichtweisen auf die Kernbegriffe und der Versuch einer Strukturierung. Basierend auf den Erkenntnissen des Anforderungskatalogs werden darauf folgend zunächst Programmiermodelle und Frameworks für die Smart Devices besprochen und verglichen. Aus der Erkenntnis heraus, dass keine der Abstraktionen, Standards oder Programmierkonzepte eine vollständig die Anforderungen abdeckende Lösung darstellt, folgt eine Auflistung der die drei konzeptionellen Kernfunktionalitäten bedienenden Arbeiten. |
| Kapitel 4
Konzept-
entwicklung | Dieser Dreiteilung folgt auch der Entwicklungsfaden und beginnt ausgehend vom Basismodell des Sentient Objects mit der Differenzierung des Konzepts. Diese zunächst kernaufgabenbezogenen Ansätze werden im letzten Abschnitt des Kapitels in einer Grundstruktur für <i>Smart-X</i> zusammengeführt. |
| Kapitel 5 | In diesem Kapitel folgt die Beschreibung der Umsetzung dieses Konzepts in zwei Varianten. Die Implementierung in Mathworks/Simulink zielt dabei auf eingebettete statische Systeme ab. Die Proof-of-Concept Implementierung in Python adressiert leistungsfähige PC-Architekturen, auf denen adaptive <i>Smart-X</i> umsetzbar sind. |
| Kapitel 6
Evaluation | Anhand eines Simulationsszenarios, einer Entwicklungsbeschreibung und einer Lokalisationanwendung aus dem Bereich der mobilen Roboter erfolgt die Validierung des <i>MOSAIC</i> -Konzepts und des prototypisch implementierten Frameworks. |
| Kapitel 7
Zusammenfass-
ung u. Ausblick | Das letzte Kapitel fasst die Ergebnisse der Arbeit zusammen und diskutiert die möglichen weiteren Entwicklungsschritte aufbauend auf den gesammelten Erkenntnissen. |

2 Klassifikation der Anforderungen

Für das im vorhergehenden Abschnitt beschriebene Konzept folgt aus dem Motivationszenario neben den bereits aufgeführten Kernanforderungen eine Zahl von darüber hinausgehenden Erfordernissen, die für die Entwicklung eines entsprechenden Programmiermodells zu berücksichtigen sind. Folglich werden in diesem Kapitel zunächst die Anforderungen näher untersucht und eine Kategorisierung durchgeführt.

Dabei wird zwischen zwei Anforderungskategorien unterschieden. Die funktionsbezogenen Erfordernisse ergeben sich aus dem Ansatz der adaptiven Verarbeitung und betreffen die Integration der Datenbeschreibung, der Fehlerdetektion und der flexiblen Fusion. Die zweite Kategorie umfasst die Anforderungen an die zu entwickelnde Architektur und deren Umsetzung. Innerhalb dieser Einteilung werden die Erfordernisse, die sich teilweise selbst nochmals in Teilanforderungen strukturieren lassen, organisiert. Anhand dieser zusammenfassenden Darstellung kann im nächsten Kapitel die Eignung der vergleichbaren Arbeiten untersucht werden.

2.1 Architekturbezogene Erfordernisse

Interne Modularität

Die Forderung nach einer Kombinierbarkeit stellt sich für *Smart-X*-Komponenten auf interner und externer Ebene. Innerhalb eines *Smart-X*-Elements finden sich viele Funktionen, die völlig gleichartig oder lediglich mit unterschiedlichen Parametersätzen in vielen Anwendungen eingesetzt werden. Dazu zählen bestimmte Algorithmen, die elementare Funktionen umsetzen und zum Beispiel die Anreicherung der Daten mit Zeitstempeln übernehmen sowie für einfache Filterungen oder die Selektion von Datensätzen verantwortlich sind. Derartige Funktionen sind in geeigneter Granularität zu organisieren, sodass sowohl eine flexible Kombinierbarkeit als auch ein hoher Grad an Wiederverwendbarkeit gegeben sind. Dafür bestehen zwei Voraussetzungen. Die Architektur eines *Smart-X* muss über generalisierte Schnittstellen die Möglichkeit bieten, variabel eine Funktion in die Verarbeitungskette zu integrieren. Innerhalb dieser Kette sind dann Mechanismen zur Steuerung der Abarbeitungsfolge als Grundlage der Adaptivität nötig. Damit wird zum Beispiel koordiniert, dass mit dem Überschreiten eines bestimmten Messwertes die Berechnungen abgebrochen oder auf einen anderen Verarbeitungspfad umgelenkt werden.

Anforderung 1 (Interne Modularität)

- a) *Die Architektur der Smart-X-Komponenten erlaubt die flexible Einbindung von wiederverwendbaren Einzelfunktionen.*
- b) *Für die Abarbeitung der Verarbeitungskette können vom Entwickler Bedingungen eingebunden werden, die den Start, die Verzweigung, den Berechnungsabbruch usw. bestimmen.*

Externe Modularität

Eine ähnliche Modularität haben die *Smart-X* auch extern zu unterstützen, wenn diese sowohl zur Entwicklungs- als auch zur Laufzeit kombiniert werden sollen, um verteilte und dynamisch veränderliche Anwendungen umzusetzen.

Mit dieser Form der Modularisierung und dem Abstraktionskonzept aus *MOSAIC* ist es dem Entwickler zudem erlaubt, mehrere *Smart-X* in einer organisatorischen Einheit zu erfassen und auf einer hierarchisch höher stehenden *Smart-X*-Komponente abzubilden. Die Darstellung vereinfacht den Entwicklungsprozess, erlaubt aber gleichzeitig den Austausch und die Wiederverwendbarkeit der *Smart-X*-Komponenten.

Anforderung 2 (Externe Modularität)

- a) *Smart-X-Komponenten sind modular organisiert und auf eine freie Kombinierbarkeit und Wiederverwendbarkeit ausgerichtet.*
- b) *Zusammengewirkte Smart-X-Komponenten lassen sich als einzelnes Element in einer hierarchisch höherwertigen Repräsentation abbilden.*

Unterstützung von Domänenspezifika

Sensor-Aktor-Systeme verzahnen verschiedene ingenieurwissenschaftliche Disziplinen in einer gemeinsamen Applikation, sodass für ein Szenario, wie in Abschnitt 1 beschrieben, Fragen der Kinematik, Dynamik, Regelung, Kommunikation, Signalverarbeitung usw. zu lösen sind. Hier gilt es neben den universell einsetzbaren Programmiersprachen wie C, C++ auch die für die einzelnen Problemfelder relevanten und etablierten Sprachen und Entwicklungsumgebungen zu berücksichtigen. Diese speziell ausgerichteten Programmierwerkzeuge werden als Domain Specific Languages (DSLs) bezeichnet, die nach [VDKV00, Seite 27] als

„... a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.“

definiert sind. DSLs abstrahieren technische Fragestellungen mit einer problembezogenen einheitlichen Terminologie, die eine effektive gemeinsame Entwicklungsarbeit von Domain-Experten erlaubt. Zum einen steigt damit die Produktivität, Wartbarkeit und

Zuverlässigkeit des Produktes und zum anderen eröffnet die Modularisierung und Abstraktion eine Wiederverwendung einzelner Elemente und Standardmethoden [Kie+96].

Im Zusammenhang mit den hier angesprochenen Robotikthemen können dies zum Beispiel Programmiersprachen mit statistischer Ausrichtung [GNU11, TIB11] oder Modellierungsumgebungen für dynamische Systeme [Ope11, Das10] sein. Daneben existieren grafische Programmiersysteme [Nat09, The11], die in Entwicklungsumgebungen eingebunden sind und mehrere Aspekte (Datenerfassung, -verarbeitung, Simulation usw.) kombinieren. Viele der DSL-Umgebungen beinhalten zur Codegenerierung eine Werkzeugkette, die eine abstrakte (grafische) Implementierung in eine plattformspezifische Programmiersprache einer Zielplattform transformiert und damit eine fehleranfällige Reimplementierung erspart.

Ein Beispiel für eine solche DSL-Kette stellt das Produktpaket Matlab/Simulink von Mathworks [The10] dar. Es benutzt die Programmiersprache m, bietet aber auch eine abstrakte grafische Programmierung an, mit der kontinuierliche und diskrete Systeme definiert werden können. Daneben umfasst die von Mathworks bereitgestellte Entwicklungsumgebung verschiedenste Werkzeuge zur Visualisierung, zum Profiling, zum Debugging und sieht eine Codegeneration vor, um aus der DSL-Implementierung einen targetspezifischen Code zu generieren [Mat10a, Mat10b]. Zudem existiert eine breite Palette an Toolboxen, die unterschiedlichste Domains berücksichtigen. Im Bereich der Regelungstechnik wird die Matlab/Simulink-Umgebung damit zu einem Quasi-Standardwerkzeug.

Zielstellung im Rahmen der Softwareentwicklung für *Smart-X*-Komponenten ist es, domänenspezifische Sprachen und Werkzeuge so zu integrieren, dass zum einen die außerhalb des zu bearbeitenden Problemfeldes liegenden Aspekte weitgehend ausgeblendet werden, während gleichzeitig die Funktionalität der DSL weitgehend ausgenutzt wird. Zum Beispiel sollen die hardwareseitigen Eigenschaften eines Sensors wie Interfaceparameterisierung, Triggerung usw. gekapselt werden, da diese für eine Signalverarbeitung nicht relevant sind. Andererseits sollen aber alle Werkzeuge und Methoden zur Signalanalyse, Filterung und Verarbeitung bereitstehen, sodass der Schwerpunkt der Arbeit des Domainexperten auf der Anwendungsentwicklung und nicht in der Schnittstellenprogrammierung und Konfiguration liegt.

Anforderung 3

- a) Die Umsetzung des MOSAIC-Konzepts in einer DSL zielt auf die Nutzung einer breiten Zahl von domänenspezifischen Bibliotheken ab.
- b) Mit der/den gewählten Programmiersprache(n) ist eine architekturübergreifende Entwicklung zu gewährleisten.

Heterogene Hardware

Die Elemente eines Sensornetzes integrieren, um eine lange Lebensdauer und einen geringen Kostenaufwand pro Knoten zu gewährleisten, häufig preiswerte Mikrocontroller, die über eine beschränkte Leistungsfähigkeit verfügen [Kah+98]. Auf der anderen Seite kann

die Verarbeitung von Messdaten im Sensor-Aktor-System auch von sehr leistungsstarken Rechnern oder hoch spezialisierten Signalprozessoren übernommen werden. Diese häufig anzutreffende Heterogenität der Knotenhardware wird auch in der Darstellung des Motivations szenarios deutlich. Im Vordergrund sichtbare Sensornetzkn timer arbeiten auf der Basis von 8-Bit-Mikrocontrollern, während die Analyse des Kamerabildes oder der Laserscannermessungen auf einer x86-Architektur umgesetzt wird.

Die Kapselung dieser unterschiedlichen Architekturen, Registerstrukturen usw. ist für die Integration des *Smart-X*-Konzeptes in eine DSL unbedingt erforderlich.

Erschwerend kommt hinzu, dass Mikrocontroller in der Regel als Controllerfamilien ausgelegt sind, wobei sich die einzelnen Mitglieder in unterschiedlichen Ausprägungen und Parametrierungen der Interfaces, des Speichers, der Kommunikationsschnittstellen o. ä. definieren. Soweit wie möglich sind Tests vorzusehen, die die Implementierbarkeit einer bestimmten Funktionalität auf einem speziellen Controller hinsichtlich seiner Interfaces und Rechenkapazität prüft.

Anforderung 4

Im Hinblick auf die zugrunde liegende Hardware gilt es, die Heterogenität der Zielplattform zu kapseln.

Simulationsfähigkeit

Für die Entwicklung verteilter Anwendungen ist die Simulation des Verhaltens einzelner Komponenten von großer Wichtigkeit. Auf diesem Wege können zum Beispiel die Signalverarbeitungsalgorithmen eines Sensors geprüft oder der Einfluss eventueller Verzögerungen beim Datenaustausch validiert werden.

Anforderung 5

Inbesondere für die Sensoren ist in deren Programmierabstraktion eine Schnittstelle für die Simulation von deren Verhalten vorzusehen.

2.2 Funktionsbezogene Erfordernisse

Beschreibung der Daten

Wie bereits im Kapitel zuvor angedeutet, ist die Bereitstellung der Attribute einer Messung eine Voraussetzung für die Umsetzung des *MOSAIC*-Konzeptes. Die für die Interpretation, Selektion, Validierung und Verarbeitung erforderlichen Informationen können entweder in einem elektronischen Datenblatt oder im Messdatensatz selbst enthalten sein, wobei erstgenannter Weg für statische Größen (Sensorkeule, Reichweite usw.) und letztgenannter für die veränderlichen (Validität, Zeitstempel, usw.) und sensorknoten-spezifischen Werte (Positionsangaben, Orientierung) vorgesehen ist. Die Programmierabstraktion hat dann sicherzustellen, dass für jeden Datensatz die Informationen zusammengeführt werden.

Im Hinblick auf die elektronischen Datenblätter sollte das Ziel darin bestehen, eine Repräsentationsform zu wählen, die aus Programmen heraus mit geringem Aufwand analysiert und generiert werden kann, aber auch vom Menschen unmittelbar lesbar ist. Damit wird zum einen die automatisierte Verarbeitung während der Entwicklung und zur Laufzeit, aber auch die einfache Editierung durch den Nutzer sichergestellt. Da die Datenblätter auf jedem Knoten abgelegt werden sollen, darf die Größe wenige kByte zudem nicht überschreiten. Diese Forderung ergibt sich aus den in Anforderung 4 formulierten Beschränkungen eines Teils der Zielformen.

Mit dem Wissen über die Parameter eines Datensatzes und deren Formate lassen sich im Entwicklungsprozess Tests integrieren, die zum Beispiel Verstöße gegen die definierten Datentypen und physikalische Einheiten automatisch detektieren und signalisieren. Damit sind Kompatibilitäts- und Schnittstellenprobleme, wie sie bei der Entwicklung von verteilten Sensor-Aktor-Systemen häufig auftreten und zum Beispiel bei der Explosion der Ariane 5 Rakete [Dow97] oder dem Verlust des Mars Climate Orbiter [EJC01] zu Totalausfällen führten, vermeidbar.

Anforderung 6

- a) *Die Eigenschaften der Smart-X-Knoten werden in Datenblättern beschrieben, deren automatisierte Auswertung sowohl in den Entwicklungsprozess als auch in die Architektur zu integrieren ist.*
- b) *Für die Datenblätter ist eine geeignete Beschreibungssprache zu definieren.*
- c) *Das Wissen um die Formate, Datentypen und Einheiten eines Datensatzes ermöglicht die kontinuierliche Konsistenzprüfung dieser Größen während der Entwicklung und zur Laufzeit.*

Verteilte Fehlerdetektion

Wie bereits im Kapitel 1 dargestellt, steigt mit der Verteiltheit und der Veränderlichkeit der Umgebungsbedingungen die Wahrscheinlichkeit von Fehlern im Sensor-Aktor-System. Dabei wirken sich, wie noch in Kapitel 4 zu zeigen ist, unterschiedliche Fehlertypen auf einzelne Bereiche einer Smart-X-Komponente mit veränderlichem Einfluss aus. Für den Entwurf eines zuverlässigen Gesamtsystems bedarf es daher einer systematischen Untersuchung und Kategorisierung der im Szenario beobachtbaren Fehlertypen. Ausgehend von einer solchen Analyse sind dann die für die Einhaltung einer bestimmten Ergebnisgüte nötigen Fehlerdetektionsmechanismen vorzusehen und deren Einschätzungen in geeigneter Weise bei der Verarbeitung der Informationen zu berücksichtigen.

Ein Hauptziel eines Fehlertoleranzkonzeptes für verteilte, eingebettete Geräte muss eine weitgehende Detektion der Fehler nahe ihrer Ursache sein. Die unzulässige Abweichung sollte bereits auf dem diese generierenden Knoten erfasst werden. Dies ist mit drei Argumenten zu begründen:

1. Ein Grund resultiert aus der angestrebten möglichst langen Lebensdauer eines eingebetteten Sensorknotens, der auf die beschränkte Kapazität einer Batterie

angewiesen ist. Geeignete Fehlertoleranzmechanismen helfen, diese Ressource zu schonen, indem beispielsweise Messdaten einen minimalen Vertrauenswert überschreiten müssen, bevor sie energieaufwendig versendet werden.

2. Sensoren stellen ihre Messdaten nicht nur einem, sondern möglicherweise mehreren Verarbeitungsknoten zur Verfügung. Führt jeder dieser Verarbeitungsknoten eine eigenständige und folglich redundante Fehleruntersuchung aus, so sind inkonsistente Bewertungen nicht auszuschließen. Gleichzeitig wird wegen der mehrfachen Berechnungen unnötig Energie konsumiert.
3. Fasst man eine *Smart-X*-Komponente als modulares Objekt auf, so sollen die spezifischen Fehlertoleranzmechanismen, die sich aus den Hardwareeigenheiten desselben ergeben, innerhalb dieses Moduls gekapselt werden. Eine Übertragung von bestimmten Fehlertoleranzfunktionalitäten mitsamt dem nötigen Parametersatz würde dieses Konzept stören.

Das Konzept der ursachennahen Fehlerdetektion wird durch die möglicherweise limitierte Rechenkapazität der eingesetzten Hardware beschränkt. Diese schließt bestimmte Fehlerdetektionsmethoden resultierend aus dem Berechnungsaufwand aus, die nicht in geeigneter Zeit oder mit hinreichender Genauigkeit ausgeführt werden können. Daneben bestehen natürlich Fehler, die nur im Zusammenspiel redundant kooperativ wirkender Sensoren zu detektieren sind und folglich auch auf nachgeordneten *Smart-X* auszuführen sind. Entsprechend müssen die bereits durchgeführten Fehlerdetektionsuntersuchungen für jeden Messwert zugeordnet werden können.

Anforderung 7

- a) *Eine Fehlertoleranz für Smart-X setzt verteilte Fehlerdetektionsmechanismen auf verschiedenen Ebenen der Sensordaten- und Informationsverarbeitung voraus. Entsprechend sind Fehlerdetektionsmechanismen als elementarer Bestandteil eines Smart-X bei der Herleitung einer Architektur zu berücksichtigen.*
- b) *Die Abbildung der variierenden, spezifischen Fehlermodelle eines Sensors oder einer Verarbeitungsapplikation auf einer einheitlichen Abstraktion stellt sicher, dass die Ergebnisse der Fehlerprüfung richtig interpretiert und gegenüber anderen Datensätzen verglichen werden können.*

Datenselektion

Mit der zu erwartenden variablen Zahl von Messungen unterscheidet sich das hier beschriebene Konzept grundsätzlich von den Standardverfahren der Messdatenverarbeitung. Dabei wird das Sensorsystem so zugeschnitten, dass zu einem bestimmten Zeitpunkt die erwarteten Messdaten zwingend für die weitere Verarbeitung bereitgestellt werden. Die dynamische Datenallokation erfordert im Unterschied dazu eine Selektion der vorliegenden Daten im Hinblick auf deren Parameter wie zum Beispiel:

- Die Relevanz von Sensoren mit einem beschränkten Überwachungsbereich verändert sich für mobile Systeme permanent. Eine statisch angeordnete Kamera verliert, wie in Abbildung 1.2 gezeigt, ihre Bedeutung, wenn sich der Roboter aus ihrem Erfassungsbereich bewegt.
- Eine Information über das Alter der Messdaten ist unter Beachtung der Dynamik des beobachteten Systems von großer Wichtigkeit. Bei einer Temperaturmessung in Gebäuden ist für eine Verzögerung von wenigen Sekunden keine Verschiebung der realen Größe zum Messwert zu erwarten. Eine sich rasch bewegende Plattform dagegen kann in dieser Zeit ihre Position signifikant verändert haben.
- Daten mit einer hohen Präzision und geringen Fehlerwahrscheinlichkeit ermöglichen eine gute Aussage über den gegenwärtigen Zustand der Umgebung oder des eigenen Systems. Folglich werden diese vor stark rauschbehafteten oder möglicherweise fehlerhaften Datensätzen bevorzugt.

Die Selektion erfolgt dabei an unterschiedlichen Stellen der Verarbeitungssequenz. Bestimmte Informationen sind beispielsweise erst nach der Kombination der Messdaten mit den zugehörigen Datenblättern verfügbar bzw. kann eine Auswahl erst unmittelbar vor der Fusion stattfinden, wenn alle in diesem Zeitschritt verfügbaren Daten gegenübergestellt werden können.

Anforderung 8

Für die Selektion der empfangenen Datensätze als Vorbereitung der Verarbeitung sind effektiv wirkende Filtermechanismen vorzusehen, die vom Nutzer einfach zu konfigurieren sind.

Flexible Verarbeitung

Ausgehend von der Vorbereitung des Datenbestandes sind die nachgeordneten Verarbeitungsalgorithmen wie zum Beispiel Filter oder Fusionsmethoden insoweit anzupassen, dass diese auf die aktuell bereitstehenden Daten zugeschnitten sind. Wenn beispielsweise für die Schätzung der Position eines Roboters zu einem Zeitpunkt kein global wirkender, externer Kamerasensor verfügbar ist, kann ausschließlich die inertielle Sensorik verwendet werden, um eine Schätzung der Positionsangabe vorzunehmen. Folglich ergibt sich eine zweifache Adaptivität in *Smart-X*-Komponenten. Das Datenmanagement ist insbesondere in aktiven Komponenten dafür verantwortlich, die von der Anwendungsfunktion geforderten Daten in bestmöglicher Qualität bereitzustellen. Gleichzeitig kann diese Funktionalität aber auch in einer hinreichend variablen Form vorgesehen werden, sodass eine Anpassung an die wirklich vorhandenen Datensätze erfolgt. Sollte kein passender Datensatz bereitstehen, kann entweder ein Ergebnis geringer Güte auf der Basis der vorhandenen Daten geschätzt werden oder aber vollständig auf eine Ausgabe verzichtet werden.

Die Verarbeitungskette kann auf unterschiedliche Weise getriggert werden. Während für einen Filter, der einen einzelnen Messwert beispielsweise auf die Einhaltung eines Schwellwertes untersucht, eine unmittelbare Verarbeitung sinnvoll ist, arbeiten Regelungsalgorithmen häufig streng periodisch. In der erstgenannten Variante wird die Verarbeitung durch das Eintreffen eines Datensatzes ausgelöst, in der zweiten durch einen Timer, der zur Entwicklungszeit parametrisiert wird. Die Abstraktion einer *Smart-X*-Komponente muss somit die Umsetzbarkeit beider Konzepte gewährleisten.

Anforderung 9

- a) *Mit der Vorverarbeitung sind redundante Werte anhand ihrer Parameter geeignet zusammenzufassen.*
- b) *Die Verarbeitung der Datensätze zum Beispiel für eine Fusion sollte an die Bereitstellung einer variablen Anzahl von Datensätzen unterschiedlicher Sensorarten angepasst sein.*

3 Stand der Forschung

In diesem Kapitel werden die verwandten Arbeiten im Bereich der Programmierkonzepte, Abstraktionen und Entwicklungsframeworks auf deren Anwendbarkeit für die adaptive Datenerfassung und Verarbeitung geprüft. In einem ersten Schritt erfolgt dazu eine Analyse der verschiedenen Bedeutungen grundlegender Begriffe wie Sensor, Transducer usw. In einem zweiten Abschnitt werden darauf aufbauend die Programmierabstraktionen für verteilte Anwendungen erfasst und deren Umsetzungen in verschiedenen Entwicklungsframeworks hinterfragt. Da, wie noch zu zeigen ist, die bestehenden Konzepte nicht den Kernaufgabenstellungen der Arbeit (Seite 6) und dem Anforderungskatalog aus Kapitel 2 vollständig entsprechen, wird eine konzeptionelle Erweiterung dieser Ansätze in Kapitel 4 notwendig. Folglich werden zusätzlich zu den Programmierabstraktionen relevante Arbeiten aus den Bereichen Datenvalidität, Datenbeschreibung und Selektion herangezogen.

3.1 Begriffsbildung

Der Gebrauch der relevanten Begriffe wie Sensor, integrierter Sensor, Smart Sensor usw. ist im Hinblick auf die Funktion, die verwendete Hardware usw. höchst unterschiedlich. Vor diesem Hintergrund versucht der vorliegende Abschnitt die Definitionen zu ordnen, in einem Schema zu erfassen und eine Bezeichnung für die Verwendung im Rahmen dieser Arbeit festzulegen.

Die Reihenfolge der Begriffsdifferenzierung ergibt sich aus zwei Gründen. Zum einen wird damit der historischen Entwicklung des Sensorbegriffes entsprochen, wobei sich aus dem einfachen Sensor oder Transducer der integrierte Baustein und darauf aufbauend der Smart Sensor entwickelte. Zum anderen folgt die Aufzählung dem steigenden Funktionsumfang der Systeme. Während für den klassischen Sensor die Erfassung einer Umgebungsgröße das dominierende Merkmal ist, wird in den als smart bezeichneten Sensoren eine Fülle an peripheren Aufgaben integriert.

3.1.1 Sensor

Als Sensor (von lat. *sentire*, dt. „fühlen“ oder „empfinden“) wird ein technisches Bauteil bezeichnet, das selektiv die Konfiguration eines Umgebungsparameters quantifizierbar macht. Resultierend aus den höchst unterschiedlichen physikalisch-chemischen Wirkprinzipien des Abbildungsprozesses sind Sensoren in der Regel auf eine bestimmte Messgröße,

zum Beispiel Wärme, Temperatur, Feuchtigkeit, Druck, Helligkeit oder Beschleunigung zugeschnitten und auch nur dafür verwendbar. Die zu messende Größe muss nicht zwingend mit der eigentlich erfassten Größe identisch sein, sofern anhand bestimmter physikalischer Gesetzmäßigkeiten eine Umrechnung möglich ist. Ein Beispiel dafür ist die Messung des Spannungsabfalls am Heißeiter, um eine Temperatur zu bestimmen. Die Messgröße wird vom Sensor auf einen für die weitere Verarbeitung vorteilhaften Ausgabewert abgebildet, wobei dies in heutigen Anwendungen zumeist elektrische Signale sind. Allerdings sind auch andere Formen der Ausgabe praktikabel, wie zum Beispiel die Kraft an der Steuerstange eines Fliehkraftreglers oder die auf den Menschen zugeschnittene visuelle Darstellung am mechanischen Kompass oder analogen Thermometer [Bib06]. Im Kontrast dazu vertreten zum Beispiel Tränkler und Obermeier [TO98] die Auffassung, dass ein Sensor immer elektrische Messsignale bereitstellt.

Eine genaue Definition des Begriffes Sensor mit einer klaren Abgrenzung zu benachbarten Bezeichnungen wie Messgrößenaufnehmer, Fühler, Messwertgeber etc. kann aus dem höchst unterschiedlichen Gebrauch in der Literatur nicht herausgearbeitet werden. Die für die Messdatenerfassung im deutschsprachigen Raum relevante DIN 1319 1-4 greift den Begriff zum Beispiel gar nicht auf und spricht stattdessen in Abschnitt 2 vom „Messaufnehmer“ als dem Beginn der Messkette [Deu95]. Im allgemeinen ingenieurwissenschaftlichen Gebrauch [Kle98] wie auch in vielen Datenblättern [Sha07, Wen03] wird als Sensor häufig das mechanisch abgeschlossene Messsystem mit einer integrierteren Messdatenverarbeitung bezeichnet, was eher den in den folgenden Abschnitten beschriebenen „Integrierten Sensoren“ oder „Smart Sensoren“ entspricht. Auf der anderen Seite, wie zum Beispiel in [Kir+02] steht die Interpretation des Sensorbegriffs für ein unmittelbares Interface zur Umgebung, dessen zentrale Aufgabe der Transformationsprozess ist [TO98]. Im Rahmen dieser Arbeit wird folgende Definition des Begriffes gebraucht, die dem modularen Konzept von *MOSAIC* entgegenkommt:

Definition 1 (Sensor)

Sensoren transformieren physikalische, chemische oder biologische Messgrößen in elektrische Signale und stellen damit das unmittelbare Interface eines Messsystems zur Umgebung dar.

Der Sensor ist insofern ein Hilfsmittel, um physikalische Größen im Sinne einer mathematischen Funktion auf einem (elektrischen) Signal abzubilden. Die Güte dieser Abstraktion richtet sich nach spezifischen Abbildungseigenschaften des Sensors, die im weiteren Verlauf der Arbeit in den angesprochenen elektronischen Datenblättern zu berücksichtigen sein werden. In Anlehnung an die Aufstellung in [Kir+02] gehören dazu:

- Erfassungsbereich - Fenster, in dem sich die Messgröße bewegen darf, um eine korrekte Erfassung zu gewährleisten.
- Ausgabeintervall - zulässiger Bereich der Ausgabewerte

- Offset - Überlagerung der Ausgabegröße mit einer variablen/konstanten Beaufschlagung
- Antwortverhalten:
 - räumlich - Einfluss der geometrischen Anordnung
 - zeitlich - temporales Folgeverhalten des Ausgabesignals zur veränderlichen Messgröße
- Auflösung - Granularität des Wandlungsprozesses im Sensor
- Kalibrierung - Definition einer Abbildungsfunktion der Messgröße auf dem Ausgabewert unter Berücksichtigung von weiteren Parametern
- Messverhalten
 - Hysterese - Beschreibung der Abhängigkeit des Ausgabewertes von der zeitlichen Entwicklung der Messgröße
 - Drift - Änderung der Ausgabewerte bei konstanter Messgröße
 - Rauschen - additiver, stochastischer Fehler, der häufig als normal verteilt angenommen wird
 - Nichtlinearität - Abweichung von einem angenommenen linearen Zusammenhang zwischen Messgröße und Ausgabewert usw.
- Fehler

Das Abbildungsverhalten definiert somit den Unterschied zwischen einem idealen und einem realen Sensorelement. Die jeweilige Ausprägung von dessen Parametern ist dem Datenblatt des Sensors zu entnehmen oder experimentell unter konkreten Einsatzbedingungen zu bestimmen, um möglicherweise unbekannte Einflussgrößen zu berücksichtigen.

3.1.2 Transducer

Für den im englischen Raum üblichen Begriff des Transducers ergibt sich ausgehend von einer allgemeingültigen Definition

„...device that converts input energy into output energy, the latter usually differing in kind but bearing a known relation to input.“ [Enc07]

ein vielschichtiges Bild. Während einige Autoren, wie Henderson und Shilcrat [HS84] sowie Artiola, Pepper und Brusseau [APB04], betonen, dass der Begriff des Transducers analog zu dem des Sensors verwendet werden kann, arbeiten andere eine Differenzierung heraus. Elmenreich [Elm02] zum Beispiel folgt der oben genannten allgemeingültigen Definition und interpretiert den Transducer als übergeordnete Abstraktion für Sensoren

und Aktoren, wobei die einzige Spezifizierung über den Wandlungsprozess erfolgt. Dieser kann sich für den als Sensor wirkenden Transducer über die Abbildung physikalischer Größen auf einem elektrischen Wert darstellen oder aber für den Aktor mit der mechanischen Reaktion auf Steuerungskommandos. In der Fortsetzung dieses generalisierenden Ansatzes kann die gesamte Verarbeitungskette einer Sensorhardware als Aneinanderkettung von Transducern aber auch als Transducer selbst betrachtet werden. Demgegenüber wird in [Sin01] lediglich die einem Sensor folgende Signalverarbeitung als Transducer interpretiert und versucht, die Energieeffizienz als ein Differenzierungsmerkmal herauszustellen. So sei der Wirkungsgrad eines Sensors im Sinne einer möglichst weitgehenden Wahrnehmung irrelevant, die nachgeordneten Transducer aber müssten das eingehende Signal effizient konvertieren. Diesem Bild entsprechend unterscheiden einige Hersteller im amerikanischen Raum in ihrer Produktpalette anhand des Ausgangsspannungsniveaus zwischen Sensor und Transducer. Ein Sensor bis zu einem Signallevel von 0.02 V gilt danach als Sensor und darüber als Transducer. Unter Benutzung der oben genannten Definition bemühen sich Luo und Kay [LK95] um eine klarere Abgrenzung zwischen dem sensorbezogenen Transducerbegriff und dem eigentlichen Fusionsvorgang und führen für letztgenannten den Begriff der „Translation“ ein.

Definition 2 (Transducer)

Der Begriff Transducer bezeichnet eine Hardware, die einen Eingangswert auf einen Ausgabewert abbildet, wobei sich diese in Art, Energieform usw. unterscheiden können. Der Begriff umfasst damit als Generalisierung sowohl Aktoren als auch Sensoren und Einzelkomponenten dieser Systeme.

3.1.3 Integrierter Sensor

Mit fortschreitender Miniaturisierung und Verbreitung von Sensorkomponenten zielt die Entwicklung auf die Umsetzung von integrierten Bauteilen, die die Diskretisierung der einzelnen Bestandteile überwindet und diese in gemeinsamen Baugruppen integriert. Mit der Verkürzung der analogen Kommunikationswege und einer sensornahen Digitalisierung werden elektrische Effekte wie Rauschen, Übersprechen sowie elektromagnetische Inferenzen reduziert und die Güte der Messung gesteigert. Gleichzeitig können der Platzbedarf und ab einer gewissen Stückzahl die Fertigungskosten reduziert werden. Mit der Kapselung in einem Bauteil sinkt zudem die Anfälligkeit gegenüber externen Störungen.

Abbildung 3.1 illustriert die verschiedenen Umsetzungsvarianten der Integration. Die Abbildung 3.1(a) stellt dabei eine häufig umgesetzte Konstellation dar, bei der zwei Baugruppen nebeneinander bestehen. Zum einen wird der Sensor mit der nachfolgenden analogen Signalverarbeitung zu einem Sensorelement [Bib06, TO98] verschmolzen. Auf der anderen Seite steht der Mikroprozessor mit der entsprechenden Peripherie, die hier durch den Analog-to-Digital-Converter (ADC) dargestellt wird. Während der Mikrocontroller mit dem ADC möglichst universell aufgestellt sein soll, ist die Signalkonditionierung

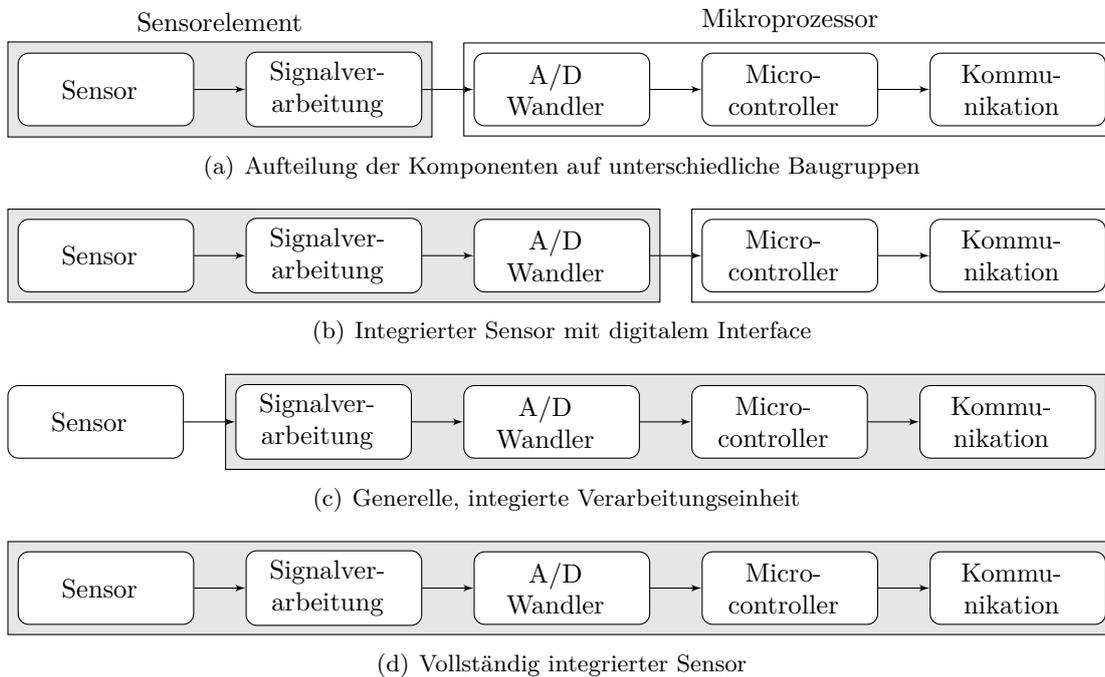


Abbildung 3.1: Ausprägungen des integrierten Sensors, erweiterte Darstellung aus [Fra00]

speziell auf die Anforderungen des spezifischen Sensors zugeschnitten. Diese Spezialisierung wird in der Einbindung des ADCs in das Sensorelement, wie in Abbildung 3.1(c) zu sehen, fortgesetzt. Damit verschiebt sich die Schnittstelle zwischen Spezialisierung und Generalisierung in Richtung des Controllers. Der umgekehrte Weg wird in Abbildung 3.1(c) gegangen, der eine unabhängige Handhabung des Sensors ermöglicht. Abbildung 3.1(d) zeigt den letzten Grad der Integrationslevel, die vollständige Umsetzung aller Bestandteile in einem Chip. Derartig hochgradig kompakte Lösungen auf der Basis eines gemeinsamen Siliziumträgers werden mit zunehmender Tendenz insbesondere im Automobilbau, das heißt also in Anwendungen mit besonders hohen Stückzahlen, verwendet [Wyc03].

Ein möglicherweise auftretender Nachteil des integrierten Sensors ist die mit der Einhausung verbundene Senkung der Sensitivität. Hier besteht ein Optimierungsproblem, das beispielsweise beim Sensoreinsatz im Automobilbereich entsteht, wenn ein Sensor-knoten mit einer Kunststoffummantelung vor Feuchtigkeit geschützt werden soll, gleichzeitig aber auch die Messgröße durch die Plastikschiicht gedämpft wird. Zudem geht mit der Integration der Elemente in eine Baugruppe eine hohe Spezialisierung einher, die neben dem Messverhalten des Sensors, der Konfiguration der Signalverarbeitung und

der Performanz des eingesetzten Mikrocontrollers auf einen Einsatzzweck zugeschnitten wird. Individuelle Anpassungen sind damit nur in beschränktem Maße möglich.

Die im Folgenden beschriebenen Konzepte des Smart Sensors und Smart Transducers sind im Grunde unabhängig vom Integrationsniveau der Hardware, die Erwartung an einen solchen Sensor zielt aber auf einen kompakten Hardwarebaustein ab, der die relevanten Funktionalitäten umsetzt.

3.1.4 Smart Sensor

Der Begriff des Smart Sensor ist, entsprechend der Darstellung in Abbildung 3.2, von unterschiedlichen Leitbildern geprägt. Zum einen werden funktionale Anforderungen als Kernelement definiert (insgesamt 67%), unter denen der Selbsttest der Sensoreinheit besonders hervorgehoben wird. Zum anderen wird die integrierte Hardware (18%) als ein Kernmerkmal benannt. In erstaunlich geringem Maße wird auf eine Definition durch Institute of Electrical and Electronics Engineers (IEEE) verwiesen (1%).

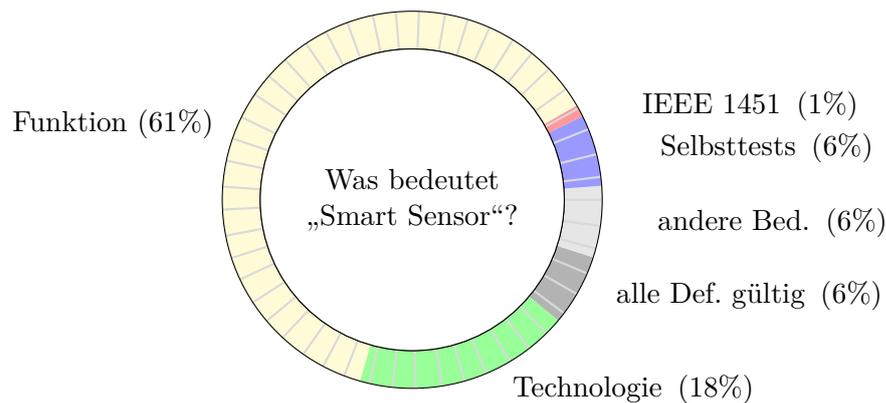


Abbildung 3.2: Umfrage unter 227 Industrieanwendern und Ingenieuren hinsichtlich der Definition eines „Smart Sensors“, übernommen aus [Int09]

Technologie Die hardwareseitige Ausstattung eines Smart Sensors deckt sich mit der eines integrierten Sensors. So stellen Tränkle und Obermeier [TO98] dar, dass der gleichbedeutend benutzte, deutsche Begriff des „intelligenten Sensors“ nach ihrer Auffassung lediglich an das Vorhandensein einer digitalen Busschnittstelle geknüpft ist. Unter anderem erweitern Kirianaki u. a. [Kir+02] diesen Hardwarekatalog und sehen im „Smart Sensors“ ein System, das neben der Hardware zur Erfassung von Messwerten auch die Voraussetzungen zu deren Verarbeitung und Kommunikation erfüllt. Hinsichtlich der Kommunikation kommen unterschiedliche drahtgebundene (Controller Area Network

(CAN), Inter-Integrated Circuit (I²C), Profibus usw.) und drahtlose (zum Beispiel Zig-Bee [Zig03]) Feldbusse oder ethernetbasierende Protokolle zum Einsatz.

Funktionale Definition Wie die Auswertung unter [Google Ngram](#) zeigt, findet der Begriff des Smart Sensor oder Intelligenen Sensors seit etwa 1975 mit der heutigen Bedeutung Eingang in die Literatur. Mit der Smart Camera definiert Schneidermann [Sch75] eine Ausformung der intelligenten Sensorik. Die Autoren interpretieren eine Kamera als intelligentes autonomes System, wenn neben der Bilderfassung anwendungsbezogene Filterfunktionen übernommen und gegebenenfalls Benachrichtigungen über bestimmte Ergebnisse dieser Berechnungen publiziert werden. Breckenridge und Katzberg [BK80] erweitern diese Definition 1980 auf einen allgemeinen Sensorbegriff und fordern, dass ein Smart Sensor folgende Merkmale erfüllen muss:

1. Umsetzung einer Berechnungsfunktionalität
2. Ableitung einer Entscheidung basierend auf den Sensordaten
3. Kommunikation mit mindestens einem anderen Smart Device

Kopetz, Holzmann und Elmenreich [KHE00] greifen den letztgenannten Punkt nicht auf, erweitern aber das Spektrum der Funktionsdefinitionen:

„The smart sensor transforms the raw sensor signal to a digital representation, checks and calibrates the signal, and transmits this digital signal via a secure communication protocol to its users.“

Abbildung 3.2 macht deutlich, dass neben der allgemeinen Forderung hinsichtlich der Funktionalität eine inhärente Fehlererkennung und Fehlertoleranz von einem Teil der Anwender intelligenter Sensoren erwartet wird. Weiler [Wei01] konkretisiert und erweitert diesen Anspruch und definiert dafür den eigensicheren intelligenten Sensor. Dieser umfasst Methoden zur Diagnose und Fehlererkennung, übermittelt dem Empfänger der Messdaten zusätzliche Informationen zum Zustand und Fehlermodus des Smart Sensors und bietet ein fehlertolerantes Verhalten, das den kurzzeitigen Ausfall des eigentlichen Sensors mit geeigneten Signalmodellen und Interpolationsmechanismen überbrücken kann.

Die geforderten Selbsttests zur Detektion eines anormalen Zustandes können in zwei Richtungen ausgeführt sein: zum einen als reine Softwarelösung, die das Messergebnis anhand sensorspezifischer Fehlermodelle auf seine Gültigkeit prüft [Ise06] und zum anderen in Form von hardwarebasierten Checks der Funktionalität. Dies kann zum Beispiel bedeuten, dass ein Sensorsystem über einen zusätzlichen physikalischen Eingang verfügt, über den definierte analoge Signale in das System eingebracht werden können, um die Schaltungstechnik, Parametrisierung und Verarbeitung zu validieren [Zil08]. In anderer Konfiguration können solche Tests auch als Online-Überwachung ausgeführt werden.

Begriff nach IEEE 1451 Wie noch im Detail in Abschnitt 3.2.1 zu beschreiben ist, definiert der IEEE 1451 Standards für verteilte Sensor/Aktor-Anwendungen auf der Basis von allgemeingültigen, netzwerkunabhängigen Schnittstellen zwischen den beteiligten Komponenten [IEE97]. Kern des Ansatzes ist Umsetzung von Portabilität, Netzwerknabhängigkeit und Interoperabilität. Dafür teilt der Standard die in Abschnitt 3.1.3 illustrierten Komponenten eines intelligenten Sensors auf zwei Grundelemente auf, das Smart Transducer Interface Modul (STIM) und den Network Capable Application Processor (NCAP). Der STIM dient dabei als Interface und Abstraktion der eigentlichen Sensorik und umfasst grundlegende Funktionalitäten zur Kalibrierung, Sensorabfrage usw. Die Messdaten werden in einem in den Abschnitten 2, 3 sowie 5 und 6 des Standards definierten Protokoll über eines der unterstützten Feldbussysteme an den NCAP weitergereicht, der die Verarbeitung und Verbreitung via ethernetbasiertem Netzwerk übernimmt [SL08].

In Übereinstimmung mit dem geringen Anteil an Votierungen der in Abbildung 3.2 gezeigten Umfrage bezweifeln verschiedene Autoren wie Sankaranarayanan [San07] das Konzept des IEEE 1451 wegen der mangelnden Durchdringung des Marktes mit Smart Sensor Applikationen. Trotz der Bemühungen von internationalen Messsystemherstellern sieht er keinen kommerziellen Erfolg des Standards mangels entsprechender Nachfrage, da ein echter Einsatz neben der Forschung nur im Bereich der Raumfahrt bemerkbar ist.

In der weiteren Entwicklung dieser Arbeit wird der Auffassung der Mehrheit der Umfrage gefolgt und der Begriff des Smart Sensor als funktionale Definition verstanden. Damit baut der Smart Sensor seine Funktionalität auf den Konzepten des integrierten Sensors auf.

Definition 3 (Smart Sensor)

Ein Smart Sensor bezeichnet einen Transducer, der neben der sensorischen Umgebungserfassung eine digitale Verarbeitung der Messdaten und die Verbreitung dieses Ergebnisses leistet. Die damit verbundene Steigerung der Güte und Aussagekraft einer Messung kann auf Selbsttests, Filter-, Fusions- oder Fehlerdetektionsalgorithmen basieren.

3.1.5 Smart Transducer

Obwohl der Standard nach IEEE 1451 in die Nähe des Begriffes Smart Sensor gerückt wird, beschreibt dieser analog zu vergleichbaren Ansätzen der Object Management Group (OMG) [Obj03] im Kern einen Smart Transducer. Dieser Begriff überführt die Konzepte der intelligenten Sensorik allgemein auf Sensor/Aktor Anwendungen, integriert also Knoten, die ohne eigne Sensorik Fusionsaufgaben wahrnehmen oder einen Aktor ansteuern. Den zweiten Punkt sprechen Delavai, Eisenmann und Elmenreich [DEE03] in ihrer Definition an:

„A smart transducer is a sensor or actuator element that is integrated with a processing unit and a communication interface. The processing unit transforms the raw sensor signal to a digital representation, checks and calibrates

the signal, and transmits the digital information to its users via a standardized communication interface.“

Ein Smart Transducer hebt sich vom Smart Sensor über den flexiblen und übergreifenden Datenaustausch ab. Damit zielt der Begriff des Smart Transducers auf die in dieser Arbeit intendierte Verbreitung von Daten in veränderlich zusammengesetzten Applikationen ab. Eine genaue Beschreibung der Definitionen des OMG-Standard folgt in Abschnitt 3.2.1.

Definition 4 (Smart Transducer)

Der Smart Transducer stellt für den Datenaustausch in verteilten Applikationen eine einheitliche Kommunikationsschnittstelle und ein Nachrichtenformat bereit, sodass ein flexibler Datenaustausch in dynamischen Umgebungen gewährleistet werden kann.

3.1.6 Einordnung des *Smart-X*

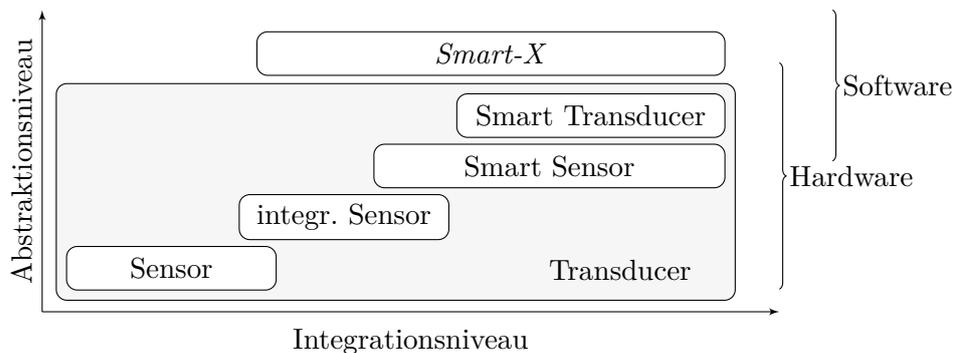


Abbildung 3.3: Abstufung des Abstraktionsgrades der verschiedenen Sensorbegriffe, motiviert aus [Bra11]

Abbildung 3.3 zeigt eine Systematik der bisher eingeführten Definitionen und ordnet den *Smart-X* in diesem Schema ein. In der kartesischen Darstellung wird das Abstraktions- und Integrationsniveau des klassischen Sensorenbegriffes, des Transducers, des integrierten Sensors sowie des Smart Transducers und Smart Sensors gegenübergestellt. Das geringste Abstraktions- und Integrationspotenzial zeigt der einfache Sensor nach Definition 1, während der Transducer als allgemeinste Abstraktion alle anderen Kategorien einschließt (grauer Bereich). Die Programmierkonzepte des Smart Transducers und des Smart Sensors bauen auf dem erreichten Integrationsniveau auf und erweitern die Abstraktion durch zusätzliche Funktionen und eine Kapselung der Knoten und Sensoreigenschaften. Der Smart Transducer erreicht mit dem die Sensorik, Verarbeitung und Aktorik erfassenden Konzept ein höheres Abstraktions- und Integrationsniveau.

Für die Umsetzung der geforderten dynamischen Datenerfassung und -verarbeitung verschmelzen im *Smart-X* verschiedene Konzepte der genannten Modelle intelligenter Sensoren wie zum Beispiel das elektronische Datenblatt des Smart Sensors oder die generische Kommunikationsschnittstelle des Smart Transducers. Gleichzeitig erweitert das Konzept des *Smart-X*, wie der Smart Transducer, die Programmierabstraktion ausgehend vom eigentlichen Sensorknoten auf Verarbeitungseinheiten und Aktoren. Im Unterschied zum Smart Sensor und in Einschränkungen auch zum Smart Transducer bezieht sich der Begriff des *Smart-X* immer auf eine Softwarekomponente, während zuvor genannte Begriffe eine physische (Sensor-)Baugruppe mit bestimmter Funktionalität meinen. Aus dieser Fokussierung des Begriffes auf eine Architektur und damit auf die Software resultiert die größere Variabilität des Ansatzes.

Das Abstraktionsniveau des *Smart-X* steigt dabei gegenüber dem Smart Transducer nochmals an, weil in der *Smart-X*-Architektur verschiedene Abstraktionen zusammengeführt sind. Unter anderem wird neben einem allgemeingültigen Datenmodell, generalisierten Hardwareschnittstellen usw. eine Validitätseinschätzung auf der Basis einer die verschiedenen Fehlertypen umfassenden Semantik integriert. Erst das Zusammenspiel der verschiedenen Abstraktionen erlaubt die angestrebte Flexibilität während der Entwicklung und die nötige Adaptivität zur Laufzeit.

Definition 5 (Smart-X)

Die Architektur des Smart-X kombiniert die funktionsbezogene Definition des Smart Sensors mit dem übergreifenden, nicht nur auf Sensoren beschränkten Anspruch des Smart Transducers. Im Smart-X werden darüber hinaus Konzepte und Abstraktionen der Fehlertoleranz und adaptiven Verarbeitung und Fusion integriert, um im Zusammenspiel dieser Ansätze verteilte, dynamisch interagierende Applikationen zu ermöglichen.

3.2 Programmierung von Smart Devices

Nach der begrifflichen Einordnung werden in den folgenden Absätzen bestehende Abstraktionen für die Umsetzung von intelligenten Sensoren vorgestellt und auf ihre Anwendbarkeit im Kontext der Anforderungen aus Abschnitt 2 überprüft. Ein Fokus der Untersuchung liegt dabei auf der Adaptivität und den Fehlertoleranzmechanismen der Modelle.

3.2.1 Standards und Spezifikationen

Im Folgenden werden relevante Umsetzungen der Programmierabstraktionen für verteilte Sensor-Aktor-Systeme vorgestellt und in ihren Eigenschaften verglichen.

IEEE-1451

Der IEEE-Standard 1451.x beschreibt eine Architektur nebst allgemeinen, netzwerkübergreifenden Schnittstellen für die Interaktion von Sensoren und Aktoren [IEE97]. Der Hauptfokus des Standards liegt auf der Verbreitung eines allgemeingültigen Interfaces für Sensoren, das Plug&Play-Vorgänge unterstützt. Dafür führt der Standard zwei Basisbausteine ein. Ein Sensorknoten gemäß IEEE 1451.x besteht aus zwei Grundelementen, dem NCAP und dem STIM. Der NCAP, dessen Aufbau und Schnittstellen in IEEE 1451.1 definiert werden, realisiert einen portablen, netzwerkunabhängigen Zugriff auf die Funktionalität und die Informationen des STIM, also die angeschlossene Sensorik. Im Fall einer drahtlosen Anbindung wird häufig vom Wireless Transducer Interface Module (WTIM) gesprochen. Das STIM oder (WTIM) Modul umfasst die notwendigen Komponenten für die Integration des Sensors, die Signalaufbereitung und -konvertierung sowie die Schnittstelle zur Kommunikation mit einem NCAP und eine komprimierte Transducer Electronic Data Sheet (TEDS) Datenblattsammlung. Für die Kommunikation mit dem NCAP wurde zunächst das Transducer Independent Interface (TII) definiert, ein auf zehn Drähten basierender Bus, der den Serial Peripheral Interface (SPI) erweitert. Zwischenzeitlich wurde der Standard um weitere Busprotokolle ergänzt und beinhaltet nun unter anderem die Spezifikation für den Datenaustausch per USB oder für WTIMs über IEEE 802.15.1 (Bluetooth) oder IEEE 802.15.4 (ZigBee). Der NCAP setzt sich aus einer STIM-Schnittstelle auf der einen und einem Netzwerkinterface auf der anderen Seite sowie einem Applikationsblock zusammen. Innerhalb der STIM-Schnittstelle ist ein TEDS-Parser integriert, der die Inhalte der Datenblätter verarbeitet und unter anderem die Correction Engine als weiteren Bestandteil des Interfaces steuert. Hier werden die rohen Sensordaten mit den vom Datenblatt vorgegebenen Einheiten kombiniert und zum Beispiel entsprechend den Kalibrierungsvorgaben angepasst.

Die mehrschichtigen elektronischen Datenblätter stellen ein herausragendes Merkmal des IEEE-Standards dar. Sie erlauben es, eine Vielzahl von Sensoren und Aktoren detailliert zu beschreiben. Das Datenblatt setzt sich dabei unter anderem aus folgenden einzelnen TEDSs zusammen:

Meta-TEDS In dieser Beschreibung werden die grundlegenden Informationen zum STIM wie das Zeitverhalten, die Anzahl der angeschlossenen Sensoren/Aktoren sowie entsprechende Kanalzuordnungen zusammengefasst. Die Informationen zum Zeitverhalten können zum Beispiel im NCAP genutzt werden, um eventuelle Ausfälle der STIMs zu erfassen.

Channel TEDS Dieser TEDS-Typ hält die spezifischen Daten jeweils eines angeschlossenen Sensors/Aktors bereit. Dazu zählt der Messbereich, die physikalische Einheit der Messung, Informationen zur Messungenaugigkeit, Anlaufzeit usw.

Users transducer name TEDS An dieser Stelle steht es dem Nutzer offen, individuelle Namenskonventionen einzutragen und für die Kategorisierung und Inventarisierung der Sensoren zu nutzen.

PHY TEDS Trotz der weitreichenden Standardisierung der Schnittstelle zwischen NCAPs und STIMs anhand verschiedener Protokolle und Übertragungsmedien werden verschiedene Kommunikationsparameter, wie zum Beispiel die Datenrate, mit einem definierten Anfangswert belegt. Danach kann, sofern eine Übereinstimmung zwischen NCAPs und STIMs besteht, auf eine individuelle Rate gewechselt werden.

Calibration TEDS Die für die Kalibrierung relevanten Daten wie zum Beispiel das letzte Kalibrierdatum, das Kalibrierungsintervall sowie die Kalibrierungsrandbedingungen werden in diesem Datenblatttyp erfasst. Die Kalibrierungsfunktion, also die Beschreibung der Abbildung von Messdaten auf Ausgabewerte, kann entweder als Look-up-Table oder als polynomiale Beziehung hinterlegt werden.

Transfer Function TEDS und **Frequency Response TEDS** geben die Möglichkeit, das dynamische Verhalten eines Sensor/Aktors zu hinterlegen. Ausgehend von einer bekannten Anregung in einem geschlossenen Regelkreis kann somit eine Prüfung des Antwortverhaltens anhand dieser mathematischen Redundanz vorgenommen werden.

End user application specific TEDS bilden lediglich den Rahmen für die Integration von nutzerspezifischen Einträgen. Für den Fall, dass diese im gesamten Netzwerk verfügbar sein sollen, muss der TEDS-Parser des NCAP entsprechend angepasst werden.

Die vier erstgenannten TEDSs sind für einen Sensor/Aktor nach dem IEEE-1451-Standard verbindlich. Von den darüber hinausgehenden, optional zum Einsatz kommenden TEDS sind nur die wichtigsten in obiger Aufstellung erfasst worden. Die Flexibilität des Datenblattmodells ergibt sich dabei aus dessen Skalierbarkeit, Anpassungsfähigkeit und Erweiterbarkeit. Ein minimales TEDS hat eine Größe von 178 Byte, die auf dem STIM im nichtflüchtigen Speicher bereitzuhalten sind. Die von einem der führenden Partner des Standards vertriebene Mess- und Programmierumgebung LabView nutzt die elektronischen Datenblätter zur Parametrisierung der Messdatenerfassung und -dokumentation. Diese können bei Sensoren, die dem Standard IEEE 1451 entsprechen, in deren Speicher abgelegt sein oder aber für konventionelle Sensoren auf einem Server liegen, wobei die Zuordnung durch den Entwickler vorgenommen werden muss [SS07, Nat09].

Eng verbunden mit dem TEDS-Konzept ist die Komponierbarkeit der Elemente zur Laufzeit. Die Plug&Play-Eigenschaften nach IEEE 1451 beziehen sich auf die Verbindung zwischen NCAP und STIM, sodass interfacekompatible STIMs auch zur Laufzeit mit beliebig vielen NCAPs verknüpft werden können. Eine flexible Integration von zusätzlichen NCAPs in ein bestehendes Netz ist dagegen vom Standard nicht vorgesehen.

Für die Implementierung des *MOSAIC*-Frameworks wurde die Methodik der Einheitencodierung aus IEEE 1451 aufgegriffen, die an dieser Stelle kurz vorgestellt werden soll. Die neun Basiseinheiten des *Système international d'unités* (SI), die allen physikalischen Einheiten zugrunde liegen und in nachfolgender Tabelle aufgeführt sind, werden in der TEDS Darstellung durch einen Gewichtungswert repräsentiert. Dieser berechnet sich zu $(2 \cdot \text{exponent}) + 128$. Damit erlaubt es die Codierung, beliebige zusammengesetzte Einheiten zu bilden, die unter Zuhilfenahme geeigneter Bibliotheken in Berechnungsoperationen einbezogen werden können.

Tabelle 3.1: Einheitencodierung nach IEEE 1451 am Beispiel von Pascal ($1 \text{ Pa} = 1 \text{ m}^{-1} \text{ kg s}^{-2}$)

	Radiant	Steradian	Meter	Kilogramm	Sekunde	Ampere	Kelvin	Mole	Candela
Exponenten	0	0	-1	1	-2	0	0	0	0
TEDS Kodierung	0	128	126	130	124	128	128	128	128

Rossi u. a. [Ros+09] beschreiben eine Implementierung des IEEE-1451-Standards. Dabei werden der STIM als Field Programmable Gate Array (FPGA) und der NCAP auf einem PC umgesetzt. Für die Kommunikation zwischen beiden Komponenten wurden dem ursprünglichen Standard entsprechende TII-Schnittstellen eingefügt. Die Autoren sind der Auffassung, dass mit der getrennten Implementierung von STIM und FPGA die Anpassungsfähigkeit an verschiedene Szenarien gewährleistet sei. So könne ein anderer Sensortyp durch die Adaption der Very High Speed Integrated Circuit Hardware Description Language (VHDL) auf hoher Abstraktionsebene umgesetzt werden. Auf der anderen Seite kann die NCAP-Implementierung mit Java zum einen auf einer Vielzahl von Systemen ausgeführt werden und zum anderen erlaubt die Hochsprache eine einfache Anpassung an individuelle Netzwerke und Protokolle für die Kommunikation der Daten. Weitere Arbeiten setzten den Standard für andere STIM-Zielplattformen oder -kommunikationsverbindungen um [CRS00, Wan+05]. Interessant im Hinblick auf die noch vorzustellende Implementierung von *MOSAIC* für Simulink ist die Arbeit von Cámara u. a. [Cám+04], die eine automatische Generation des STIM-Codes ausgehend von einem TEDS für eine vorkonfigurierte Zielplattform übernimmt.

OMG STIS

Die Object Management Group (OMG) präsentiert mit der Smart Transducer Interface Specification [Obj03] ein Framework für die Implementierung von echtzeitfähigen, ver-

teilten Anwendungen, die in heterogenen Netzen Anwendung finden [KHE00]. Grundlage dafür ist das Programmiermodell eines Services, das in Abbildung 3.4 gezeigt wird.

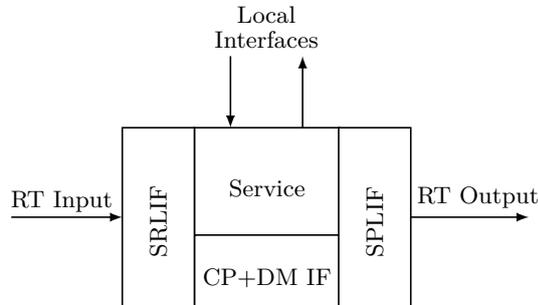


Abbildung 3.4: Interfaces eines Services nach [EPS04]

Ein Service wird über lokale Schnittstellen, echtzeitfähige Ein- und Ausgänge, sowie über Interfaces zur Konfiguration, Planung und Diagnose definiert, die die Basis eines Service Tasks bieten. Die erstgenannten Interfaces dienen zur Anbindung einer Sensorik/Aktorik von Displays oder Eingabegeräten. Die als Service Providing Linking Interfaces (SPLIF) und Service Requesting Linking Interfaces (SRLIF) bezeichneten Schnittstellen bilden die Programmierabstraktion für die Echtzeitkommunikation. Über das sowie das Diagnostic and Management Interface (CP+DM) können interne Zustände abgefragt oder Parameter des Knotens neu bestimmt werden. Ein Service kann applikationsabhängig aus einer Untermenge der genannten Komponenten bestehen [EPS04].

Die Kommunikation zwischen verschiedenen Services, die auf einer gemeinsamen Zielplattform laufen, erfolgt mittels Shared-Memory-Mechanismen, die durch ein Interface File System (IFS) abstrahiert werden. Für eingebettete Knoten wird mit Time-Triggered Protocol Class A (TTP/A) auf einen zeitgesteuerten Feldbus zurückgegriffen [KHE00]. Eine Implementierung auf der Grundlage von Universal Asynchronous Receiver Transmitter (UART) Verbindungen ist in [Elm02] beschrieben. Weiterhin können Services mittels Gateway in ebenfalls durch die OMG spezifizierte Common Object Request Broker Architecture (CORBA)-Netze eingebunden werden. Das die unterschiedliche Kommunikationshardware und -paradigmen kapselnde IFS bietet ein Adressierungsschema für bis zu 256 Knoten mit je 64 Files. Die Dateien, beziehungsweise die zugeordneten Speicherbereiche auf einem Knoten, sind zum Beispiel zur Definition der Kommunikation sowie deren Parametrisierung obligatorisch vorhanden oder für allgemeine Nutzdaten vom Nutzer festzulegen. Bemerkenswert ist die von Elmenreich [Elm02] beschriebene dynamische Erweiterung des TTP/A Netzes um weitere Knoten und damit des IFS zur Laufzeit [Elm+02].

Wie in [Pit02] ausführlich dargestellt wird, nutzt die Smart Transducer Interface Specification zwei Typen von Extensible Markup Language (XML) Beschreibungen: die

knotenspezifischen Smart Transducer Descriptions (STD) und die hierarchisch höher angesiedelten Cluster Configuration Descriptions (CCD). Wichtig für diese Arbeit ist die Einteilung in statische und nicht-statische Beschreibungsinhalte des STD. Entsprechend werden die unveränderlichen Sensorinformationen mit dynamischen Elementen wie der logischen TTP/A-Adresse kombiniert. In [EPS04] wird eine Erweiterung des Beschreibungskonzepts vorgestellt. Darin werden die Services analog zu den Methoden der SensorML [Rob06] beschrieben und in einem übergreifenden Applikations-XML zusammengeführt. Diese formellen Systembeschreibungen werden aber nicht in den Entwicklungsprozess eingebunden.

Messdaten können im Hinblick auf eine quantifizierbare oder angenommene Gültigkeit mit einem 16-stufigen Validitätswert versehen werden. Diese Granularität wurde offenbar gewählt, um den Kommunikationsaufwand nicht aufzublähen. Eine allgemeingültige Herleitung beziehungsweise Methoden zur Gewichtung oder Vergleichbarkeit werden nicht vorgestellt [KHE00].

Der Ansatz ist für *MOSAIC* aus zwei Richtungen interessant. Zum einen wird ein dynamischer Datenaustausch zwischen heterogenen Knoten mit einer übergreifenden Abstraktion sichergestellt. Zum anderen werden die Messdaten einer Validitätsprüfung unterzogen, die für die weitere Verarbeitung herangezogen wird.

3.2.2 Programmiermodelle und ihre Implementierungen

Abstract Sensor

Marzullo [Mar90] zeigt in seiner Arbeit die fehlertolerante Erfassung von gleichartigen Sensordaten in einem verteilten System. Dafür werden die als Concrete Sensors bezeichneten Sensoren in ihrer individuellen Ausprägung durch einen Abstract Sensor gekapselt. Dieser übernimmt die Abbildung der realen Messung auf einem wertkontinuierlichen Fenster, wobei eine gleich verteilte Streuung des Wertes angenommen wird. Entsprechend der Darstellung in Abbildung 3.5, fasst ein nachgeordneter fehlertoleranter Abstract Sensor n Abstract Sensors mit einer gleichartigen Messgröße zusammen und schließt dabei, ausgehend von einer vordefinierten Anzahl von maximal gleichzeitig möglichen Fehlern, auf eventuell defekte Messsysteme. Diese werden dann von der eigentlichen Fusion ausgeschlossen. Marzullo weist mathematisch nach, dass es für verschiedene Fehlertypen einer genau bestimmbare Mindestanzahl an Abstract Sensors bedarf, um diese sicher zu detektieren. Verschiedene Autoren greifen zum Beispiel L. Prasad u. a. [L+93] oder Jayasimha [Jay94] den Fehlerdetektionsansatz des Abstract Sensors auf, erweitern die Anwendbarkeit auf zweidimensionale Sensoren und stellen einen verfeinerten Algorithmus zur Fehlerdetektion vor. Qi, Iyengar und Chakrabarty [QIC01] vergleichen die Resultate dieser Ansätze anhand eines Beispiels und diskutieren die Vor- und Nachteile.

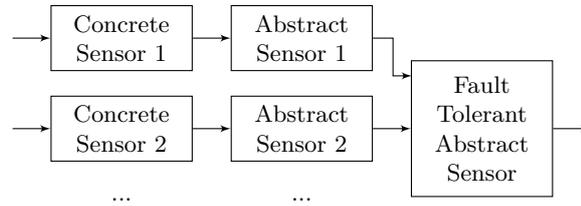


Abbildung 3.5: Marzullo's Fault Tolerant Sensor Fusion nach [Mar90]

Das Konzept des Abstract Sensor ist durch zwei Eigenschaften bestimmt: Erstens erfolgt hier die Ableitung einer definierten Ausgabe (Messwert, gleichgewichtetes Vertrauensintervall) auf der Basis von beliebigen wertkontinuierlichen Sensordaten, was die grundlegenden Sensorspezifika gegenüber dem Nutzer verbirgt. Zweitens wird auf der Basis dieser Informationen ein Selektions- und Fusionsprozess durchgeführt, der, ausgehend von einer systemspezifischen Fehlerschätzung im Zusammenspiel dieser standardisierten Daten, eine Steigerung der Güte des Messergebnisses erzielt.

Problematisch am Ansatz von Marzullo ist die Annahme der wertkontinuierlichen Messgrößen mit einer gleich verteilten Streuung für konkurrierende Messungen. Diese Festlegung schränkt die Übertragung des Fehlerdetektionsansatzes auf realistische Sensoren mit abweichenden, komplexen Messfehlermodellen stark ein. Kooperative Sensornetze oder normal verteilte, diskrete Werte kann das Verfahren in der bestehenden Form nicht handhaben. Daneben geht die Arbeit von einer vollständig synchronen Messung aus. Entsprechend sind für davon abweichende Szenarien zusätzliche Synchronisationsmethoden zu integrieren, die die Streubreite und den Messwert in Abhängigkeit von der Zeit anpassen.

Marzullo definiert mit seiner Arbeit eine Programmierabstraktion, deren Aufgabe es ist, die Eigenschaften des Sensors (Messdatenformat, Fehlerparameter usw.) zu kapseln. Die Verarbeitung dieser Informationen erfolgt in einer zentralen Einheit, deren Verarbeitungsalgorithmus grundsätzlich für eine variierende Zahl von Datensätzen geeignet ist.

Virtual Sensor

Ein Teil der Literatur bezeichnet Softwarekomponenten als virtuelle Sensoren, die nicht oder nur eingeschränkt vorhandene reale Sensoren ersetzen, vgl. zum Beispiel [KPJ06]. Diese kommen zum Einsatz, wenn eine aufwendige oder wegen der Umgebungsbedingungen technisch schwierige Instrumentierung durch ein mathematisches Modell ersetzt werden kann. Virtuelle Sensoren, in [PH00] auch als Soft-Sensoren beschrieben, werden seit etwa 20 Jahren industriell in großem Umfang genutzt [ST97].

Eine andere Definition virtueller Sensoren trifft Kirianaki u. a. [Kir+02]. Der Autor ordnet dieser Bezeichnung PC-Systeme zu, die dem Nutzer auf der Basis einer GUI

Sensorinformationen als Messwerkzeug anbieten. Es wird darauf hingewiesen, dass durch die Virtualisierung eine größere Flexibilität möglich ist als mit einem herkömmlichen Messinstrument. Auf dieser Basis einer geeigneten Hardware (Data Acquisition (DAQ)) ließen sich Visualisierung, Fusion und Auswertung umsetzen. Andere Autoren, wie zum Beispiel die Firma National Instruments, sprechen in Verbindung mit ihren derartig ausgerichteten Messtechnikprodukten [Nat09] von Virtuellen Instrumenten.

In Abgrenzung dazu finden sich bei Bose u. a. [Bos+07] der Begriff des Virtual Sensors als Programmierabstraktion im Rahmen einer hierarchischen Architektur zur Datenerfassung und Fusion. Zwischenzeitlich wird eine Implementierung im Zusammenhang mit einer Kommunikationsmiddleware als kommerzielles Softwarepaket angeboten. Virtuelle Sensoren dienen dabei als Abstraktion von Sensordaten auf drei Ebenen:

Singleton Virtual Sensors bilden das Interface zu physikalischen Sensoren und ordnen den Messwerten weitergehende Informationen (Position, ID etc.) zu. Jedem realen Sensor wird genau ein Singleton Virtual Sensor zugeordnet, wenn er ins Netzwerk integriert wird.

Basic Virtual Sensors kombinieren die Ergebnisse mehrerer gleichartiger Singleton Virtual Sensors, um ein Ausgabe mit einem höheren Vertrauensgehalt zu produzieren, oder dienen als Filter für einzelne Singleton Virtual Sensors. Basic Virtual Sensors ebenso wie die folgenden Derived Virtual Sensors werden online, ausgehend von den Anfragen der Nutzer dynamisch durch den Frameworkcontroller erzeugt.

Derived Virtual Sensors bilden die letzte und höchste Abstraktionsebene. Diese greifen auf verschiedene Basic Virtual Sensors zu, um deren Werte zu komplexen Resultaten zu kombinieren, zum Beispiel gilt Wetter=(Luftfeuchte, Temperatur). Derived Virtual Sensors können im Modell von Bose u. a. immer nur auf die Basic Virtual Sensors zugreifen. Der Austausch zwischen Derived Virtual Sensors ist nicht vorgesehen.

Die nötigen Informationen für die Interpretation der Ausgaben des verschiedenen Levels werden mit der Initialisierung in einer zentralen Datenbasis erfasst. Diese umfasst, neben der Beschreibung der Sensoren, auch eine Modellannahme der beobachtbaren Phänomene. Gleichzeitig kann hier ein für alle auszutauschenden Datensätze verbindlicher Grenzwert hinterlegt werden, der die Schwelle der Fehlerwahrscheinlichkeit festlegt, bis zu der ein Datensatz verbreitet wird.

Gegenüber der Applikation werden die Virtual Sensors durch einen Framework Controller gekapselt. Damit lassen sich die Nutzeranfragen, die in einer der Structured Query Language (SQL) nachempfundenen Sprache definiert werden (zum Beispiel `SELECT "WEATHER" from "A"`) indirekt auf den Daten bestimmter Sensoren abbilden. Dazu überprüft der Frameworkcontroller zunächst die vordefinierten Singleton Virtual Sensors, für das genannte Beispiel die Temperatur- und Luftfeuchtemessgeber im Sektor A. Danach werden die nötigen Basic Virtual Sensors und Derived Virtual Sensors generiert und als

Services innerhalb einer Open Services Gateway initiative (OSGi) Umgebung gestartet. Mit dem Ausfall einzelner Elemente erfolgt eine Rekonfiguration der Struktur.

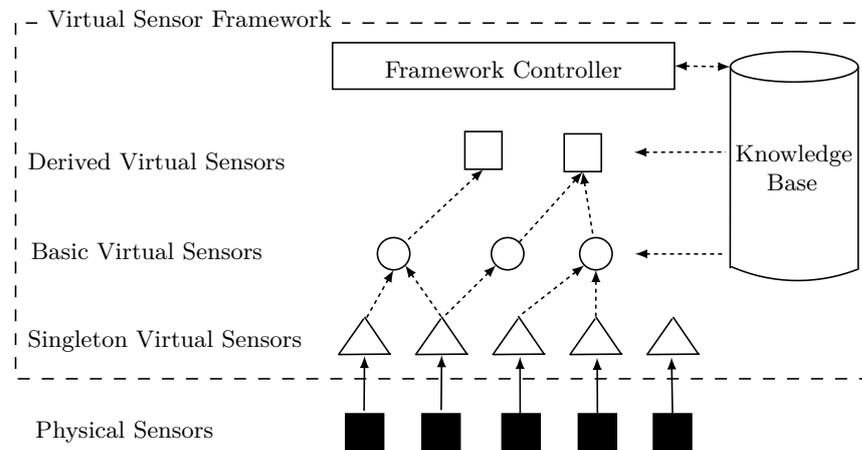


Abbildung 3.6: Hierarchie des Konzeptes der Virtual Sensors nach Bose u. a. [Bos+07]

Der Virtual Sensor trifft drei Kernforderungen des *Smart-X*, die dynamische Allokation der Daten, die Integration von Fehlerdetektionsmechanismen und eine adaptive Verarbeitungskette, die aber über eine zentrale Instanz, den Framework Controller, koordiniert wird.

Sentient Object

Die Autonomie der Komponenten bei gleichzeitiger Anonymität der Kommunikation wurde durch Casimiro, Kaiser und Verissimo [CKV04] als maßgebliche Voraussetzung für die flexible Komponierbarkeit und Interaktion in verteilten Sensor-Aktor-Systemen herausgearbeitet. Das auf diese Anforderungen orientierte Konzept des Sentient Objects wurde während des europäischen Verbundprojektes CORTEX (COoperating Real-time senTient objects: architecture and EXperimental evaluation – Projektnummer: IST-2000-26031) entwickelt und bildet die nötige Struktur ab, um Anwendungen in einer modularen und hierarchischen Struktur entwickeln zu können. Dabei stand die Abstraktion der Kommunikation im Vordergrund, die über sogenannte Ereignisse und Ereigniskanäle modelliert wird. Ein Ereignis dient nach Verissimo, Kaiser und Casimiro als Datencontainer, der aus dem Subjekt, den Attributen und dem Inhalt besteht. Das Subjekt dient der Identifikation des Inhaltes und charakterisiert den Inhalt eines Ereignisses (Distanz, Temperatur usw.) [Kai+03]. Die Attribute beschreiben den Kontext der Nachricht, also beispielsweise Zeitstempel, Positionsangaben, Güteattribute usw., die als zusätzliche Informationen der eigentlichen Messgröße beigeordnet sind. Damit wird der Messwert auf einem komplexen Datensatz als „time/value entity“ [KK90] abgebildet, wobei „ti-

me“ an dieser Stelle nicht zwingend einen Zeitwert, sondern den Gesamtbestand der zur Einordnung und Verwertung eines Messwertes nötigen Informationen umfasst. Der Austausch der Ereignisse zwischen einzelnen Sentient Objects erfolgt über Kanäle, die die individuellen Eigenschaften des unterliegenden Netzwerkes abstrahieren und kapseln.

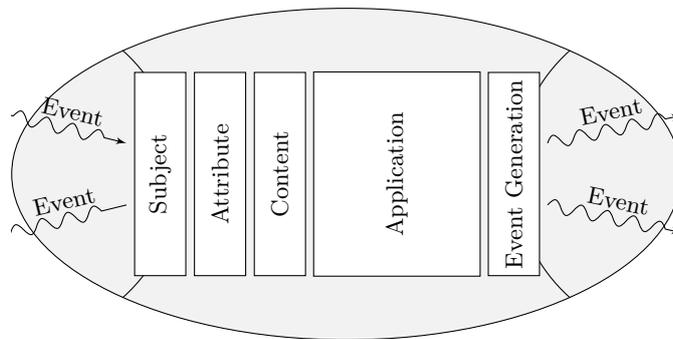


Abbildung 3.7: Grundstruktur eines Sentient Objects nach [Sch11b]

Abbildung 3.7 illustriert die Grundstruktur eines Sentient Objects. Die eingehenden Ereignisse werden in mehreren Filterstufen auf ihre Relevanz für die Anwendung untersucht. Dazu prüfen Filtereinheiten nacheinander das Subjekt des Ereignisses, die Attribute und nachfolgend den Wert des eigentlichen Datenfeldes. Die nicht verworfenen Daten werden an die eigentliche Anwendung übergeben und deren Ergebnis in ein Ereignis transformiert und über den zugeordneten Kanal publiziert. Die konkrete Umsetzung des Sentient Object Konzepts wird zum Beispiel in [Kai+09], [Kai+03] oder [Sch11b] beschrieben. Resultierend aus dem allgemeingültigen Konzept sehen Verissimo, Kaiser und Casimiro [VKC03] das Sentient Object in der Herkunftsunabhängigkeit der Ereignisse die Voraussetzung zur Umsetzung von beliebigen Komponenten einer verteilten Applikation, also neben Sensoren auch Verarbeitungseinheiten oder Aktoren.

Biegel [Bie04] erweitert das Konzept des Sentient Objects im Hinblick auf die applikationsspezifische Verarbeitung der bereitgestellten Ereignisse mit ihren attribuierten Daten. Dabei konzentriert er sich auf eine aus Bayesschen Schätzern basierende mehrgliedrige Datenfusion. Abbildung 3.8 zeigt den Datenfluss innerhalb eines Sentient Objects. Ausgehend von den Umgebungsbedingungen und der Situation des Knoten werden eingehende Ereignisse gefiltert und gegebenenfalls über einen Callback die Verarbeitungskette aktiviert, wobei sich ein Bayesscher Fusionsansatz durch die Arbeit zieht. Dessen Ergebnisse werden auf einer Inferenzmaschine abgebildet. Die Fusion erfolgt dabei in drei Stufen - Erfassung und Filterung, Fusion, regelbasierte Interpretation - wobei der Abstraktionsgrad zunimmt, dem im Absatz zuvor beschriebenen Konzept der Virtual Sensors. Letztendlich werden die Fusionsergebnisse in einer Inferenzmaschine untersucht und die

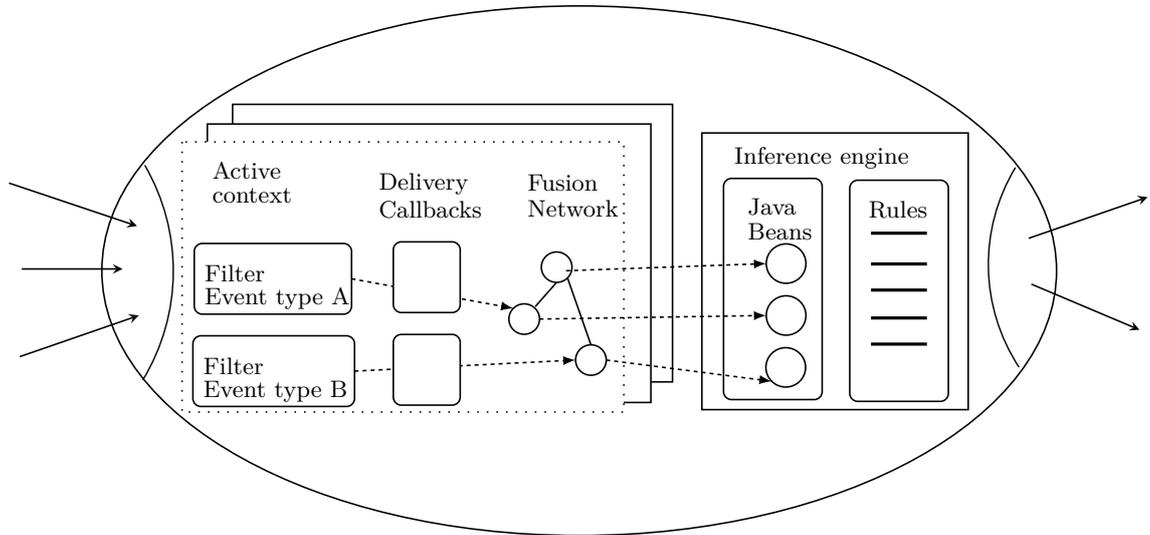


Abbildung 3.8: Datenfluss im Sentient Object (nach [Bie04])

Gültigkeit des aktuell gewählten Kontextes hinterfragt. Für sein Konzept stellt Biegel eine Entwicklungsumgebung auf der Basis von Java bereit.

Mit CODES stellt Piontek [Pio07] die Umsetzung des unter Abschnitt 3.2.2 vorgestellten Sentient Object in einer Kette von Entwicklungswerkzeugen vor. Schwerpunkt der Arbeit ist ein elektronisches Datenblatt, das die Konfiguration eines Sentient Objects umfasst und über den gesamten Produktlebenszyklus Verwendung findet. Das XML-basierte TEDS kombiniert für einen Knoten zum einen eine Beschreibung des Ereigniskanals und in einem zweiten Teil die des Ereignisses selbst. Das Datenblatt umfasst also neben Meta-informationen eine Beschreibung der Kommunikationsparameter, das Datenformat, aber eben auch material- und recyclingspezifische Angaben. Das TEDS selbst wird für jeden der intelligenten Sensoren in komprimierter Form im ROM des Mikrocontrollers abgelegt. Der für die Hinterlegung notwendige Speicherplatz der in nichtkomprimierter Form etwa 35 kByte großen XML-Datenblätter bewegt sich etwa bei 3 kByte [Pio07]. Mit der Integration eines Knotens im Netzwerk erfolgt die Übertragung an einen zentralen Server. An diesen kann jeder Knoten, der über hinreichende Ressourcen verfügt, Anfragen via Simple Object Access Protocol (SOAP) stellen, um zum Beispiel zu ermitteln, wie sich die Datenstruktur eines bestimmten Kanals zusammensetzt. Für die dafür nötige Codierung der Einheiten, Potenzen usw. greift Piontek die Systematik der Datenblätter des IEEE 1451 auf und setzt diese in einem XML-Schema um. In Quelltextauszug 3.1 ist die Definition eines Spannungswertes gezeigt, der durch die Kombinationen von beliebigen Potenzen der SI-Einheiten gebildet wird. Die Konkretisierung des eigentlichen

```

1 <Dimension>
2   <SIUnit>
3     <Meter>2</Meter>
4     <Kilogram>1</Kilogram>
5     <Second>-3</Second>
6     <Ampere>-1</Ampere>
7   </SIUnit>
8   <Magnitude>-3</Magnitude>
9   <Offset>0</Offset>
10 </Dimension>

```

Listing 3.1: Fragment der Definition eines Spannungswertes in CODES in mV
($\text{m}^2 \cdot \text{kg} \cdot \text{s}^{-3} \cdot \text{A}^{-1}$)

Wertes ergibt sich dann aus der Multiplikation der durch Magnitude spezifizierten Potenz von 10 und einem eventuell wirkenden Offsets, zum Beispiel bei der Angabe von Temperaturen auf der Basis der SI-Einheit Kelvin.

Im Hinblick auf die folgenden Kapitel dieser Arbeit ist es bemerkenswert, dass CODES anhand der Datenblätter eine Codegenerierung für einen bestimmten eingebetteten Mikrocontroller unterstützt. Allerdings erstreckt sich dieser Vorgang lediglich auf die Erzeugung der Datenstrukturen und elementaren Sensorzugriffe, deren Schnittstellen und Parametrierungen allerdings nicht in den Datenblättern enthalten sind, sondern in der Toolchain der Codegenerierung fest integriert sind. Diese nutzt die Möglichkeiten der Transformation von XML-Dokumenten mittels Extensible Stylesheet Language Transformation (XSLT). Während der Codeerstellung werden auf der Grundlage der Datenblattinformationen Konsistenzüberprüfungen ausgeführt und zum Beispiel die Übereinstimmung von Datentypen und Einheiten geprüft [Kai+08a].

Die Idee der elektronischen Datenblätter wird mit CODES in zwei Richtungen erweitert. Zum einen werden diese um dynamisch veränderliche Elemente wie zum Beispiel situationsabhängige zeitliche Gültigkeitswerte von Messwerten erweitert. Dazu werden über spezielle Ereignisse die Werte der Parameter an die zentrale Datenblattverwaltung geschickt, die dann eine Anpassung der Datensätze vornimmt. Piontek nimmt damit hinsichtlich der Parameter und Attribute eine Dreiteilung vor. Absolut fixe Daten sind in den statischen Teilen der Datenblätter enthalten, während gelegentlich veränderliche Größen in deren dynamischen Bereichen adaptiv eingepflegt werden. Mit jedem Ereignis gelangen weiterhin die sogenannten Attribute als dritte Kategorie der zusätzlichen Informationen zum Empfänger der Nachrichten.

Eine zweite wichtige Neuerung sehen Kaiser und Piontek [KP06] in der Einführung von ereignisdiskreten Sensoren in die Beschreibungssprache. Dabei unterscheiden sie zwischen abhängigen und unabhängigen Zuständen. Die letztgenannte Kategorie wird in einer Aufzählung zulässiger Zustände und die erstgenannte mit einer kompletten Zustandsmaschine beschrieben.

In der Arbeit von [Sch11b] wird die architekturübergreifende Middleware Family of Adaptive Middleware for autonomOUS Sentient Objects (FAMOUSO) [Sch08] vorgestellt, die das Sentient Objects als Programmiermodell aufgreift. Zielstellung war dabei die Überwindung der Heterogenität der ausführenden Plattformen und der unterliegenden Kommunikationsnetze. Entsprechend wird eine Middleware präsentiert, die 8-Bit-Mikrocontroller bis PC-Architekturen als Zielplattformen abdeckt und über verschiedene Netze, TCP/IP, CAN, ZigBEE, RS232 usw. arbeitet. Um die unterschiedlichen Netze und Systeme zu kapseln, wurde ein adaptives Schichtensystem entworfen, das die Ereigniskanalabstraktion in verschiedenen Ausprägungen umsetzt. Dafür definiert die Arbeit eine Beschreibungssprache für Dienstgüteangebote und Dienstgüteanforderungen an den einzelnen Kanal, die während des Entwurfes und zur Laufzeit zur Prüfung der Gültigkeit eines Kanals genutzt werden. Die Beschreibungssprache wird zudem als Mittel der Ereignisattributierung eingesetzt und eine, sofern möglich, hardwarenahe Filterung einzelner Ereignisse damit umgesetzt.

Logical Sensor

Shilcrat, Panangaden und Henderson [SPH84] beschrieben in ihrer Arbeit den Logical Sensor als Abstraktion der einzelnen Komponenten eines Sensornetzes. Zielstellung war die Kapselung der physikalischen Natur des Sensors sowie der Mechanismen für die Fehlerüberwachung und Neuintegration von Sensoren. Abbildung 3.9 zeigt eine erweiterte Form des Logical Sensors, den Instrumented Logical Sensor Systems (ILSS) aus Henderson und Dekhil [HD98]. Dessen Grundbestandteile sind demnach:

- ein eindeutiger Sensorname für die Adressierung des Knoten,
- $0 - n$ Eingabevektoren/ein charakteristischer Ausgabevektor,
- ein Eingangsbefehlssatz zur Parametrisierung des Knotens selbst/ein Ausgangsbefehlssatz, um andere Knoten in ihrem Verhalten zu manipulieren,
- ein Interpretermodul, das die Kontrollbefehle in konkrete Aktionen innerhalb des Logical Sensors umsetzt,
- eine oder mehrere Verarbeitungseinheiten, die die eingehenden Informationen auf dem Ausgabevektor (Characteristic Output Vector (COV)) abbilden und
- ein Selektor, der die Auswahl der Verarbeitungseinheiten in Abhängigkeit vom Fehlerstatus übernimmt,
- ein Monitor, der die Gültigkeit der Ausgabevektoren prüft,
- Embedded Tests, die rekursiv anhand eines mathematischen Modells die Plausibilität eingehender Messdaten einschätzen und

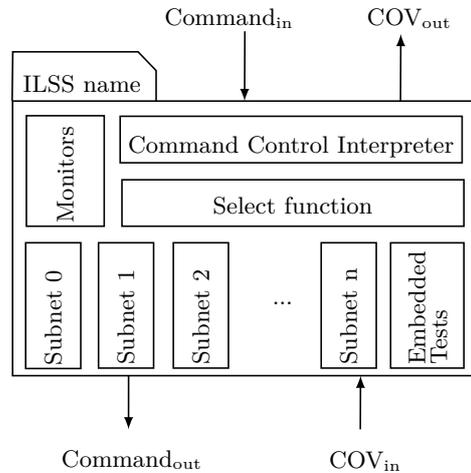


Abbildung 3.9: Basisstruktur des Instrumented Logical Sensor nach [HD98] (COV = Characteristic Output Vector)

- eine Schnittstelle für Callbackfunktionen zur Überwachung der einzelnen Elemente des COV

Damit setzten die ILSS die bereits in den vorhergehenden Arbeiten deutlich gewordene Abstraktion der Nachrichten fort. In diesem Fall werden die auszutauschenden Informationen im Ausgabevektor in **data** und **performance measures** gegliedert, die dem Inhalt und den Attributen der Ereignisse des Sentient Objects entsprechen. Die Kommunikationskanäle werden entsprechend Abbildung 3.9 als **subnets** bezeichnet, oberhalb derer, eine Selektionsfunktion arbeitet. Abweichend zu den bisherigen Kommunikationsansätzen können die **commands** und das Interpretermodul als Bestandteile eines Servicekonzepts verstanden werden. Unklar an der Darstellung der Architektur ist die Einordnung der eigentlichen Funktionalität, die durch den ILSS umgesetzt werden soll.

Unterschiedliche Logical Sensors können in beliebigen Mustern hierarchisch strukturiert werden. Dazu besteht eine Konfigurationsprache, die den strukturellen Aufbau der Applikation definiert. So werden zum Beispiel mit `serial(ILSS1, ILSS2)` Sequenzen von ILSS aneinander gefügt. Diese Beziehungen werden allerdings, wie in [DH98] beschrieben, zur Compilezeit festgelegt, wobei innerhalb des ILSS dann eine Auswahl über der Menge der empfangenen Daten stattfinden kann.

Schilp u. a. [Sch+04] entwickeln einen differenzierteren Logischen Sensor auf der Basis des Logical Sensors von Henderson und differenzieren vier Typen:

- 1. Art** Logische Sensoren 1. Art sind jeweils nur einem physischen Sensor zugeordnet. Analog zur Idee des Abstract Sensor, des Virtual Basis Sensor usw. dient diese Ebene als Schnittstelle für die realen Messwandler. Neben anderen sieht Ehren-

straßer [Ehr07] an dieser Stelle einen Zustandsautomaten mit den Zuständen Init, Loop und Error vor.

2. **Art** Sensoren dieser Ebene verfügen über keine physikalische Schnittstelle, sondern fassen ausschließlich die Resultate von Logischen Sensoren 1. Art zusammen. Dazu werden einheitliche Messgrößen fusioniert.
3. **Art** Im Unterschied dazu sind Logische Sensoren der 3. Art flexibler und greifen auf differenzierte Daten zurück, um diese zusammenzuführen.
4. **Art** Logische Sensoren der 4. Art sind nicht an physikalische Sensoren geknüpft, sondern bestimmen anhand der Werte anderer Logischer Sensoren und eines physikalisch-mathematischen Modells Prozessgrößen. Ehrenstraßer [Ehr07] nutzt für diese Kategorie zusätzlich den Begriff des virtuellen Sensors und folgt damit der ersten unter Abschnitt 3.2.2 getroffenen Begriffsbedeutung.

Das Konzept von Hendersons ILSS wurde in einem Framework und zugehöriger Werkzeugkette umgesetzt. Dabei zielt die Implementierung weniger auf typische ressourcenbeschränkte Sensorknoten als auf leistungsstarke Controller. Der Datenaustausch erfolgt auf der Basis von Sharded Memory oder via Component Object Model (COM) und Distributed Component Object Model (DCOM). Für die Anwendungsentwicklung wurde eine grafische Programmierungsumgebung entwickelt [HD98]. Die von Henderson nur am Rande berücksichtigte Aufnahme von Messdaten wurde unter anderem in der Arbeit von Naish [Nai00] erweitert.

Mitchell [Mit07] nutzt eine vereinfachte Darstellung des ILSS als Grundlage seiner Fusionstopologien und zeigt, dass sich mit dieser Abstraktion eines Sensor- oder Fusionsknotens komplexe Verarbeitungsketten umsetzen lassen. Dafür stellt er drei Grundmuster vor: die serielle, die parallele und die iterative Anordnung, die sich auf verschiedenen Ebenen beliebig kombinieren lassen.

Fusion Channel

In [Aga+02] beschreiben die Autoren eine Programmierabstraktion, den Fusion Channel, der die Erfassung, Verarbeitung und Verbreitung von Sensordaten umsetzt. Wie in Abbildung 3.10 dargestellt, besteht dieser aus Zugriffsfunktionen auf Eingangsbuffer, einer Fusionsfunktion sowie einem Ausgangsspeichermanagement. Der Fusion Channel ist damit die logische Fortsetzung der für die Kommunikation genutzten, als *channel* bezeichnete Abstraktion. Mittels dieser Kommunikationskanäle tauschen die einzelnen Fusion Channels *items*, mit Zeitstempeln angereicherte Daten, aus. Die Interaktion mit den Input Buffers kann dabei beliebig konfiguriert werden, also zum Beispiel ereignisgetrieben oder per Abfrage arbeiten. Zudem können neben dem Datenaustausch auch Konfigurations- und Requestbefehle übermittelt werden, die spezielle Speicherzugriffe

wie konkrete Anforderungen hinsichtlich des Zeitstempels der Items definieren, bevor diese übersandt werden.

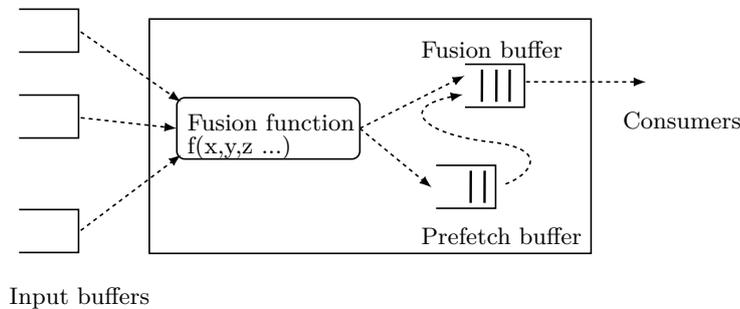


Abbildung 3.10: Fusion Channel (nach [Aga+02])

Die Zahl der eingehenden Kanäle wird zur Übersetzungszeit definiert. Das Fusion Channel Programmiermodell bietet ein vielseitig konfigurierbares Fusionsmanagement. Die Triggerung und der Fusionsalgorithmus einer neuen Berechnung können dabei sowohl vom Eintreffen eines Items abhängen als auch durch Steuerkommandos sowie Speicherzugriffe ausgelöst werden. Inwieweit bei der Fusion auf einzelne Kanäle verzichtet werden kann, um ein Ergebnis zu bestimmen, ist in der Definition der Fusion hinterlegt. Die Autoren sprechen dabei von mehreren, nebeneinander bereitstehenden *partial fusions*. Eine Erweiterung von bereits laufende Fusionsthreads um weitere Eingänge ist nicht vorgesehen, eine dynamische Adaption des Fusionsprozesses ist damit nicht möglich.

Die Kommunikation basiert auf der von Ramachandran entwickelten Middleware Stampede [APR02]. Ein Stampede-Programm verbindet unterschiedliche Threads dynamisch, die zeitgestempelte Items als Datenpakete austauschen. Dieser Austausch geschieht via get/put über eine Kanalabstraktion, Queues oder gemeinsame Registerzugriffe. Stampede wurde für Cluster von x86-Systemen entwickelt. Der Datenaustausch erfolgt über gemeinsame Speicherzugriffe des Clusters oder verschiedene Ethernetprotokolle. Hinsichtlich der Zeitsynchronisierung wird die Einbindung eines entsprechenden Protokolls vorausgesetzt.

In [Kum+03] wird das Konzept der Fusion Channels aufgegriffen und in ein Framework, das neben einer fehlertoleranten Fusion auf eine dynamische Bestimmung des Ausführungsortes abzielt, integriert. Es wird ein Task Graph, der die einzelnen Schritte der Aggregation und Fusion beschreibt, auf der verfügbaren Hardware abgebildet und diese Zuordnung beständig geprüft. Dazu werden Kostenfunktionen ausgewertet, die die Performance der Knoten, die Auslastung der Kommunikationspfade und den Energievorrat auswerten.

3.2.3 Zusammenfassung

Die Tabelle 6.2 stellt die Programmierabstraktionen und Standards dem Anforderungskatalog des Kapitels 2 gegenüber. Für die Bewertung der Abdeckung wird eine dreistufige Skale benutzt. Ein freies Tabellenfeld markiert eine geringe oder fehlende Berücksichtigung eines Aspektes. Ein leerer Kreis repräsentiert eine eingeschränkte und ein ausgefüllter Kreis eine weitreichende Integration der Forderung.

Der Abstract Sensor bietet mit seinem Fehlertoleranzkonzept einen auf sehr eingeschränkte Randbedingungen zugeschnittenen Detektionsmechanismus, der eine zuverlässige Selektion anhand einer veränderlichen Güteschranke erlaubt. Die Sensorspezifika werden hinter einem übergreifenden Modell verborgen, das einen um die Parameter des Rauschverhaltens erweiterten Datensatz bereitstellt. Der Abstract Sensor beinhaltet damit Konzepte zur Abstraktion für Sensorik, Messdaten und Fehlerdetektionsmethoden. Im Hinblick auf die vorliegende Arbeit ist insbesondere der um zusätzliche Informationen erweiterte Messdatensatz relevant, der die Grundlage der hierarchisch höher angesiedelten Bewertung darstellt. Die Übertragbarkeit auf realistische Sensormodelle und über einen einfachen Mittelwert hinausgehende Fusionsansätze ist aber, bedingt durch die mathematischen Annahmen des Selektionsalgorithmus, gering.

Das Sentient Object unterstützt die Applikationsentwicklung auf der Basis einer netzwerkunabhängigen Kommunikationsabstraktion für die Nachrichten und den Transport. Damit lassen sich komplexe verteilte Anwendungen umsetzen. Das Modell des Sentient Objects greift damit die Überlegungen von Marzullo [Mar90] hinsichtlich der erweiterten Daten zur korrekten Einordnung, Evaluation und Verarbeitung der Daten auf und systematisiert diesen Ansatz im Ereignis, das über (virtuelle) Kommunikationskanäle ausgetauscht wird. Eine Abstraktion der Sensorik/Aktorik, der Targetspezifika oder der Fehlerdetektion bietet das Sentient Object folglich nicht. Relevant für die Arbeit am *MOSAIC*-Framework ist zum einen die Verarbeitungskette, der die geforderten Fehlertoleranzmechanismen zwar fehlen, die aber die Grundlagen für einen feingliedrigen standardisierten Auswahlprozess legt. Im Weiteren bietet sich das Konzept der Ereignisse und des Ereigniskanals für die lose Kommunikation innerhalb eines verteilten Sensor-Aktor-Systems an.

Die in Bose u. a. [Bos+07] beschriebenen Fehlerdetektionsansätze legen hinsichtlich der Ausprägung der Sensoren hohe Anforderungen fest. Da alle Sensortypen durch ein einfaches Tupel in ihren Eigenschaften beschrieben sind, wird für Sensoren einer Messgröße von einem identischen Verhalten im Hinblick auf Präzision, Rauschen usw. ausgegangen. Ebenso wird angenommen, dass mit dem Starten des Systems alle Sensoren verfügbar sind. Dynamisch hinzukommende Sensoren sind von dem Ansatz nicht erfasst. Fehlertoleranz ist nur auf der mittleren Ebene in Form eines kreuzweisen Vergleiches der Sensordaten vorgesehen. Von der dafür nötigen synchronen Messaufnahme wird ausgegangen. Die automatische Konfiguration des Netzes der virtuellen Sensoren ist nur möglich, weil relativ einfache physikalische Phänomene beobachtet werden sollten. Die

Tabelle 3.2: Merkmale vergleichbarer Programmiermodelle im Hinblick auf die Anforderungen des *MOSAIC*-Frameworks (◦ = eingeschränkt, ● = umfangreich berücksichtigt/implementiert)

		Standards		Programmiermodelle			
		IEEE 1451	OMG STIS	Abstract Sensor	Sentient Object	Virtual Sensors	Instr. Logical Sensors
Implementierungen		●	●	●	◦	◦	●
Nachrichtenabstrakt.				Ereignis	Tupel	COV	Item
Kommunikationsabstrakt.				Kanal		Subnet	Kanal
1	a) Modularität	◦	◦	◦	◦	●	◦
	b) Ablaufsteuerung	●	●			●	●
2	a) Kombinierbarkeit	◦	●	◦	●	◦	●
	b) Rekursivität		◦		●	●	●
3	a) domainspez. Imp.	◦	◦		◦	◦	◦
	b) vers. Architekturen	●	◦		●	◦	
4	Hardwarekapselung	●	●	◦	●	●	◦
5	Simulierbarkeit	◦		●	●		
6	a) Datenmodell	●	●	◦	●	●	◦
	b) Beschreibungssprache	●	●		◦	◦	
	c) Konsistenzprüfung		◦		●		
7	a) Fehlerdetektion	◦	◦	◦		◦	◦
	b) einheitl. Semantik			◦			
8	adaptive Datenselektion		◦		◦	●	◦
9	a) Vorverarbeitung	◦	◦		◦	●	◦
	b) adaptive Fusion					◦	◦

Bereitstellung der relevanten (Sensor-)Parameter für die Virtual Sensors steht damit konträr zu den beiden vorangegangenen Programmiermodellen, die Informationen wie das Messrauschen, Validität, zeitliche Gültigkeit usw. mit jeder Nachricht verbreiten und damit unabhängig von der hier beschriebenen zentralistischen und statischen Datenhaltung sind. In die Entwicklung des *MOSAIC*-Frameworks werden beide Ansätze einbezogen werden, wobei für eine differenzierte Betrachtung auf Kapitel 4 verwiesen sei.

Die Instrumented Logical Sensors bieten im Vergleich zu den anderen Abstraktionen die am weitesten entwickelte interne Struktur eines intelligenten Sensors. Entsprechend sind die Funktionalitäten auf einzelne Komponenten aufgeteilt, die im modularen Zusammenspiel das Funktionsspektrum eines Smart Sensors abdecken. Dazu zählen Netzwerkschnittstellen, Filter und Selektionsmethoden, eine Verarbeitungskomponente sowie Fehlertests. Diese Grundbausteine werden sich in der Herleitung der *MOSAIC*-Architektur analog in differenzierterer Weise wiederfinden. Einer dieser Bausteine, die Schnittstelle für Selbsttests, hebt den ILSS in dieser Hinsicht von den anderen Programmiermodellen ab. Allerdings zielen die bestehenden Implementierungen auf leistungsstarke Knotenhardware ab und tauschen Informationen über komplexe Client-Server-Mechanismen aus.

Als Programmierabstraktion für verteilte Systeme sind die Fusion Channels im Hinblick auf ihre flexibel konfigurierbare und zumindest im Bereich statischer Vorgaben adaptive Verarbeitung bemerkenswert. Damit besteht die Möglichkeit, auf Abruf rasch ein Ergebnis zu generieren, das zwar von geringerer Güte, aber unmittelbar verfügbar ist, ähnlich den Ansätzen in [NGM98]. Unabhängig davon kann eine „normale“ vollständige Fusion zu einem späteren Zeitpunkt ein präziseres Ergebnis liefern.

3.3 Beschreibungskonzepte für Sensordaten

3.3.1 SensorML

Die Sensor Model Language (SensorML) stellt eine Spezifikationssprache für verteilte Sensorsysteme dar, die von einfachen zentral operierenden Messanlagen über geowissenschaftliche Systeme und ganze Schiffe bis zu Satelliten reichen [Ope07]. Ziel der Entwicklung von SensorML, die als Bestandteil der Sensor Web Enablement (SWE) des Open Geospatial Consortium (OGC) entwickelt wurde, war insbesondere die Umsetzbarkeit von internetbasierten Sensorapplikationen. Um dafür die technologischen Grundlagen zu legen, sieht SensorML die Beschreibung von Messwerten mit Einheiten, Qualitätsattributen, Lokalisierungsangaben usw. vor, aber auch die Modellierung von Schnittstellen, Abbildungsfunktionen von Ein- auf Ausgängen sowie die nichtfunktionalen Eigenschaften. Die Verbindungen zwischen den einzelnen Modellierungen werden durch vordefinierte Datentypen sichergestellt. SensorML stellt mit diesen abstrakten Beschreibungen die Informationsbasis für übergeordnete Applikationen dar. Dies können Anwendungen für

Sensormanagementaufgaben innerhalb großer Applikationen, Servicediscoverymechanismen, Untersuchungen von Performanzwerten wie Genauigkeit, Laufzeit usw. sein.

Tabelle 3.3: Modelle in SensorML

	nichtphysikalisch	physikalisch
atomar	Prozess	Komponente (Detector)
verkettet	Prozesskette	System (Sensor)

Die Beschreibungsmodelle von Sensor ML werden analog zur Darstellung von Tabelle 3.3 in nichtphysikalische und physikalische sowie atomare und verkettete Elemente eingeteilt, wobei einzelne Begriffe auch hierarchisch übergreifend benutzt werden können. Diese Struktur ist begründet in den unterschiedlichen Schnittstellen, die für die Elemente in Abhängigkeit von der Repräsentation eingeführt werden. Physikalische Elemente interagieren über ein Interface mit der Umgebung, dies können sowohl Sensoren, Aktoren oder Netzwerkschnittstellen sein. Für deren Beschreibung wird zum Beispiel für die Kommunikationsschnittstellen auf ein erweitertes Open Systems Interconnection (OSI) Referenzmodell verwiesen.

Komponente Eine atomare Einheit wird dabei als Komponente bezeichnet, wenn diese der Definition eines Transducers folgend Eingänge auf Ausgänge abbildet und dabei von spezifischen Parametern und nichtfunktionalen Eigenschaften abhängig ist. Eine spezielle Form davon ist der Detektor, der in SensorML analog zu dem in dieser Arbeit gebrauchten Verständnis eines Sensors benutzt wird. Weitere Elemente, die als Komponente beschrieben werden können, sind Aktoren.

System Mit der Konkatenation von Komponenten und anderen Elementen entsteht im Sinne der SensorML ein System, für das bei der Einbindung eines Detektors und einer Kommunikationsschnittstelle von einem Sensor gesprochen wird.

Prozess Nicht-physikalische Prozesse bilden einen Zusammenhang zwischen Eingangs- und Ausgangsgrößen ab und lassen sich beispielsweise durch externe Programmaufrufe oder einen MathML-Ausdruck spezifizieren. Dieser Typ verfügt über keine Schnittstellen zur Umgebung und repräsentiert beispielsweise einen Filter.

Prozesskette Die Prozesskette kombiniert analog zum System einzelne Prozesse in einer Verarbeitungsfolge.

In der Kombination aus Prozessketten und Systemen lassen sich komplexe Anwendungen modellieren, wobei das Abbildungskonzept eine vollständige Darstellung des verteilten Sensor-Aktor-Systems erlaubt.

Für die Beschreibung der genannten Elementtypen hält SensorML eine umfangreiche XML-Bibliothek bereit. An dieser Stelle sollen, der Ausrichtung dieser Arbeit entsprechend, die Kernelemente dieser Beschreibung für Detector und Sensor Modelle einführend vorgestellt werden. Hinsichtlich der umfangreichen Möglichkeiten der gesamten SensorML-Beschreibungssprache sei auf [BR06] sowie die Beispielimplementierungen wie [Rob06] verwiesen.

Der Detector wird in drei XML-Beschreibungen definiert: das Eingangsphänomen, das Ausgabeformat und das spezifische Verhalten der Komponente. Diese Informationen werden um die Metadaten ergänzt:

Metadata Diese Datenstruktur umfasst insbesondere die Einträge für vordefinierte Identifikatoren wie `intendedApplication` oder `sensorType`, die für Service-Discovery-Mechanismen hinterlegt sind.

Reference Frame Mit dem Reference Frame wird das Bezugssystem des Detectors definiert. Dies erfolgt in einer textuellen Beschreibung von dessen Lage innerhalb des Systems. Entsprechende Relationen werden damit transparent.

Inputs/Outputs In diesem Abschnitt werden sogenannte Phänomene als Eingabegrößen und Ausgaben in ihrem Format, Einheiten usw. definiert. Ein Detector kann dabei immer nur ein Phänomene überwachen und einen Ausgabewert bedienen. Ein komplexeres Sensorverhalten muss über Systeme oder Detector-Arrays beschrieben werden.

Response Parameter In dieser Kategorie werden optionale Sensorparameter quantifiziert. SensorML bietet dabei einen deutlich größeren Umfang an vorgefertigten Datentypen und Beschreibungsmechanismen für die Kalibrierung, das Zeitverhalten des Sensors oder den Überwachungsbereich als beispielsweise die IEEE-1451-Datenblätter.

Die Beschreibung eines Systems fasst mehrere Subkomponenten in einer gemeinsamen Struktur zusammen, greift bestimmte Elemente der Detector-Beschreibung auf und ergänzt diese. Diese Modelle werden ebenfalls mit erweiterten Metadaten der Reference Frames und der Ein- und Ausgänge charakterisiert. Anders als beim Detector können innerhalb der Ausgabewerte Gruppen gebildet und einzelne Messungen in neuen Strukturen (zum Beispiel ein Messwert mit einem Zeitstempel) zusammengefasst werden. Innerhalb des Systems können verschiedene Komponenten oder Prozesse/Prozessketten eingebunden sein. Die Definition und Zuordnung sowie die Verbindung dieser internen Bestandteile ermöglicht die feingranulare Abbildung von komplizierten Verarbeitungsketten für Sensorsysteme. Aufbauend auf den Reference Frames werden in der

Systembeschreibung zudem die geografischen Beziehungen definiert. Dies kann in stark hierarchisch aufgebauten komplexen Applikationen dazu führen, dass eine Kette von Relativbeziehungen durchlaufen werden muss, bevor die konkrete Position eines Sensors bestimmt werden kann. SensorML unterstützt dabei auch veränderliche Positionsangaben für mobile Systeme. Im letzten Punkt des Beschreibungsdokuments wird die Kommunikation hinsichtlich der Protokolle, Medien und physikalischen Verbindungselemente konfiguriert. Dabei wird auf ein erweitertes OSI-Schichtenmodell zurückgegriffen.

3.3.2 OpenRave

Die Neuimplementierung des Rave Frameworks im Rahmen der Arbeit von Diankov [Dia10] bietet auf der Basis der Open Dynamics Engine (ODE) eine Entwicklungsumgebung für Roboter manipulatoren. Der Schwerpunkt liegt dabei auf der Berechnung von Trajektorien mit unterschiedlichen Kinematikansätzen und der Simulation der auf diesem Weg erzeugten komplexen Verfahrensbewegungen. Dafür können präzise geometrische Roboter- bzw. Sensormodelle in eine detailreiche Umgebung eingebettet werden und mit dieser unter frei konfigurierbaren physikalischen Bedingungen interagieren. Für die Evaluation können reale Manipulatoren und Sensoren mit der Simulationsumgebung verknüpft werden.

Im Hinblick auf das *MOSAIC*-Projekt ist die Beschreibungsform der Modelle, insbesondere die der Sensorik und deren Interfaces interessant. OpenRave erweitert an dieser Stelle nämlich das unter der Bezeichnung Collaborative Design Activity (COLLADA) präsente XML-basierte, offene Austauschformat zwischen 3D-Programmen wie Blender, LightWave 3D, Cinema 4D, Google SketchUp usw. auf robotikspezifische Belange. Damit lassen sich dann die Umgebungsgeometrien, physikalische Eigenschaften und Relationen einzelner Komponenten untereinander definieren. In Bezug auf die Simulation des physikalischen Verhaltens bietet COLLADA Interfaces für Standardengines, wie ODE oder grafikartenorientierte Bibliotheken.

OpenRave bietet für sieben Sensoren entsprechende XML-Schemata, die die unterschiedlichen Strukturen für die Beschreibung von Sensoreigenschaften und Interfaces implementieren. Quelltextauszug 3.2 zeigt den Inhalt dieser Spezifikation am Beispiel des *BaseLaserscanner2D*. Im Unterschied zu Quelltextauszug 3.3 werden hier keine Nachrichten beschrieben, sondern allgemeine Sensorinformationen strukturiert bereitgehalten. Folglich umfasst Zeile 9-13 die Definition des Interfaces, das in einer untergeordneten Beschreibung konkretisiert wird. Die Zeilen 2-8 listen maximale und minimale Messbereiche sowie Aussagen zum Zeitverhalten des Sensors auf.

Fokus der Beschreibungen in OpenRave ist die lokale Verwendung der XML-Daten für Berechnungs- und Simulationszwecke. Die über die Middlewareinterfaces von COLLADA mögliche Verbreitung von Datenblättern ist nicht vorgesehen.

```
1 <sensortype="base_laser2d" id="ExampleLaser1">
2   <angle_min<float>-90</float></angle_min>
3   <angle_max<float>90</float></angle_max>
4   <range_min<float>0.01</float></range_min>
5   <range_max<float>4.0</float></range_max>
6   <angle_increment<float>1</float></angle_increment>
7   <time_increment<float>0.0005</float></time_increment>
8   <measurement_time<float>0.025</float></measurement_time>
9   <interface_type>
10    <techniqueprofile="OpenRAVE">
11     <interface>BaseLaser2D</interface>
12    </technique>
13  </interface_type>
14 </sensor>
```

Listing 3.2: Beschreibung eines Laserscanners in OpenRAVE

3.3.3 AutomationML

Die Konzepte aus COLLADA werden durch AutomationML (Automation Markup Language) erweitert. Zielstellung der Entwicklung war ein XML-basiertes Datenformat für technische Anlagen, insbesondere im Hinblick auf den Austausch von Engineering-Daten, Programmen, elektrischen Plänen, Robotersteuerungen usw.

AutomationML beschreibt die Elemente der technischen Anlage aus verschiedenen Perspektiven und erlaubt einen hierarchischen Aufbau. Die einzelnen Eigenschaftskomplexe werden mit unterschiedlichen Standards erfasst:

- Attribute und Beziehungen von Objekten werden nach CAEX (IEC 62424) definiert,
- die geometrischen Relationen und kinematische Abhängigkeiten werden mit COLLADA begründet,
- das Systemverhalten wird über PLCopen implementiert [Hun+10].

Wie von Rossdeutscher, Zuern und Berger [RZB10] und Diedrich, Lüder und Hundt [DLH11] beschrieben, bildet AutomationML die Grundlage einer Vielzahl von Projekten zur automatisierten Roboterbeschreibung und -entwicklung. Der übergreifenden Anspruch, sowohl die Funktionalität aber auch die geometrisch-physikalischen Eigenschaften eines komplexen Systems abzubilden, erfordert die Konzeption verschiedener Hardwareabstraktionen analog zu den Anforderungen von *MOSAIC*.

3.3.4 Netzwerkspezifische Beschreibungskonzepte

Es existiert eine Vielzahl von feldbusabhängigen Beschreibungssprachen wie Device Descriptions (DD) von HART [HAR10] oder Electronic Data Sheets (EDS) für CANopen [CAN05]. Diese streng netzwerkspezifisch ausgerichteten Datenblätter dienen damit aber nicht der im Rahmen dieser Arbeit avisierten übergreifenden Kommunikation in heterogenen Umgebungen. Zudem sind diese Beschreibungen lediglich als Programmier- und Parametrisierungsunterstützung vorgesehen, erlauben in der Regel kein Service Discovery zur Laufzeit oder gar darauf aufbauend eine Berücksichtigung neuer Informationsquellen in den Applikationen.

Analog bestehen im Bereich webbasierter Anwendungen eine Vielzahl von Protokollen mit unterschiedlich ausgeprägten Device- oder Datenbeschreibungen. Beispielsweise sei an dieser Stelle auf Web Services Description Language (WSDL) [Mor+06] und Universal Description, Discovery and Integration (UDDI) [Bel+02] verwiesen. Auch bei diesen Arbeiten ist eine übergreifende Verwendbarkeit in den angestrebten Szenarien nicht möglich, da dem eingeschränkte Hardwareperformance und unterschiedliche Feldbusse gegenüberstehen.

3.3.5 TinyDB

Anwendungen für Sensornetze im Sinne des Smart Dust sind durch drei entscheidende Einschränkungen geprägt: eine limitierte Rechenleistung der Knoten, eine eingeschränkte Kommunikationsbandbreite und endliche Energievorräte, die effektiv für eine möglichst lange Laufzeit eingesetzt werden sollen. Konträr dazu erwartet der Nutzer eine komfortable API für den Zugriff auf die Messdaten, über die die Verbreitung der Anforderung an alle Knoten und die Erfassung der Resultate umgesetzt werden kann. Dieses Szenario, das von vielen Sensornetzapplikationen bedient wird, dient einer zentralisierten Datenerfassung mit einer Informationssenke und steht damit im Kontrast zu dem in der Motivation dieser Arbeit formulierten Aufgabenstellungen.

TinyDB [Mad08] bietet auf TinyOS aufsetzend ein Interface, das die Formulierung dieser Anfragen in einer SQL ähnlichen Semantik erlaubt, ohne diese in C-Code implementieren zu müssen. Die Anfragen können (periodisch wiederkehrend) auf eine Momentaufnahme (Snapshot, Interval Queries) abzielen, die Historie der Messdaten (Queries over Stored Data) berücksichtigen oder die Kriterien definieren, bei deren Erfüllung asynchron ein Ereignis ausgelöst wird (Event-based Queries). TinyDB integriert im Hinblick auf den Kommunikationsaufwand Aggregationstechniken, die ausgehend von einer bekannten Netzstruktur die Daten bei ihrem Weg durch das Netz so weit wie möglich fusionieren, um die Datenmenge zu reduzieren. Wie Madden u. a. [Mad+05] beschreiben, erhöht dieser Ansatz die Lebensdauer des Netzes erheblich, wobei aber mit den vier bereitgestellten Funktionen (*min*, *max*, *mean*, *median*) nur eine sehr einfache Messdatenaggregation ermöglicht wird.

Die Ursprungshardware, für die TinyOS und TinyDB entwickelt wurden, die Berkeley Motes, verwenden einen fixen Satz einfacher und nichtredundanter Sensortypen wie Beschleunigungs-, Licht- oder Temperatursensoren. Erst diese Beschränkung ermöglicht die abstrahierten Abfragen. Szenarien mit heterogenen Sensoren und daraus resultierend unterschiedlichsten Formaten lassen sich mit diesem Konzept, das Messwerte auf einzelne skalare Größen reduziert, kaum fassen.

Bemerkenswert für die Arbeiten am *MOSAIC*-Framework ist der konzeptionelle Ansatz. Der Nutzer formuliert mit einer abstrakten möglicherweise auch komplexen Anfrage seine Informationswünsche an ein verteiltes System [CH05]. Damit wird die Filterung der verfügbaren Informationen, um die Bandbreite zu schonen, aus dem nachfragenden Knoten in das Netzwerk verschoben. Dies setzt voraus, dass in jedem Knoten alle Informationen über einen Sensor und die Messdaten für diese Selektion vorhanden sind. Die Arbeit von Madden, Franklin und Hellerstein [MFH03] beschreibt diesen Weg anhand einfacher Metadaten wie zum Beispiel des Messbereiches. Im Falle einer einfachen Messwertfilterung mittels einer min/max-Schranke mag dies möglich sein, aber für komplexe Anfragen, die zum Beispiel den Observationsbereich eines Sensors einbeziehen, kann dies schon aus Gründen des begrenzten Speichers und der limitierten Rechenperformance nicht im Netzwerk erfolgen. Die Filterung der Informationen muss also zweigeteilt vorgenommen werden: zum einen auf der Seite des Senders, um Bandbreite zu sparen und zum anderen auf der Seite des Empfängers, der allerdings ein Wissen um die Spezifika des Senders benötigt.

3.3.6 ROS

Das mit dem irreführenden Namen Robot Operating System (ROS) bezeichnete open-source Framework kombiniert Konzepte für den Datenaustausch in verteilten Anwendungen mit einer umfangreichen Bibliothek von Hardwareabstraktionen, Gerätetreibern für einschlägige Sensorik/Aktorik und Methoden für robotikspezifischen Aufgabenstellungen [Qui+09]. Ausgangspunkt der Entwicklung war das Stanford Artificial Intelligence Laboratory. Heute wird die Entwicklung hauptsächlich durch die Firma Willow Garage in Verbindung einer Open Source Community vorangetrieben. Als Programmiersprachen werden C++ und Python originär unterstützt, wobei zusätzliche Programmierinterfaces für weitere Sprachen (Octave, Lisp) bestehen. Die Kommunikation innerhalb der verteilten ROS-Knoten kann zum einen über Publish/Subscribe Mechanismen abgewickelt werden. Dabei werden die Nachrichten auf sogenannten Messages abgebildet, die für ein bestimmtes Topic einheitlich aufgebaut sind. Auf der anderen Seite kann der Nutzer die Kommunikation über synchron ablaufende Services umsetzen. In beiden Fällen geht das Konzept von ROS davon aus, dass zur Designzeit alle relevanten Topics bekannt und in der Applikation fest hinterlegt sind.

Das Format einer Message kann sowohl individuell im Programmcode codiert sein oder aber über eine textuelle Beschreibung, wie in Quelltextauszug 3.3 am Beispiel der Stan-

```

1 Header header
2 float32 angle_min      # start angle of the scan [rad]
3 float32 angle_max      # end angle of the scan [rad]
4 float32 angle_increment # angular distance between measurements [rad]
5 float32 scan_time      # time between scans [seconds]
6 float32 range_min      # minimum range value [m]
7 float32 range_max      # maximum range value [m]
8 float32 [] ranges      # range data [m]

```

Listing 3.3: Definitionsdatei der Standard-Laserscanner-Message in ROS

dar Laserscannernachricht gezeigt, definiert werden. Vorteil der letztgenannten Variante ist die allgemeine Verfügbarkeit dieser Definition im gesamten ROS-Netz, sodass andere Entwickler diese im Sinne einer Wiederverwendbarkeit und Allgemeingültigkeit benutzen können. Da die Beschreibungen in verschiedenen Ebenen aufeinander aufbauen können, sind somit komplexe Datenstrukturen möglich. Dabei hält das Framework neben den universellen Typen eine Zahl von quasi standardisierten Formaten vor, die von häufig verwendeten Bibliotheken definiert wurden. Ein Beispiel dafür stellt Quelltextauszug 3.3 dar, der neben dem obligatorischen Header mit Timer- und Knoteninformationen (Zeile 1) sowie Parametern des Sensors (Zeile 2-7) ein Array aus Messwerten (Zeile 8) enthält.

Zwei wesentliche Aspekte werden an diesem Beispiel deutlich. Zum einen müssen in Ermangelung eines Datenblattkonzeptes, wie zum Beispiel in CODES, alle Daten, die für die weitere Verarbeitung wichtig sein könnten, mit jeder Message versendet werden. Im Beispiel macht dies einen Overhead von 6 `float32` Werten aus. Für eingebettete Sensorknoten innerhalb eines Feldbusnetzes würde dies eine unzulässige Performance- und Bandbreitenverschwendung darstellen. Der zweite Kritikpunkt betrifft die Integration von Einheiten in dieser Beschreibungsform. Da die Einheiten lediglich als Kommentar und in autorenenabhängiger Ausprägung beigefügt sind, lassen diese sich automatisiert kaum erfassen und damit nicht zur Interpretation oder Eingabenüberprüfung heranziehen.

Die Stärken von ROS liegen in der Vielzahl von Werkzeugen zur Anwendungsentwicklung, der breiten Entwicklergemeinschaft und der umfangreichen Bibliotheken für autonome Robotikanwendungen. Neben ROS existiert eine große Zahl von vergleichbaren Projekten, wobei aber nur das Microsoft Robotik Studio [Mic11] eine Verbreitung auf ähnlich hohem Niveau erzielt hat. Das Microsoft-Produkt bietet zwar differenziertere und variantenreichere Kommunikationsmechanismen, umfasst aber nicht die Bandbreite an robotikrelevanten Paketen.

3.3.7 Zusammenfassung

Die zuvor genannten Implementierungen umfassen drei für die weitere Entwicklung des Konzeptes wichtige Anregungen:

- Die Möglichkeit der Beschreibung mathematischer Zusammenhänge wie in SensorML vereinfacht die Charakterisierung der Sensorspezifika zum Beispiel im Hinblick auf die Beschreibung des Rauschens oder des nichtlinearen Abbildungsverhaltens erheblich.
- Es sollte eine klare Trennung zwischen statischen Sensoreigenschaften und den veränderlichen Attributen eines Datensatzes bestehen. Damit lässt sich vermeiden, dass ein ressourcenbeschränkter Knoten alle Parameter jeder Nachricht beizufügen hat.
- Die Lesbarkeit einer Beschreibungsdatei steht im Gegensatz zu den möglicherweise sehr komplexen Daten, Paramter usw. Mit geeigneten vorgegebenen Mustern, zum Beispiel im Hinblick auf häufig verwendete Einheiten, sollte diese Aufblähung beschränkt werden.

3.4 Fehlermodelle und Detektionsverfahren

Um die Heterogenität der Fehler der beteiligten Sensoren für eine adaptive Verarbeitung greifbar machen zu können, ist zum einen eine effektive Fehlerdetektion auf mehreren Ebenen – sensorlokal und global – nötig. Zum anderen bedarf es einer Fehlerkategorisierung und -semantik, die die individuellen Eigenschaften des Transducers und der Verarbeitungskette kapseln. Im Folgenden wird daher zunächst ein Überblick über etablierte Detektions- und Identifikationstechniken gegeben und anschließend der Stand der Technik im Bereich der Fehlerabstraktion dargestellt.

3.4.1 Methoden zur Fehlerdetektion

Einen guten Überblick zur Fehlertoleranz aus dem Blickwinkel der Regelungstechnik gibt Isermann [Ise05]. Die als Fault Detection and Isolation (FDI) bezeichnete Werkzeugkette definiert verschiedene Stufen der Fehlerbehandlung. Als erste Stufe sind die Methoden der Fehlerdetektion dafür verantwortlich den eigentlichen Zustand einer Störung festzustellen. Wenn also beispielsweise ein Roboter seine Distanz zu einem Hindernis durch drei heterogene Sensoren misst, kann ein Fehlerzustand im einfachsten Fall durch eine signifikante Abweichung der Messungen voneinander festgestellt werden. In der zweiten Stufe der FDI wird die Herkunft des Fehlers unter dem Begriff Fehlerisolation hinterfragt. Im Rahmen der klassischen FDI-Methoden werden dafür mathematische Modelle des Systems benutzt und ein zu erwartender Messwert bestimmt. Anhand der Abweichung wird die Identifikation fehlerhafter Komponenten durchgeführt. Dem Beispiel folgend wird aus der Kinematik des Roboters und den Bewegungsbefehlen eine Abbildung der Distanzentwicklung erzeugt und für jeden Sensor ein Residuum bestimmt. Aus der Gesamtheit der Residuen wird eine Einschätzung des Fehlerzustandes der Sensoren vollzogen. Im letzten

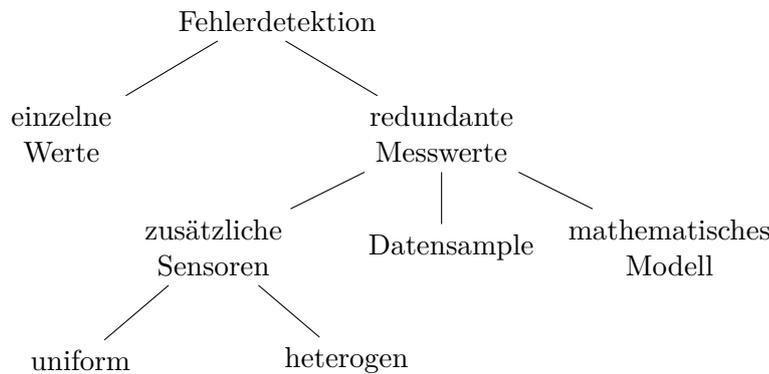


Abbildung 3.11: Fehlerdetektionsansätze in Abhängigkeit von der Sensorkonfiguration eines Systems

Schritt, der Fehleridentifikation, werden der Typ, die Parameter und die Auswirkungen eines Fehlers hinterfragt und davon ausgehend ein Fehlerzustand der Regelschleife abgeleitet [Bla+03]. In der Praxis ist mitunter keine klare Trennung zwischen den einzelnen Schritten der FDI möglich, da die Methoden eng ineinander und mit den eigentlichen Regelungstechnikalgorithmen verknüpft sind. Diese Verflechtung verbietet den Einsatz in einem dynamischen System, wie es im Rahmen dieser Arbeit angestrebt wird.

Systematische Überblicke über Fehlerdetektionsmechanismen geben Meulen, Vachtsevanos u. a. [Meu04, Vac+06]. Letztendlich basiert jede Fehlerdetektion auf dem Vergleich einer Messung mit einer Referenz, wobei diese Redundanz auf verschiedenem Wege erzielt werden kann. Im ersten Fall können dies zusätzliche, kooperativ wirkenden Sensoren sein. Gegen diesen Ansatz sprechen der zusätzliche Energieverbrauch, der notwendige Einbauraum etc. Eine alternative Möglichkeit eröffnet die mathematische Redundanz. Wie bereits im vorherigen Absatz angedeutet, wird dabei das Systemverhalten modelliert und nachgebildet, um einen theoretischen Sensorwert zu bestimmen. Voraussetzung dafür ist eine hinreichende Kenntnis über die Zusammenhänge und Parameter des Systems und eine genügend leistungsfähige Rechenkomponente. Die dritte Möglichkeit ist die Erzeugung von Redundanz aus der Historie der Messwerte, also über Fenster von n Samples.

Die Methoden der Fehlerdetektion reichen dabei von einfachen Tests, die das Überschreiten des Messbereiches erfassen, bis zu komplexen stochastischen Filteralgorithmen. Im Hinblick auf die beschränkten Ressourcen eines Sensorknotens sind diese aber nicht immer lokal umsetzbar, sondern müssen teilweise auf leistungsstärkere Knoten im Netzwerk ausgelagert werden. Damit muss der Anwendungsentwickler auf eine Ausgewogenheit zwischen erreichbarer Detektionswahrscheinlichkeit, Ausführungszeit und Kommunikationsaufwand achten. Eine automatisierte Lösung dieses Problems ist in [Aga+02]

durch Agarwalla u. a. präsentiert worden. Die Autoren beschreiben eine selbständige Suche nach einer optimalen Ausführungslokalität für einen Code, sodass die Ressourcen optimal genutzt werden. Im Vordergrund dieser Arbeit standen aber Fusionstrategien und keine Fehlerdetektionsmethoden.

Hardwareredundanz

Hardwareredundanz wird in sicherheitskritischen Applikationen auf unterschiedlichen Ebenen umgesetzt [Nel90]. Redundante heterogene oder homogene Sensoren erfassen einen Umgebungswert oder damit verbundene Größen. Fehlerhafte Sensoren können dann über den Vergleich und eine Majoritätsentscheidung eines k-aus-n Voters festgestellt werden [BPOARLS07]. Damit ist festgelegt, dass die maximal als fehlerhaft detektierbare Zahl von Sensoren durch die Zahl der redundanten Sensoren bestimmt wird. Im Hinblick auf eine dynamische Applikation ist dies eine erhebliche Einschränkung. In [Mar90] abstrahiert Marzullo zunächst das Messverhalten eines Sensors durch die Streubreite und den Messwert, um dieses zusätzliche Wissen in die Detektion einfließen zu lassen.

Eine andere Form der Ausnutzung der Hardwareredundanz beschreiben Koushanfar, Potkonjak und Sangiovanni-Vincentelli [KPSV03]. Hier werden mit großen Rechenaufwand alle Kombinationen der redundanten Messungen fusioniert und die unterschiedlichen Resultate auf Ausreißer hin untersucht. Wenn die Außerachtlassung eines Messwertes eine Stabilisierung des Ergebnisses bewirkt, so wird dieser als fehlerhaft markiert.

Forschungen im Bereich der Fehlertoleranz in Sensornetzen zielen häufig auf die beschränkte Kommunikationsbandbreite und die beschränkte Rechenperformanz der eingebetteten Sensoren ab. Beide Punkte unterbinden einen kompletten Austausch aller Messungen und deren lokale Verarbeitung, um eine konsistente Einschätzung der Umgebung zu gewinnen [Aky+02]. Mögliche Lösungen basieren auf unterschiedlichen Clusterkonzepten, die eine lokale Auswertung vornehmen, um deren Resultate auf einer höheren Ebene weiterzuverarbeiten [GY03]. Ein anderer Ansatz, der im Rahmen dieser Arbeit entwickelt wurde, kombiniert lokale Detektionsmechanismen mit dem an alle Sensorknoten überstellten zentralen Fusionsergebnis [ZD10]. Damit ist eine präzise Bewertung der Messdaten anhand eines validen Referenzwertes möglich.

Mathematische Redundanz

Ein anderer Ansatz generiert aus einer mathematischen Redundanz simulierte Messungen als Referenzgrößen. Zu diesem Zweck leitet der Entwickler ein Modell aus einer vereinfachten Annahme über das beobachtete System ab und nutzt dafür physikalische Gleichungen für Masse, Kraft, Impuls etc., um das Systemverhalten nachzubilden. Für stochastische Systeme werden solche Modelle dann mit (verteilten) Kalman oder Particle Filtern kombiniert [ZJJ02]. Diese Ansätze berücksichtigen unter anderem das

Messrauschen der Sensoren und die Unsicherheit der Modellannahmen, um eine optimale Schätzung im Hinblick auf die Wahrscheinlichkeit des Ergebnisses zu erlangen. Die Modelle ermöglichen es aus bekannten Eingangsgrößen, wie den Motorkommandos, auf das Verhalten des Roboters und damit auf die zu erwartenden Messwerte der Sensoren zu schließen [Bla+03]. Der Fehlerdetektionsansatz analysiert die Residuen aus diesem Vergleich und hinterfragt die Signifikanz der Abweichung. Dafür existiert eine große Zahl von Modellierungstechniken und Analysemethoden zur Validierung der Residuen, die auf unterschiedliche Modellannahmen, Senorkonzepte und Grade an Verteiltheit zugeschnitten sind [Ise05, Pat97]. Die Ableitung der möglichen Fehlerzustände aus den Residuen reicht von einfachen Schwellwertentscheidungen bis hin zu regelbasierte Implementierungen oder Fuzzy-Techniken [FKS97].

Die Nutzung der Vorhersagen eines modellbasierten Fusionsalgorithmus zur Einschätzung der Gültigkeit von Messwerten wird in [SF88] vorgestellt. Dabei wird ein Abstandswert zwischen der Apriori-Schätzung und dem Messwert unter Berücksichtigung von deren statistischen Eigenschaften berechnet und daraus entweder eine binäre Entscheidung [SN04] oder eine kontinuierliche Validitätseinschätzung bestimmt. In [AAM01] werden für diese Abbildung unterschiedliche statistische Abstandsmaße und Heuristiken diskutiert und verglichen.

Signalanalyse

Analog zur mathematischen Modellbildung nutzt die Signalanalyse Annahmen über die Parameter eines Fensters von Messungen als Referenzwert. Die einfachste Variante prüft anhand eines einzelnen Wertes die Einhaltung eines zulässigen Messbereiches. Unter Zuhilfenahme eines weiteren Wertes kann die Änderung und damit die Steigung des Signalverlaufs auf atypisches Verhalten hin geprüft werden. Für eine größere Zahl von Messungen kann ein laufender Mittelwert, Median etc., berechnet und zur Validierung benutzt werden. Das Kriterium für die Einhaltung von bestimmten Parametern muss dabei nicht als einfacher Grenzwert definiert sein, sondern sollte im Hinblick auf die Unsicherheit der Messungen mit statistischen Mitteln wie x-Perzentil, Varianz etc. [HKU01] auf seine Signifikanz untersucht werden.

Im Unterschied zur systemmodellbezogenen Validierung ist der Ansatz der signalorientierten Messwertprüfung unabhängig von Unwägbarkeiten des Systemverhaltens. Insbesondere in den angestrebten dynamischen Szenarien mit einer großen Zahl an Freiheitsgraden ist die präzise Nachbildung eines Sensors schwierig. Die Signalanalyse ist an dieser Stelle robuster und einfacher umzusetzen. Die Änderung der Signalparameter kann dabei in unterschiedlichen Richtungen interpretiert werden. Aus dem Blickwinkel der Fehlerdetektion wird der Sensor als fehlerhaft eingestuft, wenn sich zum Beispiel die Autokorrelationsfunktion des Signals verändert. Im Unterschied dazu gehen Maschinenbau- oder Bauingenieure im Bereich der Maschinenüberwachung oder des Bauwerksmonitoring den umgekehrten Weg [WMF00, DFP98]. Hier wird angenommen, dass das beobach-

tete System seine Eigenschaften verändert hat und sich in einem veränderten, eventuell in einem fehlerhaften Modus befindet. Die Identifikation des eigentlichen Fehlers hängt also unmittelbar von den Annahmen des Nutzers ab, eine Validierung im Szenario ist nur mit zusätzlicher Messtechnik möglich.

3.4.2 Fehlermodelle

Eine grundlegende Einteilung treffen die englischen Begriffe Failure, Fault und Error. Faults sind in jeder technischen Applikation inhärent vorhanden und bezeichnen Bauteile, Codeabschnitte usw., die unter bestimmten Bedingungen ein Fehlverhalten auslösen können. Der Error beschreibt das Erreichen dieses Zustandes, wobei sich aus der Manifestation eines objektiven Fehlverhaltens ein Failure ergibt. Die Einordnung einer Störung in dieser Hierarchie hängt vom Blickwinkel der Betrachtung ab. Eine auf ein Sensorelement wirkende Störung, die für den Sensor selbst einen Failure darstellt, weil ein nicht korrekter Messwert ausgegeben wird, kann auf der Ebene des Roboters als Fault bzw. Error eingestuft werden, sofern das System in seinem Verhalten nicht beeinträchtigt ist [ALR01].

Um eine sensorübergreifende Darstellung von Fehler umzusetzen, wird die Abweichung des Messwertes von der realen Größe in geeigneten Modellen gefasst. Um diese zu systematisieren, führt Souza [Sou07] fünf Kategorien ein und ordnet Fehler als Crash, Auslassungs-, Zeit-, Wert- und Zugriffsfehler:

- Crash oder Auslassungsmodelle sind durch den kompletten Ausfall eines Systems oder das Ausbleiben von Daten gekennzeichnet.
- Messwertbezogene Modelle beschreiben die Dauer der Einwirkung und die zeitliche Veränderlichkeit der Größe des Fehlers. Entsprechend ergeben sich in vielen Kategorisierungsansätzen zum Beispiel aus [Her+08, Bla+03, Jay94] Kernmodelle wie Stuck-At, Drift und Outlier.
- Treten Störungen bei der Zuordnung eines Messwertes zu einem Messzeitpunkt auf, so ergibt sich ein Fehler in Abhängigkeit von der Dynamik des Systems.
- Der Begriff der Zugriffsfehler aus [Sou07] zielt auf Byzantinische Fehler, die eine inkonsistente Verbreitung der Messdaten an verschiedene Nachfrager beschreiben [LSP82].

Andere zusammenfassende Darstellungen zu Sensorfehlern und deren Modellierung sind in [Fra90, Ni+09] enthalten. Anhand des Modells eines Sensorfehlers können die Detektionsmethoden ausgewählt und parameterisiert werden.

3.4.3 Fehlersemantik

Wie lassen sich die heterogenen und für ein Sensorsystem kombiniert auftretenden Fehlermodelle in einem verteilten System handhaben? Dazu bedarf es einer Fehlersemantik, die der Verarbeitung die Bewertung eines Datensatzes erlaubt.

Im einfachsten Fall wird von einem Fail-Silent-System ausgegangen, bei dem für alle Elemente und Ausgabewerte kontinuierlich eine Prüfung auf die relevanten Fehler vollzogen wird. Mit der Detektion einer Störung werden keine Daten mehr publiziert. Sämtliche Fehler werden in eine einzige nach außen sichtbare Fehlerart umgewandelt – die komplette Abschaltung der betroffenen Komponente. Voraussetzung dieses Konzeptes ist eine sichere Detektion der unterschiedlichen Fehler [ALR01]. Pike u. a. [Pik+04] machen die Entscheidung über den Übergang in den Fail-Silent-Zustand neben der Messwertbeurteilung von zwei weiteren Attributen des Knotens abhängig, die sich aus dessen Einbindung im Netzwerk ergeben. Die bisher versendeten Nachrichten müssen von mindestens einem als fehlerfrei eingeschätzten Knoten akzeptiert worden sein und alle die Nachricht empfangenen Knoten haben signalisiert, dass ein einheitlicher Datensatz empfangen wurde. Damit wird die Entscheidung über den Fehlerzustand nicht mehr nur sensorlokal, sondern übergreifend bestimmt.

Eine differenziertere Fehlerdarstellung erlaubt die Systematik von Sharma, Golubchik und Govindan [SGG07]. Sie bildet die oben genannten messwertbezogenen Fehlermodelle auf drei Metamodelle ab: „Short“, das eine sprunghafte Änderung des Wertes signalisiert, „Noise“, mit dem stochastische Fehler abgebildet werden und „Constant“, in dem ein zeitabhängiger Bias berücksichtigt wird.

In den Arbeiten von [Pio07, Suk+, EPS04] wird das Ergebnis der Fehlerprüfung in einer Validitätseinschätzung, die durch einen einzelnen, kontinuierlichen Wert repräsentiert wird, abgebildet. Dieser Wert wird jeder Messung beigelegt und überträgt die Entscheidung über den Umgang mit einem möglicherweise fehlerhaften Wert, entgegen dem Fail-Silent-Ansatz, an die nachgeordneten Knoten. Die Bestimmung dieses Validitätswertes, der eine objektive und mit anderen Knoten vergleichbare Größe darstellt, wird unter anderem in [RN04] in seiner aktuellen Entwicklung diskutiert. Für eine binäre Entscheidung stellt sich zum Beispiel die Frage nach der Auslegung des Grenzwertes, ab dem eine Störung angenommen wird [Fra93]. Wird diese auf einem zu geringen Niveau festgesetzt, steigt die Zahl von unberechtigten Fehlerannahmen. Für einen zu hohen Wert besteht die Gefahr nicht erkannter Fehlersituationen. Entsprechend zielen die Arbeiten von [Fra93] und [GA00] auf einen adaptiven Grenzwert ab und bedienen sich dabei der Prinzipien der Fuzzy-Systeme.

3.4.4 Zusammenfassung

Aus der Veränderlichkeit und Heterogenität der aggregierten Sensordaten folgt ein nicht statisches Spektrum möglicher Messfehler. Die Literatur bietet zwar eine große Zahl an

Detektions- und Identifikationsmethoden, die in den folgenden Abschnitten überblicksartig vorgestellt werden sollen, letztendlich werden aber isolierte und auf einen konkreten Anwendungsfall zugeschnittene Methoden präsentiert. Die für die dynamische fehlertolerante Datenaggregation in verteilten Systemen nötige Aneinanderreihung von sensor-lokalen Techniken, clusterbezogenen Detektionsmethoden und multi-modale Redundanz nutzenden Ansätzen kann in keiner der ausgewerteten Arbeiten erkannt werden. Um eine solche verteilte Detektionskette umsetzen zu können, bedarf es einer gemeinsamen Semantik der Fehlereinschätzung, einer darauf aufbauenden Datensatzselektion und einer flexiblen Integration der Fehlerdetektionsmethoden in die *Smart-X*-Architektur.

3.5 Datenselektion und adaptive Verarbeitung

3.5.1 Selektion

Generell kann von der These ausgegangen werden, dass mit steigendem Informationsangebot die Güte einer daraus fusionierten Größe wächst. Dabei zeigt die Arbeit von Nahin und L. [NL80], dass der Einfluss zusätzlicher Sensoren ab einer bestimmten Anzahl von bereits vorhanden Informationsquellen marginal wird. In Abbildung 3.12 wird von einem Mehrheitsentscheider ausgegangen, wobei die Güte der Messung auf der x-Achse abgetragen ist. Die y-Achse zeigt die Änderung der Wahrscheinlichkeit, dass eine korrekte Entscheidung getroffen wurde.

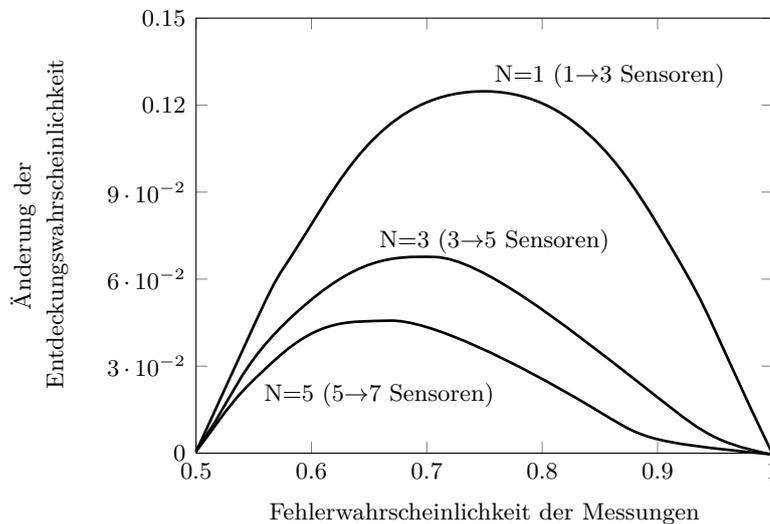


Abbildung 3.12: Änderung der Aussagengüte eines Voters in Abhängigkeit von der Zahl der Sensoren, nach [NL80]

Eine maximale Steigerung lässt sich also erzielen, wenn statt einem, drei Sensoren verwendet werden und die Wahrscheinlichkeit einer korrekten Messung für jeden Sensor bei etwa 0,75 liegt. Eine Steigerung von fünf auf sieben Sensoren bewirkt eine deutlich geringere Verbesserung, nämlich von 4,1 Prozent für Sensoren, die mit einer Wahrscheinlichkeit von 0,66 ein fehlerfreies Resultat generieren. Es ist zu prüfen, ob der erhöhte Aufwand, der sich aus dem Einsatz von zwei zusätzlichen Sensoren im Hinblick auf Bau- raum, die Energieversorgung und die Beschaffungskosten erbracht werden muss, durch die gering verbesserte Güte des Ergebnisses gerechtfertigt ist. Die genannte Untersuchung gibt einen guten Eindruck von der Notwendigkeit einer Abstimmung zwischen der Zahl der eingebundenen Sensoren und dem zu erwartenden Fusionsresultat, lässt sich aber wegen des einfachen Sensor- und Fusionsmodells nur schwer auf reale Systeme in multi-modalen Fusionsszenarien übertragen.

Wegen dieser Vielschichtigkeit sind zumeist keine zusammenfassenden Aussagen wie in Abbildung 3.12 möglich, sondern es muss eine differenzierte Betrachtung für alle verschiedenen Kombinationen von Sensoren erfolgen, um eine bestimmten Kriterien entsprechende, optimale Zusammenstellung zu erzielen. Die Zahl der zu untersuchenden Kombinationen v lässt sich dabei mit einer erweiterten Permutationsformel

$$v = \sum_{k=1}^{k=n} \frac{n!}{(n-k)!k!} = 2^n - 1 \quad (3.1)$$

abschätzen, wobei n die Anzahl der verfügbaren Sensoren repräsentiert und k die Laufvariable darstellt, die die Menge der ausgewählten Sensoren von 1 bis n definiert. Für fünf Sensoren ergeben sich damit 15 mögliche Kombinationen, wobei die Auswahl einzelner Sensoren genauso wie die Integration aller Sensorelemente erfasst ist. Aus 10 Sensoren folgt bereits eine Zahl von 1023 möglichen Kombinationen.

Allein diese Größenordnung, die deutlich unter dem für intelligente Umgebungen erwarteten Datenaufkommen liegt, lässt auf eine Herausforderung des Selektionsprozesses schließen. Der Aufwand für die eigentliche Auswahl steigt mit der Zahl der Sensoren rasch an und übertrifft unter Umständen den Berechnungsbedarf für die eigentlich zu umgehende vollständige Fusion aller Daten. Folglich muss eine Selektion den Suchraum effektiv einschränken und zum Beispiel Temperatursensoren vor einer Positionierungsaufgabe verbergen oder vermutlich fehlerbehaftete Messungen ohne aufwendige Prüfungen verwerfen. Die dafür notwendige Konfiguration der Selektionsmechanismen kann entweder statisch zur Compilezeit oder dynamisch zur Laufzeit erfolgen. Im Folgenden werden daher bestehende Arbeiten aus beiden Bereichen gegenübergestellt.

Sensorselektion zur Compilezeit

Der in der Literatur als statische Sensorselektion oder Sensoreinsatzplanung [MB01, MEU07] beschriebene Vorgang legt die für eine konkrete Applikation nötigen Sensoren

während der Entwicklung fest. Ausgangspunkt dafür ist eine Analyse der Messaufgabe, die Beschreibung der Randbedingungen des Szenarios, die Definition der erwarteten Messgüte, Taktfrequenz sowie die Schnittstellen des Systems [MÖ9]. In [YS00] beschreiben die Autoren eine Strukturierung der Eigenschaften von Sensoren, die als Auswahlkriterien in der Entwicklung eines mechatronischen Systems dienen können. Sie werden mit Gewichtungsfunktionen belegt und einer Performanceschätzung gegenübergestellt. Diese Vorgänge sind jedoch Bestandteil des Entwicklungsprozesses und für die adaptive Selektion nicht relevant.

Wenn aber anhand einer mathematischen Analyse die Auswirkungen eines Sensorsystems auf das Gesamtsystem untersucht werden, so lassen sich diese Konzepte auch auf die avisierte dynamische Auswahl übertragen. Marc van de Wal und Bram de Jager [MB01] fassen die Arbeiten aus dem Fokus einer regelungstechnikspezifischen Sensorvalidierung zusammen und identifizieren zum Beispiel die Stabilität des Regelkreises, resultierende zu erwartende Regelfehler usw. als Parameter einer solchen Auswahl. Hinterfragt werden müssen dabei unterschiedliche Zustände des Systems (Initialisierung, Anfahren, Dauerbetrieb usw.) oder aber die möglicherweise eintretende wechselseitige Beeinflussung bestimmter Sensoren oder Aktorsysteme, die entsprechend nicht zur gleichen Zeit betrieben werden.

Ein weiterer Gesichtspunkt ist die optimale Positionierung der Sensorik bezüglich des zu beobachtenden Phänomens. Zielstellung ist hierbei die Abdeckung eines bestimmten Überwachungsbereiches [Mit07] oder aber die effiziente Erfassung einer spezifischen Größe [VW+00, MZ05]. Die Beantwortung dieser Frage hängt in starkem Maße vom konkreten Verhalten des observierten Systems ab. Um beispielsweise die Verformung einer Brücke zu überwachen, ist es notwendig, anhand eines physikalischen Modells die besonders stark betroffenen Bereiche zu ermitteln und diese gezielt zu überwachen. Daneben existiert ein ganzes Feld von Algorithmen aus der Optimierungsrechnung, die auf der Basis von System- und Fehlermodellen optimale Sensorpositionen bestimmen. In der Regel gehen diese Arbeiten von sehr idealistischen Annahmen - omnidirektionale Sensoren, keine Laufzeiten der Signale [ZJJ02] - aus. Abweichend dazu entwirft zum Beispiel [MEU07] aufbauend auf [Vij+06] ein Sensorscheduling.

Sensorselektion zur Laufzeit

Im Unterschied zur statischen entwicklungsbegleitenden Auswahl validiert die Sensorselektion zur Laufzeit neben den fixen Sensorparametern auch die individuellen Informationen der eintreffenden Datensätze. Dabei ist die Selektion eng mit dem Sensormanagement [XS02, Ben02] im Sinne des Datenfusionsmodells des Joint Directors of Laboratories (JDL) verknüpft, das im nachfolgenden Abschnitt einleitend beschrieben wird. Ausgehend von einer Validierung der Messwerte werden in einem Regelkreis die Parameter der Sensorik angepasst und zum Beispiel der Fokus eines Kamerasystems neu gesetzt, Crowley und Demazeau [CD93] sprechen dabei von „Control of Perception“. Die

dafür bereitzustellenden Informationen wie auch die Analysemethoden lassen sich analog für die Selektion von Messdaten vor einer Fusion benutzen.

Shalom und Fortmann [SF88] definieren im Zusammenhang mit dem Kalman-Filter einen als „validation gate“ bezeichneten Selektionsansatz, der vor der Einbeziehung eines Messwertes in die Validierungsphase eine Einschätzung von dessen Gültigkeit anhand der Vorhersage übernimmt. Diese Idee wird mehrfach aufgegriffen und weiterentwickelt [SN04].

Tabelle 3.4: Beispiele für Optimierungskriterien aus der Sensorselektion nach [DS04] und Guibas und Zhao [GZ04]

MOE Kategorie	Beispiele
Detektion	Sensorreichweite Anteil der detektierten Ziele in bestimmter Distanz Anzahl der undetektierten Ziele im Observationsbereich Abdeckung des Observationsbereiches
Target Tracking	Varianz des Tracks Zahl der Tracks/Zahl der Ziele (Zuordnungsproblem)
Identifikation	Anteil der korrekt identifizierten Ziele Reichweite der Identifikation
Emission	Gesamtsendeleistung Sendeleistung pro Zeit
Rechenaufwand	Dauer Speicher Busbandbreite
Abbildungsverhalten	Antwortzeit des Sensors Auflösung

Dabei stellt sich die Frage, welche Kriterien für die Beurteilung eines Datensatzes herangezogen werden müssen und somit Eingang in die Datenbeschreibung finden sollten. David L. Hall und Sonya A. H. McMullen [DS04] definieren für die Relevanzbewertung den Begriff der Measure of the Effectiveness (MOE) und fassen darunter einen Kriterienkatalog zusammen, der mögliche Heuristiken für das Tracking eines Roboters aufführt.

In Sensornetzen stehen die Arbeiten zur Selektion, wie Tabelle 3.4 in der vierten Kategorie zeigt, von Datensätzen in unmittelbarem Zusammenhang mit der Reduktion des Kommunikationsaufwandes. Beispielhaft vorgestellt sei dafür die Arbeit von Chu, Hauss-

ecker u. a. [CH+02], die einen kommunikationsreduzierenden Auswahlprozess eines Clusterhauptknotens beschreibt. Dabei wird die Unterschreitung eines global vorgegebenen Unsicherheitsniveaus angestrebt und die verfügbaren Sensoren darauf hin untersucht, welche maximale Auswirkung sie darauf haben. Spezifisch für die Sensornetze ist dabei, dass für diese Berechnung davon ausgegangen wird, dass der eigentliche Messwert nicht vorliegt, da dieser sonst eventuell überflüssigerweise kommuniziert werden müsste. Analog schreiben Zhao, J. Und J. [ZJJ02, Seite 4] dazu:

„Our goal is to predict the information utility of a piece of nonlocal sensor data before obtaining the data. In practice, the prediction must be based on the currently available information: the current belief state, the characteristics of the sensor of interest which includes information such as the sensor position and sensing modality that can be established beforehand.“

Die Untersuchung des Einflusses auf die Unsicherheit der Gesamtschätzung basiert also allein auf dem Wissen um die Unsicherheit der Messung des einzelnen Sensors. In [CH+02] werden Messungen simuliert und die durchschnittliche Verbesserung, die beste Steigerung der Güte oder die schlechteste Auswirkung als Kriterium für die Filterung realer Messungen verwendet.

Der Selektionsvorgang wird in der Arbeit von Kumar u. a. [Kum+08] durch einen zusätzlichen Middlewayerlayer gekapselt, welcher Ausgaben realer Sensoren auf einen Mobile Virtual Sensor (MVS) abbildet. Dieser übernimmt den Auswahlprozess auf der Basis der vom Anwender vorgegebenen Informationen. Als Applikationsbeispiel diente die Verfolgung von Personen mit mehreren Kameras. Anhand einer statischen und den MVSs bekannten Karte werden die für einen konkreten Aufenthaltsort relevanten Kameras ausgewählt, eine gemeinsame Schätzung zur Position errechnet und das Resultat an die Applikation ausgegeben.

In [GB97] werden die statische und die dynamische Selektion zu einem hybriden Konzept verschmolzen und auf der Basis probabilistischer Annahmen offline eine Look-Up-Tabelle erstellt, die ausgehend von einzelnen Zuständen und den möglicherweise eingehenden Sensordaten den daraus resultierenden Informationsgewinn analysiert. Entsprechend entsteht ein vordefiniertes System, wobei die Berechnungsdauer und die Unsicherheit der Zustandsschätzung gegeneinander aufgerechnet werden. Beispielanwendung ist eine Montageaufgabe für einen Manipulator, wobei ein fester Sensorsatz angenommen und für verschiedene Systemzustände variabel ausgewertet wird.

3.5.2 Fusion

Die Zusammenführung einzelner Datensätze zielt auf die Erhöhung der Validität und der Fehlertoleranz ab, aber auch auf die Bestimmung von Systemgrößen, die über ein physikalisches Modell gewonnen werden und nicht als Messwerte vorliegen. Ein Beispiel dafür ist die Berechnung der Orientierung auf der Basis eines Drehratengebers. Ausgehend von

der zu erwartenden Variabilität der verfügbaren Sensoren und Messdatensätze werden im Folgenden Methoden der adaptiven Fusion kategorisiert. Zuvor wird untersucht, wie sich der Ansatz in die Basiskonzepte der Datenfusion einfügt.

Fusionsarchitekturen

Das JDL-Modell stellt eine Struktur von Verarbeitungsebenen für Sensormessungen bereit, die auf fünf Ebenen gegliedert ist:

Vorverarbeitung (Level 0) Mit der Vorverarbeitung, die individuell für jeden angeschlossenen Sensor erfolgt, werden diese gefiltert und linearisiert, Fehler detektiert usw. Die Vorverarbeitung entlastet die objektbezogene Datenfusion.

Objektverarbeitung (Level 1) Die Zusammenführung aller Informationen, die sich auf ein einzelnes Objekt, einen Roboter, einen Sensor usw. beziehen, erfolgt an dieser Stelle.

Situationverarbeitung (Level 2) Auf dieser Ebene werden die aus dem vorhergehenden Level gewonnen Informationen über die einzelnen Objekte unter Berücksichtigung der Relationen zueinander in einer übergreifenden Situationsdarstellung zusammengefasst.

Threadverarbeitung (Level 3) Anhand von mathematischen Modellen werden in dieser Ebene Aussagen zur zukünftigen Entwicklung der Umgebung getroffen.

Prozessmanagement (Level 4) Das Prozessmanagement umfasst unter anderem die Akquise und Konfiguration der Sensoren auf der Basis einer Einschätzung der Effektivität der Verarbeitung.

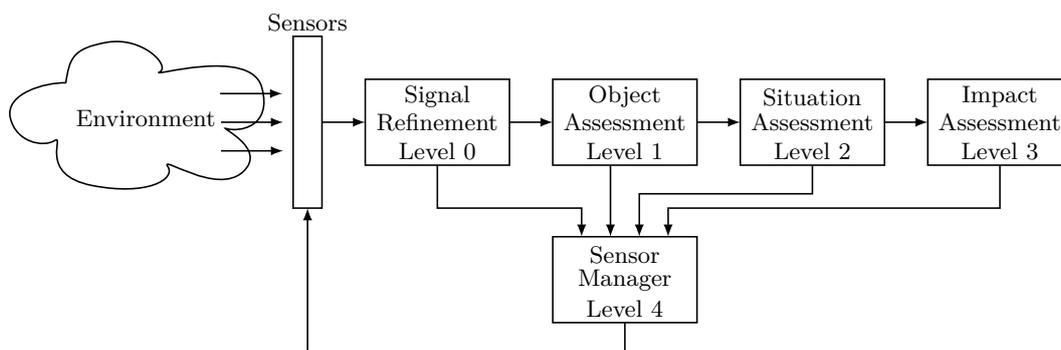


Abbildung 3.13: JDL Ebenen der Datenfusion nach [XS02]

Aus dem Motivationsszenario von *MOSAIC* folgt, dass von der angestrebten Architektur die Ebenen 0 bis 2 und 4 zwingend abzudecken sind. Die Mechanismen der Level 0 und

4 sollten dabei weitgehend automatisiert umgesetzt werden, sodass dem Anwendungsentwickler die Implementierung der eigentlichen Anwendung in Level 1 und 2 obliegt.

Adaptive Fusionsansätze

Adaptivität im Bereich der Datenfusion bezieht sich in erster Linie auf die Anpassung der Fusionsparameter unter Berücksichtigung eines geänderten Kontextes [EAM02]. Dazu zählt beispielsweise die Korrektur der Rauschannahmen eines Sensors über die Änderung seiner Kovarianzmatrizen oder der zugehörigen Gewichtungsfaktoren im Fusionsalgorithmus. Ein weiterer Grund zur Anpassung kann die Veränderung bestimmter Modellparameter und dabei insbesondere die Modellunsicherheit sein. Hu u. a. [Hu+03] beschreiben einen Kalman-Filter zur Positionsschätzung von Automobilen. Die Unsicherheit hinsichtlich des Bewegungsmodells lässt sich unter anderem durch einen weiteren Filter minimieren, der die Parameter des Modells schätzt und so eine rasche Anpassung an tatsächliche Zustandsänderungen (Wechsel von Bremsvorgang auf positive Beschleunigung) gewährleistet. Andere Ansätze verwenden für die Schätzung der Parameter Fuzzy-Systeme [Loe+04] oder neuronale Netze.

Eine alternative Vorgehensweise kann in der Integration verschiedener Modelle für ein beobachtetes System liegen. In diesem Fall wird von sogenannten Filterbänken gesprochen, die jeweils einen bestimmten Zustand des beobachteten Systems abbilden, in ihrer Kombination ein nicht gaußsches Rauschen normalisieren [QM10, SS06] oder eine bestimmte Sensorkonfiguration umsetzen [DMC00]. Letztgenanntes Beispiel hält sieben Kalman-Filter für unterschiedliche Sensormessdaten eines autonom navigierenden U-Bootes bereit. Die Schwankung der Zahl der verfügbaren Messwerte resultiert aus den unterschiedlichen Lagen des Bootes im Wasser. Die Integration multipler Modelle wird parallel in der Fehlerdetektion und -identifikation benutzt, indem für die Steuervorgaben die zu erwartenden Ausgaben des Systems bestimmt und diese mit den Reaktionen des Modells verglichen werden.

Ein weiterer, in den bisherigen Arbeiten häufig thematisierter Punkt, ist das Out of Sequence Measurement (OOSM) Problem, also das Eintreffen von Messdaten, deren Entstehungszeitpunkt vor dem zeitlichen Bezugspunkt der aktuellen Schätzung liegt [MM03]. Der Entwickler hat beim Entwurf des Systems entsprechend zu entscheiden, ob diese nachträglich verfügbaren Informationen ignoriert oder in die Schätzung aufgenommen werden [BS02]. Für den Fall eines Systems mit hoher Dynamik ist die von einem verspätet eintreffenden Datensatz her motivierte Neuberechnung der Schätzungen ab dem Entstehungszeitpunkt zwingend [Mau+06]. Dafür existiert eine Vielzahl von approximierenden und analytischen Lösungen, zusammengefasst unter anderem in [SS06], von denen beispielhaft hier auf die Arbeit von [Lar+98] mit einer auf der Approximation des Messwertes basierenden Lösung verwiesen sei, die durchaus auch auf eingebetteten Systemen umgesetzt werden kann. Rhéaume und Benaskeur [RB07] stellen dazu einen interessanten Ansatz vor, der es ermöglicht, vor der Integration eines verspäteten Wer-

tes eine Abschätzung zu dessen Einfluss auf das Fusionsergebnis zu treffen. Dazu wird offline eine Tabelle erzeugt, aus der in Abhängigkeit des Verhältnisses zwischen Modellrauschen und Sensorrauschen ein Maß der Verbesserung abgelesen werden kann. Diese Herangehensweise stellt gleichzeitig eine interessante Methode der Datenselektion dar.

3.5.3 Zusammenfassung

Anhand der Darstellung der Selektionsmechanismen und der Methoden zur adaptiven Datenverarbeitung wird deutlich, dass die an das *MOSAIC*-Framework gesetzten Anforderungen mit keiner der Lösungen umfassend abgedeckt werden können. Dies resultiert aus dem starken Zuschnitt vieler bestehender Lösungen auf eine konkrete Applikation. Die für den Entwurf einer Architektur für *Smart-X* nötige Generalität ist damit nicht gegeben. So ist dem Autor insbesondere keine Beschreibung einer Filterkette bekannt geworden, die eine die gesamte Verarbeitung begleitende Selektionsmöglichkeit bietet. Trotzdem lassen die zuvor genannten Arbeiten im Hinblick auf die Selektion drei Leitlinien sichtbar werden. Die erste Kategorie untersucht statisch den Sensortyp, also die grundlegende Eignung. Die zweite Filterkategorie wertet die Parameter des Sensors zuzüglich der Positions- und Orientierungsangaben aus. Letztendlich folgt die Untersuchung des eigentlichen Messwertes.

Die beschriebenen Selektionsmechanismen liefern eine wichtige Anregung für die weitere Arbeit und finden sich systematisch ausgebaut in als elementarer Bestandteil in der *Smart-X*-Architektur wieder. Demgegenüber dient die Betrachtung der adaptiven Fusionsansätze eher der Definition der Anforderungen für die Integration der anwendungsorientierten Verarbeitungsfunktionalität. Der Entwurf der Architektur muss entsprechend darauf ausgerichtet sein, dass alle notwendigen Informationen für die Verarbeitung eines Datensatzes bereitstehen.

4 Konzepte des *MOSAIC*-Frameworks

Dieses Kapitel greift die im Motivationsszenario benannten ersten drei Kernfragestellungen dieser Arbeit auf: Datenvalidität/Fehlerdetektion, Datenbeschreibung sowie Selektion und dynamische Fusion. Ausgehend von den Erkenntnissen aus dem Stand der Forschung und dem Anforderungskatalog des Kapitels 2 wird die konzeptionelle Grundlage des *MOSAIC*-Frameworks entwickelt.

4.1 Basisabstraktionen

4.1.1 Smart-X

Die Dynamik der Datenakquisition und -verarbeitung der *Smart-X* wirkt in zwei Richtungen. Zum einen müssen die unterschiedlichen Datenquellen, also Sensoren oder Fusionsknoten, aggregiert werden, um darauf aufbauend deren Daten, die in variierender Zahl eintreffen, in die Applikation einzubeziehen. Die Adaptivität zielt damit auf eine flexible Integration von verschiedenen Sensoren zur Laufzeit und ein anpassungsfähiges (Mess-)Datenmanagement ab. Abbildung 4.1 illustriert diese Erweiterung gegenüber der üblichen statischen Sensorkonfiguration durch das *MOSAIC*-Framework in einer 2x2-Matrix. Diese differenziert zwei Möglichkeiten zum Zustandekommen einer variablen Zahl von Datensätzen zur Laufzeit:

- Die Abszisse kategorisiert Systeme, für die die Zahl der Datensätze, die von einer fest definierten Sensorkonfiguration generiert werden, schwankt. Diese Variabilität kann mehrere Gründe haben. Erstens ist der Sensor aus der Verteiltheit des Systems möglicherweise gar nicht erreichbar. Zweitens genügen die kommunizierten Datensätze eventuell nicht den (Güte-)Anforderungen des verarbeitenden Knotens oder sind drittens gar nicht relevant. Ein Verarbeitungsalgorithmus ist entsprechend auf diese veränderliche Zahl auszurichten.
- Eine weitergehendere Steigerung der Variabilität ergibt sich, wenn, wie entlang der Ordinate gezeigt, auch die Art und Anzahl der vorhandenen Sensortypen veränderlich ist. Ausgehend von der Mobilität und den damit variabel verfügbaren Sensoren wird die Sensorkonfiguration permanent angepasst. Dies setzt einen größeren Aufwand im Hinblick das Discovery der Knoten, die Datenselektion und -verarbeitung voraus.

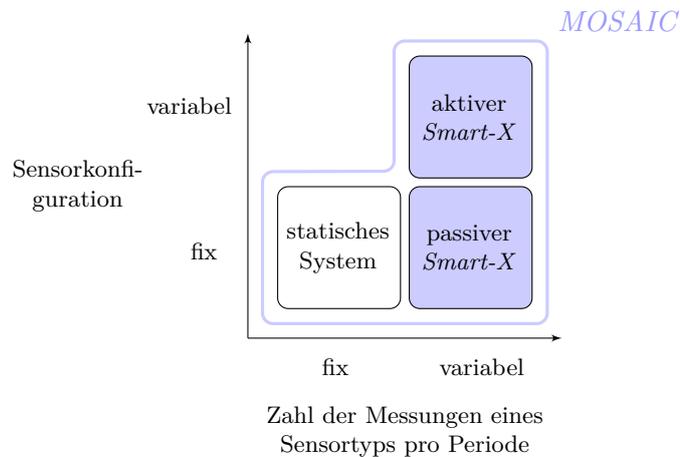


Abbildung 4.1: Erweiterung der üblichen statischen Systemkonfiguration anhand des *MOSAIC*-Konzepts auf eine variable Anzahl von Messungen und eingebundenen Sensoren

Innerhalb dieses Diagramms finden sich die unterschiedlichen Ausprägungen eines *Smart-X*-Knotens wieder. Einfache *Smart-X* als eingebettete Sensorknoten arbeiten analog zum Stand der Technik als statisches System. *Smart Fusion* und *Smart Aktoren* können, wie die blau markierten Flächen zeigen, auf zwei Arten implementiert werden. So genannte passive *Smart-X* berücksichtigen nur Daten von Sensoren, deren Typ zur Übersetzungszeit vom Entwickler fest vorgegeben wurde. Die Dynamik erstreckt sich in diesem Fall nur auf die veränderliche Zahl der Daten, die pro Messzyklus bereitstehen. Mithilfe der inhärenten und im Strukturmuster vorgesehenen Filter und Selektionsmethoden sollen diese Schwankungen aufgefangen und der Verarbeitungsfunktion die bestmögliche Kombination übergeben werden. Mit einer sinkenden Zahl von Sensoren sinkt in der Regel die Güte des Resultats der Berechnungen. Die andere Variante, der aktive *Smart-X*, überprüft seine intelligente Umgebung permanent auf möglicherweise zusätzlich verfügbare Datenquellen unabhängig vom Sensortyp, die einer abstrakten Vorgabe an Kriterien, zum Beispiel die Erfassung aller Daten, die Distanz- oder Positionsmessungen enthalten, folgt. Mit diesem flexiblen Informationsmanagement erweitert der aktive *Smart-X*-Knoten die Anpassungsfähigkeit nochmals. Die Zahl der verfügbaren Messungen variiert mit einer wechselnden Zahl von Sensortypen, sodass die frei gebliebene Position in der Matrix von Abbildung 4.1 keine Relevanz hat.

Neben dem variierenden Grad an Variabilität (statisch, passiv, aktiv) decken *Smart-X*-Knoten verschiedenen Funktionen in einem verteilten Sensor-Aktor-System ab:

Smart-X-Sensor Der *Smart-X*-Sensor definiert eine im Hinblick auf die verteilte Fehlerdetektion und die modulare Verarbeitungskette erweiterte Ausprägung des be-

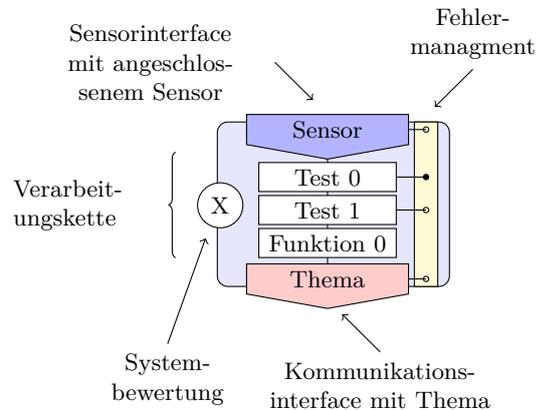
kannten Smart Sensors. Es wird dabei davon ausgegangen, dass als Ausführungs-umgebung ein performanzschwacher Controller dient, der die Möglichkeiten der Signalverarbeitung und der Fehlerdetektion beschränkt. Unter der Maßgabe einer weitgehenden Wiederverwendbarkeit ist ein *Smart-X*-Sensor entweder mit lediglich einem Sensor oder einem Array aus gleichartigen Sensoren verbunden.

Smart-X-Fusion Entscheidend für den *Smart-X*-Fusionsknoten ist, dass er über keine eigenen physischen Sensoren verfügt, sondern auf netzwerkkommunizierte Daten angewiesen ist, diese verarbeitet und versendet. Dabei sind unterschiedlichste Anwendungen denkbar, die von einfachen binären Filtern bis hin zu komplexen Fusionsalgorithmen reichen. Ebenso kann zwischen aktivem und passivem Akquiseverhalten, wie in Abbildung 4.1 aufgezeigt, unterschieden werden. Ein Sonderfall der *Smart-X*-Fusionsknoten sind virtuelle Sensoren. Unter Berücksichtigung der inhaltlichen Diskussion zu diesem Begriff in Kapitel 3 wird damit im Rahmen dieser Arbeit die Nachbildung der Messwertausgaben eines realen Sensors bezeichnet. Diese Ausgaben können auf dem Abspielen aufgezeichneter Werte, einem zufälligen Resultat oder aber auf in virtuellen Umgebungen gewonnenen Informationen beruhen. Virtuelle Sensoren sind wegen ihrer Flexibilität und Austauschbarkeit ein wichtiges Hilfsmittel für die Anwendungsentwicklung.

Smart-X-Aktor *Smart-X*-Aktoren schließen die Kette der *Smart-X*-Komponenten. Sie bilden über das Netzwerkinterface eingehende Informationen auf eine physische Aktorschnittstelle ab. Dabei umfasst der *Smart-X*-Aktor im Funktionsbereich Methoden zur Prüfung der eingehenden Aktorkommandos.

Trotz dieser Unterschiede sollen *Smart-X*-Knoten einen gemeinsamen Aufbau vorweisen. Dazu wird an dieser Stelle zunächst eine Basisstruktur entworfen, die im Kapitel 5 weiter differenziert wird. In Abbildung 4.1.1 wurde ein *Smart-X*-Sensor auf dem höchsten Abstraktionsgrad abgebildet. Entsprechend sind weder die innere Struktur noch das Zusammenspiel der Abstraktionen oder die individuelle Ausprägung, abgesehen von der Funktionsbezeichnung, abgebildet. Auf der Basis dieser Struktur lassen sich Netze von *Smart-X* übersichtlich darstellen und strukturieren. Im Hinblick auf die Anwendungsentwicklung wird dieses grafische Strukturierungsmittel aufgegriffen, das sich, wie noch in Kapitel 5 zu zeigen ist, auch in den grafischen Werkzeugen der Implementierungskette selbst wiederfindet.

Die Darstellung umfasst die Abstraktionen in den für diese Arbeit festgelegten Farben. Das gelb markierte Fehlermanagement zieht sich als querschneidende Funktionalität nicht nur übergreifend durch die gesamte Applikation, sondern ist auch Architekturelement über der gesamten Tiefe des Knotens. Die Fehlerinformationen der anderen Verarbeitungsfunktionen laufen, wie durch die kurzen Striche „-o“ angedeutet, hier zusammen, damit eine ganzheitliche Einschätzung zum aktuellen Fehlerzustand des Systems möglich wird. In engem Zusammenhang dazu steht die fehlerbezogene Kennzahl zur Bewertung

Abbildung 4.2: Grafische Darstellung eines *Smart-X*-Sensors mit seinen Kernelementen

des *Smart-X*, die im weiteren Verlauf als Risk Priority Number (RPN) bezeichnet wird, deren Herleitung in Abschnitt 4.2 beschrieben wird. Die Schnittstellen zur physischen Umgebung werden blau markiert und mit der Identifikation des Sensors, die als Referenz auf dessen entsprechendes Datenblatt dient, dargestellt. Die Verarbeitungskette integriert zwei Fehlerdetektionsmethoden sowie eine Verarbeitungsfunktion. Das Ergebnis des Verarbeitungsprozesses wird gemeinsam mit der gemeinsamen Validitätseinschätzung und zusätzlichen Attributen sowie dem Datenblatt unter einem Thema im Netzwerk verbreitet. Das Thema, das den Inhalt einer Nachricht charakterisiert, bezieht sich auf den realen Sensortyp und die nachgeordneten Verarbeitungsfunktionen. Identische *Smart-X* verbreiten ihre Informationen unter einem einheitlichen Thema. Die Netzwerkschnittstelle ist rot markiert. Die Pfeilform der Sensorschnittstelle und des Netzwerkinterfaces illustriert die Verarbeitungsrichtung der Nachrichten von oben nach unten. Mit dem Austausch der Sensorschnittstelle und gegen eine Kommunikationsschnittstelle lassen sich *Smart-X*-Fusionsknoten realisieren. Analog kann die unten liegende Kommunikationsschnittstelle gegen ein Aktorinterface ausgetauscht werden. Die interne Struktur muss also so gewählt werden, dass die Änderung der Schnittstellen ohne eine Anpassung der eigentlichen Anwendung vollzogen werden können. Dazu zählt insbesondere die Repräsentation der Datensätze, die unabhängig von der Art der Schnittstelle erfolgt.

Folglich bauen auch die *Smart-X*-Fusionsknoten und die *Smart-X*-Aktoren auf dem vorgestellten Architekturmuster auf. Abbildung 4.3 erweitert die einzelne Darstellung eines *Smart-X*-Sensors und zeigt ein beispielhaftes Szenario, das alle Varianten – *Smart-X*-Sensor, *Smart-X*-Fusionsknoten und *Smart-X*-Aktor – kombiniert. Die Ausgaben des Ultra Sonic (US)-Sensors bedienen eine Abstandsregelung, wobei diese durch den passiven Fusionsknoten in Bezug auf die Temperatur kalibriert werden. Die statische Konfiguration der Netzwerkschnittstelle ist durch die dezidierte Vorgabe der relevanten Informa-

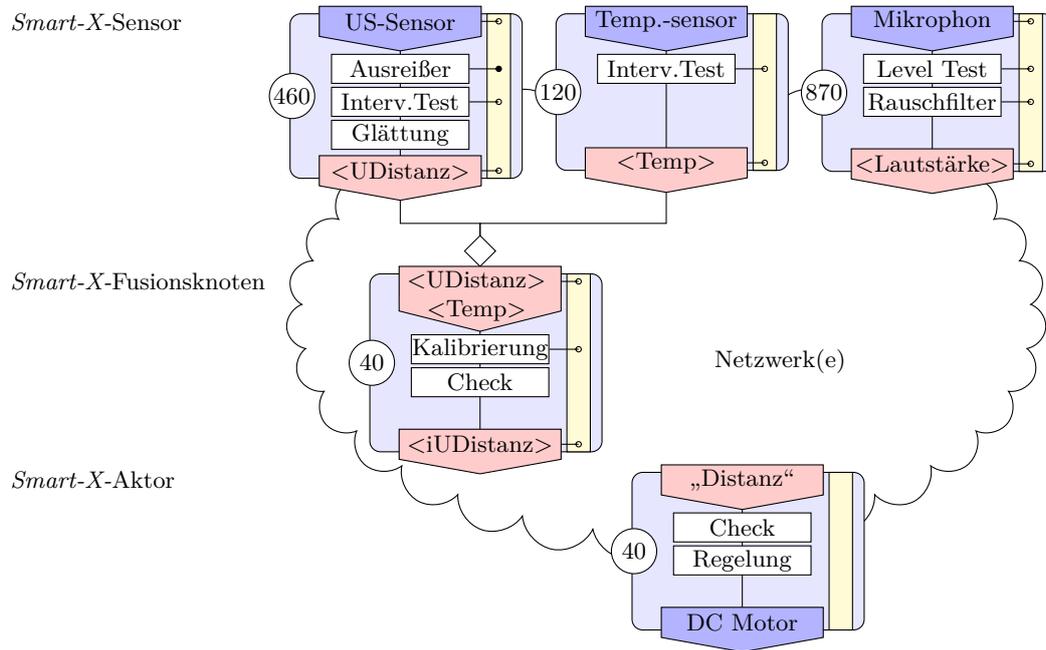


Abbildung 4.3: Beispielhafte Darstellung einer Applikation aus verschiedenen aktiven und passiven *Smart-X*-zur Abstandsregelung auf der Basis eines temperaturkalibrierten Distanzmesswertes

tionsthemen `<UDistanz>` und `<Temp>` vorgegeben. Diese Zuordnung wird grafisch durch eine aus der Unified Modeling Language (UML) entlehnte Verbindungsform abgebildet, die die Assoziation herstellt. Der Fusionsknoten nutzt das Wissen um die physikalischen Beziehungen der Datensätze, berechnet einen kalibrierten Distanzwert, der auf die Einhaltung des Messintervalls geprüft wird, und verbreitet diesen unter dem Thema `<iUDistanz>` im Netzwerk. Da der störende Einfluss der Temperatur auf die Ultraschall-distanzmessung nun abgeschaltet ist, steigt das Vertrauen in die Ausgaben des *Smart-X*. Daher sinkt der Wert der statischen Systemeinschätzung auf „90“ gegenüber der Kenngröße für den vorgelagerten Sensor mit „460“ deutlich. Dieser Wert signalisiert den nachgeordneten Knoten, dass der zu erwartende Datensatz und dessen Validitätsbewertung von hoher Güte sind. Die Herleitung der konkreten Werte für verschiedene Sensorkonstellationen wird in Abschnitt 4.2 diskutiert. Im Unterschied zu der statischen Kommunikationskonfiguration des Fusionsknotens alloziert der *Smart-X*-Aktor seine Messdaten aktiv. Dies wird anhand der Schnittstelle über die abweichende Notation deutlich, wenn also alle verfügbaren Themen auf eine Übereinstimmung mit einem vorgegebenen Muster „Distanz“ geprüft werden. Die Messungen des dritten Sensors, des Mikrofons, sind nach dieser Vorgabe für die Verarbeitung irrelevant. Entsprechend werden nur die Mess-

daten der beiden Themenkanäle $\langle iUDistanz \rangle$ und $\langle UDistanz \rangle$ berücksichtigt. Sofern also temperaturkalibrierte Werte zur Verfügung stehen, werden diese zur Regelung der Distanz herangezogen, im anderen Fall die einfachen Messdaten. Liegen die Daten innerhalb der spezifizierten Vorgaben werden diese an den Regler übergeben und die Aktoren angesteuert.

4.1.2 Programmiermodell

Insbesondere für den geforderten nahtlosen Datenaustausch, die flexible Kombination der Elemente und der Funktionalität sowie die Anpassungsfähigkeit gegenüber Veränderungen der Systemstruktur bedarf es einer generellen Programmierabstraktion, in die die Konzepte der *Smart-X* einzubetten sind. Die Untersuchung der verschiedenen Ansätze aus dem Kapitel zum Stand der Technik machte deutlich, dass keine diese Anforderungen im Hinblick auf Kommunikation und Funktionalität komplett abdeckende Lösung existiert. Für das *MOSAIC*-Framework wurde daher insbesondere ein Programmiermodell gesucht, auf das die oben genannte Struktur aufbauen kann.

Das von Veríssimo u. a. [Ver+02] vorgestellte und im Abschnitt der verwandten Arbeiten diskutierte Sentient Object bietet eine solche Abstraktion. *MOSAIC* greift dieses Konzept unabhängiger Entitäten, die über Kanäle und Ereignisse miteinander kommunizieren, auf und passt sie den Anforderungen der adaptiven Fusion an. Erst mit der Integration des Sentient Object löst sich die Sichtweise der *Smart-X*-Elemente, vom Blick auf die Hardware und die Funktionalität. Darin liegt die Grundlage der Interpretation als Softwarekomponente [Pio07]. Entsprechend kann ein Knoten, der physisch als Smart Sensor bezeichnet wird, durchaus rekursiv mehrere Sentient Objects und damit *Smart-X* umfassen, um seine Aufgabe zu erfüllen. Der Datenaustausch und die Nachrichten zwischen den *Smart-X* folgen dabei den Abstraktionen des Sentient Objects. So werden Datensätze von Sensoren gleichen Typs und gleicher nachgeordneter Verarbeitungsfunktionalität auf ein einheitliches Subject abgebildet. Diesem Ereignistyp ist ein die Nachrichtenpakete beschreibendes Nachrichtendatenblatt zugeordnet. Sind beispielsweise mehrere Knoten mit einem GP2D120 Sensor ausgestattet, vollführen aber unterschiedliche Berechnungen und Fehlerdetektionsmethoden, so werden den unterschiedlichen Ausgaben verschiedene Subjects und damit Ereignisdatenblätter zugeordnet. Anhand dieser Vorgabe wird die Trennung zwischen dem hardwareorientierten Systemdatenblatt, das während der Entwicklung benötigt wird, sowie dem funktions- und kommunikationsbezogenen Nachrichtendatenblatt deutlich. Alle Varianten des GP2D120-Sensors basieren auf der gleichen Hardware und damit dem gleichen Systemdatenblatt, generieren aber bedingt durch unterschiedliche Verarbeitungsmethoden ungleiche Nachrichten, die entsprechend den zugeordneten Datenblättern strukturiert sind. In Abweichung zu vorhergehenden Arbeiten werden jedem Ereignis neben dem eigentlichen Messwert drei zwingende Kerninformationen zugeordnet: ein Zeitstempel, eine Positionsangabe der Messaufnahme und eine Gültigkeitsschätzung. Weitere Attribute können optional hinzugefügt werden.

Das Sentient Object definiert die Hülle eines *Smart-X*, dessen Architektur die Funktionsebene für das Programmiermodell ausbildet. Entsprechend werden die Methoden der adaptiven Datenerfassung auf den Kommunikationsabstraktionen des Sentient Object aufsetzen.

Im Folgenden werden nun die konzeptionellen Voraussetzungen für diesen Architektur-entwurf im Hinblick auf die Fehlertoleranz, die Selbstbeschreibung und die dynamische Datenallokation und Verarbeitung erarbeitet. Dabei folgt die Analyse und Konzeptentwicklung dieser Reihenfolge, weil die Ergebnisse der Überlegungen zur Fehlertoleranz in die Selbstbeschreibung einfließen und diese wiederum eine Voraussetzung der Datenselktion und -verarbeitung ist.

4.2 Fehlerdetektion und Datenvalidierung

Der Vorgang der Abbildung einer physikalischen Größe auf einer digitalen Repräsentation, die vorverarbeitet in einem Netzwerk bereitgestellt wird, ist zumindest im analogen Fall immer mit einem Fehler, also einer Abweichung des Ausgabewertes von der realen Größe verbunden. Der Fehlerbegriff nach Laprie [Lap85] differenziert dies und bezeichnet die Abweichung zwischen Spezifikation und Ausgabe als Fehler. Dies hat zur Folge, dass zum Beispiel ein Messrauschen, dessen Parameter im Idealfall für einen Sensor bekannt sind, nicht als Fehler zu werten ist. Erst wenn das Signal mit unbekanntem Ausreißern überlagert wird, ist nach dieser Herangehensweise von einem Fehler zu sprechen.

Dieser Ansicht wird in der vorliegenden Arbeit nicht gefolgt, sondern vielmehr die Differenz zwischen Ausgabewert und realer Größe als Fehler betrachtet. Der Grund liegt dabei in der angestrebten Kapselung der individuellen Spezifikationen, Fehlermodelle und deren Parametern eines jeden Sensors. Eine zwischen verschiedenen Sensoren vergleichbare Einschätzung zur Güte einer Messung kann sich immer nur auf den möglichen Fehler im Hinblick auf die reale Größe beziehen. Damit werden die individuellen Sensorspezifikationen zum Rauschen, über Sättigungseffekte, Nichtlinearitäten usw. Bestandteil der Fehlerbetrachtung.

In der englischen Sprache wird sehr präzise zwischen Fault und Failure, also einem internen Fehlerzustand und einer im Kontext der Gesamtapplikation bemerkbaren Abweichung vom Spezifikationszustand unterschieden. Die Frage, auf welchen der Begriffe die im Folgenden benutzte Fehlerdefinition Bezug nimmt, lässt sich nur anhand der hierarchischen Ebene klären, in der eine Fehlerbetrachtung erfolgt. Ein Messfehler kann, sofern der einzelne Sensor als System aufgefasst wird, als Failure gelten. Wird die nachgeordnete Verarbeitung mit einbezogen, wenn also das Messergebnis zum Beispiel in eine Regelschleife einfließt, so stellt der Sensorfehler einen Fault dar. Dieser muss als kurzfristige Störung, beispielsweise wegen der Trägheit des Gesamtsystems, nicht zwingend zum Ausfall und damit zum Failure führen. Die Entscheidung, ob ein Fault oder Failure vorliegt und wie dieser zu erfassen und zu behandeln ist, obliegt somit dem Anwendungs-

entwickler. Er wird dabei aber durch die Fehlerdetektions- und Validierungsansätze aus *MOSAIC* unterstützt.

Im Weiteren sollen die Vielschichtigkeit der Fehlerquellen, die Fehlerauswirkungen auf einen Messwert und die möglichen Detektionsmethoden untersucht werden, um darauf aufbauend das Fehlerkonzept des *MOSAIC*-Frameworks herzuleiten.

4.2.1 Herkunft von Fehlern

Im Rahmen der ersten im großen Umfang durchgeführten Arbeiten im Bereich der Sensornetze wurden verschiedene Tierarten sowie die Umgebungsparameter auf der Insel „Great Duck Island“ über Monate überwacht. Szewczyk u. a. analysierten die Resultate der unterschiedlichen Messungen und klassifizierten bis zu 60 % der Daten als fehlerhaft [Sze+04]. Welche Ursachen stehen hinter diesem hohen Fehleranteil? Fehlerhafte Messungen eines intelligenten Sensors im Sinne der Definition 3 lassen sich auf externe und interne Fehlerquellen zurückführen. Abbildung 4.4 zeigt die elementaren Hardwarekomponenten eines Smart Sensors – Sensor, Verarbeitungseinheit und Netzwerkschnittstelle – und ordnet diesen vier Fehlertypen zu:

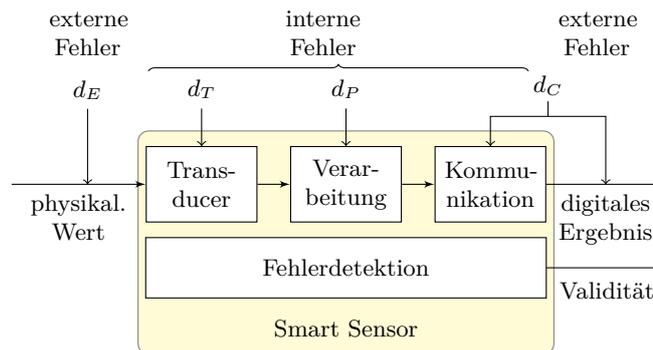


Abbildung 4.4: Kategorien von Sensorstörungen

1. *Externe Fehler* liegen in allen Parametern d_E begründet, die die Umgebungsbedingungen eines Sensors so beeinflussen, dass dessen Messung eine Abweichung aufweist. Beispiele für solche externen Störungen können zum einen direkt im physikalischen Prinzip der Messung begründet sein. Wenn also eine Distanzmessung mittels Ultraschalllaufzeitmessung unter der Annahme einer bekannten Schallgeschwindigkeit, die von der Temperatur abhängig ist, erfolgt, bewirkt eine Veränderung dieser Größe eine Fehlmessung. Zum anderen können bestimmte externe Parameter den Messprozess indirekt stören. Ein Beispiel dafür ist die Störanfälligkeit von Infrarotdistanzsensoren auf starkes Fremdlicht. Wie in [DZK10] beschrieben,

kann eine solche Störung die Verschiebung der gemessenen Werte um bis zu 25 % gegenüber der korrekten Distanz bewirken.

2. Der Sensor, als unmittelbare Schnittstelle zur Umgebung projiziert den physikalischen Wert auf ein elektrisches Signal. Dieser Wandlungsprozess kann durch elektrische oder mechanische *Sensorfehler* d_T negativ beeinflusst werden. Beispiele dafür sind das nichtlineare Abbildungsverhalten von Sensoren, aber auch allgemeine Störungen der Hardware, wie eine unterbrochene Stromversorgung, eine instabile Referenzspannung des Analog-Digital-Wandlers usw.
3. *Verarbeitungsfehler* d_P umfassen Soft- oder Hardwarefehler der Verarbeitungseinheit. Diese schließen neben den genannten generellen Faktoren, wie zum Beispiel der Energieversorgung, auch Programmierfehler oder aber Hardwarefehler ein.
4. Die korrekte Verbreitung der Messdaten kann sowohl durch externe als auch durch interne *Kommunikationsfehler* d_C gestört werden. Interne Fehler werden dabei durch eine gestörte Kommunikationshardware generiert, externe aber durch äußere Einflüsse wie Interferenzen, eine beschränkte Bandbreite etc. hervorgerufen. Beide Fehlertypen können inhaltlich gestörte Messungen, einen verzögerten Nachrichtenversand oder den vollständigen Verlust eines Datensatzes produzieren.

Die Identifikation von Fehlern wird dadurch erschwert, dass sich häufig mehrere Fehler unterschiedlicher Herkunft überlagern wie beispielsweise ein Sensorrauschen mit elektromagnetischen Feldern der Umgebung, sodass in der Summe eine gesteigerte Rauschamplitude steht. Analog lässt sich aus dem Ausbleiben einer Nachricht nicht auf den Grund dafür schließen, dieser kann in der Verarbeitungseinheit, der Kommunikationshardware oder den die Kommunikation beeinflussenden Umgebungsbedingungen liegen. An diesem Beispiel wird deutlich, dass die Fehlerdetektion den Ursprung eines Fehler unter Umständen nicht konkret eingrenzen kann. Die Detektion des Fehlerzustandes ist aber unabhängig davon möglich.

Vorgreifend auf diese Systematisierung im nächsten Abschnitt soll zunächst darauf hingewiesen werden, dass ein Fehler entsprechend Abbildung 4.1 ausgehend von seiner Auftretenscharakteristik (permanent, temporär) und dem Wissen um seine Parameter einer unterschiedlichen Verarbeitungskette zur Detektion, Parameterabschätzung und Filterung bedarf. An dieser Stelle sei noch einmal darauf hingewiesen, dass Fehler bekannter Parameter, die sonst zur Spezifikation gerechnet werden, im Sinne einer abstrakten und übergreifenden Abbildung der Störgrößen mit in die Fehlersystematik aufgenommen werden. Der Fehlertyp I aus Abbildung 4.1 stellt eben diesen typischen Sensorfehler dar, der zum Beispiel eine Messung mit einem bekannten Rauschen überlagert. Damit sind Methoden zur Detektion oder Parameterbestimmung nicht erforderlich, der Anwendungsentwickler kann unmittelbar entsprechende Filter in die Anwendung integrieren. Sind einzelne Parameter des Fehlers nicht oder nur annähernd bekannt, ist zunächst

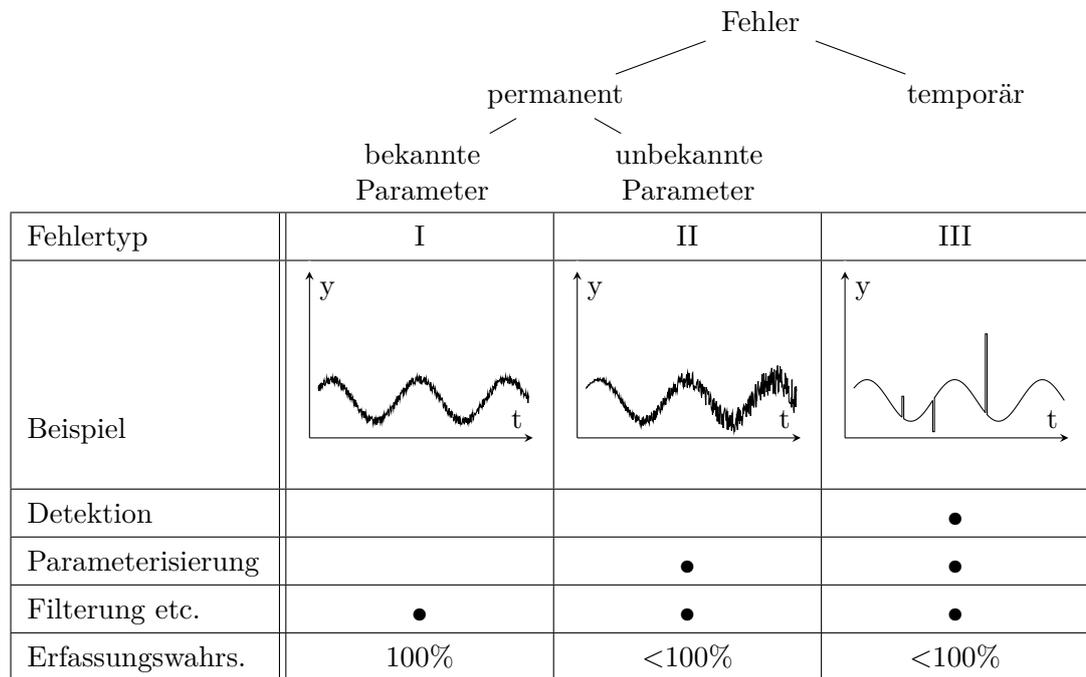
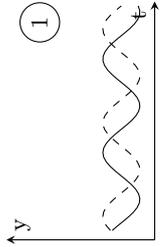
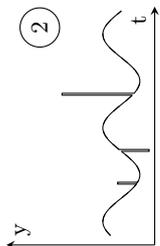
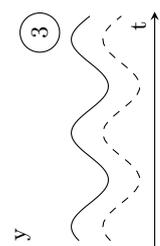
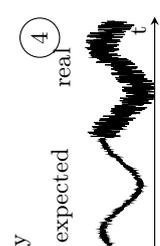
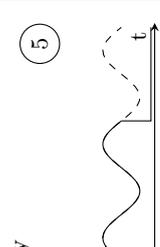
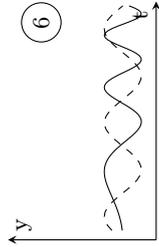
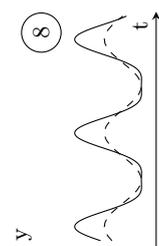
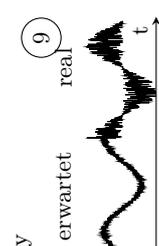
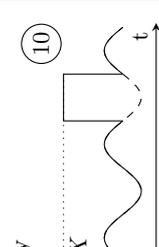
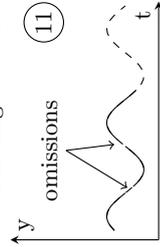
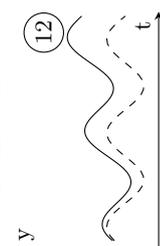
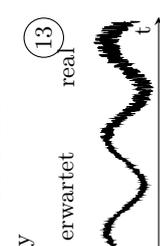
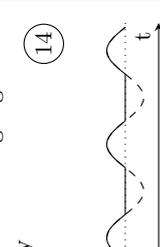


Abbildung 4.5: Fehlertypen im Hinblick auf die notwendigen Signalverarbeitungsschritte

eine Abschätzung der Größenordnung des Fehlers erforderlich. Mit dieser zusätzlichen Funktionalität ist aber nicht das Niveau der Güte eines Fehlers vom Typ I erreichbar, da nicht von einer perfekten Abschätzung ausgegangen werden und folglich die nachgeordnete Filterung weniger erfolgreich wirken kann. Im Beispiel der Abbildung 4.1 wäre dies die Frage nach der Varianz des verrauschten Signals, die sich wie in der dritten Spalte sichtbar verändert und als (unsichere) Steuergröße eines stochastischen Filters genutzt werden wird. Für nichtdeterministisch auftretende Fehler wird diese Unsicherheit noch größer, da zunächst der Fehler selbst detektiert werden muss, bevor eine Abschätzung hinsichtlich seiner Größe erfolgen kann. Die Ausreißer im Signalverlauf der letzten Spalte aus Abbildung 4.1 lassen sich zwar wegen der fehlenden Stetigkeit des Signalverlaufes gut erfassen, für komplexere Fehlercharakteristiken schließt dieser Schritt aber eine weitere Unwägbarkeit und damit eine sinkende Güte eines Messwertes ein.

Für die später einzuführende Kategorisierung eines Sensors ist die sich aus der Einteilung ergebende Erfassungswahrscheinlichkeit höchst relevant. Folglich müssen die im folgenden Kapitel einzuführenden Fehlermodelle auch auf die Zugehörigkeit zu einem Fehlertyp hin untersucht werden.

Tabelle 4.1: Fehlerkategorisierung für Sensoren (der korrekte Wert wird durch die gestrichelte Linie repräsentiert, der ausgegebene durch den nicht unterbrochenen Graphen)

Verzögerung	Offset			Konstant
	sporadisch	permanent	stochastisch	
konstant $e = \Delta t \cdot \max(dy/dt)$  <p>1</p>	Ausreißer $e = A$  <p>2</p>	konstant $e = const = 0$  <p>3</p>	konstant expected real  <p>4</p>	Stuck-At-Zero $e = y$  <p>5</p>
variabel $\max(e) = \max(y) - \min(y)$  <p>6</p>	Spike $\max(e) = A$  <p>7</p>	wertkorreliert $e = f(y) - y$  <p>8</p>	wertkorreliert erwartet real  <p>9</p>	Stuck-At-X $e = y - X$  <p>10</p>
Verbindungsabbruch omissions  <p>11</p>		zeitkorreliert $e = f(t) - y$  <p>12</p>	zeitkorreliert erwartet real  <p>13</p>	Sättigung $\forall y > X : e = y - X$  <p>14</p>

4.2.2 Fehlerabstraktionen

In diesem Abschnitt werden die typischen Sensorfehler in ihrer Auswirkung auf den Signalverlauf abstrahiert und mathematischen Modellen zugeordnet.

Verzögerungen

Eine Sensormessung ist nur dann in einem Fusionsalgorithmus verwendbar, wenn zu dem eigentlichen Messwert auch der Zeitpunkt der Aufnahme vermerkt wird. Folglich können Messfehler in zwei Richtungen entstehen: zum einen durch einen Messwert, der die reale Größe nicht korrekt abbildet oder zum anderen durch einen falschen Zeitstempel, der die korrekte Messung auf dem Zeitstrahl verschiebt. Dies entspricht der bereits angesprochenen Interpretation einer Messung als Time-Value Entity. Die erste Spalte der Tabelle 4.1 beschreibt eben diese zeitliche Zuordnung, während die übrigen Kategorien wertmäßige Fehler charakterisieren.

Alle Komponenten eines Smart Sensors können eine Verzögerung hervorrufen. Dies kann durch den eigentlichen Sensor geschehen, durch eine zeitaufwendige Signalverarbeitung oder durch die Kommunikation bedingt sein. Mit der Zuordnung eines Zeitstempels zu jeder Messung lässt sich diese Verzögerung aber quantifizieren. Der Fehler besteht also in der falschen Zuordnung einer Messung zu einem Zeitstempel. Unter der Annahme, dass in der verteilten Anwendung ein Synchronisationsprotokoll für vergleichbare Zeitzuordnungen sorgt, lässt sich der maximale Fehler anhand der maximalen Veränderung der Messgröße über dem maximalen Synchronisationsfehler annehmen. Ohne ein solches Protokoll wird der Zuordnungsfehler wegen der Drift der Uhren mit wachsender Laufzeit zunehmen.

Tabelle 4.1 unterscheidet zwischen drei Typen von Verzögerungen, welche mit den Markern ①, ⑥ und ⑪ dargestellt sind. Die erste Kategorie weist einen konstanten Fehler auf. Dieser kann daraus resultieren, dass die Verzögerung Δt des eigentlichen Sensors nicht im Zeitstempel enthalten ist. Unter Beachtung der Dynamik der beobachteten Umgebungsgröße ergibt sich damit ein maximaler Fehler e , der in seiner Größe mit $\max(e) = \Delta t \cdot (dy/dt)$ abgeschätzt werden kann, wobei dy/dt die Geschwindigkeit der Änderung des Wertes y meint. Realistischer sind dynamisch veränderliche Verzögerungen, die sich, wie bereits angeführt, aus der ungenauen Synchronisation ergeben können ⑥. Das letzte Fehlermodell definiert kommunikationsbedingte Verzögerungen ⑪. Diese können temporärer oder permanenter Natur sein. Permanente Verbindungsverluste werden als Verzögerungen mit einem unendlichen Wert für Δt modelliert. Wird von periodisch übermittelten Nachrichten ausgegangen, so lassen sich diese Fehler sicher detektieren.

Sporadischer Offset

Während die Verzögerung den Fehler durch eine falsche Zeitzuordnung entstehen lässt, begründen sich die folgenden Fehlerkategorien auf einer Verschiebung im Wertebereich. Sporadische Offsets können in den physikalischen Eigenschaften des Messprozesses begründet sein und durch externe und interne Störungen ausgelöst werden. Ein typisches Beispiel für sporadische Offsets sind Ausreißer ②, die als einzelne losgelöste Samples eine signifikante Abweichung vom realen Wert zeigen [Ni+09]. Beispielsweise sind Ultraschallmessungen häufig mit Ausreißern überlagert, die auf der Vielzahl der Reflexionen der Umgebung bei großem Öffnungswinkel basieren [ZK09]. Der maximale Fehler wird durch die Amplitude A des Ausreißers bestimmt, wobei deren Wert in der Regel stochastischer Natur ist.

Die zweite Kategorie in dieser Spalte – Spikes – ⑦ unterscheidet sich von der ersten durch den kontinuierlichen Anstieg der Abweichung bis hin zum Maximum A . Spikes können als erweiterte Variante des Ausreißers betrachtet werden, wobei diese Störung über mehr als ein Sample auf der Grundlage von Induktivitäten und Kapazitäten gebildet wird. Ein Spike zeigt hinsichtlich der Abweichung vom realen Wert einen geringeren Anstieg und ist somit schwieriger zu detektieren.

Spike und Outlier gehören wegen des sporadischen Auftretens zum Fehlertyp III.

Permanenter Offset

Die zweite Spalte der Offset-Fehler umfasst die permanenten Störungen einer Messung und unterscheidet zwischen wertmäßig konstanten und variablen Effekten.

Konstante Fehler ③ führen zu einer kontinuierlichen Überlagerung des Messwertes mit einem statischen Offset, der über mehrere Samples anhält. Ein solcher Fehler kann sowohl auf bekannten Parametern beruhen als auch von unbekannter Größe sein, womit die Fehlertypen I bis III umfasst werden. Ein Beispiel für einen Fehler mit bekannten Parametern ist die Temperaturabhängigkeit eines Ultraschallsensors. Mit der Kalibrierung des Sensors auf die Umgebungsbedingungen kann der Einfluss des Fehlers beschränkt werden. Unbekannte konstante Offset-Fehler entstehen zum Beispiel durch externe Störeinflüsse wie Fremdlicht bei einem Infrarot Sensor [DZK10]. Der resultierende Fehler ist definiert durch den Offset O .

Der variierende Offset kann in zeit- oder wertkorrelierte Effekte eingeteilt werden. Für Fehler der erstgenannten Gruppe ⑧ wird eine Funktion $f(y)$ modelliert, die eine lineare oder nichtlineare Abbildung des Ausgabewertes auf der realen Größe darstellt. Der Fehler ergibt sich dann zu $e = f(y) - y$. Fehler, die sich in Abhängigkeit von der Zeit als Offset auswirken ⑫, entstehen zum Beispiel durch Alterungsprozesse des Sensors oder über kürzere Horizonte betrachtet durch die Fehlerrate der Uhr. Als Fehlerfunktion ergibt sich dann $e = f(t) - y$. Die Größe des Wertefehlers, der durch die Ungenauigkeit

der Uhr hervorgerufen wird, kann, wie bereits erwähnt, durch geeignete Synchronisationsprotokolle begrenzt werden [SY04].

Permanente Offsetfehler können als alle drei Fehlertypen des Schemas nach Abbildung 4.5 auftreten.

Stochastischer Offset

Sensormessungen sind häufig mit einem Rauschen, das von unterschiedlicher Herkunft sein kann, überlagert. Ein analoger Wert kann durch thermisches, Interferenz- oder Quantisierungsrauschen verzerrt werden. Die Amplitude des Messrauschens wird üblicherweise mit einer oder im Fall abweichender Verteilungen als Kombination aus mehreren Standardverteilungsfunktionen modelliert. Es existiert eine große Zahl von Methoden zur Bestimmung der entsprechenden Verteilungsparameter, wobei häufig von einer Normalverteilung ausgegangen wird. Diese erlaubt dann die einfachere Verarbeitung und zum Beispiel die Nutzung von etablierten Techniken, wie den Kalman-Filter.

Das Rauschen kann dabei als Fehlertyp I mit bekannten Parametern auftreten, aber auch mit veränderlichen Parametern als Fehlertyp II die Messwerte überlagern. Die mehrfarbige Darstellung der Fehlerkategorien ④, ⑨ und ⑬ illustriert dies. Die Varianz der Parameter kann dabei vom Messwert ⑨ oder der Zeit ⑬ abhängen.

Konstant

Die Gruppe der als Konstant bezeichneten Fehlermodelle besteht aus den Stuck-at-Fehlern und den ein Sättigungsverhalten zeigenden Signalverläufen.

„Stuck-At“ Fehler sind definiert als eine Folge von Sensorausgaben auf konstantem Niveau. Sobald der Messwert auf einen GND-Pegel sinkt, wird von Stuck-At-Zero gesprochen ⑤. Natürlich sind aber auch alle im Ausgabespektrum des Sensors liegenden Werte möglich ⑩. Ein Stuck-At-Fehler kann permanent wirken oder aber als temporärer Effekt auftreten. Die Fehlersystematik von Sharma, Golubchik und Govindan [SGG07] fasst die Gesamtheit der Stuck-At-Fehler unter dem Begriff der CONSTANT-Fehler zusammen.

Die Sättigung beschreibt das Abschneiden des Messwertes an einer bestimmten Schranke, wie in Abbildung ⑬ dargestellt. Dieser Fehlertyp tritt zum Beispiel auf, wenn die Referenzspannung eines Analog-Digital-Wandlers überschritten wird. Der maximale Fehler ergibt sich dabei aus $\forall y > X : e = y - X$.

4.2.3 Fehlerdetektion

Fehlerdetektionsmethoden erlauben die Validierung der Messwerte und bilden die Grundlage des MOSAIC-Fehlertoleranzkonzeptes. Ausgehend vom Überblick in Abschnitt 3.4

muss für jedes Fehlermodell ein geeigneter Ansatz ausgewählt werden, der auch die Kombination mehrerer Detektionsmethoden umfassen kann. Zusätzlich sind häufig geeignete Parameter zu bestimmen. Wegen des starken Anwendungsbezugs soll in Tabelle 4.2 lediglich eine grundsätzliche Zuordnung zwischen den Fehlerkategorien aus Tabelle 4.1 und möglichen Ansätzen zu deren Detektion erfolgen. Die letzte Spalte der Aufstellung zeigt dabei die Ausführungsumgebung der Detektionsmethoden an. Gegen eine lokale Fehlerprüfung auf einem Sensorknoten können zwei Dinge sprechen: zum einen die möglicherweise zu geringe Rechenleistung der Hardware und zum anderen das Fehlen der nötigen (redundanten) Messwerte.

Ausgehend von dieser Tabelle sind für individuelle Sensormodelle und zu erwartende Signalparameter geeignete Methoden auszuwählen.

4.2.4 Sensor- und Messdatenvalidierung

Im angestrebten dynamischen Szenario soll jedem Ereignis eine Einschätzung seiner Gültigkeit und damit die Wahrscheinlichkeit der Überlagerung mit einem Fehler zugeordnet werden. Dieser Validitätswert basiert dabei auf folgenden Merkmalen der beteiligten Sensoren und der hinzugezogenen Fehlerdetektionsmethoden:

Spezifische Fehlermodelle Die Häufigkeit und das Niveau möglicher fehlerhafter Messungen definieren im entscheidenden Maße die Gültigkeit der Ausgabewerte.

Wahrscheinlichkeit der Entdeckung Ein Fehler mit einer geringen Entdeckungswahrscheinlichkeit wirkt sich analog gesehen negativer auf die Validität aus als ein möglicherweise zwar mit hoher Häufigkeit auftretender, aber sicher zu detektierender Fehler. Wurde für ein bestimmtes Fehlermodell keine Fehlerdetektionsmethode vorgesehen, so ist von einer Detektionswahrscheinlichkeit von 0 auszugehen.

Ergebnis der Fehlerprüfung Die Fehlerdetektionsmethoden, die vom Entwickler den Fehlermodellen zugeordnet wurden, generieren wie zum Beispiel bei einer Bereichsprüfung eine binäre Ausgabe (innerhalb/außerhalb) oder im Fall einer statistischen Signaluntersuchung eine Wahrscheinlichkeit, dass ein Fehler vorliegt. Werden mehrere Fehlerdetektionsmethoden angewendet, ergibt sich somit ein Vektor aus Gültigkeitseinschätzungen.

Die beiden erstgenannten Punkte können für Sensoren als statisch angesehen werden, da Fehlermodelle und die Wahrscheinlichkeit der Entdeckung als konstant angenommen werden. Demgegenüber wird das Ergebnis der eigentlichen Fehlerdetektionsmethode(n) mit jeder Messung variiert. Dabei wird die Aussagekraft einer solchen Gültigkeitseinschätzung durch die beiden statischen Elemente bestimmt. Wenn für einen Sensor, der im Betrieb mehrere Fehler zeigt, lediglich ein einfacher Ausreißertest implementiert wurde, kann der Gültigkeitseinschätzung auf der Basis des einzelnen Tests nur

Tabelle 4.2: Fehlerdetektionsmethoden für die Fehlerkategorien der Tabelle 4.1 im Hinblick auf die Ausführungsumgebung (lokal = unmittelbar auf dem Sensor-knoten, global = auf nachgeordneten leistungsstarken Fusionsknoten)

ID	Fehler-kategorie	Detektionsansatz	Lokalisation
① ⑥ ⑪	Verzögerung	Eine falsche Zeitzuordnung ist durch redundante Messungen mit korrektem Zeitstempel und mathematischen Modellen als Referenz detektierbar.	global
② ⑦	Sporadischer Offset	Ausreißer und Spikes, die Werte außerhalb des Messbereiches produzieren, sind durch eine einfache Schwellwertprüfung erfassbar. Innerhalb des Messbereiches ist eine (stochastisch untermauerte) Gradientenanalyse, Bewertungen des Frequenzganges usw. denkbar. Dabei sind Ausreißer wegen der deutlichen Flanken zumeist einfacher detektierbar als Spikes. Zudem können die Fehler dieser Kategorien mithilfe von Systemmodellen erfasst werden.	lokal/global
③ ⑧ ⑫	Permanenter Offset	Ohne eine redundante Sensormessung ist zum Beispiel der Offset der Ausgabe eines mit einer falschen Referenzspannung arbeitenden ADC nicht detektierbar. Zudem helfen mathematische Modelle nur für den Fall eines sprunghaften Auftretens.	global
④ ⑨ ⑬	Stochastischer Offset	Die Detektion von unbekanntem Rausch einflüssen erfordert eine erhebliche Anstrengung. Dabei werden vorrangig statistische Methoden eingesetzt, die an ein Systemmodell gekoppelt sind.	global
⑤ ⑩ ⑭	Konstant	Für die Erkennung dieses Fehlertyps können sowohl mathematische Modelle, Signalanalysen oder redundante Messungen genutzt werden. Allerdings ist erst im Zusammenspiel eine Detektion mit hoher Wahrscheinlichkeit möglich, die unmittelbar mit dem Fehlereintritt greift.	lokal/global

in geringem Maße vertraut werden. Wenn hingegen umfangreiche Fehlerdetektionstests für die Gültigkeitsannahme durchgeführt wurden, so steigt deren Vertrauenswürdigkeit. Entsprechend wurde für das *MOSAIC*-Framework ein kombiniertes Konzept zur Kapselung der spezifischen Fehlermodelle umgesetzt, das eine objektive Validitätseinschätzung einer Messung erlaubt. Dieser Wert wird als dynamische Datensatzvalidität bezeichnet, während die von den Fehlermodellen und Detektionsmethoden abhängige Einschätzung als statische Systemvalidität bezeichnet wird. Die statische Bewertung dient der Beschreibung der Gültigkeit der dynamischen Validitätsaussage.

Statische Systemvalidität

Folglich ist eine Klassifikation zu entwickeln, die eine abstrakte, statische Güteeinschätzung eines Messsystems anhand der Fehlerauftretswahrscheinlichkeit, Fehleramplitude und Fehlerdetektionsansätze liefert. Die statische Systemvalidität rückt damit in die Nähe der von Powell [Pow92] vorgestellten und durch Verissimo und Kopetz [VK93] aufgegriffenen Begriff des Coverage. Unter dieser Bezeichnung wird für alle Fehler vor dem Hintergrund der Umgebungsbedingungen (Netzwerke, Hardware, Infrastruktur usw.) und der Funktionalität (Programme, Algorithmen, Protokolle usw.) eine Auftretenswahrscheinlichkeit für den Fall bestimmt, dass der Fehler nicht toleriert wird. In der Summe ergibt sich dann die Wahrscheinlichkeit, dass das System ausfällt. Der Fokus dieser Arbeit liegt allerdings auf Rechnerarchitekturen und Kommunikationsnetzen, der spezielle Bezug zu sensorspezifischen Fehlern ist nicht gegeben. Entsprechend lassen sich deren Fehlermodelle schlecht auf eine einzige Wahrscheinlichkeit abbilden.

Für den erweiterten Ansatz im Sinne dieser Arbeit wird zudem auf die Failure Modes and Effects Analysis (FMEA) [TM03] zurückgegriffen. Diese Methode dient der Identifikation, Validierung und Klassifikation von Fehlern im Rahmen der industriellen Produktentwicklung. Zu diesem Zweck kombiniert die Methode Techniken zur Dekomposition der zu untersuchenden Systeme, die Analyse potenzieller Fehler, deren Bewertung und die Ableitung notwendiger Schritte zu Reduzierung derselben. Die FMEA wurde erstmalig in der Automobilindustrie eingesetzt und zielte dort auf die Systematisierung der Fehlerbetrachtung ab, die eine Vergleichbarkeit unterschiedlicher Fehlerzustände im Hinblick auf die Auswirkung erlaubt. Dabei wird für jedes Fehlermodell die Fehlerauftretenswahrscheinlichkeit (*Occurrence* (O)), die Fehlerbedeutung (*Sensitivity* (S)) und die Detektionswahrscheinlichkeit (*Detection* (D)) bestimmt und in einer RPN kombiniert. Jeder der drei Werte wird mit einem Index von 1 bis 10 belegt. Die RPN wird durch eine Multiplikation dieser drei Werte bestimmt. Ein permanent wirkender Fehler $O = 10$ mit einer hohen Auswirkung auf das System $S = 10$ und einer geringen Detektionswahrscheinlichkeit $D = 10$ führt somit zu einem RPN von 1000, wobei dieser Wert ein außerordentlich unzuverlässiges System anzeigt. Umgekehrt markiert ein niemals erreichbarer RPN von 1 eine fehlerfrei arbeitende Komponente. Als Maßstab für die Kategorisierung der Werte für O, D und S werden standardisierte Klassifikati-

Tabelle 4.3: Adaptierter VDA 4.2 Standard zur Klassifikation von Sensorfehlern

Index	Fehleramplitude pro Messbereich (F)	Auftreten ppm (O)	Detektionswahrscheinlichkeit (D)
10	≥ 1	100.000	>50 %
9		50.000	
8		20.000	75 %
7	≥ 0.1	10.000	
6		5.000	90 %
5		2.000	
4	≥ 0.01	1.000	
3		100	99 %
2		50	
1	≥ 0.001	1	99.99 %

onsvorschriften zum Beispiel des Verbandes der Automobilindustrie (VDA) verwendet. Die Version 4.2. des VDA [Ver96] definiert, basierend auf IEC 61508 „Fault detection probability“, eine verbindliche FMEA Indexzuordnung.

Um den RPN der FMEA als Indikator der statischen Systemvalidierung auf Sensorsystemen anwenden zu können, erfolgte die in Tabelle 4.3 gezeigte Anpassung dieses Schemas im Sinne der Sensordatenvalidierung. Die textuelle Beschreibung der Fehlerbedeutung S aus VDA 4.2 wird für den Fokus der Sensorfehler durch die Größe des Messfehlers F ersetzt. Um eine Normierung zu erzielen, wird dafür das Verhältnis aus Überwachungsbereich und maximalen Messfehlern gebildet. Ein Verhältnis von größer gleich eins bedeutet also, dass ein Sensor vorliegt, dessen Fehler der Größe des gesamten Messbereiches entsprechen kann. Diese relative Angabe entkoppelt den Fehler von der maximalen Amplitude und macht ihn für verschiedene Sensorsysteme vergleichbar.

Hinsichtlich der Problematik der Abschätzung der Detektionswahrscheinlichkeit sei auf Abbildung 4.6 verwiesen. Beide Diagramme zeigen einen sinusförmigen Signalverlauf, wobei die Messung mit Ausreißern überlagert ist. Der graue Bereich markiert das Fenster zulässiger Messwerte. Die Ausreißer, die den Messbereich verlassen, können mit geringerem Aufwand zuverlässig erkannt werden. Für den in Abbildung 4.6(b) dargestellten Signalverlauf trifft dies nicht zu. Hier bedarf es eines Modells zum Rauschverhalten und zur Dynamik des beobachteten Systems, um die Ausreißer, die innerhalb des gültigen Messbereiches liegen, zu erfassen. Für eine sichere Detektion müssen dem Fehlermodell

Ausreißer offenbar mehrere Detektionsmethoden zugeordnet werden. Entsprechend den Regeln der Wahrscheinlichkeitsrechnung werden die Detektionswahrscheinlichkeiten D_0 und D_1 für die beiden einzelnen Verfahren dann mit

$$D(D_1 \cup D_2 \cup \dots \cup D_n) = 1 - (1 - D_n) \dots (1 - D_1) \cdot (1 - D_2) \quad (4.1)$$

zusammengefasst.

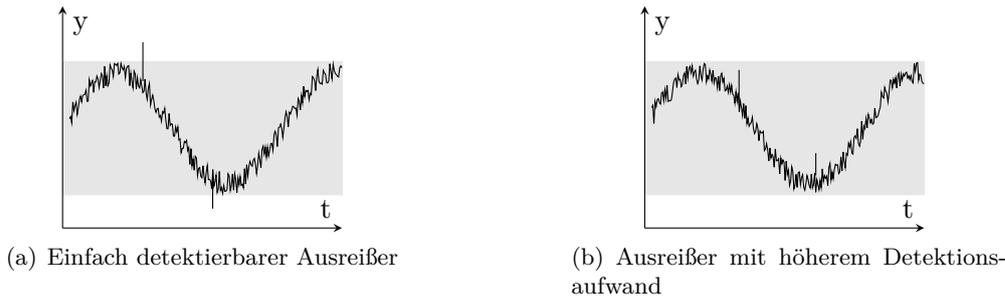


Abbildung 4.6: Beispiele für Detektierbarkeit von Ausreißern im Signalverlauf

Im Ergebnis einer *MOSAIC*-spezifischen FMEA-Untersuchung ergibt sich eine Darstellung, wie in Tabelle 4.4 beispielhaft ausgeführt für den GP2D120. In der ersten und zweiten Spalte sind die Fehlermodelle und die zugehörigen Indizes aus Abbildung 4.5 aufgeführt. Die dritte Spalte fasst die Indizes der Fehleramplituden (F) zusammen. Da diese für Stück-At-Fehler den gesamten Messbereich umfassen kann, ist für diese Fehlerkategorie jeweils eine 10 eingetragen. Die Fehlerhäufigkeit (O) wurde entsprechend der dritten Spalte der Tabelle 4.3 kodiert. Die darauf folgende Spalte gibt den resultierenden RPN wieder, wenn davon ausgegangen wird, dass keine Detektionsmethode integriert wird. In diesem Fall liegt die Detektionswahrscheinlichkeit bei null. Diesem Wert ist ein FMEA-Index von 10 zugeordnet. Wenn damit im Hinblick auf die Gesamteinschätzung des Sensorsystems der Maximalwert der Spalte im letzten Drittel der zulässigen RPN-Werte liegt, deutet dies auf einen sehr unzuverlässigen Sensor hin.

Erst mit der Einbeziehung von Detektionsmethoden wird der Einfluss von Fehlern reduziert. Die Tabelle 4.4 ordnet den vier Fehlertypen jeweils mehrere Methoden zu. Dabei wird durch die Farbgebung unterschieden, wo diese Methode ausgeführt werden kann. Weiße Zeilen repräsentieren sensorlokal umsetzbare Algorithmen, während hellgraue Methoden wegen hoher Rechenanforderungen wahrscheinlich nicht auf einem Sensorknoten umsetzbar sind. Die dunkelgrauen Zeilen kennzeichnen Detektionskonzepte, die auf mehr als einem Datensatz basieren, also redundante Informationen nutzen. Mit der Integration und Berücksichtigung der Detektionsmethoden und der zu erwartenden Detektionswahrscheinlichkeit sinkt der RPN-Wert stark ab. Im Beispiel wird dies durch den Test auf Ausreißer deutlich, wo der RPN von 600 mit einem Test auf dem Sensorknoten auf 240

Tabelle 4.4: Beispiele für die Fehlerklassifikation anhand der Fehlerkategorien des GP2D120-Distanzsensors mit den zugehörigen Detektionsmethoden

Fehler				Detektion			
Type	Index	Fehleramplitude (F)	Auftretens- wahrscheinlichkeit (O)	RPN ohne Detektion ($F \cdot O \cdot 10$)	Methode	Detektions- wahrscheinlichkeit (D)	RPN mit Detektion ($F \cdot O \cdot D$)
Stuck-At	⑤	10	4	400	Gradienten-Test	6	240
					Varianztest	3	120
					Vergleich	1	40
Ausreißer	②	10	6	600	Intervall-Check	3	180
					Gradienten-Test	6	240
					Σ	1	60
					Vergleich	1	60
Offset	③	8	9	720	Signalanalyse	7	504
					Vergleich	1	72
Spikes	⑦	8	8	640	Gradienten-Test	6	384
					Vergleich	1	56

reduziert wird. Für Ausreißer sind im Beispiel zwei sensorlokale Detektionsverfahren vorgesehen, die nebeneinander arbeitend die Wahrscheinlichkeit einer korrekten Detektion steigern. Die Detektionswahrscheinlichkeiten $D_0 = 0,9$ und $D_1 = 0,99$ werden mit der Gleichung 4.1 kombiniert. Daraus ergibt sich eine Gesamtdetektionswahrscheinlichkeit Σ von 99,99, die gleichbedeutend mit der Indexzuordnung von eins ist. Ausreißer können also im resultierenden *Smart-X*-Sensor sehr gut detektiert werden.

Um eine Aussage über ein System zu treffen, werden nun für jeden Fehlertyp die RPN-Werte zusammengetragen. Für einen einzelnen, eingebetteten Sensorknoten, auf dem die in Tabelle 4.4 mit weißen Zeilen markierten Fehlerdetektionsmethoden zum Einsatz kommen, ergibt sich ein Maximalwert für den RPN von 720. Zwar werden, wie gezeigt, Ausreißer ($RPN = 60$) und weniger gut auch Stuck-At-Fehler ($RPN = 240$) detektiert, Offseteinflüsse aber bleiben wegen fehlender Methoden komplett unerkannt ($RPN =$

720). Dieser maximale RPN-Wert wird im Rahmen dieser Arbeit als Kriterium für die statische Vertrauenswürdigkeit eines Systems definiert. Offenkundig liegt im Beispiel ein relativ unsicherer Sensor vor.

Zur Verbesserung der statischen Validität wird das Szenario, wie in Abbildung 4.7 gezeigt, um zwei zusätzliche, gleichartige Sensoren erweitert, die einen gemeinsamen Überwachungsraum abdecken. Damit besteht nun die Möglichkeit, über Vergleiche die Messdaten gegeneinander zu prüfen und gegebenenfalls zu verwerfen. Zudem bringt der eingesetzte Knoten auch genügend Rechenkapazitäten mit, um alle aufgeführten Algorithmen ausführen zu können. Damit sinkt der maximale RPN auf einen Wert von 72. Unterschiedliche Ansätze zur FMEA sehen bei einem RPN-Wert von 100 im Bereich der Medizintechnik und Automobilindustrie die Grenze für zulässige Komponenten. Der Anwender einer *MOSAIC*-Implementierung hat diesen Wert szenarienspezifisch festzulegen.

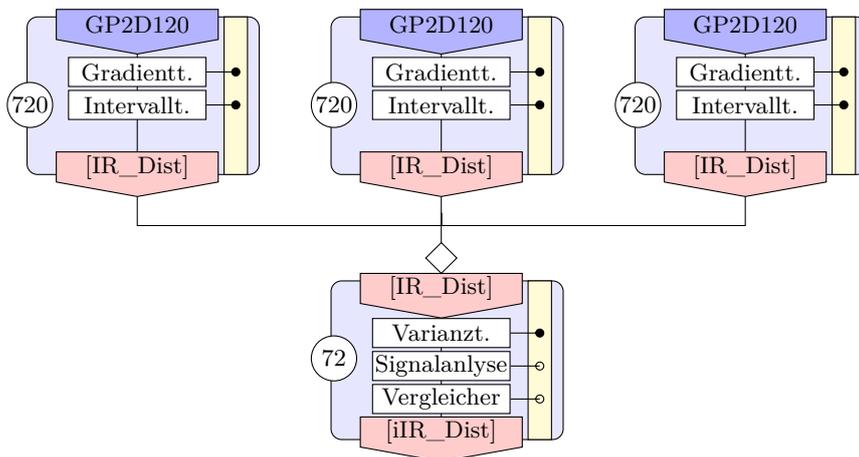


Abbildung 4.7: Beispielhafte Umsetzung einer fehler tolerantanten Kollisionsvermeidung

Im genannten Beispiel erlaubt die homogene Struktur eine einfache Ableitung der Systemvalidität des Fusionsknotens aus den RPN-Werten der Sensoren. Für komplexere Szenarien müssen jeweils neue Analysetabellen wie Tabelle 4.4 für den Fusionsknoten aufgestellt werden.

Für den dazu nötigen ersten Schritt, die Aufstellung und Parametrisierung der Fehlermodelle, kann ausgehend von den Fehleramplituden der Sensoren eine analytische Abschätzung des Fehlerhorizontes des fusionierten Ergebnisses hergeleitet werden. Dafür bieten sich die Methoden der Fehlerfortpflanzungsrechnung [Bar07] an. Dabei werden mittels der linearen Anteile einer Taylorreihe die quantitativen Einflüsse von fehlerhaften

Messgrößen abgeschätzt. Für eine Funktion $y = f(x_0, x_1, \dots, x_n)$ lautet die entsprechende Formel

$$\Delta y = \frac{\partial y}{\partial x_0} \cdot \Delta x_0 + \dots + \frac{\partial y}{\partial x_n} \cdot \Delta x_n \quad (4.2)$$

Die damit bestimmbaren Fehleramplituden sind zu normieren und zu klassifizieren. Mit der Zuordnung und Indizierung der Fehlerdetektionsmethoden kann dann auch für komplexe Systeme ein RPN berechnet werden.

Dynamische Datensatzvalidität

Der RPN dient zur Einschätzung der Gültigkeit der Validitätsinformationen, die jedem Datensatz beigeordnet sind. Die Datensatzvalidität selbst basiert dabei auf den Ergebnissen der Fehlerdetektionsmethoden. Diese können eine boolesche Aussage oder aber eine Wahrscheinlichkeit generieren. Da, wie in Tabelle 4.4 gezeigt, eine umfassende Detektionsstrategie mehrere Methoden umfasst, bedarf es einer Abbildungsvorschrift, um die Einzeleinschätzungen auf einer generellen Aussage abzubilden, die einen analogen Wert zwischen 0 und 1 darstellt. Dabei sind die Einzelbewertungen individueller Fehlerdetektionsmethoden in zwei Gruppen einzuteilen:

Dominante Einschätzungen wirken unmittelbar auf das Gesamtergebnis. Ein Messwert außerhalb des Messbereiches kann unabhängig von den Ergebnissen anderer Detektionsmethoden als sicherer Indikator für eine fehlerhafte Messung angenommen werden. Natürlich setzt eine solche Aussage eine mit hoher Wahrscheinlichkeit richtige Klassifikation voraus, die in einer binären Aussage mündet.

Kumulative Einschätzungen werden durch eine umfassende Betrachtung anhand der Einzelergebnisse anderer Detektionsmethoden validiert und dann gegebenenfalls mit diesen zur generellen Validitätseinschätzung fusioniert. Da die Einzeleinschätzungen nicht im Netzwerk kommuniziert werden, setzt eine Einschätzung dieser Kategorie also mindestens eine weitere Detektionsmethode im gleichen Knoten voraus. Wie in Abbildung 4.7 gezeigt, werden die dominant wirkenden Testverfahren in der MOSAIC-Symbolik mit einem vollen Kreis, die kumulativen mit einem leeren Kreis bezüglich des Fehlermanagers gekennzeichnet. Aus der Grafik ist also ablesbar, dass die Fehlereinschätzung der Signalanalyse und die des Vergleichers gegeneinander abgewogen werden, während der Varianztest eine unmittelbar auf die Gesamteinschätzung wirkende Bewertung trifft. Im Unterschied dazu generieren die beiden anderen Untersuchungen einen Wahrscheinlichkeitswert, der sich auf die Gültigkeit der Messung bezieht. Diese Einschätzungen sind durch die Stochastik des Signals unsicher und bedürfen einer gegenseitigen Prüfung.

Die Zuordnung einer Fehlerdetektionsmethode zu den beiden Gruppen hängt in starkem Maße von den Parametern der Detektionsmethode und dem Verhalten des obser-

vierten Signals ab. Analog kann auch keine allgemeingültige Vorschrift für die Zusammenfassung der Einzelbewertungen eingeführt werden. In [ZDK11a] und [ZD10] wurden regelbasierte Methoden verwendet, es sind aber auch andere Ansätze auf der Basis von neuronalen Netzen, Fuzzy Logic o. ä. denkbar. Aufgabe des *MOSAIC*-Frameworks ist es also, die Voraussetzung für das Handling zu schaffen, das dann vom Anwendungsentwickler ausgefüllt werden kann.

Zusammenfassend kann festgestellt werden, dass der Ansatz der flexiblen Datenfusion in verteilten Systemen erst durch ein umfassendes Fehlertoleranzkonzept möglich wird. Dazu werden für jeden Datensatz zwei Informationen bereitgestellt. Aus den individuellen Sensorfehlermodellen und den zugeordneten Detektionsverfahren wird mit dem RPN eine Einschätzung des Sensorsystems bestimmt. Daneben wird für jeden Messwert eine Datensatzvalidität bestimmt, die die Ergebnisse der Einzelprüfungen zusammenfasst. Die komplexe Fehlercharakteristik eines Sensors wird also in diesen beiden Werten gekapselt und erlaubt in der Abstraktion der einheitlichen Semantik eine systemweite Vergleichbarkeit und Verwendbarkeit.

4.3 Datenbeschreibung

Die angestrebte dynamische Fusion ist neben dem reinen Messwert darauf angewiesen, weitere Informationen zu dessen Einordnung bereitgestellt zu bekommen. Diese zusätzlichen Informationen, im Sinne des Sentient-Object-Programmiermodells als Attribute bezeichnet, reichen von einfachen Zeitstempeln bis zu komplexen Informationen wie dem Überwachungsbereich eines Sensors. Die Informationen dazu werden neben dem Anwendungskontext (Position, Fehlertoleranzmethoden usw.) durch die Charakteristika der Hardware bestimmt. Die Hardwarespezifikation wiederum fasst die drei Kernbestandteile Sensor, Controller und Kommunikation in ihren individuellen Ausprägungen zusammen. Für den physischen Sensor sind dies Informationen über die angebotenen Schnittstellen, Fehlermodelle, das Rauschverhalten usw. Analoge Beschreibungen bestehen für die Zielplattform einer *Smart-X*-Applikation und die Kommunikationsschnittstelle. Die Kombination aus den Datenblättern eines Sensors, einer Zielplattform und einer Kommunikationsschnittstelle, die im Folgenden als Systemdatenblätter bezeichnet werden, charakterisiert somit einen *Smart-X*-Knoten und kann für die Codegenerierung eines der gewählten Spezifikation entsprechenden Knotens verwendet werden [Bra+10]. Die Umsetzung dieser abstrakten Beschreibung in einer domänenspezifischen Sprache wird in Abschnitt 5.2 beschrieben.

Für das Ereignisdatenblatt werden die Hardwarebeschreibungen mit dem anwendungsspezifischen Kontext kombiniert und irrelevante Bestandteile wie zum Beispiel die Schnittstellenparameter des eigentlichen Sensors entfernt. *MOSAIC* folgt dabei dem Datenblattkonzept und definiert die XML-basierten Nachrichtendatenblätter, die zur Laufzeit für die Anreicherung der Ereignisse genutzt werden. Welche Informationen aus welcher

Datenquelle sind dabei für die Nachrichtendatenblätter relevant? Tabelle 4.5 gibt dazu einen Überblick und kategorisiert die Informationen entsprechend.

Die Inhalte der Systemdatenblätter sind in den ersten drei Spalten aufgeführt. Für die Zusammenstellung eines Ereignisdatenblattes sind aber nur die im oberen Teil enthaltenen Informationen relevant. Das konkrete Ausgabeformat eines Sensors zum Beispiel ist durch die Anwendung, die auf diesem Knoten läuft, verdeckt und somit für die weitere Verarbeitung ohne Bedeutung. Ein Beispiel dafür ist der sensorseitige Ausgabewert in Volt, der durch die Anwendungsfunktion in eine Distanz gewandelt wird. Ebenso werden die Schnittstellen und deren Parameter zwar im Codegenerationsprozess benötigt, sind aber ohne Bedeutung für die Interpretation der Nachrichten. Die Attribute, die sich aus den Systemdatenblättern ergeben, sind im Einzelnen:

Sensor - Fehlermodelle Ausgehend von den im vorhergehenden Abschnitt vorgestellten Konzepten zur Fehlertoleranz umfassen die Sensordatenblätter eine Aufstellung der für den zugeordneten Sensor zu erwartenden Fehlercharakteristik. Dazu zählen die Fehlerhäufigkeit und die Fehleramplitude entsprechend der Systematik aus Tabelle 4.3.

Sensor - Überwachungsbereich Zur Einschätzung der Relevanz eines Sensors ist es nötig, den Ausschnitt der Umgebung, der von diesem überwacht wird, zu kennen.

Kommunikation - Interaktionsmodell Im Bereich der Regelungs- und Steuerungstechnik arbeiten Sensoren in der Regel zeitgesteuert und publizieren in bestimmten Abständen ein Resultat. Andererseits sind aber auch pollende Abfragen oder ereignisgesteuerte Interaktionsmechanismen (mit dem Überschreiten des Messwertes o. ä.) möglich. Ein Fusionsknoten, der die Datensätze eines solchen *Smart-X* empfängt, kann mit diesem Wissen Rückschlüsse auf den Zustand des publizierenden *Smart-X* ableiten, wenn dieser zum Beispiel über mehrere Perioden keine Nachricht übermittelt.

Neben den bestimmten Hardwarekomponenten zuordenbaren Attributen, folgt eine Vielzahl von Parametern aus der Anwendungsspezifikation. Diese werden aus der Applikation heraus oder über die Lokalisation bestimmt, wie die Spalten vier und fünf der Tabelle 4.5 zeigen. Die Informationen dafür sind implizit im Code enthalten.

Anwendung - Sensortyp/Thema Durch die Adaption der rohen Messdaten mit den Anwendungsfunktionen entsteht der für die weiteren *Smart-X* sichtbare Sensortyp. Sensoren eines Typs verfügen also neben identischer Hardware auch über eine gleichartige Applikation und werden durch ein gemeinsames Datenblatt beschrieben. Um die Fülle der Informationen zu systematisieren, werden Namensräume oder Themen eingeführt, anhand derer die Nachrichten einem Datenblatt zugeordnet werden können.

Anwendung - Detektionsmethoden Die Verarbeitungskette wird, sofern Detektionsmethoden darin vorkommen, den Sensorfehlern zugeordnet. Damit lassen sich, wie im vorhergehenden Kapitel beschrieben, die Messdaten qualitativ bewerten, gleichzeitig kann aber auch zur Laufzeit anhand dieser Informationen geprüft werden, ob ein bestimmter Fehlertyp des Sensors bei einem Datensatz schon geprüft wurde.

Anwendung - Rauschverhalten Die Verarbeitung der Messdaten produziert ein verändertes, im Normalfall reduziertes Rauschverhalten, sodass das Sensorfehlermodell in dieser Hinsicht angepasst werden muss. Die Beschreibung des Rauschverhaltens ist gleichzeitig Basis der Angabe der Unsicherheit eines Messwertes.

Anwendung - Datenformat Das Format der Nachrichten, das im Kommunikationsdatenblatt definiert ist, legt den Aufbau des Datensatzes fest. Während der Entwicklung einer *Smart-X*-Komponente ist damit eine permanente interne Konsistenzprüfung möglich. Zur Laufzeit dient dieses Wissen der Erfassung und Interpretation der auszutauschenden Daten.

Neben der Funktionalität des Knotens wird die Frage der Einbeziehung in eine Fusionsapplikation auch durch seine Position und Orientierung im Hinblick auf ein Bezugssystem bestimmt. Letztendlich folgen drei weitere Attribute aus dem Kontext eines Datensatzes, nämlich der Zeitstempel, die konkrete Validitätsschätzung und das Maß der Unsicherheit.

4.3.1 XML-Darstellung

Wie können derartig heterogene Informationen in einem Beschreibungsformat, das der Anforderung 6 genügt, strukturiert werden? Im Rahmen dieser Arbeit werden dafür XML-basierte Dateien benutzt. XML ist anders als die Datenblattformate vergleichbarer Arbeiten (IEEE 1451) textbasiert und damit in einem einfachen Editor ohne spezielle Tools les- und bearbeitbar. Gleichzeitig kann aber für ein XML-Dokument durch eine eigene Spezifikationssprache ein Regelwerk vorgegeben werden, das die automatisierte Prüfung und Verarbeitung gegenüber einfachen Textdokumenten, wie sie zum Beispiel in ROS Verwendung finden, entscheidend vereinfacht. Es ist somit möglich einfache Konsistenzprüfungen durchzuführen, ohne individuelle Parser entwickeln zu müssen. Diese können allerdings zusätzlich einbezogen werden, um zum Beispiel komplexe zielplattformspezifische Parameter zu prüfen.

Der Aufbau eines Datenblattes wird durch die Vorgabe von XML-Schemas definiert. Innerhalb eines XML-Schemas können Datentypen, Relationen, Vererbungen usw. festgelegt werden, die dazu dienen, die Struktur einer XML-Datei vorzugeben, wobei sehr komplexe Strukturen entwickelt werden können. Im Hinblick auf die Erstellung eines Datenblattes ist es zum Beispiel möglich, zwingend erforderliche Informationen, Optionen oder Auswahlmöglichkeiten zu definieren und damit bereits mit den XML-inhärenten

Tabelle 4.5: Herkunft der für die dynamische Fusion nötigen Attribute mit Einteilung nach relevanten (oberer Teilenteil) und nicht relevanten (unterer Teilenteil)

Hardware/System-Datenblätter		Anwendungsspezifika			Kontext
Sensor	Target	Kommunikation	Funktionalität	Lokalisation	
Fehlermodelle Überwachungs- bereich		Interakt.-modell	Sensortyp-/Thema Detektionsmeth. Rauschverhalten Datenformat Interaktions- modell	Position Orientierung Bezugssystem	Zeitstempel Validität Unsicherheit RPN
Schnittstellen Rauschverhalten Datenformat Transformationen Interaktionsmod. Zeitverhalten	Schnittstellen Datentypen	Schnittstellen			

Mechanismen einen Leitfaden für die Erfassung der relevanten Informationen zu definieren. Die Organisation und Strukturierung der Daten erfolgt dabei anhand von vordefinierten Tags, die als Datenbezeichner fungieren und aus den XML-Schemas vorgegeben werden. Diese Tags können mit zusätzlichen Attributen versehen werden, die Datentypen, Kardinalitäten usw. bestimmen. Diese abstrahierende Zuordnung von spezifischem Wissen zu vordefinierten Mustern eignet sich für die Strukturierung des Wissens als Grundlage für die fehlertolerante und adaptive Datenverarbeitung.

Mit der Umsetzung der Datenblätter als XML-Dokument ist eine gute Lesbarkeit und Vergleichbarkeit derselben gegeben, gleichzeitig kann aber auch aus vielen Programmiersprachen automatisiert auf die beinhalteten Informationen zurückgegriffen werden. Daneben besteht mit XSLT eine Kette von erprobten Werkzeugen, die eine unmittelbare Transformation von XML-Dateien in andere Formate, wie Dokumentationen oder Quellcodes, unterstützen. In den folgenden Darstellungen wurde aus Gründen der Übersichtlichkeit auf die Darstellung der Namensräume der XML-Schemata verzichtet.

Definition der physikalischen Einheiten

Alle Daten mit einem Bezug auf eine physikalische Einheit werden innerhalb des *MOSAIC* mit einer entsprechenden Kodierung versehen. Dazu werden die Konzepte des IEEE 1451 aufgegriffen und motiviert durch die Arbeit von Piontek [Pio07] ein entsprechendes XML-Schema hinterlegt. Abbildung 4.8 zeigt auszugsweise die entsprechende XML-Schemadatei. Der Datentyp `ExtUnits_Type` umfasst dabei die Informationen wie Offset, Magnitude, Skalierungsfaktor und Potenzen um beliebige SI-Einheiten gemäß der Gleichung

$$value_{phy} = scaling \cdot (value + offset) \cdot 10^{Magnitude} \cdot \prod_k^{k \leq 10} (SI_unit_k)^{Potenz_k} \quad (4.3)$$

als eine einheitenbehaftete Größe darstellen zu können. Dazu wird neben der Skalierung des eigentlichen Wertes die Einheit aus den 10 SI-Standardeinheiten als Produkt gebildet. Der entsprechende, in Abbildung 4.8 auf der rechten Seite mit seinen Elementen dargestellte, generalisierte Typ `SIUnit_Type` ist zur Erweiterung der Verifikationsmöglichkeiten einer Eingabe um weitere Basiseinheitstypen ergänzt worden. Wie noch in späteren XML-Schemata zu sehen sein wird, sind eigene Datentypen für Distanz oder Orientierungsvorgaben (`SIUnit_Distance_Type`, `SIUnit_Orientation_Type`) und Zeitangaben `SIUnit_Time_Type` eingefügt worden. Neben einheitenbehafteten Werten können auch einheitenlose Daten mit dem Schema abgebildet werden, um zum Beispiel Zähler ebenfalls abbilden zu können. Dazu besteht die Wahlmöglichkeit zwischen `Quantities` und dem mit `TRUE` vorbelegten `isDimensionless` Wert.

In Erweiterung der in der Arbeit von Kaiser und Piontek [KP06] beschriebenen Umsetzungskonzepte ist zum einen eine verfeinerte modulare Struktur aufgestellt und zugleich

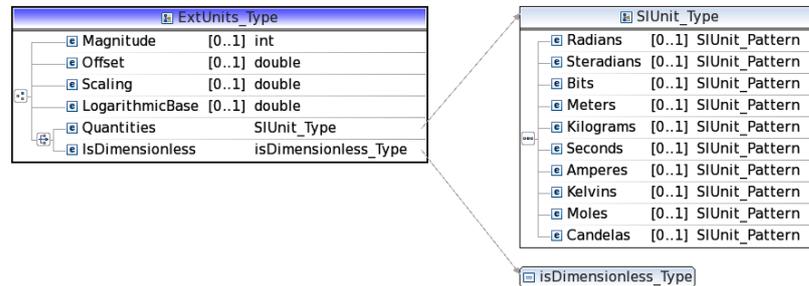


Abbildung 4.8: XML-Schema der Einheitencodierung innerhalb des MOSAIC-Frameworks

unter Ausnutzung der XML-Spezifikationsmöglichkeiten eine Vielzahl an logischen Eingabefiltern integriert worden. So wird beispielsweise über den eigenen Datentypen `SIUnit_Pattern` mit den Mitteln regulärer Ausdrücke sichergestellt, dass die Werte der Potenzen der Einheiten nur mit Werten belegt werden dürfen, die ein Vielfaches von $\pm 0,5$ darstellen, um also m^2 oder \sqrt{Hz} umzusetzen. Daneben konzentrierte sich die Arbeit eher auf die Kommunikation als auf Aspekte der Datenfusion und Fehlertoleranz. Entsprechend ist dort der Zuschnitt der XML-Tags gewählt.

Abbildungsbeschreibung

Für die Beschreibung eines spezifischen Verhaltens oder der mathematischen Ausprägung einer bestimmten Eigenschaft ist es erforderlich, Relationen zwischen Werten in den Datenblättern abbilden zu können. Damit sind dann zum Beispiel möglich Nichtlinearitäten, die Parameter des Rauschens oder eine Abbildungsfunktion von einem Messwert auf einer bestimmten Einheit zu definieren.

Das MOSAIC-Framework bietet, wie Abbildung 4.9 mit der Aufspaltung auf drei komplexe Datentypen zeigt, mehrere Varianten dafür:

Funktionsbasierte Beschreibung Besteht zum Beispiel für die Transformation der Spannungsausgaben eines Sensors auf einen Ausgabewert, wie eine Distanz, eine die Abbildungsvorschrift in Form eines Polynoms, so kann dies im Datenblatt in einer MathML-Syntax festgeschrieben werden. Der MathML-Standard bietet vorrangig eine auf die Darstellung mathematischer Beziehungen orientierte Sprache. Gleichzeitig besteht aber auch in vielen Programmiersprachen eine Funktionalität zur analytischen Auswertung derartiger Ausdrücke, sodass eine automatisierte Berechnung auf dieser Basis möglich ist. Zudem kann die syntaktische Korrektheit der Eingaben mit einfachen Mitteln permanent überwacht werden.

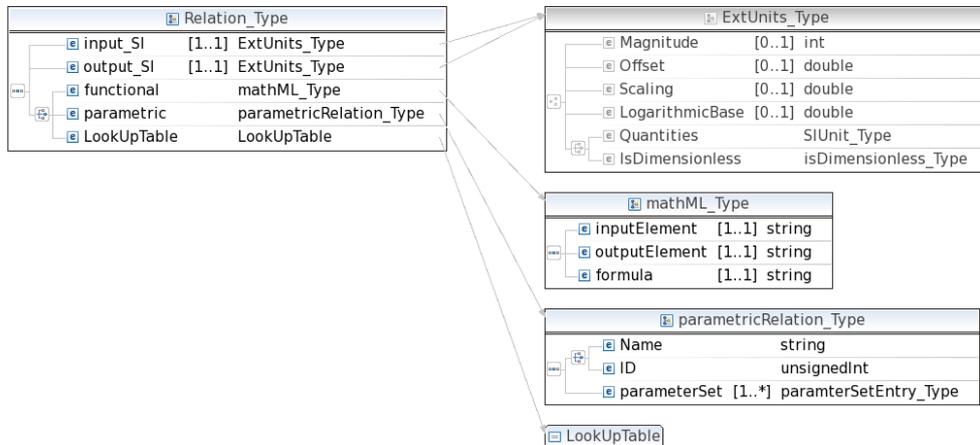


Abbildung 4.9: XML-Schema der Darstellung von Relationen zwischen unterschiedlichen Werten

Parametrische Abbildung Sofern eine Standardfunktion repräsentiert werden soll, genügt die Angabe einer entsprechenden ID bzw. einer Namenskennung und der unmittelbaren Parameter `parameterList`. Hierbei sind insbesondere die bekannten Verteilungsfunktionen zur Beschreibung des Rauschverhaltens hinterlegt. Mit der Definition zulässiger Tupel wie zum Beispiel `Name=NormalDistribution` mit `parameterList=[μ, σ]` oder `(UniformDistribution,[min, max])` wird auch hier eine erweiterbare aber vordefinierte Schnittstelle geschaffen.

Definition via Lookup-Table Fehlt das Wissen um eine funktionale Darstellung des Zusammenhangs zwischen zwei Größen, muss auf eine Lookup-Table zurückgegriffen werden. Diese stellt die Werte der Eingangsgröße den zugehörigen Werten der Ausgangsgröße mit einer vom Nutzer festzulegenden Granularität gegenüber, sodass für beliebige Eingangswerte mittels Interpolation eine Annäherung vollzogen werden kann. Zur Definition einer solchen Struktur besteht in *MOSAIC* der XML-Datentyp `LookupTable`, der es erlaubt, XML-Standarddatentypen in Zweierpaaren in einer Liste zu organisieren.

Das XML-Schema umfasst die drei zuvor genannten Typen von Relationen und bildet diese auf geeigneten Formaten ab. Hinzu kommen die Einheiten der Eingangs- und Ausgangsgrößen, wie Abbildung 4.9 anhand des Typs `Relation_Type` zeigt.

4.3.2 Beschreibung der Attribute im Datenblatt

Auf der Basis der im vorhergehenden Abschnitt getroffenen generellen Festlegungen zur Umsetzung und Struktur der elektronischen Datenblätter innerhalb des *MOSAIC*-

Frameworks sollen nunmehr die einzelnen Attribute besprochen werden, die in der Tabelle 4.5 hinsichtlich ihrer Herkunft zusammenfassend dargestellt wurden. Zur Illustration der Anwendung werden für den bereits genannten Infrarotdistanzsensor GP2D120 [Sha07] die entsprechenden Datenblattauszüge dargestellt.

Fehlermodelle

Die im vorherigen Abschnitt dieses Kapitels besprochenen Fehlermodelle und Semantiken werden in den Datenblättern entsprechend abgebildet. Dazu werden die Elemente der Fehlermodellsemantik in ein XML-Schema, welches in Abbildung 4.10 dargestellt ist, überführt. Die entsprechende XML-Struktur fasst die sensorspezifischen Parameter, wie Fehlerhäufigkeit und Fehleramplitude, mit den jeweiligen Indizes gemäß Tabelle 4.3 zusammen. Durch die spezielle Typdefinition `FaultCategory_Type`, `FaultIndex_Type` kann die Eingabe der Fehlerkategorien (1-14) aus der Definition der Tabelle 4.1 und die Indizierung (1-10) auf Korrektheit geprüft werden.

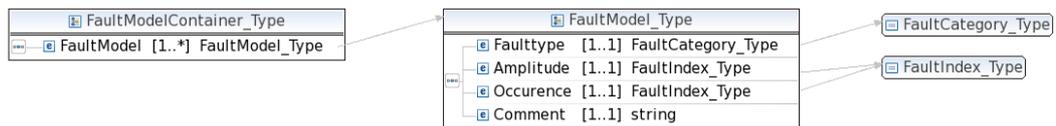


Abbildung 4.10: XML-Schema zur Erfassung der Sensorfehler

Quelltextauszug 4.1 illustriert die konkrete Umsetzung beispielhaft für einen GP2D120-Infrarotdistanzsensor anhand von zwei der vier Fehlermodelle dieses Sensors (Outlier (2), Stuck-At (5)). Die vollständige Auflistung der Fehlermodelle erfolgte bereits in Tabelle 4.4 im Zusammenhang mit der Erarbeitung des Fehlerkonzepts.

Überwachungsbereiche

Die Erfassungsbereiche eines Sensors sind höchst unterschiedlich ausgeprägt. Der von einem Laserscanner überstrichene Bereich markiert, ohne dass sich Hindernisse darin befinden, einen Kreisausschnitt von geringer Höhe. Dagegen überwacht ein Ultraschallsensor seine Umgebung innerhalb eines mitunter nichtsymmetrischen und von vielen Haupt- und Nebenkeulen gekennzeichneten Erfassungsbereiches. Die Information über diesen Überwachungsbereich ist für die adaptive Fusion von hoher Relevanz, da mit diesem Wissen entschieden werden kann, ob ein Sensor, selbst wenn er einen passenden Messwert bereitstellt, überhaupt in der Lage ist, eine gültige Messung für einen bestimmten Ausschnitt der Umgebung umzusetzen.

Um dieses wichtige Selektionskriterium in adaptiven Szenarien auswerten zu können, sind zwei Informationen erforderlich. Zum einen muss für die Messung eine Positions- und Richtungsangabe und zum anderen eine Abstraktion der Sensorkeule bereitstehen.

```

1 <FaultModels>
2   <FaultModel>
3     <Faulttype>2</Faulttype>
4     <Amplitude>10</Amplitude>
5     <Occurence>6</Occurence>
6     <Comment>Outlier</Comment>
7   </FaultModel>
8   <FaultModel>
9     <Faulttype>3</Faulttype>
10    <Amplitude>8</Amplitude>
11    <Occurence>9</Occurence>
12    <Comment>Offset</Comment>
13  </FaultModel>
14  ...
15 </FaultModels>

```

Listing 4.1: , Datenblattauszug eines Infrarotdistanzensors GP2D120-Sensor mit der Beschreibung der Sensorfehler als Abbildung der Aufstellung aus Tabelle 4.4

Hierfür wurde im Rahmen dieser Arbeit ein Pyramidenstumpfmodell entworfen, das sich auf viele Sensormodelle anwenden lässt. Der Pyramidenstumpf wird der Darstellung in Abbildung 4.12 und im Quelltextauszug 4.2 folgend aus sechs Distanzmaßen gebildet. Aus einem Pyramidenkörper lassen sich, wie Abbildung 4.13 illustriert, Überwachungsbereichsgeometrien einfacher Sensoren approximieren. Komplexere Konturen, etwa das zweidimensionale Abbildungsverhalten eines Laserscanners oder die komplizierte Keulenstruktur eines Ultraschallsensors, müssen aus mehreren dieser Basisbausteine zusammengesetzt werden. Für die Untersuchung der Frage, ob beispielsweise der obere Roboter in einem der Überwachungsbereiche der Sensoren erscheint, wurde eine Transformation abgeleitet, die mit einer einzigen Koordinatentransformation die Normierung der fraglichen Punkte gegenüber dem Sensorbeam ermöglicht. Darauf aufbauend lässt sich die Frage, ob ein bestimmter Sensor zur Validierung der Positionsannahme des Roboters in Abbildung 4.13 herangezogen werden kann, auch auf einem eingebetteten System bestimmen.

In Abbildung 4.13 wird das Wissen um die Lokalisation und die Sensorkeule kombiniert genutzt, um diese und die aktuellen Messwerte mit den Mitteln der Augmented Reality (AR) darzustellen. Diese Möglichkeit, basierend auf den Ereignisdatenblättern des *MOSAIC*-Frameworks, erlaubt eine Visualisierung der Sensorsysteme, wobei sich diese Technik, wie in [Die+10] beschrieben, zur interaktiven Entwicklung oder zur Illustration der Messdaten während des Betriebs nutzen lässt.

Bemerkenswert ist die Veränderung der Idee der Information durch das Wissen um die Position und die Form des Überwachungsbereiches. Zum Beispiel im Hinblick auf einen Distanzsensor wird der einfache Abstandswert, den der Sensor eigentlich liefert, mit diesem Wissen zu einer Lokalisationsangabe. Damit steigt der Abstraktionsgrad

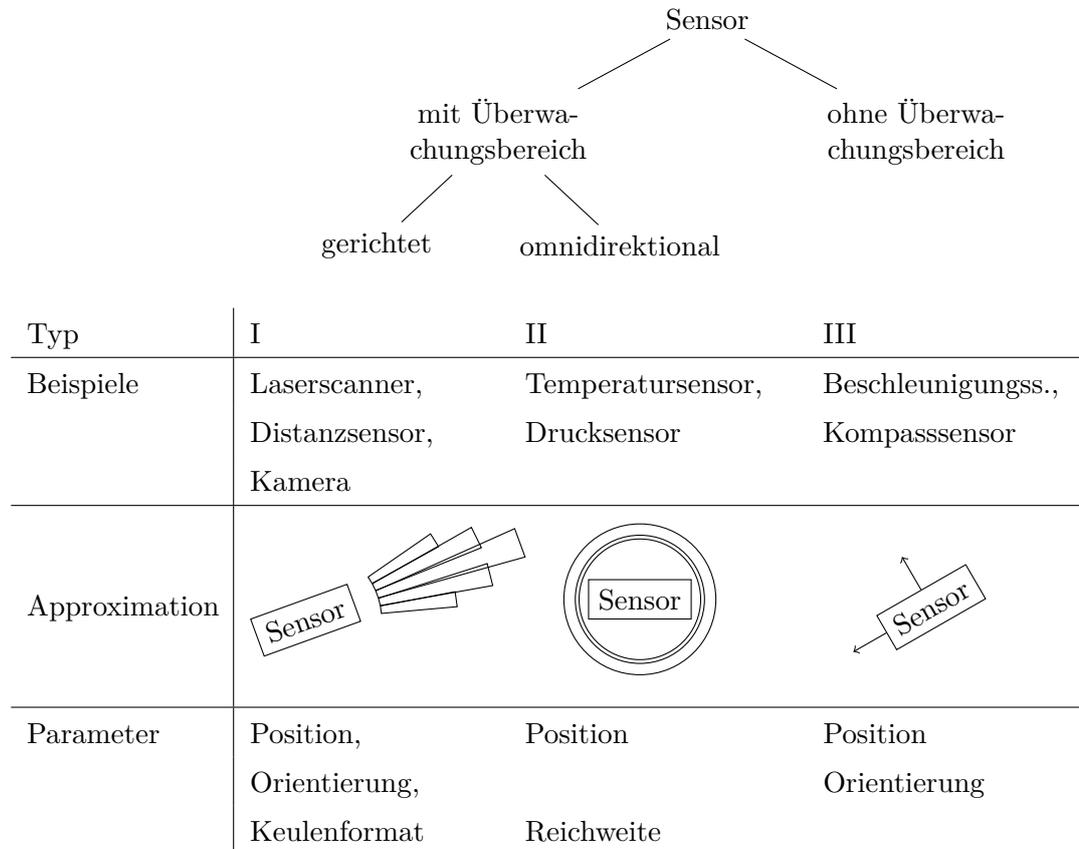


Abbildung 4.11: Kategorien von Sensoren im Hinblick auf Position, Orientierung und Überwachungsbereich

entscheidend, denn die Aussage eines solchen Ereignisses kann umgekehrt auch dahin gehend von Bedeutung sein, dass eine bestimmte Position unbesetzt ist. Dabei muss man allerdings bedenken, dass andere Sensoren wie 2D-Kameras die Tiefe als Information nicht liefern können. In diesem Fall trägt diese Komponente des Positionsvektors eine unendliche Unsicherheit.

Interaktionsmodelle und Zeitverhalten

Die Einbindung eines Sensorknotens in das Netzwerk erfolgt in Abhängigkeit von dessen Interaktionsmodell, das ereignisorientiert oder pollingbasiert ablaufen kann. Sensoren der ersten Kategorie stellen einen neuen Datensatz periodisch oder bei der Erfüllung bestimmter Umgebungskriterien bereit und publizieren diesen im Netz. Auf der Empfängerseite stehen dann entsprechende Callback-Mechanismen für die Verarbeitung des

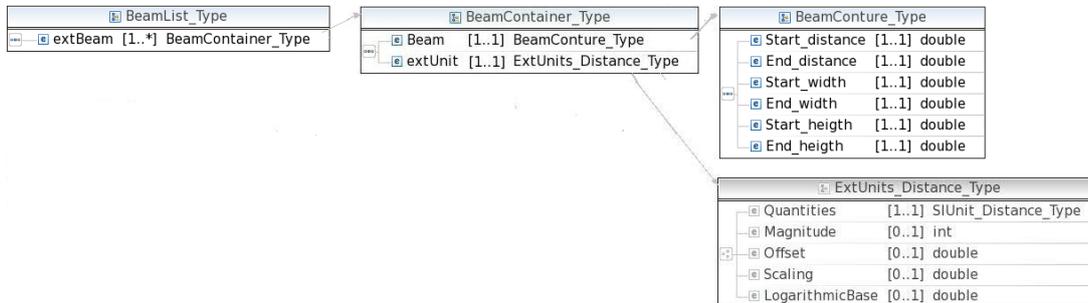


Abbildung 4.12: XML-Schema zur Erfassung des Sensorüberwachungsbereiches anhand einer Liste von Pyramidenstümpfen, die sich jeweils aus sechs Distanzangaben zusammensetzen, die mit einer Einheit der Entfernung versehen sind

```

1 <SensorTyp>1<\SensorTyp>
2 <BeamList >
3   <extBeam>
4     <Beam>
5       <Start_distance>0.2</Start_distance>
6       <End_distance>0.8</End_distance>
7       <Start_width>0.02</Start_width>
8       <End_width>0.05</End_width>
9       <Start_height>0.02</Start_height>
10      <End_height>0.08</End_height>
11     </Beam>
12     <extUnit>
13       <Quantities>
14         <Meters>0.0</Meters>
15       </Quantities>
16     </extUnit>
17   </extBeam>
18 </BeamList>
  
```

Listing 4.2: Datenblattauszug eines GP2D120-Sensors mit Kategorisierung und Beschreibung des Erfassungsbereiches

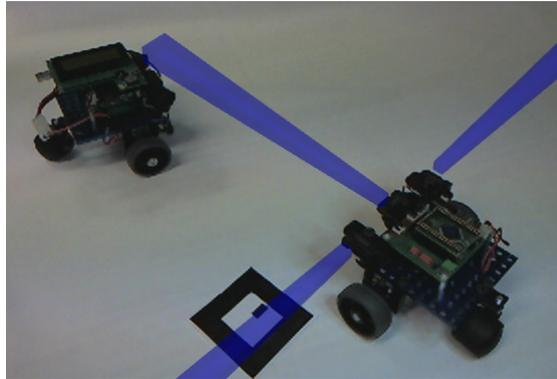


Abbildung 4.13: Darstellung des Pyramidenstumpfmodells mit Hilfe der Augmented Reality für einen Roboter mit drei GP2D120-Infrarotsensoren

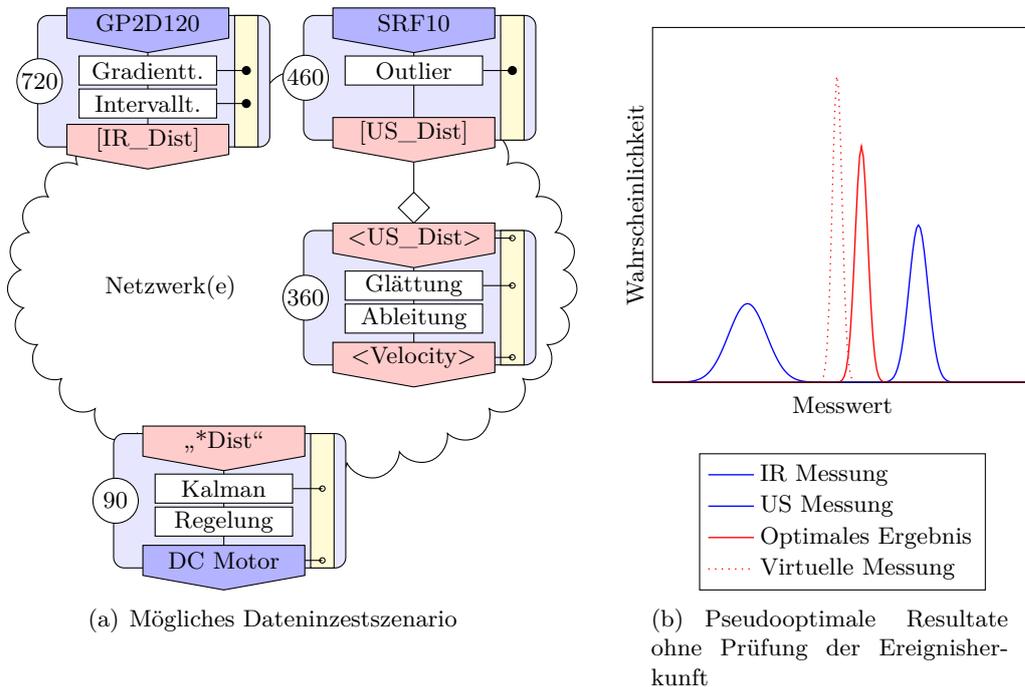
Datensatzes bereit. Für zyklisch arbeitende Sensoren ist es bei bekannter Periodendauer möglich eine erste Fehlerprüfung dahingehend umzusetzen, dass ein Timer die Frequenz der eingehenden Messdaten überwacht. Beim nichtdeterministischen Nachrichtenversand fehlt diese Möglichkeit.

Dem stehen Sensoren gegenüber, die aus der Anwendung heraus über das Netzwerk abgefragt werden. Von großer Bedeutung ist hierbei die Rate, mit der der Sensor eine neue Messung bereitstellt, da diese durch die Abfrageperiode nicht unterschritten werden sollte, um die Erzeugung einer vermeintlichen Stabilität der Messdaten auszuschließen. Eine Abschätzung des zeitlichen Antwortverhaltens auf eine Anfrage hin ermöglicht hier eine Fehlerdetektion. Dieser zeitliche Versatz setzt sich aus der Erfassungs- und Verarbeitungsdauer des *Smart-X*-Sensors und der Verzögerung beim Datentransfer zusammen. Erstere Größe wird dabei in der Regel als statisch, die Netzwerkverzögerung aber als nichtdeterministisch angenommen, was die Einschätzung des Sensorzustandes erschwert.

Im Datenblatt sind also neben dem Vermerk über das Interaktionsmodell Information zur Periodizität des Sensors vorzusehen und im Fall eines pollingbasierten Sensors die Antwortzeiten des *Smart-X* zu integrieren.

Datenherkunft

Durch die adaptive Einbeziehung von Sensormessungen in die Verarbeitung besteht die Gefahr der mehrfachen Integration Messungen und damit der verbundenen Verfälschung des Ergebnisses. Diese Pseudomessungen entstehen, wenn ein weiterer Knoten Messdaten eines Sensors verarbeitet und das Resultat dem Anforderungsprofil eines Fusionsknotens entspricht. Mitchell [Mit07] weist auf dieses grundsätzliche Problem hin und nennt den Vorgang „Dateninzest“, ohne allerdings Lösungsmöglichkeiten vorzustellen. Ein denkbare Szenario wird in Abbildung 4.14(a) präsentiert. Ein Ultraschallsensor



(a) Mögliches Dateninzestszenario

(b) Pseudooptimale Resultate ohne Prüfung der Ereignisherkunft

Abbildung 4.14: Darstellung eines möglichen Dateninzestszenarios in adaptiven Fusionsanwendungen

und ein GP2D120 sind an einem Roboter befestigt und erfassen redundant die Position des Systems. Die Messdaten des Ultraschallsensors werden durch einen passiven *Smart-X*-Fusionsknoten erfasst, nach einer Prüfung der Werte abgeleitet und das Resultat als Bewegungsgeschwindigkeit gegenüber möglichen Hindernissen ausgegeben. Unabhängig davon arbeitet eine aktiver *Smart-X*-Fusionsknoten, dessen Autor vom zuvor genannten *Smart-X*-keine Kenntnis hat. Da auch das Ergebnis der Geschwindigkeitsberechnung zumindest indirekt zum Suchschema „Dist“ passt, würden die originalen Messwerte mit einem vermeintlich unabhängigen Datensatz verknüpft werden. Das Ergebnis wird in Abbildung 4.14(b) dargestellt. Wird für die beiden Messwerte von einem normal verteilten Messrauschen (blaue Linien) ausgegangen und die inverse Varianz als Kriterium einer gewichteten Mittelwertbildung gemäß

$$y = \frac{1}{\sigma_2 + \sigma_1} \cdot \left(\frac{1}{\sigma_1} y_1 + \frac{1}{\sigma_2} y_2 \right) \quad (4.4)$$

benutzt, so ergibt sich eine rauschoptimierte Schätzung (rote Linie). Wird aber auch die Geschwindigkeitsschätzung in die Berechnung einbezogen, folgt mit der mehrfachen

Integration der vermeintlich unabhängigen Werte eine fehlerhafte, pseudooptimale Schätzung (durchbrochene rote Linie in Abbildung 4.14(b)).

Das *MOSAIC*-Framework sieht daher eine Auszeichnung der originären Sensoren vor, deren Messdaten die Grundlage für weitere Ergebnisse sind. Für *Smart-X*-Sensoren und passive *Smart-X*-Fusionsknoten mit einem definierten Sensor oder einer Themenzuordnung ist diese Frage rasch zu beantworten und kann im Datenblatt eines Themas abgelegt werden. Für aktive *Smart-X*-Fusionsknoten, die entsprechend der Einteilung aus Abbildung 4.1 selbstständig die verfügbaren Daten allozieren, müssen diese Informationen jedem Ergebnisdatensatz beigefügt werden, um Dateninzests zu verhindern.

Detektionsmethoden

Analog zur Abbildung der relevanten Fehlermodelle in einer XML-Struktur werden auch die Detektionsmethoden festgehalten. Für die eindeutige Zuordnung zwischen Fehlermodell und Detektionsmethode wird wiederum der entsprechende Referenzindex **Fault-Category_Type** angegeben und für diesen die Detektionswahrscheinlichkeit **Detection-Probability** festgelegt. Gleichzeitig wird für die Detektionsmethode bestimmt, ob im Hinblick auf die laufende Berechnung der Validität eine dominante Einschätzung vorliegt. Dabei können auch mehrere Detektionsmethoden einem Sensorfehler zugeordnet werden.

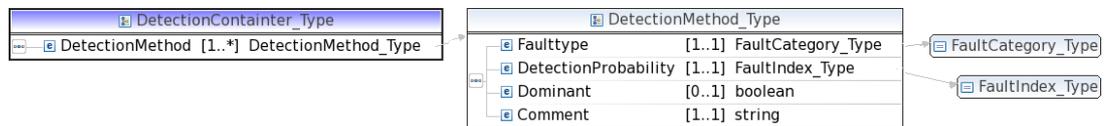


Abbildung 4.15: XML-Schema der Darstellung der Zuordnung von Detektionsmethoden zu den Sensorfehlern

Die Anwendung des XSD-Schemas auf ein konkretes Beispiel ist im Quelltextauszug 4.3 dargestellt, der sich wiederum auf die Tabelle 4.4 bezieht. Neben zwei fehlerorientierten Funktionen ist eine weitere Methode – **Mean Calculation** – ohne Validitätsausgabe eingebunden. Der ab Zeile 7 definierte **Interval Check** produziert eine dominante Validitätseinschätzung. Im Fall eines durch diese Methode detektierten Fehlers wird diese Bewertung als generelle Einschätzung des Datensatzes weitergegeben.

In Kombination mit den Informationen der entsprechenden Sensorfehler aus Quelltextauszug 4.1 lässt sich nunmehr automatisiert eine fehlertoleranzbezogene Einschätzung des *Smart-X*-Sensors bestimmen. So kann der RPN-Wert sehr einfach aus den Werten des Datenblattes bestimmt werden. Zur Laufzeit können intelligente *Smart-X*-Fusionsknoten überprüfen, welche potentiellen Fehler bereits abgeprüft wurden und die Verarbeitung entsprechend zuschneiden.

```

1 <DetectionMethods>
2   <DetectionMethod>
3     <Faulttype>2</Faulttype>
4     <DetectionProbability>3</DetectionProbability>
5     <Comment type="str">Variance Test<\Comment>
6   </DetectionMethod>
7   <DetectionMethod>
8     <Faulttype>5</Faulttype>
9     <DetectionProbability>1</DetectionProbability>
10    <Dominant>1</Dominant>
11    <Comment type="str">Interval Check<\Comment>
12  </DetectionMethod>
13  <DetectionMethod>
14    <Faulttype>0</Faulttype>
15    <DetectionProbability>10</DetectionProbability>
16    <Comment type="str">Mean Calculation<\Comment>
17  </DetectionMethod>
18  ...
19 </DetectionMethods>

```

Listing 4.3: Datenblattauszug eines GP2D120-Sensors mit Beschreibung der Detektionsmethoden

Datenformat

Ein Ereignis setzt sich zwingend aus dem eigentlichen Messwert oder Fusionsresultat, der dynamischen Validitätseinschätzung und dem Zeitstempel sowie einer Orts- und gegebenenfalls Orientierungsangabe zusammen. Abbildung 4.16 zeigt diese Mindestanforderungen anhand der entsprechenden Kardinalitäten zwischen den unterschiedlichen XML-Schematadefinitionen. Es können weitere Attribute wie die Unsicherheit, ein Datensatzindex usw. hinzukommen. Für die korrekte Interpretierbarkeit sind für jedes dieser Elemente die Zahl der Einträge, der Datentyp und die physikalische Einheit anzugeben.

Lokalisationsbezüge

Hinsichtlich der Lokalisierung wird in der vorliegenden Arbeit von einem dreidimensionalen Raum ausgegangen. Die Sensoren sind dabei, wie Abbildung 4.11 im oberen Teil zeigt, in zwei Kategorien zu teilen – die ungerichteten und die gerichteten. Sensoren, die eine richtungsunabhängige Größe wie Druck oder Temperatur omnidirektional messen, benötigen lediglich eine Angabe der Position, sofern eine gleichförmige Sensibilität des Transducers vorliegt. Die Messdaten anderer Sensoren lassen sich dagegen nur verwenden, wenn zur Positionsangabe auch eine Richtungsinformation bereitgestellt wird.

Das *MOSAIC*-Framework ist ein Teil der Forschungsfelder der Arbeitsgruppe für Betriebssysteme und eingebettete Systeme der Otto-von-Guericke-Universität Magdeburg.

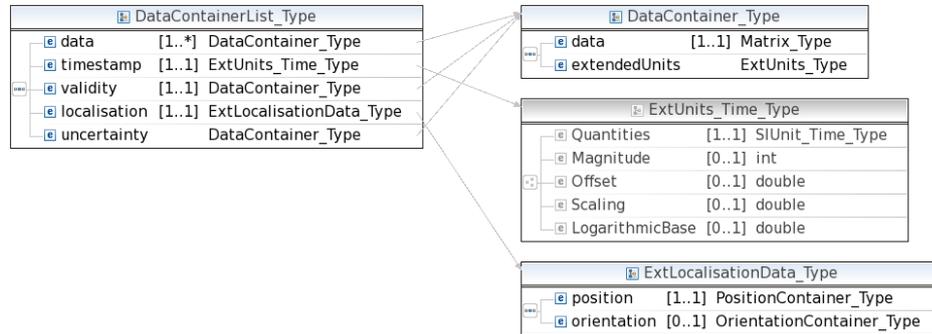


Abbildung 4.16: XML-Schema zur Beschreibung des Ereignisformates

Im unmittelbaren Zusammenhang mit den Kernkonzepten stehen die in [DZK11b] beschriebenen Modellierungen der Umgebung von Robotersystemen. In Erweiterung des Szenarios aus Kapitel 1 werden in dieser Arbeit von einer intelligenten Umgebung auch die notwendigen Informationen zu deren physikalisch-geometrischer Beschaffenheit bereitgestellt.

Über die Vorgabe eines Umgebungsmodells bzw. einer Roboterstruktur lassen sich die Positions- und Orientierungsangaben eines Sensors explizit einordnen. Dabei verweist das Datenblatt des Sensors auf die übergeordnete Modellinstanz. Dies kann zum Beispiel eine mobile Plattform oder ein Teil der Modellhierarchie der Umgebung sein. Damit ist dann gewährleistet, dass mit Kenntnis dieses Modells jederzeit eine korrekte Einordnung des Sensorüberwachungsbereiches möglich ist. Fehlt das Umgebungsmodell, sind die externen Sensoren in Unkenntnis der Referenzpunkte der Positionsangabe nicht verwendbar.

Der in [DZK11a] vorgestellte Ansatz eines umfassenden Umgebungsmodells eröffnet im Zusammenspiel mit den Konzepten von *MOSAIC* weitreichende Fehlerdetektionsmöglichkeiten für die Sensorik. Mit der modellhaften Abbildung der Umgebung steht eine zusätzliche Referenz zur Überprüfung der Messwerte bereit, sodass mit dem Wissen um das spezifische Verhalten des Sensors anhand des virtuellen Abbildes eine Vorhersage der Messung getroffen werden kann. Der Abwägungsprozess, ob bei einer signifikanten Abweichung von virtuellen und realen Messungen ein Modellfehler, eine Störung des Systems oder ein Messfehler vorliegt, wird durch die Daten unterstützt, die das *MOSAIC*-Framework bereitstellt.

4.3.3 Einbindung der Datenblätter in *MOSAIC*

Das in diesem Abschnitt vorgestellte, elektronische Ereignisdatenblatt wird innerhalb des *MOSAIC*-Frameworks in folgenden Bereichen eingesetzt, die für die dynamische Fusion unerlässlich sind:

Ereignisidentifikation Voraussetzung für die dynamische Erfassung der Messdaten ist die Untersuchung der Ereigniskanäle auf ihre Relevanz für eine konkrete Aufgabenstellung. Ausgangspunkt ist dabei die Frage, ob diese einem vorgegebenen Muster entsprechen, also Daten einer bestimmten physikalischen Einheit oder einer Mindestvalidität umfassen. Mithilfe der Ereignisdatenblätter, die die Ereignisse beschreiben, kann der Indikator eines Ereignisses oder Kanals, das Subjekt, auf konkreten Anforderungsanforderungen abgebildet werden.

Datenbeschreibung Nach der Allokation der Daten erfolgt deren Interpretation. Dazu werden die in den Datenblättern beschriebenen Nachrichtenformate ausgewertet und auf die empfangenen Ereignisse angewendet.

Fehlerprüfung Verteilte Sensor-Aktor-Systeme stellen komplexe Anwendungen dar, deren Entwicklung bereits Fehlererkennungsmechanismen beinhalten sollte. Mithilfe der Datenblattspezifikationen lassen sich zum Beispiel inkonsistente Datenformate oder Einheiten entdecken oder fehlende Attribute in einem Ereignis detektieren.

Bewertung der Validitätsschätzung Da die Datenblätter der Ereignisse eine Übersicht über die für den individuellen Sensor zu erwartenden Fehlermodelle bereithalten, lassen sich im Zusammenhang mit dem nachrichteninhärenten Validitätswert Aussagen zur Gültigkeit treffen, die für die nachfolgende Verarbeitungskette eine wichtige Selektionsgröße darstellt.

Zwar wurde in [Pio07] ein ähnlich übergreifender Ansatz gezeigt, dabei ist aber ein einzelnes Datenblatt in verschiedenen Richtungen genutzt worden, während innerhalb von *MOSAIC* klar zwischen Ereignis und Systemdatenblättern für die Entwicklung getrennt wird. Die Ballung der Spezifikationen in einer einzigen Beschreibungsdatei bringt zwei Nachteile mit sich. Ein großer Teil der für den Entwicklungsprozess relevanten Informationen wie zum Beispiel Schnittstellenparameter zwischen Sensor und Target, Spezifika der Zielplattform oder Kommunikationsdetails sind zur Laufzeit für die Identifikation und Interpretation der im Netzwerk verbreiteten Daten unerheblich. Im Sinne der verwendeten eingebetteten Geräte wurde aber auf eine möglichst kompakte Struktur der Ereignisdatenblätter abgezielt, sodass eine Trennung zwischen System- und Ereignisdatenblättern zwingend erforderlich ist. Der zweite Kritikpunkt an einem einzelnen übergreifenden Datenblatt bezieht sich auf die fehlende Modularität. Die beschriebene Einteilung der Systemdatenblätter erlaubt eine flexible Kombinierbarkeit von Sensoren und Zielplattformen. Eine einmal vorgenommene Charakterisierung eines Sensors, einer Zielplattform usw. ist also in vollem Umfang in einem neuen oder neu konfigurierten Projekt wiederverwendbar.

4.4 Adaptive Verarbeitung

Die adaptive Verarbeitung der eingehenden Datensätze steht vor zwei Grundproblemen. Zum einen müssen die relevanten und das Ergebnis signifikant positiv beeinflussenden Ergebnisse erkannt werden. Zum anderen sind diese in einem Algorithmus zu verarbeiten, für den zur Designzeit die Zahl der verfügbaren Daten nicht bekannt ist. Folglich besteht die Aufgabe darin, zunächst ein Konzept für die Selektion der Datensätze zu entwerfen und darauf aufbauend eine adaptive Verarbeitung von Sensordaten zu entwickeln.

Um eine mehrschichtige Selektionsstruktur in die *Smart-X*-Architektur zu integrieren, bedarf es der Untersuchung von drei Aspekten der Ereignisfilterung. Zunächst müssen die Kriterien der Selektion erkannt und die Filtermethoden herausgearbeitet werden. Im Weiteren stellt sich, den verteilten Charakter der Anwendung berücksichtigend, die Frage nach dem Filterort und der Filtermethodik. Letztendlich ist zu diskutieren, wie die Filterkonfiguration vorgenommen werden kann.

4.4.1 Selektion

Kriterienkatalog

Welche Attribute oder Ereignisdatenblattinformationen kommen als Selektionskriterien infrage? Ausgehend vom Anspruch des Motivationsszenarios und der Diskussionen zum verbindlichen Inhalt eines Ereignisses lassen sich sechs Attribute als selektionsrelevant herausarbeiten:

1. Einheit des eigentlichen Messwertes,
2. Risk Priority Number,
3. Überwachungsbereich des Sensors in Verbindung mit dessen Position,
4. Alter der Daten,
5. Unsicherheitsniveau der Daten,
6. aktuelle Validität.

Diese sollen in den folgenden Absätzen weiterer differenziert und mögliche generelle Empfehlungen zur Definition der zugehörigen Grenzwerte abgeleitet werden.

Einheiten Die Einheit des eigentlichen Messwertes eines Datensatzes stellt das wichtigste Selektionskriterium dar. Anhand der Datenblätter kann für jede Messung die zugehörige physikalische Einheit bestimmt werden, wobei entsprechend der Abbildungskonvention SI-Einheiten benutzt werden. Wenn also eine Abstandsregelung alle grundsätzlich relevanten Messungen erfassen möchte, so können die Datensätze, die keine mit

„Meter“ oder den entsprechenden Varianten ausgezeichnete Messung umfassen, aus-
 gesondert werden. Gleichermaßen können Messdaten aber entsprechend der Strukturierung
 aus Abbildung 4.17 indirekt wegen physikalischer Zusammenhänge von Interesse sein.

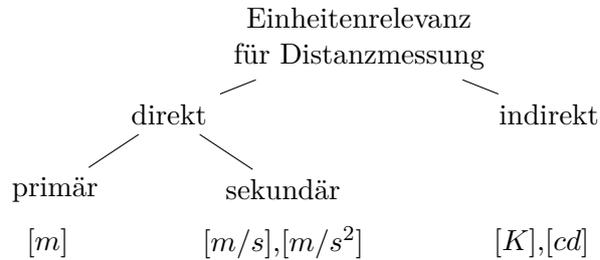


Abbildung 4.17: Systematik der Einbeziehung von Datensätzen in Abhängigkeit von der physikalischen Einheit des Messwertes am Beispiel einer Distanzmessung

Wie die Abbildung zeigt, wird zwischen primärer und sekundärer sowie direkter und indirekter Relevanz bezüglich der Einheit des Messwertes unterschieden. Die primär zu berücksichtigenden Datensätze zeichnen sich durch eine mit der Anforderung unmittelbar übereinstimmende Einheit des Messwertes aus. Sekundäre Relevanz ist durch die Ableitung oder eine Integration der nachgefragten Einheit definiert. So zum Beispiel sind die Daten eines Beschleunigungssensors zweimal und die der Odometrie einmal zu integrieren, um dann Eingang in eine Abstandsbestimmung zu finden. Im Gegensatz zu diesen Größen fließen andere Messwerte nur über bestimmte physikalische Modellannahmen des Sensors oder des Systems in die Berechnung ein. Ein Beispiel dafür wäre die Auswirkung der Batteriespannung auf die Geschwindigkeit des Roboters oder der Temperatureinfluss auf Ultraschallmessungen. Diese Daten werden als indirekte Größen bezeichnet. Der Entwickler eines aktiven Fusionsknotens hat abzuwägen, welchen Grad an Abweichung zur eigentlich gewünschten, physikalischen Einheit durch die Anwendung toleriert werden soll. Eine weitgehende Öffnung der Schnittstellen für Ereignisse mit Messdaten verschiedener Einheiten erfordert eine entsprechende Verarbeitungsfähigkeit, die variable mathematische Modelle, das nötige Expertenwissen usw. beinhalten muss. Möglicherweise sind auch hier adaptive Konzepte möglich, wenn in Situationen mit wenigen Ereignissen die einheitenbezogene Suchmaske erweitert und umgekehrt, bei einer ausreichenden Datenlage, verringert wird.

Risk Priority Number Die zulässige Größe einer RPN, wie sie im Rahmen dieser Arbeit für die Fehlerbewertung angewendet wird, liegt in der Literatur, abhängig vom Anwendungsszenario, im unteren Zehntel des erreichbaren Spektrums [Pep06]. Systeme mit einem Wert größer als 40 sollten nur in Einzelfällen zum Einsatz kommen, eine RPN

größer als 100 gilt als nicht tolerabel [BW08]. Diese engen Maßstäbe, die zudem auf die Kategorisierungen eines Industrieproduktes zugeschnitten sind, lassen sich im Hinblick auf dynamische Sensorapplikationen nicht anwenden.

Andere Autoren ziehen eine hybride Bewertung vor und nutzen neben dem eigentlichen RPN-Wert auch dessen Komponenten (O, P, D) für eine Klassifikation [Wer11]. Dazu definiert der Anwender eine Risikomatrix über den einzelnen Kenngrößen der FMEA, im Sinne dieser Arbeit also mit der Auftretenswahrscheinlichkeit, der Fehleramplitude und der Detektionswahrscheinlichkeit. Die dreidimensionale Struktur wird dann genutzt, um Bereiche tolerierbarer Systemanfälligkeit zu markieren. Die Festlegungen der Areale sind auf das Sicherheitslevel der Anwendung zuzuschneiden, konkrete Standards sind in diesem Bereich für sensorische Anwendungen nicht bekannt.

Abhängig vom Einsatzszenario hat der Entwickler eigene Schwellwerte für die FMEA-basierte Zurückweisung von Ereignissen zu definieren. Mit der Beschreibung der Fehlermodelle und der implementierten Fehlerdetektionsmethoden in den Ereignisdatenblättern kann die Konfiguration dieser Selektionskriterien zur Laufzeit angepasst werden.

Überwachungsbereich Der Überwachungsbereich wird in *MOSAIC* als Pyramidenstumpf modelliert, wobei von einem im Ursprung des Koordinatensystems liegenden Sensor ausgegangen wird. Die Prüfung der Relevanz eines Ereignisses anhand des Sensorüberwachungsbereiches zielt auf die Frage, ob ein oder mehrere geometrische Punkte, physikalische Größen usw. eine gemeinsame Schnittmenge mit dem Überwachungsbereich aufweisen. Diese Vergleichsgrößen sind dabei veränderlich, wie möglicherweise auch die Lage der Sensoren selbst, was eine permanente Überprüfung erforderlich macht.

Vor der Bewertung werden die Sensorposition und Orientierung aus dem Ereignis sowie die Art und Form der Sensorkeulen aus dem Datenblatt ausgelesen. Wenn eine stochastische Systemgröße als Kriterium dient, wird ein Sample von n Zufallswerten entsprechend der zugehörigen Kovarianzmatrix erzeugt. Diese drei Elemente – Sensorposition, Sensorkeule und Systemzustand – werden an die Validierungsfunktion übergeben. Zunächst wird die Zahl der nötigen Rechenoperationen für die Transformation der Sensorkeule ins globale Koordinatensystem bestimmt. In einem zweiten Schritt wird der Aufwand für die umgekehrte Transformation der Systemgrößensamples in das Sensorkoordinatensystem durchgeführt, um darauf aufbauend eine die Rechenleistung schonende Strategie zu wählen.

Alter der Daten Da jedem Ereignis ein Zeitstempel zugeordnet ist, lassen sich die durch die mess- und kommunikationsbedingte Verzögerung ungeordnet erfassten Datensätze in eine chronologische korrekte Ordnung überführen. Stehen mit dem Beginn der Verarbeitung nunmehr n unterschiedlich alte Ereignisse bereit, so sind mehrere Vorgehensweisen möglich. Ist die Dynamik des beobachteten Systems gering, so kann der zeitliche Versatz vernachlässigt werden. Wenn dies nicht der Fall ist, hat der Anwendungsentwickler

zwischen einer datensatzorientierten oder einer fusionsorientierten Synchronisierung zu entscheiden [KAN05]. Erstere nutzt ein analytisches Modell, um die Ereignisse auf einem Zeitpunkt abzubilden und daraufhin einen Fusionsalgorithmus auf die so vorbehandelten Daten anzuwenden. Die zweite Variante verrechnet jedes Ereignis in der chronologischen Reihenfolge. Letztgenannter Ansatz ist mit der Umsetzung der Methoden von Kalman und Bayes sehr verbreitet.

Dabei stellt sich aber das Problem, wie mit veralteten Datensätzen umzugehen ist, die hinter dem Zeithorizont der aktuellen Schätzung liegen. Die Einbeziehung solcher Datensätze in ein analytisches Modell wird nach Bar-Shalom [BS02] als Retrodiktion bezeichnet. Dieser Abweichung von der kontinuierlichen Folge erhöht den Speicheraufwand für die gängigen analytischen Modelle erheblich, die wie zum Beispiel der Kalman-Filter ausdrücklich auf dem Markow-Kriterium aufbauen und daher neben dem aktuellen Zustand und dessen Kovarianz keine weiteren Daten ablegen müssen. Um veraltete Ereignisse berücksichtigen zu können, müssen diese beiden Informationen für die zulässige Zahl von Rückwärtsschritten ebenso gespeichert werden wie alle auf den Track bezogenen Ereignisse. Der Aufwand steigt rasch an, sodass ein Kriterium notwendig ist, mit dem sich ausgehend vom Alter des Ereignisses und dem zeitlichen Abstand zum Zeithorizont des Filters der Aufwand gegen den Erfolg, nämlich die verbesserte Güte der Schätzung, abwägen lässt. Dabei wird im Rahmen dieser Arbeit auf ein Verfahren aufgesetzt, das von Rhéaume und Benaskeur in [RB07] vorgestellt wurde und den großen Vorteil bietet, dass der in vergleichbaren Arbeiten große Online-Berechnungsaufwand offline ausgeführt wird. Die für die Bewertung notwendigen Kenndaten werden dann aus einer Lookup-Tabelle entnommen.

Im Kommenden soll der Ansatz kurz verdeutlicht werden. Ausgangspunkt ist dabei ein beispielhaftes lineares Modell eines Bewegungsmodells mit stochastisch wirkenden und als normal verteilt angenommenen Beschleunigungen w_k . Das Modell umfasst im Zustandsvektor die Schätzungen der Position x und der Geschwindigkeit \dot{x} . Ähnliche Modelle wurden zur Evaluation des *MOSAIC*-Frameworks in den Veröffentlichungen [ZDK11a] und [ZK09] verwendet.

$$x_{k+1} = \begin{pmatrix} 1 & t \\ 0 & 0 \end{pmatrix} x_k + \begin{pmatrix} t^2/2 \\ t \end{pmatrix} w_k \quad (4.5)$$

Wendet man nun den Kalman-Filter mit seinen zwei Bestandteilen, der Prädiktion und der Validierung, an, so kann ein Verhältnis zwischen der zu erwartenden Unsicherheit für den Positionsanteil der Schätzung ohne Einbindung des Messwertes $\sigma_{x\rho}^2$ und unter Berücksichtigung des Messwertes σ_x^2 angegeben werden mit:

$$\left(\frac{\sigma_{x\rho}^2}{\sigma_x^2} \right)_{k+1} = \left(1 + \frac{\sigma_x^2 + 2\sigma_{x\dot{x}}^2 t + \sigma_{\dot{x}\dot{x}}^2 t^2 + \frac{1}{4}\sigma_v^2 t^4}{\sigma_r^2} \right)_k \quad (4.6)$$

Diese für das oben genannte Modell charakteristische Gleichung entspricht den Erwartungen. Die aus dem vorhergehenden Berechnungszyklus k folgenden Kovarianzmatrixanteile $(\sigma_x^2, \sigma_{x\dot{x}}^2, \sigma_{\dot{x}\dot{x}}^2)_k$ und die Modellunsicherheit σ_v^2 bewirken eine potenziert steigende Unsicherheit der Positionsschätzung über der Zeit, die der Messunsicherheit σ_r^2 gegenübersteht. Dabei ist die rechte Seite in jedem Fall größer als eins und die Unsicherheit somit bei der Einbindung der Messung stets kleiner als die Unsicherheit ohne Berücksichtigung einer Messung. Die Frage ist, in welchem Maß diese Verbesserung ausfällt und ob dies den Retrodiktionsaufwand lohnt. Zur Beantwortung dieser Frage kann ausgehend vom Verhältnis der Modellunsicherheit und der Messunsicherheit σ_v^2/σ_r^2 eine Tabelle erstellt werden, die im Hinblick auf verschiedene Zeitschritte t das Verbesserungspotenzial erfasst. Diese Tabelle wird für ein Modell anhand der charakteristischen Gleichung sowie der Modell- und der Sensorparameter offline bestimmt. Damit ist eine objektive Einschätzung der Auswirkung der Einbeziehung eines veralteten Datensatzes möglich.

Unsicherheitsniveau des Messdatensatzes Für die Selektion anhand der Unsicherheit des Datensatzes kann zum einen ein einfacher Schwellwert benutzt werden, der Ereignisse, deren Messdaten ein Signal-Rauschverhältnis übersteigen, von der weiteren Verarbeitung ausschließen. Da das Rauschmodell eines Sensors Bestandteil des Ereignisdatenblattes ist, kann die Konfiguration dieses Filters auch zur Laufzeit geschehen.

Im Weiteren kann die Entscheidung über die Nichtbeachtung eines Ereignisses analog zu der im Absatz zuvor beschriebenen analytischen Herangehensweise umgesetzt werden. In diesem Fall wird die Look-up-Tabelle um eine Dimension erweitert, sodass nun neben der Zeit auch das Rauschniveau der Sensormessdaten übergeben wird. Das Ergebnis, die potenzielle Verbesserung der Schätzung, kann dann benutzt werden, um zwischen Datensätzen mit unterschiedlichem Alter und Rauschniveau eine optimale Auswahl zu treffen.

Validität Die Vertrauenswürdigkeit eines Ereignisses ist, wie in Abschnitt 4.2 beschrieben, von zwei Größen abhängig: der Systemvalidität und der Ereignisvalidität. Eine hohe Systemvalidität des ein Datensatz produzierenden Sensors garantiert eine hohe Ereignisvalidität. In einem Szenario, das die sichere Umgebungserfassung eines Roboterarmmanipulators untersucht [ZDK11b], ist als ein Ansatz der Verschmelzung der beiden Werte eine multiplikative Verknüpfung vorgenommen worden. Für andere Szenarien sind aber auch individuelle Filterungen oder andere Abbildungskonzepte vorstellbar.

Ausführungsumgebung

Die Selektion der Datensätze kann innerhalb eines *Smart-X*-Netzwerkes unterschiedlich implementiert werden: zum einen im Sensorknoten selbst und zum anderen im Fusionsknoten. Beide Ansätze bringen Vor- und Nachteile mit sich.

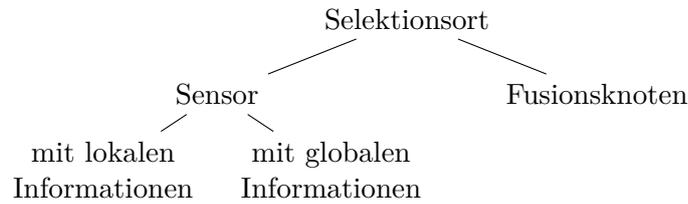


Abbildung 4.18: Systematik der Einbeziehung von Datensätzen in Abhängigkeit von der physikalischen Einheit des Messwertes

Die unmittelbarste Form der Selektion im Sensorknoten geschieht über die Periodendauer von dessen Anwendung und die Häufigkeit des Nachrichtenversands. Ein permanent Nachrichten publizierender Knoten belastet das Netz unnötig, folglich sollte die Wiederholrate auf die Dynamik des beobachteten Systems abgestimmt sein und dem Nyquist-Theorem folgen. Weitere Kriterien, anhand derer die Verbreitung ungeeigneter Messdaten gesteuert werden kann, sind die Validität und die Unsicherheit einer Messung. Unter der Maßgabe, dass in einem dynamischen *Smart-X*-Netzwerk möglicherweise leistungsstarke Knoten auch stark fehlerbehaftete Messungen für die Weiterverarbeitung nutzen können, sollte eine statische Reduzierung der versendeten Messdaten auf der Seite der Sensorknoten behutsam eingesetzt werden.

Demgegenüber wurde in [ZD10] eine dynamische Form der sensorseitigen Filterung vorgestellt. Die auf dem *Smart-X*-Sensor laufende Applikation wurde um ein Sentient Object ergänzt, das die Validierung der Messwerte anhand eines über das Netzwerk empfangenen, fusionierten Datensatzes übernimmt. Damit lassen sich zwei Szenarien umsetzen. Im ersten Fall wird über das Netzwerk ein fusionierter Wert mit einer höheren Validität als der Messwert selbst empfangen. Damit kann im Sensorknoten die Messung auf unzulässige Abweichungen und damit Fehler geprüft werden. Als fehlerhaft eingestufte Messungen werden nicht versandt. Grundsätzlich ist es aber auch möglich, diesen Mechanismus für eine softwareseitige Sensorsteuerung zu nutzen. Wie in [ZD10] angedeutet, kann mit einem entsprechenden, als Kriterium dienenden Datensatz eine Region of Interest (ROI) festgelegt werden. Dafür können virtuelle Daten erzeugt werden, um mit dieser Definition von außen sicherzustellen, dass nur die diesen Messbereich betreffenden Messdaten eines *Smart-X*-Sensors eine akzeptable Validitätsbewertung erhalten. Bei der Beurteilung der Frage, ob ein Messwert mit dem vorgegebenen Datensatz korrespondiert, ist wiederum das Rauschverhalten der Messung zu berücksichtigen.

Die Vorteile der sensorseitigen Filterung gegenüber einer Differenzierung im Fusionsknoten lassen sich wie folgt gliedern:

- Mit der Fehlerdetektion unmittelbar im *Smart-X*-Sensor ist eine konsistente Beurteilung der Messdaten in der gesamten Applikation gewährleistet. Durch die

„Zentralisation“ führt nicht jede *Smart-X*-Komponenten eine eigene Detektion mit wohlmöglich unterschiedlichem Ergebnis durch.

- Durch die Einbeziehung von validen externen Einschätzungen eines Systems reduziert sich in der Fehlerdetektion und Filterung der Messdaten eines Sensors die Initialisierungs- und Einschwingzeit. Ist beispielsweise zur Glättung der Messwerte des Sensors ein Kalman-Filter implementiert, so kann als inertielle Schätzung der Fusionswert übernommen werden. Das Sensorsystem ist mit seinen Glättungsmechanismen unmittelbar einsatzbereit.
- Auf der Basis der höhervaliden Ergebnisse einer nachgeordneten Verarbeitung lassen sich die Einflüsse der Umgebung wie Temperaturschwankungen, Fremdlicht usw. detektieren und reduzieren.
- Durch die genannten Mechanismen werden vermeintlich fehlerhafte oder außerhalb einer ROI liegende Messwerte nicht im Netz verbreitet und somit die Bandbreite und die Ressourcen der anderen Knoten geschont.

Die Möglichkeiten der sensorlokalen Fehlerdetektion sind allerdings durch die beschränkte Rechenkapazität der Sensorknoten limitiert. Komplexe Fehlertoleranzmethoden lassen sich nur auf leistungsstärkeren *Smart-X*-Fusionsknoten umsetzen. An dieser Stelle wird die Notwendigkeit der aktiven Datenaggregation deutlich. Ein Wert mit höherer Validität wird wegen der gegenüber dem Ausgangssatzen geänderten Attribute unter einem anderen Thema im Netz verbreitet. Ein verarbeitender *Smart-X*, der von der Integration und den Ergebnissen der Fehlerdetektion des zusätzlichen Knotens profitieren soll, muss seine Datenerfassung dieser Veränderung anpassen können.

Mechanismen

Die denkbaren Filtermechanismen von Ereignissen greifen sehr unterschiedlich in die Verarbeitungskette innerhalb eines *Smart-X* ein. Die Selektionsmethoden sind dabei in Ansätzen kategorisierbar,

- die für eine kommunikationshardwarenahe Filterung der Ereignisse sorgen,
- die auf ein individuelles Ereignis wirken und darüber entscheiden, ob dieser Datensatz in der weiteren Verarbeitung Beachtung findet,
- die eine Auswahl aus einem Satz von Ereignissen treffen und
- die keinen Ereigniszugriff, sondern die Verarbeitungskette steuern und diese gegebenenfalls nicht starten, unterbrechen usw.

Selektionsfolge in *MOSAIC*

Die Aufgabe des Selektionskonzeptes ist die organisierte Integration der verschiedenen Filtermechanismen innerhalb der Verarbeitungskette für die adaptive Verarbeitung. Dabei folgen die Überlegungen den fünf Stufen, die ein Ereignis innerhalb eines *Smart-X* durchläuft. Tabelle 4.6 illustriert diesen Pfad und bezeichnet die zugeordneten Selektionsmechanismen und -kriterien, die im Folgenden beschrieben werden.

Verarbeitungsstufe	Selektion	
	Mechanismus	Kriterien
Kom.-konfiguration.	themenbasiert	Maßeinheit der Messwerte RPN
Ereignisankunft	individuell	Alter Validität Unsicherheit Position
Verarbeitung	kontextorientiert alternativ steuernd	Systemzustand „jüngster“ „validester“ Anzahl der Ereignisse
Versand	verbreitungsorientiert	Validität

Tabelle 4.6: Selektionsstufen innerhalb einer *Smart-X*-Komponente

Themenbasiert Ausgangspunkt dieser Selektionskette ist die adaptive Erfassung der Ereignisse über die Netzwerkschnittstellen. Basierend auf den Informationen des Ereignisdatenblattes erfolgt bei deren Konfiguration eine erste Differenzierung. Unter Ausnutzung der kommunikationsinhärenten Filter werden zunächst Datensätze nicht relevanter Themen von der weiteren Verarbeitung ausgeschlossen. Die Festlegung der Themen erfolgt in passiven *Smart-X*-Komponenten zur Entwicklungszeit, in aktiven zur Laufzeit. Für letztere Variante ist es somit notwendig, permanent die Datenblätter der neu verfügbaren *Smart-X*-Sensoren zu prüfen und das Thema gegebenenfalls als applikationsrelevant zu kennzeichnen.

Im Hinblick auf die Implementierung sollte die themenbasierte Selektion soweit wie möglich auf die Kommunikationsmiddleware und die zugehörige Hardware ausgelagert werden, um diese Filterung möglichst effizient abzuwickeln und die eigentliche Anwendung vor einer Überflutung mit irrelevanten Messdaten zu bewahren.

Individuell Mit dem Eintreffen eines als relevant erkannten Datensatzes wird basierend auf dem Kriterienkatalog, der im Abschnitt 4.4.1 definiert wurde, eine Entscheidung über die weitere Verwendung getroffen. Besteht das einzelne Ereignis den Test, so wird es an die Datenhaltung übergeben. Dafür wird die attributive Selektion beim Starten mit einem festen Prüfmuster initialisiert. Ein Beispiel für eine attributive Selektionsfunktion ist die Filterung auf eine bestimmte Validität oder ein maximales Alter des Ereignisses hin.

Kontextorientiert Im Unterschied zur individuellen Filterung, die statische Filtervorgaben verwendet, nutzt die kontextorientierte Selektion veränderliche Muster. Diese Werte ergeben sich zumeist aus dem Modus des *Smart-X* oder einer Zustandsgröße. Ein Beispiel für die kontextorientierte Selektion ist die Filterung der Ereignisse anhand des Überwachungsbereiches des Sensors im Hinblick auf die gegenwärtige Positionsschätzung. Die kontextorientierte Selektion wird daher unmittelbar vor dem Beginn der Verarbeitung gestartet. Im Unterschied zur attributiven Selektion wird jeweils das Prüfkriterium als Kontextparameter beim Aufruf zusätzlich zum eigentlichen Ereignis übergeben.

Alternativ Während der Verarbeitung wird vergleichend aus mehreren Datensätzen eine Auswahl getroffen. Es werden also Anfragen wie „jüngster Datensatz“ oder nach „drei Datensätzen mit der höchsten Validität“ eingehen. Die Datensätze, die keine Verwendung finden, werden jeweils für eine vom Entwickler festzulegende Zahl von zukünftigen Fusionszyklen gespeichert und dann verworfen.

Steuernd Die verarbeitungsorientierte Selektion bewirkt indirekt eine Auswahl, wenn die Verarbeitungskette, zum Beispiel mit dem Erkennen eines schwerwiegenden Fehlers, gestoppt oder ausgehend von fehlenden Ereignissen gar nicht erst gestartet wird.

Verbreitungsbezogen Als letzte Schranke limitiert die verbreitungsbezogene Selektion den Versand eines Ereignisses im Netz. Kriterium aus dem obigen Katalog kann an dieser Stelle vorrangig eine zu geringe Validität sein, weil erst unmittelbar vor der Verbreitung das zusammenfassende Ergebnis der individuellen Fehlereinschätzungen bereitsteht.

Die Umsetzung und Konfiguration dieser Selektionskette ist im Kapitel 5.3.4 anhand eines Programmbeispiels dargestellt.

Parameterdefinition

Die vorangegangene Aufzählung macht deutlich, dass die Selektionsmethoden eine unterschiedliche Komplexität hinsichtlich der Kriterien erlauben. Attributive Filter geben lediglich das Ergebnis einer booleschen Entscheidung bezüglich der Einhaltung eines der

genannten Parameter wieder. Ähnlich einfach wird die themenorientierte Selektion parametrisiert: Entweder erfolgt die Vorgabe eines konkreten Subjekts einer passiven Komponente oder es wird für einen aktiven *Smart-X* eine Maßeinheit vorgeben. Die den Programmablauf steuernden Filter entscheiden demgegenüber anhand von möglicherweise sehr komplizierten Bedingungen. Während erstgenannte gut in eine abstrakte Beschreibung überführbar sind und der Fusionsaufruf damit gekapselt werden kann, lassen letztere wegen Komplexität eine solche Vereinfachung der Entwicklung nicht zu. *MOSAIC* greift diese unterschiedlichen Herangehensweisen auf und integriert vielschichtig in das Framework eingebettete, einfach zu parametrisierende themenorientierte und alternative Selektionsfilter. Für die anderen Selektionsmethoden sind in den Implementierungen Platzhalter vorgesehen, die vom Anwendungsentwickler überschrieben und mit spezifischem Code versehen werden können.

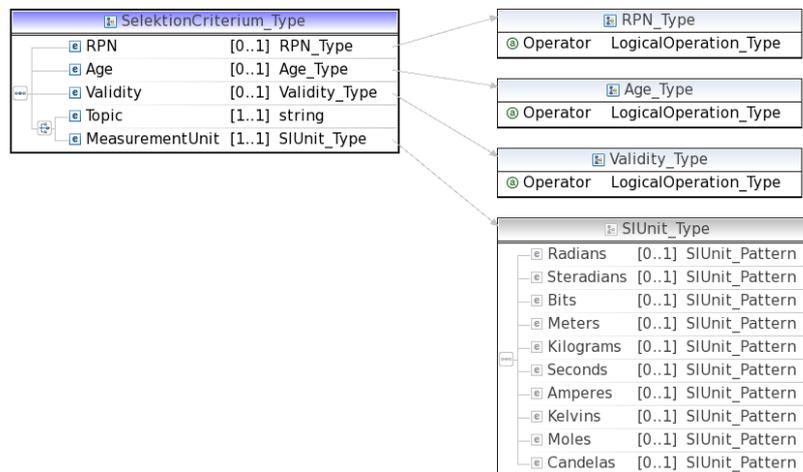


Abbildung 4.19: XML-Schema der Kriteriendefinition für die individuelle Selektion

Die zuvor erläuterte Datenblattkonfiguration aufgreifend wurde ein XML-Schema entworfen, das die Kriteriendefinition des themenorientierten und alternativen Filters übernimmt. Abbildung 4.19 illustriert die fünf Kriterien, wobei hinsichtlich der themenorientierten Selektion entweder ein passiver *Smart-X* über ein Thema *Topic* oder ein aktiver Knoten über die Festlegung der physikalischen Einheit der Messgröße *MeasurementUnit* definiert werden kann. Wie noch im folgenden Kapitel detailliert beschrieben wird, umfassen die Typen für die Kriteriendefinition des RPN, Alters oder der Validitätsdefinition jeweils einen Operator sowie einen Grenzwert. Das XML-Schema erlaubt dabei eine unmittelbare Prüfung der Anwendervorgaben.

4.4.2 Adaptive Fusion

Die Selektion stellt der Fusion einen oder mehrere Ereignisspeicher bereit, die eine variable Zahl von noch nicht abgerufenen Ereignissen umfassen. Diese Speicher werden im weiteren Verlauf der Arbeit auch als Puffer bezeichnet. Die Unterschiedlichkeit der Datensätze selbst wird durch das Ereignismodell gekapselt. Mit der Konfiguration des *Smart-X* und der Selektion wird die Verschiedenartigkeit der Messdaten in den bestimmte Themen zusammenfassenden Puffern kanalisiert. Damit stehen diese für die Anwendung als virtuelle Sensoren bereit, die beispielsweise alle relevanten Distanz- oder Temperaturmessungen unabhängig von der Herkunft der Daten anbieten. Mit dieser Abstraktion werden die Sensormessungen des Knotens den netzwerkbasierten Datensätzen gleichgestellt, was eine herkunftslose Verarbeitung garantiert. Die Aufgabe der Fusion besteht darin, zunächst die Ereignisse, die in einem einzelnen Ereignisspeicher liegen, zusammenzufassen, um diese Resultate dann anschließend pufferübergreifend zu verschmelzen. Damit greifen nacheinander alle drei der von Durrant-Whyte in [DW88] unterschiedenen Fusionskonzepte:

- Bei einer *komplementären Fusion* werden die Daten von Sensoren mit unterschiedlichen Überwachungsbereichen in ein gemeinsames Weltbild übertragen, ohne selbst voneinander abhängig zu sein. Dabei soll die Vollständigkeit der Abbildung der Umgebung in einer virtuellen Repräsentation erhöht werden. Ein etabliertes Szenario für diese Fusionsvariante sind Systeme zur Kollisionsvermeidung, bei denen den einzelnen Richtungen jeweils ein Sensor zugeordnet ist. Dieser Ansatz lässt sich zwar mit der *MOSAIC*-Struktur umsetzen, ist aber im Hinblick auf die adaptive Fusion weniger interessant, da die Unabhängigkeit der Datensätze voneinander ohnehin eine flexible Integration erlaubt.
- Die *konkurrierenden Fusion* erfasst gleichzeitig mehrere Sensoren mit einem gleichen Überwachungsbereich, die dabei Messdaten gleichen Typs (Temperatur, Distanz, Beschleunigung) liefern. Das Ziel der Verknüpfung ist in diesem Fall die Steigerung der Genauigkeit des gemeinsamen Resultats. Die Abbildung eines Puffers auf einen gemeinsamen Datensatz stellt eine konkurrierende Fusion dar.
- Werden die Messdaten mehrerer Sensoren kombiniert, so lassen sich anhand von physikalischen Beziehungen Ergebnisse generieren, die andere Größen repräsentieren als die Ausgabedaten. In diesem Fall wird von einer *kooperativen Fusion* gesprochen. Ein Beispiel ist ein Stereokamerasystem, welches zwei Einzelsensoren zusammenfasst, um eine Tiefenabbildung zu gewinnen. Mit lediglich einem der beteiligten Transducer ist diese Information nicht erlangbar. Während bei diesem Beispiel geometrische Relationen benutzt werden, bedienen sich viele kooperative Fusionsanwendungen der physikalischen Beziehungen zwischen den Messgrößen, um diese zu verknüpfen. Die Algorithmen der konkurrierenden Fusion, wie die

Temperaturkalibrierung einer Ultraschallmessung aus Abbildung 4.3, erlauben die Zusammenfassung mehrerer Puffer.

Für eine kooperative Fusion wird dabei ein höherer Grad an Adaptivität verlangt als für einen konkurrierenden Ansatz. Dessen Anpassungsfähigkeit beschränkt sich auf eine variable Zahl von Datensätzen. Diese Herausforderung lässt sich mit dem bei Weitem am häufigsten gebrauchten konkurrierenden Fusionsalgorithmus, dem gewichteten Mittelwert, einfach umsetzen. Wird auf eine iterative Form dieses Verfahrens aufgebaut, so kann die Berechnung mit einer beliebigen Anzahl an Datensätzen erfolgen. Häufig wird die Berechnung aber mit dem Erreichen einer stabilen Varianz abgebrochen.

Die Adaptivität der kooperativen Fusion ist dann gefragt, wenn ein Puffer leer bleibt, eine bestimmte Eingangsgröße also nicht zur Verfügung steht. Dies kann zwei Folgen haben. In bestimmten Konstellationen, wie bei der Temperaturkalibrierung, kann das Ergebnis nur bestimmt werden, wenn alle Eingangswerte (Ultraschallmessung, Temperatur) vorliegen. Im anderen Fall kann der Ausfall durch eine Anpassung des Fusionsalgorithmus aufgefangen und trotzdem ein Ergebnis generiert werden, das aber sehr wahrscheinlich eine geringere Güte aufweist.

In der Diplomarbeit von Pfahl [Pfa11] wurden daher Möglichkeiten gesucht, den als kooperativen Fusionsansatz weit verbreitenden Kalman-Filter in diesem Sinne zur Laufzeit anpassen zu können. An dieser Stelle soll die Grundidee dieser Arbeit kurz verdeutlicht werden. Der Kalman-Filter, der seine erste Verwendung im Apollo-Programm der NASA fand, erlaubt die Fusion verrauschter Sensormessungen und nutzt dafür ein analytisches Modell des beobachteten Systems. Dabei basiert der Kalman-Filter auf der Annahme, dass sowohl das Sensorrauschen als auch die Prozessunsicherheit normal verteilt sind. Zudem geht die ursprüngliche Form des Algorithmus von einem linearen Systemmodell aus. Kollidieren bestimmte Anwendungsspezifikationen, wie im Kapitel 6 anhand einer Roboterlokalisierung gezeigt, mit diesen Voraussetzungen, ist der Einsatz spezieller Variante des Filters zum Beispiel der Extended Kalman-Filter (EKF), Unscented Kalman-Filter (UKF), zu prüfen. Der Kalman-Filter wird unter der Annahme des Markow-Kriteriums rekursiv angewendet und ist damit auch, wie in [Pfa11] anhand einer detaillierten Analyse gezeigt, für eingebettete Geräte praktikabel.

Ausgangspunkt ist die Zustandsgleichung, die entsprechend Gleichung 4.7 aufgebaut ist und den Übergang von einem Zustand zum nächsten beschreibt. Der Zustandsvektor x_{k-1} beschreibt durch eine bestimmte Anzahl von Variablen (z. B. x/y-Position, Richtung, Geschwindigkeit) den Systemzustand und wird durch die Transitionsmatrix A und den Eingangsvektor u_{k-1} mit dem Eingangsmodell B manipuliert. Für das Modell wird ein normal verteiltes Systemrauschen angenommen, das der Vektor w_{k-1} repräsentiert.

$$x_k = A \cdot x_{k-1} + B \cdot u_{k-1} + w_{k-1} \quad (4.7)$$

Mit dem Systemmodell lässt sich eine Schätzung des Systemzustandes für einen beliebigen Zeitpunkt bestimmen. Um diese Modellannahmen zu verifizieren, werden diese mit

entsprechenden Messwerten verglichen. Die Abbildungsvorschrift geht aus Gleichung 4.8 hervor. Die Sensorwerte z_k werden mit der Messmatrix H auf den Zustandsvektor überführt. Es wird davon ausgegangen, dass die Sensordaten mit einem stochastischen Anteil v_k beaufschlagt sind.

$$z_k = H_k \cdot x_k + v_k \quad (4.8)$$

Die periodisch wiederkehrende Abarbeitung des Kalman-Filters erfolgt in zwei Phasen, der Vorhersagephase, in der eine neue Position und deren Unsicherheit geschätzt werden. In der Korrekturphase erfolgt die Zusammenführung von Apriori-Schätzungen und den Messdaten.

In [Pfa11] wurde ein Konzept vorgestellt und validiert, das die Anpassung der H -Matrix zur Laufzeit ermöglicht. Ein entsprechender Manager erfasst die Art und Anzahl der einzubindenden Sensoren und entwirft davon ausgehend die Messmatrix H für jeden Fusionsschritt neu. Dazu werden vor der Vorhersagephase die Puffer analysiert und geprüft, ob die Ausführung des Filters generell möglich ist. Darauf aufbauend wird baukastenartig die Messmatrix erstellt und die Kovarianzmatrix R der Sensorfehler aufgebaut. Das Konzept konnte nach einer Erprobung in MathWorks Simulink durch eine geschickte Implementierung auf einem 8-Bit-AVR-Mikrocontroller umgesetzt werden. Die Validierung erfolgte anhand einer Roboternavigationsaufgabe, wobei die einzelnen Sensoren zur Laufzeit an- und abgeschaltet werden konnten. Es wurde gezeigt, dass auf dem eingebetteten Mikrocontroller maximal drei Puffer bei einer Periodendauer von 150 ms verwaltet und genutzt werden konnten. Diese Zeitvorgabe ergab sich aus den Regelungsparametern der Antriebe des Roboters.

4.5 Smart-X-Architektur

Ausgehend von der Entwicklung von Konzepten zur Datenbeschreibung, Validierung sowie adaptiven Selektion und Verarbeitung stellt sich dieser Abschnitt der Aufgabe, die abgeleiteten Anforderungen in eine Architektur zu überführen.

Abbildung 4.20 illustriert den Entwurf und zeigt anhand der Orientierung von oben nach unten den Pfad eines oder mehrerer Ereignisse durch den *Smart-X*, analog zur abstrakteren Darstellung in Abbildung 4.1.1. Das dort gezeigte grob strukturierte Zusammenspiel von Inputlayer, Verarbeitungsalgorithmen, Fehlerdetektion und Outputlayer ist in Abbildung 4.20 analog präsent wurde aber ausgehend von den Ergebnissen der vorangegangenen Diskussionen weiter differenziert. Damit bildet die Darstellung der Abbildung 4.20 den Grundstock der *MOSAIC*-Architektur, die im folgenden Kapitel weiter ausgeformt und letztendlich implementiert werden kann.

Hinsichtlich der Farbgebung in der Abbildung wurden folgende Konventionen getroffen. Die grau markierten Komponenten heben interne Funktionalitäten hervor, die abgesehen von einer Parametrisierung durch den Anwendungsentwickler nicht unmittelbar

genutzt werden. Die Methoden mit einer grünen Farbgebung stellen öffentliche Funktionen dar, die vom Anwendungsentwickler überschrieben werden können. Die roten und blauen Schnittstellen sollen die Heterogenität im Hinblick auf die Netzwerk-, Sensor- und Aktortreiber illustrieren.

Ausgangspunkt der Überlegungen ist die Zielstellung, im Sinne einer Austauschbarkeit der Schnittstellen alle Informationen unabhängig von ihrer Herkunft gleich zu behandeln. Vor diesem Hintergrund wurde die Idee des Sentient Object, das abstrahierte Nachrichten über virtuelle Kanäle empfängt und verbreitet, aufgegriffen. Folglich besteht die erste Aufgabe der Datenverarbeitung innerhalb eines *Smart-X* darin, Messdaten und Nachrichten in ein das Ereignismodell widerspiegelndes Format zu übertragen, in dem die unterschiedlichen Datentypen, Dateninhalte, Attribute usw. flexibel gehandhabt werden. Diese Aufgabe wird von der Methode `Msg2Event` übernommen. Die Farbwahl in Abbildung 4.20 verdeutlicht dabei den auf der Basis der Datenblätter EDS automatisierten Übergang von heterogenen Daten der Sensor- oder Netzwerkschnittstellen auf ein in *MOSAIC* standardisiertes Ereignisformat. Das erste Element der Selektionskette aus Abschnitt 4.4.1, die themenbasierte Selektion, erfolgt für passive *Smart-X* indirekt mit der Konfiguration der Hardwareschnittstellen, für aktive aber zur Laufzeit. Die rechte Seite des Inputlayers in Abbildung 4.20 zeigt ein solches, mit gegenwärtig zwei Netzwerkinputs belegtes aktives Netzwerkinterface. Die weitere Vorverarbeitung durchlaufen alle Ereignisse gleichermaßen, wobei sich die individuelle Selektion als zweite Filterstufe unmittelbar an die Erfassung eines Ereignisses anschließt. Da sich an dieser Stelle die Filterung auf die bekannten und für jedes Ereignis verbindlichen Attribute bezieht, bietet sich hier eine automatisierte Filterung an. Der Nutzer definiert in einer abstrakten Darstellung die Kriterien `age<1s` oder `validity>0.6`, um damit die Selektionsstufe zu initialisieren. Mit dem Bestehen dieser Attributprüfung werden die Ereignisse einem Puffer zugeordnet. Nach dem Durchlaufen der vom Entwickler überschreibbaren kontextorientierten und alternativen Filter wird die gegenwärtige Relevanz eines Ereignisses geprüft und eine Auswahl aus den Puffern unabhängig voneinander für die weitere Verarbeitung ausgewählt. Damit ist die Abbildung der ursprünglich heterogenen und willkürlich verfügbaren Messdaten abgeschlossen, die (konkurrierende und/oder kombinierende) Fusion kann nun auf einen Satz von relevanten und abstrahierten Daten aufbauen.

Die Abarbeitung der Verarbeitungskette kann periodisch oder ausgehend vom Eintreffen bestimmter Datensätze gestartet werden. Erstgenannter Ansatz ist insbesondere häufig bei Regelalgorithmen der Fall. Die Berechnung beginnt dabei unabhängig von den im Datenspeicher verfügbaren Datensätzen, sodass die Verarbeitungskette an die aktuelle Situation angepasst werden muss. Im Unterschied dazu werden bei der zweiten Triggerform im Hinblick auf die verfügbaren Datensätze Muster vorgegeben. Dies kann im einfachsten Fall das Eintreffen eines einzelnen Datensatzes sein, es sind aber auch komplexere Varianten denkbar, insbesondere wenn mehrere Zustandsgrößen von einem Fusionsalgorithmus erfasst werden. Definiert beispielsweise eine Lokalisationsschätzung

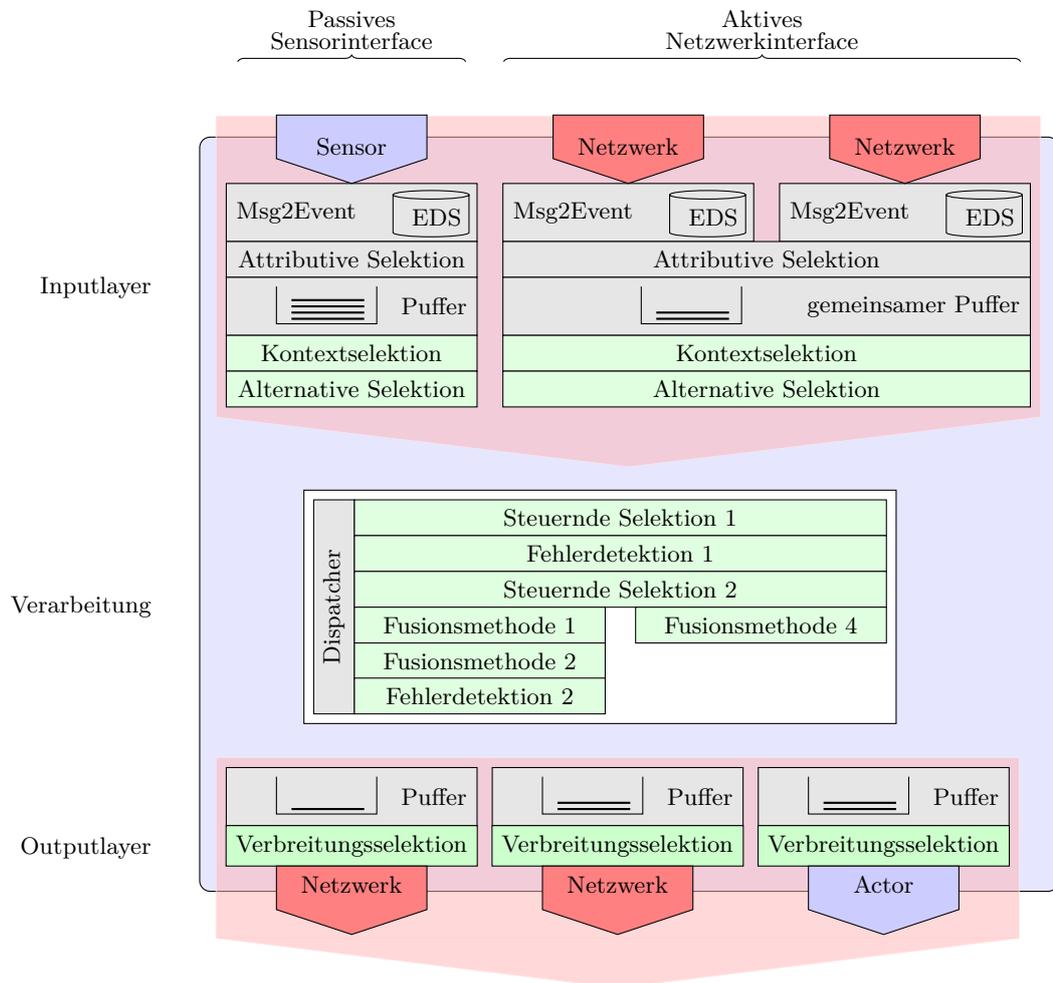


Abbildung 4.20: Architektur eines *Smart-X*, wobei die grauen Bestandteile inhärente Methoden implizieren, die vom Nutzer nur konfigurierbar sind, während die grünen Komponenten frei überschrieben werden können

einen Zustandsvektor, der Richtungs- und Positionsdaten sowie deren Ableitungen umfasst, so kann als Bedingung für die Verarbeitung eine Untermenge daraus festgelegt werden. Es genügt also, wenn im Datenspeicher Richtungs- und Geschwindigkeitsdaten vorliegen, um eine neue Schätzung zu berechnen.

Die Konfiguration dieser verarbeitungsorientierten Selektion wird vom Nutzer mit der in der Verarbeitungskette integrierbaren **Steuernden Selektion** vorgenommen. Damit können komplexe und sich unterteilende Verarbeitungsketten in *MOSAIC* codiert werden. In Abbildung 4.20 prüft die **Steuernde Selektion 1** die in den Puffern vorhan-

denen Daten und kommuniziert das Ergebnis an die Verarbeitungssteuerung, die dann gegebenenfalls die Berechnung abbricht. Abhängig vom Ergebnis der **Fehlerdetektion** entscheidet die **Steuernde Selektion 2** darüber, welche Methode – **Fusionsmethode 1** oder **Fusionsmethode 3** – als nächste aufgerufen wird. Die Ergebnisse der in der Verarbeitungskette liegenden Fehlerprüfungen werden zudem an die Verarbeitungssteuerung übermittelt, die daraus eine ganzheitliche Bewertung des aktuellen Ausgabeereignisses generiert. Diese Information wird im Ereignis abgespeichert, bevor es in den Ausgabe-puffern abgelegt wird.

Der Outputlayer weist eine ähnliche Struktur wie der Inputlayer auf, die produzierten Ereignisse werden in einem Puffer abgelegt, der deren Produktion und die Verbreitung entkoppelt. Unmittelbar vor dem Versand des aktuellsten Ereignisses aus jedem Puffer wird mit der als **Verarbeitungsselektion** bezeichneten Funktion geprüft, ob es bestimmten Qualitätseigenschaften entspricht und möglicherweise wegen einer geringen Validität von der Verbreitung abgesehen werden sollte.

Grundsätzlich können analog zum Inputlayer auch im Outputlayer mehrere Interfaces für Kanäle im Sinne des Sentient Object oder Aktoren eingebunden werden. Der Nutzer sollte dabei in angemessener Form dem Teile-und-herrsche-Grundsatz folgen und bemüht sein, *Smart-X* jeweils mit nur einem Ausgabeinterface zu entwerfen. Komplexe Anwendungen sind dann über modulare *Smart-X*-Kombinationen umsetzbar, was den Grad der Wiederverwendbarkeit erhöht.

5 Implementation

Ausgehend von der zuvor erfolgten Beschreibung der Konzepte und der Basisarchitektur des *MOSAIC*-Frameworks wird in diesem Kapitel dessen Implementierung beschrieben. Da, wie noch zu zeigen sein wird, eine vollständige Umsetzung aller Ansätze vor dem Hintergrund der Heterogenität der *Smart-X*-Komponenten bei gleichzeitiger Domainorientierung nicht praktikabel ist, wurden zwei Umsetzungsstrategien für verschiedene Programmiersprachen und Zielplattformen verfolgt.

Zum einen wurde eine Werkzeugkette für die domänenspezifische Programmierung von *Smart-X*-Sensoren unter Simulink entwickelt, wobei die modellgetriebene Anwendungsentwicklung für eingebettete Geräte im Vordergrund stand. Auf dieser Basis lassen sich nunmehr preiswerte Sensorknoten als *Smart-X*-Komponenten entwickeln und programmieren. Die adaptiven Selektions- und Fusionsansätze, die unter anderem wegen der zur Laufzeit nötigen Zugriffe auf die Inhalte der XML-Datenblätter leistungsfähigere Architekturen voraussetzen, wurden für PC-Architekturen in einer Python-Implementierung prototypisch umgesetzt. Gleichzeitig lassen sich aber mit der Python-Implementierung auch *Smart-X*-Sensoren umsetzen, wenn wie im dritten Validierungsszenario gezeigt, entsprechende PC-basierte Messsysteme zum Einsatz kommen. Die Kombination beider Implementierungen ermöglicht die rasche Umsetzung von *Smart-X*-Elementen unterschiedlicher Ausprägung.

5.1 Übergreifende Vorgaben

5.1.1 Werkzeugkette

Im Folgenden soll zunächst die generelle Werkzeugkette vorgestellt werden, die für beide Implementierungen mit unterschiedlichen Schwerpunkten und Ausprägungen als Leitlinie galt. Zur Umsetzung einer Architektur gilt es, die vielschichtigen Heterogenitäten innerhalb eines *Smart-X* in entsprechenden Abstraktionen zu kapseln. Die Informationen dazu sind für Sensoren, die Zielplattform und die Kommunikation in den Systemdatenblättern enthalten. Es gilt also dieses Wissen in eine allgemeingültige Programmstruktur zu überführen, die vom Anwender mit der eigentlichen Funktionalität zu füllen ist. Diese Transformation ist, wie Abbildung 5.1 zeigt, in einem aus fünf teilweise optionalen Schritten bestehenden Entwicklungsprozess organisiert worden. Dieser ist grundsätzlich in zwei Phasen eingeteilt, die Spezifikation und die Implementierung.

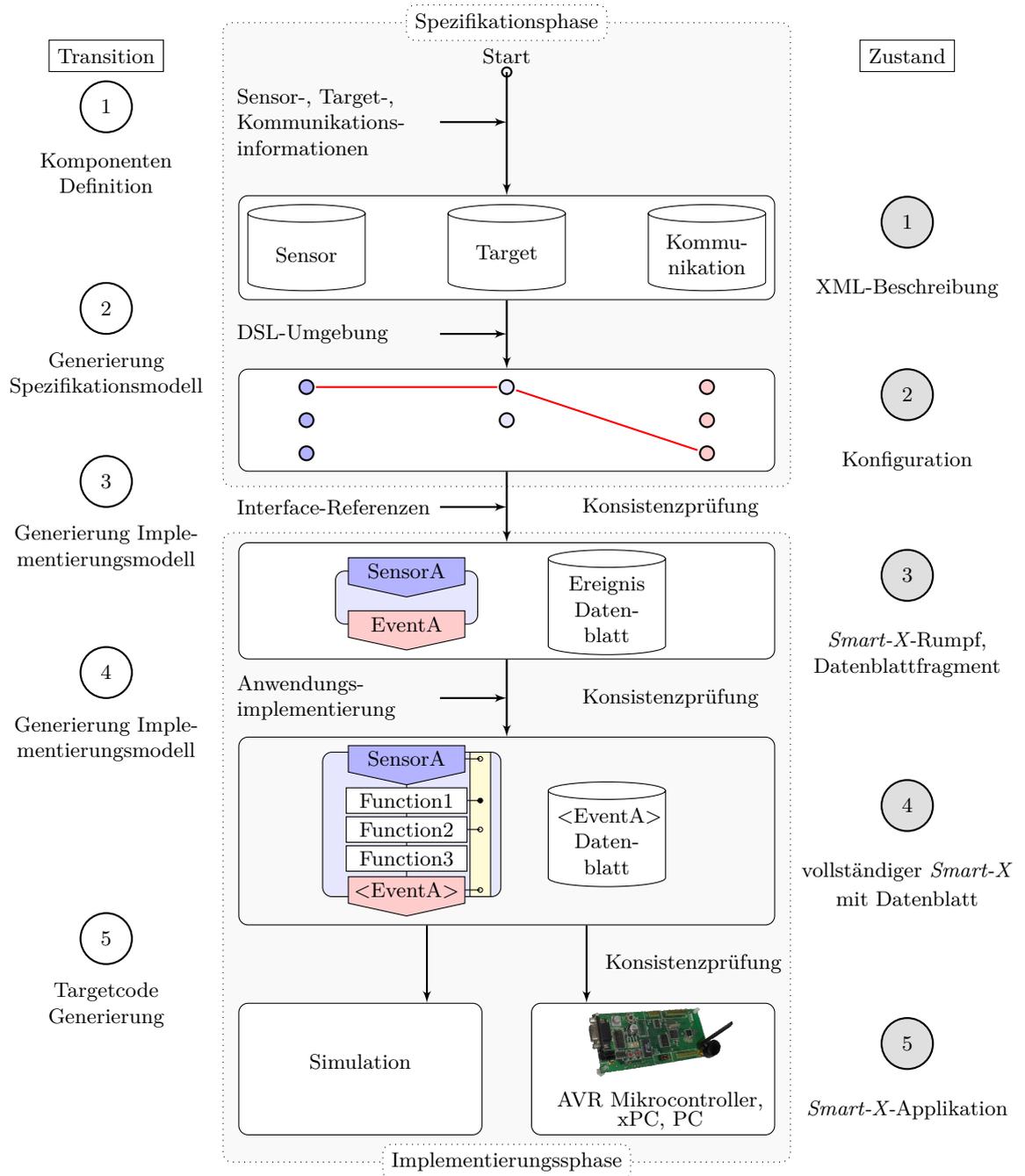


Abbildung 5.1: Struktur der Werkzeugkette mit den fünf Entwicklungszuständen (Ziffern im grauen Kreis) und der zugehörigen Fehlerprüfung während der Transition (Ziffern im weißen Kreis)

Während der Spezifikation (Zustände 1 und 2 in Abbildung 5.1) erfolgt die Konfiguration des *Smart-X* ausgehend von den Systemdatenblättern einer *Smart-X*-Komponente. Aus dem Fundus der Datenblätter kann der Nutzer des Frameworks im ersten Schritt die gewünschten Systemkomponenten zusammenstellen, in dem ein Sensordatenblatt, eine Beschreibung der Zielplattform und eine Datei mit der Definition der Kommunikationsverbindungen ausgewählt werden (Zustand 2). Im zweiten Schritt bestimmt der Anwender die individuelle Systemkonfiguration innerhalb der durch die Datenblätter vorgegebenen Schnittstellen und Parameter. Ein Sensor, der beispielsweise mit einer digitalen und einer analogen Schnittstelle ausgestattet ist, wird mit den entsprechenden Interfaces der Zielplattform verknüpft. Dabei werden die Eingaben des Nutzers auf Konsistenz geprüft und zum Beispiel erkannt, dass eine für die Zielplattform ausgewählte Kommunikationsgeschwindigkeit vom zugeordneten Sensor gar nicht unterstützt wird (Transition 3). Auf der Basis dieser Selektion erfolgt die Generierung der Rumpfimplementierung eines *Smart-X*-Sensors und ein Basisentwurf des zugehörigen Nachrichtendatenblattes. Dies enthält zu diesem Zeitpunkt die Informationen, die aus den Systemdatenblättern entsprechend der Aufstellung aus Tabelle 4.5 entnommen werden. Im Anschluss an die Implementierungsphase ist es durch den Nutzer zum Beispiel für den RPN entsprechend der Applikation zu ergänzen. Der Rahmen eines *Smart-X*, welcher im Zustand 3 bereitsteht, bindet dafür die im Datenblatt der Zielplattform beschriebenen plattformspezifischen Bibliotheken und Treiberschnittstellen ein.

In der zweiten Phase, die als Implementierung bezeichnet wird, integriert der Nutzer die Anwendungsfunktionen, wobei in beiden Implementierungsvarianten eine Vielzahl von Bibliotheken und Methoden aus der jeweiligen Programmiersprache oder Entwicklungsumgebung einbezogen werden kann. Simulink bietet zum Beispiel eine Fülle von Methoden zur Signalverarbeitung und Analyse, während unter Python unter anderem die Pakete des ROS-Frameworks bereitstehen. Unterstützend wirkt dabei die Vorgabe der internen Struktur eines *Smart-X* auf der Anwendungsebene, die die Verzahnung der Funktionsmodule und die Einbettung der Fehlerdetektionsmethoden definiert. Im Anschluss an die Implementierung können die Simulationsmöglichkeiten aus ROS/Python und Simulink genutzt werden, um die einzelnen Funktionalitäten zu erproben oder Parameter zu konfigurieren. Daneben wirken die Konsistenzprüfungen, die zum Beispiel die Vollständigkeit von Nachrichten überprüfen und somit die Einhaltung der Spezifikation des Nachrichtenformats überwachen. Die Simulationen sind dabei auch hardwareübergreifend im Sinne der Software-in-the-Loop (SiL) möglich. Um letztendlich einen auf der Zielplattform ausführbaren Programmcode zu erhalten, ist für die hochgradig abstrahierende, grafische Programmierung aus Simulink ein weiterer Codegenerationsprozess nötig.

5.1.2 Kommunikationsmiddleware

Zur Umsetzung der im Anforderungskatalog definierten Erwartungen an die Kommunikationsinfrastruktur wurden zwei Kommunikationsmiddlewareimplementierungen mit ihren Schnittstellen in das *MOSAIC*-Framework integriert: die in der Arbeitsgruppe EOS der Universität Magdeburg entwickelte Family of Adaptive Middleware for autonomOUS Sentient Objects (FAMOUSO) [Sch11b] und das bereits erwähnte Robot Operating System (ROS) [Wil]. Grundsätzlich ist das Konzept dieser Arbeit unabhängig von der zugrundeliegenden Kommunikationsspezifikation, für die Auswahl der beiden Middlewares sprachen folgende Gründe: Sowohl FAMOUSO als auch ROS folgend dem ereignisbasierten Kommunikationsparadigma und tauschen Informationen nach dem Publish/Subscribe-Modell aus. Die Entkopplung der Kontrollflussabhängigkeiten und die Abbildung der Nachrichten auf Themen (in FAMOUSO als *Subjects*, in ROS als *Topics* bezeichnet) erlauben einen dynamischen 1-zu-m Datenaustausch, der für die Umsetzung einer adaptiven Datenverarbeitung unablässig ist. Beide Middlewareimplementierungen erlauben die Anreicherung einer Messung mit entsprechenden Attributen. In ROS bestehen dafür komplexe Datentypen, die entweder als Standarddatentypen hinterlegt oder vom Nutzer in einer in Abschnitt 3.3.6 vorgestellten Sprache individuell definiert werden. FAMOUSO bietet dafür ein Attributeframework, das neben dem effizienten Austausch der Informationen zusätzlich Filterfunktionen und Kontrollmechanismen anbietet [SF11]. Der Grund für die duale Integration einer Kommunikationsmiddleware in *MOSAIC* war die unterschiedliche Abdeckung diverser Kommunikationsnetze und Zielplattformen. Während ROS auf leistungsfähige Architekturen zugeschnitten ist, zielt FAMOUSO auf eine übergreifende Integration ab und kann auf 8-Bit-Systemen genauso ausgeführt werden, wie auf PCs. Daneben unterstützt es CAN, UDP-Multicast, ZigBee und AWDS als Kommunikationsnetze, während ROS originär ethernet-basierte Kommunikation bedient. FAMOUSO unterstützt Echtzeitkommunikation und bietet damit die Grundlage für die Umsetzung zeitlich zwingend deterministischer Regelkreise, wie sie im Motivationskapitel beschrieben wurden. Die Vorteile von ROS liegen im Bereich der Anwendungsentwicklung. So werden zum einen vielfältige Werkzeuge für die Visualisierung, Datenaufzeichnung und Simulation bereitgestellt. Zum anderen sind für vielfältige Anforderungen und Anwendungen im Sensor- und Robotikbereich bereits entsprechende Bibliotheken verfügbar.

Die Verknüpfung der beiden Ansätze erfolgte im Rahmen des JANUS-Projektes, das einen Austausch von Formaten und Kanalbindungen von FAMOUSO nach ROS und umgekehrt umsetzt. Besonderes Augenmerk galt dabei der adaptiven Datenaggregation, die zur Laufzeit die Integration neuer Knoten registriert und deren Informationen verbreitet. Für diese Aufgabe wurde ein entsprechendes Gateway entwickelt, das anhand der *MOSAIC* -Ereignisdatenblätter automatisiert Publisher im jeweils anderen Netz erzeugt. Das JANUS-Projekt ist zwischenzeitlich als ROS-Package registriert und als Open Source auf der Projektwebsite [CJ11] verfügbar.

5.2 Umsetzung in Simulink

Das Matlab/Simulink-Paket der Firma MathWorks stellt in vielen Bereichen ingenieurwissenschaftlichen Arbeitens, dabei insbesondere im Bereich der Regelungstechnik, einen Standard da. Grundlage dafür ist eine vielfältige Landschaft von Erweiterungen und Toolboxen zum Beispiel für Bildverarbeitungsprojekte, maschinelles Lernen oder numerisches Rechnen, aber auch Schnittstellen zu Messkarten und Bussystemen. Mithilfe von speziellen Toolboxen eignet sich Matlab/Simulink auch für die Softwareentwicklung von eingebetteter Hardware.

Die auf einem hohen Abstraktionsgrad beruhende Programmierung, die Vielzahl der domänenspezifischen Bibliotheken und die bereits integrierten Werkzeuge für die Softwareentwicklung für eingebettete Geräte machten diese Programmiersprache für eine *MOSAIC*-Umsetzung interessant.

5.2.1 Matlab/Simulink

Matlab basiert auf der Skriptsprache „m“, die eine imperative und objektorientierte Programmierung unterstützt, wobei die Funktionalität auch mittels verschiedener Interfaces um C- oder FORTRAN-Module erweitert werden kann. Neben dieser Kombination verschiedener Sprachen stellt Matlab zahlreiche spezialisierten Erweiterungen zur Verfügung und bietet gleichzeitig eine integrierte Entwicklungsumgebung mit einer Vielzahl von programmierbaren Tools zum Profiling, zur Datenanalyse und Visualisierung [The10].

Aufbauend auf Matlab bildet Simulink eine Plattform für die modellbasierte Simulation dynamischer Systeme. Diese können im Hinblick auf Zeit und Werte ein kontinuierliches, diskretes oder hybrides Verhalten zeigen. Die Programmierung erfolgt dabei, grafisch über Blöcke mit Verbindungspfeilen für den Datenaustausch. Diese repräsentieren bestimmte Funktionalitäten oder Gerätetreiber. Dabei können mit den Blöcken hierarchische Strukturen zur Organisation und Kapselung komplexen Verhaltens auf bis zu 64 Ebenen umgesetzt werden. Zur Umsetzung eines bestimmten Modellverhaltens können zum einen vordefinierte Blöcke der vorhandenen Standardbibliothek oder aber spezifischer Toolboxen entnommen. Ein individuelles Verhalten lässt sich über eingebetteten Matlabcode oder sogenannten „S-Functions“, die in m, C, C++, Ada oder FORTRAN programmiert sein können, hinterlegen. Mit Simulink wird der Nutzer zwar auf bestimmte Paradigmen eingeschränkt, gleichzeitig abstrahiert die grafische Programmierung zum Beispiel Hardwareschnittstellen, Schedulingdetails und Simulationsparameter [The11]. Damit vereinfacht sich die Konfiguration der Ausführungsumgebung und der Schnittstellen, was die Anwendungsentwicklung in den Vordergrund hebt.

Der Real-Time Workshop (RTW) erlaubt die Generation von C/C++ Programmcode aus den Simulink-Modellen [Mat10a]. Dieser Code und folglich auch der Codegenerationsprozess können auf verschiedene Zielplattformen zugeschnitten sein. Der automatisierte Codegenerationsprozess und Kompilierungsvorgang wird über eigene Skripte

gesteuert und kann damit sehr flexibel gestaltet werden. Diese Individualisierbarkeit wird mit dem Embedded Real-Time Workshop (E-RTW) zusätzlich erweitert [Mat10b]. Der E-RTW zielt dabei auf die Generation von besonders kompaktem und performantem Programmcode für ressourcenbeschränkte Systeme ab. Allerdings ist die Zahl der standardmäßig in vollem Umfang unterstützten 8-Bit-Mikrocontroller gering und eine individuelle Erstellung der Werkzeugkette nötig. An dieser Stelle wird für diese Arbeit auf ein vorhergehendes Projekt zur Code-Generation zurückgegriffen [Bra09].

5.2.2 Werkzeugkette in Simulink

Die Implementierung der Werkzeugkette für die Erstellung von *Smart-X*-Sensoren in Simulink basiert auf dem SadaS-Projekt [Bra11] und wurde in [Bra+10] beschrieben. Grundlage der Implementierung ist ein Komponentenmodell, mit dem sich alle Bestandteile des *Smart-X*-Sensors, wie Eingabe, Signalverarbeitung, Verarbeitung und Ausgabe, beschreiben lassen. Eine Komponentendefinition umfasst dabei sowohl die spezifischen Schnittstellen als auch nichtfunktionale und funktionale Eigenschaften.

- Jede Schnittstellendefinition konfiguriert die Interaktionsmöglichkeiten mit anderen Komponenten und umfasst dabei die Art und Parametrisierung der Schnittstelle sowie die Beschreibung der auszutauschenden Daten.
- Nicht-funktionale Charakteristika legen den Typ der Schnittstellen, minimale und maximale Werte usw. fest.
- Funktionale Charakteristika ergeben sich aus dem Verhalten einer Komponente, sprich der Transformation zwischen Eingaben und Ausgaben. Dafür kann entweder eine individuelle Funktion, ein Simulink-Modell oder ein Standard-Simulink-Block referenziert werden.

Das Komponentenmodell definiert eine Abstraktion, die zum einen die Heterogenität der Sensorik und Zielplattform kapselt und gleichzeitig die Datenhaltung für die Modell- und Codegenerierung gewährleistet.

Spezifikationsphase

In der zweiten Transition in Abbildung 5.1 erzeugt der Modellgenerator ein Simulink-Modell, das der Systemspezifikation dient. Mit dieser Ausnutzung der domänenspezifischen Umgebung für die Systemdefinition geht das SadaS-Projekt über bestehende Konzepte für die Entwicklung in Simulink hinaus. Ausgehend vom Laden der Datenblätter der Sensorik, Zielplattform und Netzwerkschnittstelle bestimmt der Entwickler die für die Anwendung relevanten Komponenten und kombiniert diese. Die Daten werden mittels XML-Parser aus den Datenblättern eingelesen und in eine simulinkspezifische Darstellung überführt, sodass die Spezifikation in der für den Anwendungsentwickler bekannten

Umgebung stattfindet und nicht auf ein externes Werkzeug verlagert wird. Darauf aufbauend erfolgt die Generation und Integration von Submodellen in das Simulink-Modell, wie in Abbildung 5.2 illustriert. Mit der Abbildung als Submodell wird die Konfiguration der Komponente für den Anwender in eine grafische Repräsentation überführt. Die nicht-funktionale Definition des Schnittstellentyps ist durch die Lage am Block erkennbar, Eingänge sind links und Ausgänge rechts am Modell angeordnet. Die im Datenblatt vorgesehenen Schnittstellenparameter können über ein grafisches Interface, das jedem Knoten zugeordnet ist, eingesehen und konfiguriert werden. Die Datentypen der auszutauschenden Informationen sind als interne *structs* in der Blockdarstellung gekapselt und entstammen der Attributbeschreibung des XML-Datenblattes.

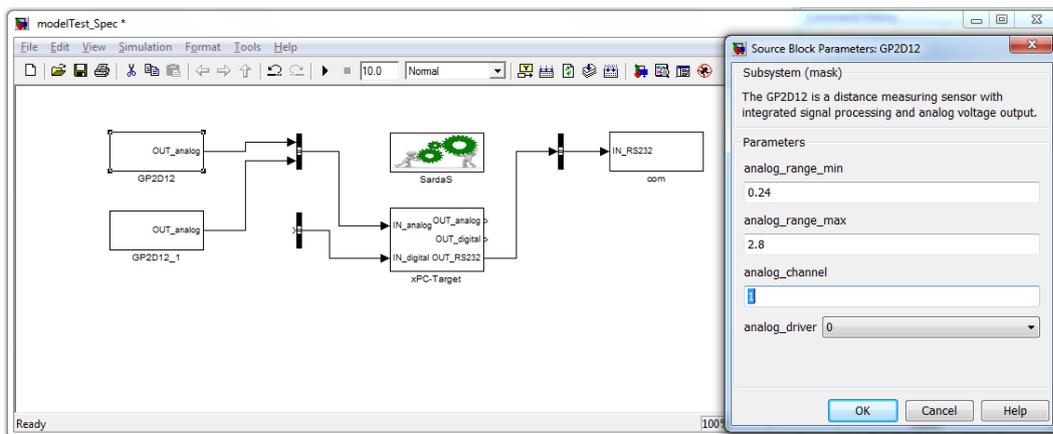


Abbildung 5.2: Struktur eines Spezifikationsmodells in Simulink

Ausgehend von dieser Darstellungsform nutzt der Anwendungsentwickler nun die simulinkinhärenten Programmiermittel, die grafische Verbindung der Aus- und Eingänge, um deren Interaktion zu definieren und damit einen individuellen Knoten zu parametrisieren. Dieser umfasst wie die Notation aus Abbildung 5.3 mindestens eine Messdatenerfassung, eine Verarbeitungs- und Kommunikationskomponente, kann aber auch aus komplexeren Verarbeitungsketten aufgebaut sein. Neben den eigentlichen Verbindungen werden die Schnittstellenparameter der ausgewählten Ein- und Ausgänge im Rahmen der durch die Datenblätter vorgegebenen Auswahlmöglichkeiten konfiguriert. Am Ende dieses Vorganges ist im Schema der Abbildung 5.1 der Zustand 2 erreicht.

In Vorbereitung der Modellierungsphase erfolgt mit der Erzeugung des *Smart-X*-Rumpfes eine erste Konsistenzprüfung. Entsprechend der vollständigen Umsetzung der Spezifikation in der Simulink-Entwicklungsumgebung wird diese wie eine Simulation gestartet. Die zugehörigen Callbackmechanismen der Submodelle werden dann für die Prüfung herangezogen. Diese wird auf zwei Ebenen durchgeführt. Zum einen wird untersucht, ob die spezifizierte Komposition eines *Smart-X* jeweils mindestens eine Instanz

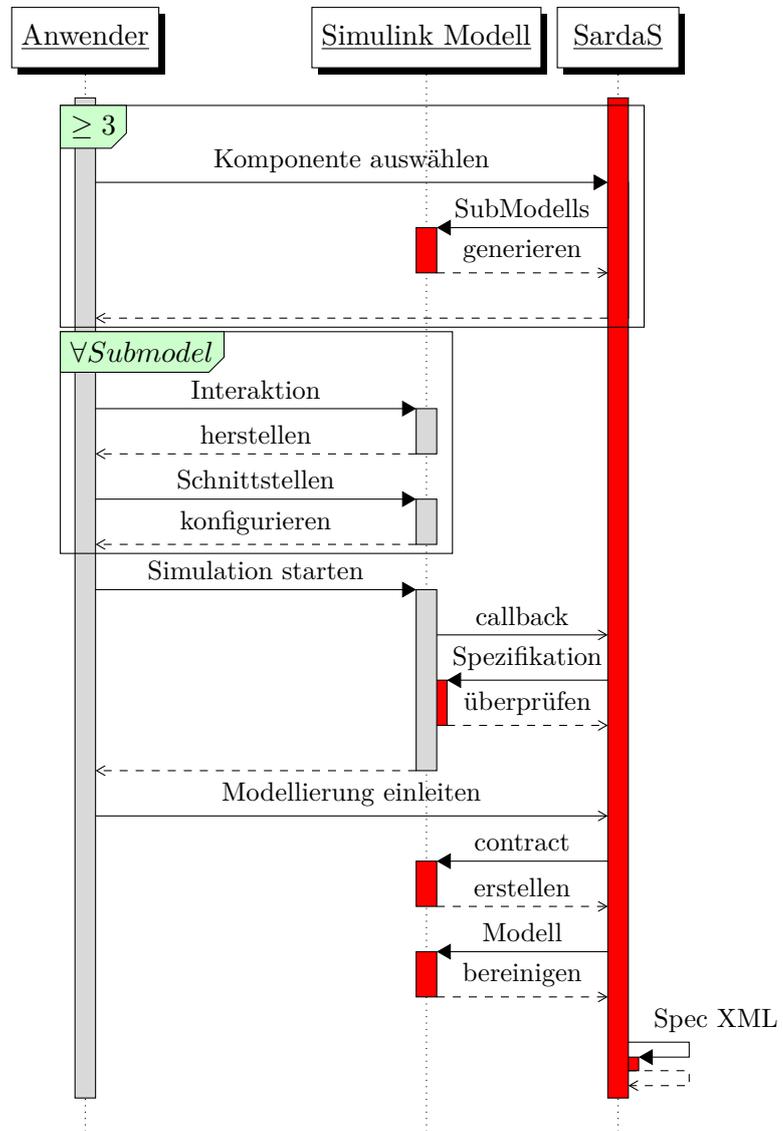


Abbildung 5.3: Ablauf der Spezifikationsphase in Simulink aus [Bra11]

der Kernkomponenten beinhaltet. Dazu wurde in [Bra11] eine formale Grammatik entwickelt, die als Basis für eine Prüfung des Signalflusses dient. Zum anderen werden die Schnittstellendefinitionen der verbundenen Ein- und Ausgänge im Hinblick auf Datentypen und Parameter validiert.

Darauf folgend wird die Modellgeneration als zweiter Bestandteil der Transition 4 in Abbildung 5.1 vorbereitet. Als Resultat der Konsistenzprüfung steht eine Liste der gültigen Signalpfade bereit, die als sogenannter *contract* in eine XML-Spezifikationsbeschreibung überführt wird. In Kombination mit dem Simulink-Modell, das die Komponenten selbst umfasst, ist die Spezifikation damit vollständig gesichert.

Nun ändert sich der Charakter der Modelldarstellung. Während in der Spezifikationsphase die Verbindungs Pfeile eine bestimmte Schnittstelle repräsentieren, wird das Modell während der Generationsphase umstrukturiert und bereinigt. Gleichzeitig werden die funktionalen und nichtfunktionalen Charakteristika zugeordnet, sodass zum Beispiel eine einen Sensor repräsentierende Box nun einen konkreten Code kapselt, der für den Zugriff auf die reale Hardware verantwortlich ist. Die Subsysteme stellen jetzt die Schnittstellen dar, während die Verbindungs Pfeile die Datenübermittlung anzeigen. Damit wird jetzt der ursprünglichen Simulink-Programmierweise folgend die Anwendungsentwicklung begonnen.

An dieser Stelle wird der Neuerungs Wert des Implementierungskonzeptes im Hinblick auf die bestehenden Entwicklungsansätze innerhalb von Simulink deutlich. Die übliche Umsetzung einer Anwendung von Simulink beginnt mit der manuellen Zusammenstellung des Systems in Form einer Auswahl von konkreten Funktionen oder Hardware repräsentierenden Blöcken. Die teilautomatisierte Umsetzung dieses Vorganges innerhalb der dem Anwender vertrauten Umgebung reduziert durch die Konsistenzprüfungen die Wahrscheinlichkeit von Konfigurationsfehlern und steigert die Transparenz dieses Entwicklungsschritts. Insbesondere bei der Entwicklung von Anwendungen für eingebettete Systeme, die nicht durch die Standardbibliotheken von MathWorks unterstützt werden, wären die physikalischen Schnittstellen als individuell zu implementierende Funktionen zu integrieren. Deren spezifische Parameter sind dann im Code eingebettet und eine Konsistenzprüfung wegen des fehlenden Wissens um die Parameter nicht umsetzbar. Erst mit der Integration der Datenblätter und eines systematischen und übergreifenden Spezifikationsmodells werden die notwendigen Daten bereitgehalten. Damit wird eine vereinfachte Austauschbarkeit der Komponenten gleicher Schnittstellenkonfiguration möglich.

Implementierungsphase

Der nach der Generation vorliegende Rumpf eines *Smart-X* wird nun mit der eigentlichen Funktionalität versehen und zum Beispiel die Fehlerdetektion integriert. Dafür stehen nun alle Entwicklungsbibliotheken von Simulink zur Verfügung, sodass auch komplexe Algorithmen der Datenverarbeitung, Filterung und Fusion usw. durch deren Kapselung und Abbildung auf einer grafischen Repräsentation einfach per *drag and drop* eingefügt

werden können. Für aufwendige Verarbeitungsketten, die eine Vielzahl von Submodellen umfassen, können bereits bestehende Implementierungen referenziert werden, wie im Sequenzdiagramm in Abbildung 5.5 angedeutet wird.

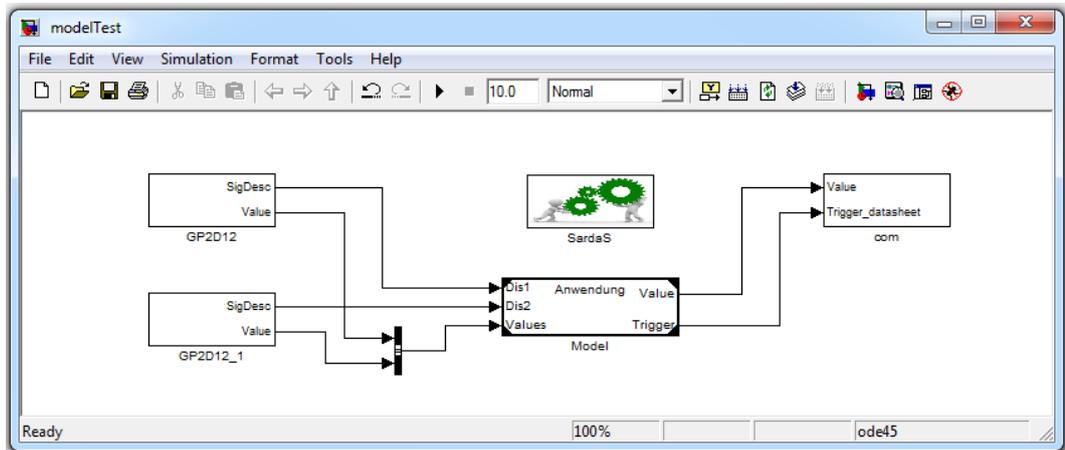


Abbildung 5.4: Struktur eines *Smart-X*-Modells in Simulink

Im Hinblick auf die weitere Entwicklung sollen noch zwei Mechanismen hervorgehoben werden, die den Entwicklungsprozess unterstützen und wichtiger Bestandteil des SardaS-Projekts waren. Um zu vermeiden, dass die zweistufige Struktur der Entwicklungskette eine strikte Trennung zwischen Spezifikation und Verhaltensimplementierung definiert, wurden sogenannte Profile etabliert. Diese können in der Implementierungsphase definiert werden und erlauben die Einbeziehung mehrere Varianten einer Spezifikation, sofern die Schnittstellendefinitionen und Parameter einen Austausch zulassen. Ein Sensor A, der einen Spannungswert produziert, kann unproblematisch gegen einen Sensor B, der gleiche Schnittstellen bietet, ausgetauscht werden. Damit kann der Hardwarezuschnitt eines Projektes flexibel variiert und umgeschaltet werden, um zum Beispiel die Auswirkung des genannten Sensorwechsels auf die Güte des Ausgabewertes zu erproben.

Simulink-Modelle lassen sich in unterschiedlichem Kontext ausführen. Üblicherweise wird eine Implementierung zunächst der Simulationsumgebung getestet, wobei der Anwender durch eine Vielzahl von Tools zum Monitoring, zur Visualisierung usw. unterstützt wird. Darauf folgend kann mittels automatisierter Generation der Code für eine Zielplattform erzeugt werden. Eine Variante stellt das xPC-Target dar. Mit dieser speziellen Simulink-Toolbox wird die Ausführung des Modells auf einem x86-Target-Computer, unter Zuhilfenahme entsprechender Treiberbibliotheken für Mess- und Kommunikationsschnittstellen möglich, wobei alle Debug- und Monitoringmöglichkeiten der Simulation erhalten bleiben. Das Modell läuft dabei unter einem Echtzeitbetriebssystem kann zur Laufzeit verändert und parametrisiert werden. Damit lassen sich mit dem xPC-Target

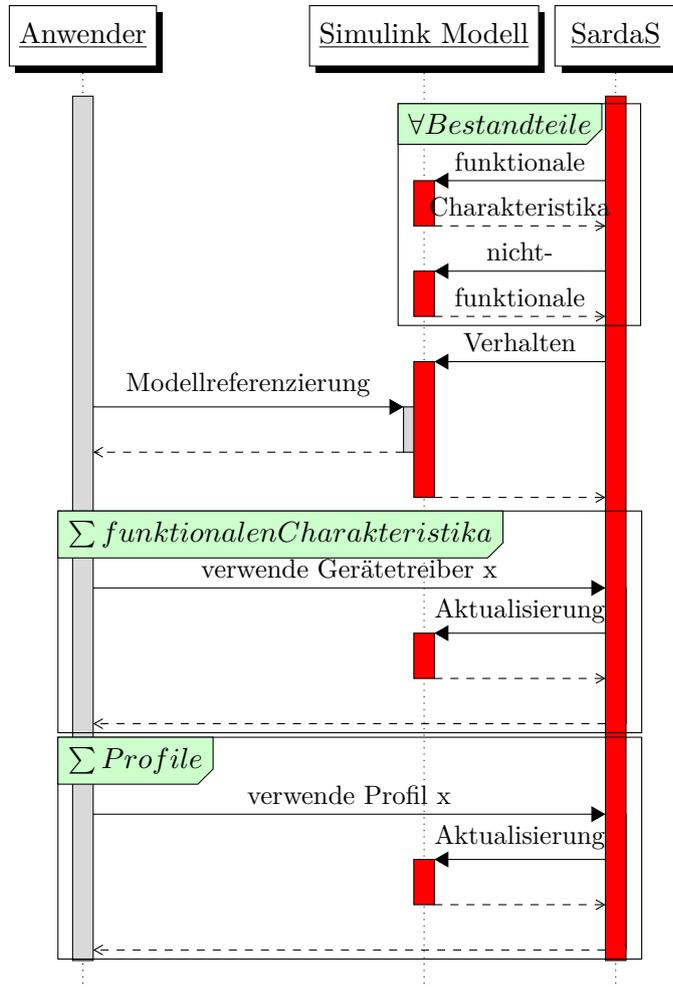


Abbildung 5.5: Ablauf der Implementierungsphase in Simulink aus [Bra11]

Steuerungs- und Regelungsanwendung komfortabel testen sowie Hardware-in-the-Loop-Szenarien rasch umsetzen. Um diese Kette – Simulation, xPC-Target und eigentliche Zielplattform – zu unterstützen, umfasst ein Submodell des SardaS-Projektes mehrere Sätze von funktionalen Eigenschaften. Für einen Sensor ist diese eine C-Funktion, die für die Nachbildung von dessen Ausgaben während einer Simulationsausführung verantwortlich ist sowie einen Gerätetreiber für das xPC-Target und die Zielplattform umfasst. Damit kann beliebig zwischen den verschiedenen Ausführungsumgebungen gewechselt werden, um deren Vorteile dem Anwendungsentwickler zugänglich zu machen.

Die Simulink-Implementierung des *MOSAIC*-Konzepts deckt die Werkzeugkette der Abbildung 5.1 komplett ab. Aufbauend auf den Systemdatenblättern unterstützt diese

einen zweistufigen Entwicklungsprozess, der konsequent die vorhandenen Hilfsmittel und Mechanismen nutzt und erweitert.

Die Validierung erfolgte wie in [Bra11] beschrieben anhand eines Lokalisationssystems für kleine Robotersysteme. Der Entwurf kombinierte acht Infrarotsensoren, um aus deren Messdaten eine Positionsschätzung zu bestimmen [Zug+11].

5.3 Python

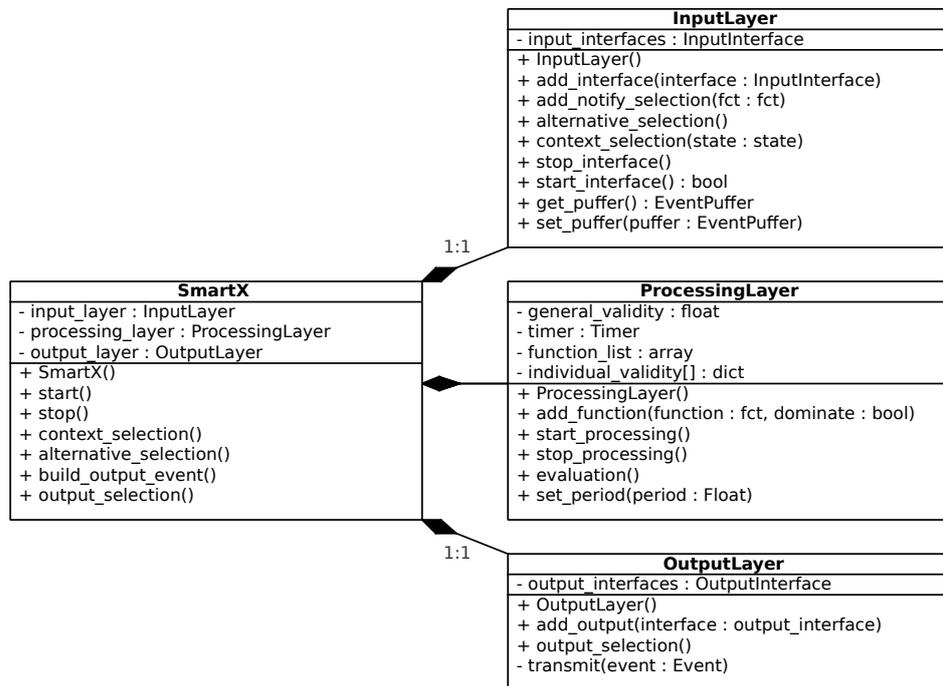
Für die Umsetzung in der Skriptsprache Python sprachen neben einer großen Zahl von Bibliotheken für die Erfassung, Verarbeitung und Analyse von Daten unterschiedlicher Form insbesondere die Verfügbarkeit von Interpretern für unterschiedliche Architekturen und Betriebssysteme. So existieren beispielsweise auch Interpreter für 8-Bit-Zielplattformen, wobei diese aber nur einen eingeschränkten Sprachumfang bedienen. Ein entscheidendes Argument war auch die Tatsache, dass der überwiegende Anteil der ROS-Packages diese Sprache unterstützt.

5.3.1 Basisstruktur

Eine *Smart-X*-Komponente lässt sich, wie Abbildung 5.6 zeigt, in drei Kernbestandteile aufteilen – einen `InputLayer`, der die Eingangsschnittstellen zusammenfasst, einen `ProcessingLayer`, in dem die eigentliche Anwendungsfunktionalität eingebracht wird und den `OutputLayer`. Dieser bindet die Ausgabeschnittstellen wie Hardwaretreiber für Netzwerke und Aktoren. Für die Umsetzung eines konkreten *Smart-X* wird eine individuelle Klasse implementiert, die von der Basisklasse `SmartX` erbt. Diese setzt die oben genannte Struktur um und definiert die Basismethoden zur Verarbeitung der Ereignisse, die in die Verarbeitungskette eingebunden sind. Für die konkrete Implementierung überschreibt der Nutzer diese und ersetzt die abstrakte Methode `context_selection` beispielsweise durch eine positionsabhängige Filterung der Ereignisse. In dem diesem Abschnitt folgenden Implementierungsbeispiel im Quelltextauszug 5.7 wird das generelle Vorgehen verdeutlicht.

Die Umsetzung in Python deckt alle Varianten von *Smart-X* ab. Als Sonderfall der Fusionsapplikationen lassen sich auch abgewandelte Typen wie virtuelle Sensoren implementieren. Häufig zum Einsatz kommende Visualisierungsknoten treten in der *Smart-X*-Lesart als Aktoren auf. Erstgenannte integrieren einen unbesetzten `Inputlayer` und produzieren Messdaten anhand einer Simulationsumgebung, die in einer Verarbeitungsfunktion gekapselt wird. Die zweite abgeleitete Variante dient der visuellen Darstellung von Daten der verteilten Anwendung, publiziert aber selbst keine Ausgaben. Entsprechend werden in diesem Fall keine Schnittstellen im `Outputlayer` eingehängt.

Die Klasse `InputLayer` integriert die eigentlichen `input_interfaces`, die über die Methode `add_interface` eingebunden werden. Ein Interface kann dabei eine Netzwerkschnittstelle oder ein Messwerterfassungssystem sein. Die über diese Schnittstellen emp-

Abbildung 5.6: Klassendiagramm eines *Smart-X*-Komponente

fangenen Messdaten oder Nachrichten werden als Ereignisse in zugehörigen Puffern abgelegt, wobei ein Puffer auch durch mehrere Interfaces befüllt werden kann. Zwar wird durch die vorliegende Umsetzung des Frameworks die Möglichkeit eingeräumt, eine Vielzahl von Interfaces in eine einzelne *Smart-X*-Implementierung zu integrieren und damit hochkomplexe Anwendungen monolithisch umzusetzen, im Sinne einer Wiederverwendbarkeit sollte aber eher eine modulare Struktur aus verschiedenen, spezialisierten *Smart-X* angestrebt werden.

Für die eventbasierte Initiierung der Abarbeitung kann der Entwickler mit `add_notify_selection` eine Methode definieren, in der die Informationen über eingetroffene Datensätze zusammenlaufen, sodass über ein Auslösen der Verarbeitung in Abhängigkeit der vorhandenen Daten entschieden werden kann. Diese als verarbeitungsorientierte Selektion bezeichnete Funktionalität lässt die Anwendung zum Beispiel darauf warten, dass eine bestimmte Anzahl von Messdaten oder ein Datensatz einer gewissen Güte eingetroffen ist. Neben der triggerspezifischen, verarbeitungsorientierten Selektion bilden die `input_interfaces` weitere drei der sechs Selektionsstufen der Ereignisverarbeitung ab, die im Konzeptkapitel beschrieben wurden. Die Konfiguration der themenbasierten und die individuelle Selektion erfolgt mittels XML-Beschreibung und wird im folgenden Abschnitt dargestellt. Die Methode `context_selection` und die `alternativ_selecti-`

on werden unmittelbar vor dem Starten der Abarbeitung ausgeführt. Die erstgenannte untersucht anhand des aktuellen Systemzustandes die Relevanz der Ereignisse, analysiert also beispielsweise den Überwachungsbereich eines Sensors im Hinblick auf die Position des Roboters. Die alternative Selektion wägt dann bestehende Ereignisse gegeneinander ab, um beispielsweise nur die jüngsten, validesten usw. im Speicher zu belassen.

Die (De-)Aktivierungsfunktionen der Schnittstellen werden in den Funktionen `start_interface` und `stop_interface` gebündelt und an die `start_smartX` Methode der `SmartX`-Klasse geknüpft.

Der `processing_layer` dient als Container der Verarbeitungsfunktionen und umfasst die für deren Ausführung notwendigen Methoden. Die Einbindung der Funktionalität erfolgt über deren Registrierung in der `function_list` mit der Methode `add_function`. Für die Steuerung der Abarbeitung stehen zwei Mechanismen bereit, die am Ende dieses Kapitels in Beispielen vorgestellt werden. Eine Variante ist es, den Steuercode unmittelbar in die Funktionalität zu integrieren. Demgegenüber bietet das Decorator-Konzept eine klare Trennung von Verarbeitung und Steuerung. Zudem lassen sich damit iterative Berechnungsmethoden im Sinne [WPI] einfach implementieren, die mit dem Erreichen eines bestimmten Güteniveaus oder nach einer bestimmten Laufzeit die Berechnung stoppen. Wiederum steht es dem Entwickler frei, die Granularität der Methoden nach seinen Vorstellungen auszulegen, wobei jede Methode aber nur eine Validitätseinschätzung liefern kann. Anhand der Eintragungen `dom` in der Übergabefunktion `add_function` wird ein Validitätsvektor erzeugt, der die Fehlereinschätzungen der einzelnen Funktionen aufnimmt. Bleibt `dom` unbesetzt, erfolgt keine Bewertung, während mit `dominate=True` eine dominante Bewertung markiert wird. Die Auswertung des Validitätsvektors und die Berechnung einer generellen Validitätseinschätzung erfolgt mit dem Aufruf der Methode `evaluation` unmittelbar nach dem Ende der Verarbeitungskette. Das Ergebnis wird im Attribut `general_validity` für eine Zeitreihenanalyse gespeichert.

Der `output_layer` weist eine ähnliche Struktur auf wie der `input_layer` und kombiniert Netzwerk und Aktorschnittstellen mit entsprechenden Puffern, die die durch die Verarbeitungskette bereitgestellten Ereignisse aufnehmen. Vor deren Übernahme in den Puffer greift die letzte Form der Selektion von Ereignissen. Der Entwickler kann hierfür mit der Funktion `output_selection` beispielsweise verhindern, dass Messdaten geringer Validität versandt werden, um Bandbreite und Energie zu sparen.

Die Pythonimplementierung bietet damit einen Bauplan für die Umsetzung einer *Smart-X*-Komponente in dem eine Zahl von Klassen und Methoden vorgegeben werden, deren Instanzen vom Nutzer zu konfigurieren und mit Verarbeitungslogik zu versehen sind. Die in Simulink im wesentlichen grafisch erfolgende Umsetzung der Spezifikationsphase erfolgt hier in textueller Form während der Instanziierung der individuellen Kindklasse der Basisklasse `SmartX`. Die Implementierungsphase erstreckt sich über die Definition der Selektionsmethoden und die Umsetzung der eigentlichen Verarbeitungskette. Die Pythonimplementierung bietet, entsprechend ihrer Ausrichtung auf Fusionsknoten, für den bereitgestellten *Smart-X*-Rumpf breitere Möglichkeiten zur Integration

von Verarbeitungs- und Selektionsmethoden als das auf eingebettete Systeme orientierte Framework für Simulink.

Eine komplette Beschreibung der Struktur des Frameworks würde die Idee einer Einführung sprengen. Im Weiteren sollen daher zwei Aspekte der Pythonimplementierung des Frameworks genauer vorgestellt werden. Zum einen die Implementierung der `input_interfaces` und der Aufbau der Verarbeitungskette als Grundlagen einer adaptiven Fusionsstrategie. Anschließend wird anhand einer Beispielimplementierung eines Knotens das generelle Vorgehen dargestellt.

5.3.2 Eingangsschnittstellen

Die Aufgabe der Klasse `InputInterface` besteht darin, die unterschiedlichen Datenquellen zu kapseln, die Informationen auf einem einheitlichen Format abzubilden und zu speichern sowie die themenorientierte und individuelle Selektion auszuführen. Die interne Struktur der Klasse ist in Abbildung 5.7 dargestellt. Die Definition und Konfiguration einer Eingangsschnittstelle erfolgt mittels Datenblättern und einer Selektionsdefinition, die ebenfalls in XML-Notation übergeben wird. Dabei werden, drei Typen differenziert, die beispielhaft in ihrer Deklaration in Quelltextauszug 5.5 dargestellt sind:

Passive Netzwerkschnittstellen greifen über eine Netzwerkschnittstelle auf einen konkreten Kanal zu, der Ereignisse eines speziellen Sensortyps verbreitet. Dieser lässt sich über ein Ereignisdatenblatt `channel_ed`s und die unterliegende Middleware `mw` bestimmen. Gegenwärtig werden in der *MOSAIC*-Implementierung ROS und FAMOUSO unterstützt. Hinzu kommen die Kriterien der individuellen Selektion, die sich auf die in Abschnitt 4.4.1 als selektionsrelevant identifizierten Attribute wie Maßeinheit, Alter, Validität, usw. beziehen. Für die Definition dieser Parameter wurde ein XML-Schema entwickelt, das es erlaubt, die Kriterien in einer XML-Datei `selection_criteria` zu erfassen. Damit kann die individuelle Filterung der eingehenden Ereignisse automatisiert umgesetzt werden. Der letzte Parameter der Schnittstellendefinition verweist auf die nach dem Eintreffen eines Ereignisses aufzurufende Funktion `startProcessing`, aus der heraus die Triggerung der Verarbeitungskette erfolgt. Damit lassen sich komplexe Anwendungen umsetzen, die zeitgesteuerte und eventbasierte Triggerung kombinieren. Beispielsweise ist es möglich, für einen bestimmten Ereignistyp zwingend die Verarbeitung auslösen zu lassen, wenn gleichzeitig aber auch sichergestellt werden soll, dass mit einer bestimmten Periode zwingend eine Ausgabe erfolgt.

Beispielhaft zur obigen Schnittstellendefinition ist im Quelltextauszug 5.2 eine Selektionsparametrisierung dargestellt. Das maximale Alter eines Datensatzes wird auf eine Sekunde beschränkt. Ein entsprechender Parser bildet diesen Wert und den zugehörigen Operator auf einer Vergleichsoperation ab, anhand der jedes eingehende Ereignis geprüft und gegebenenfalls verworfen wird.

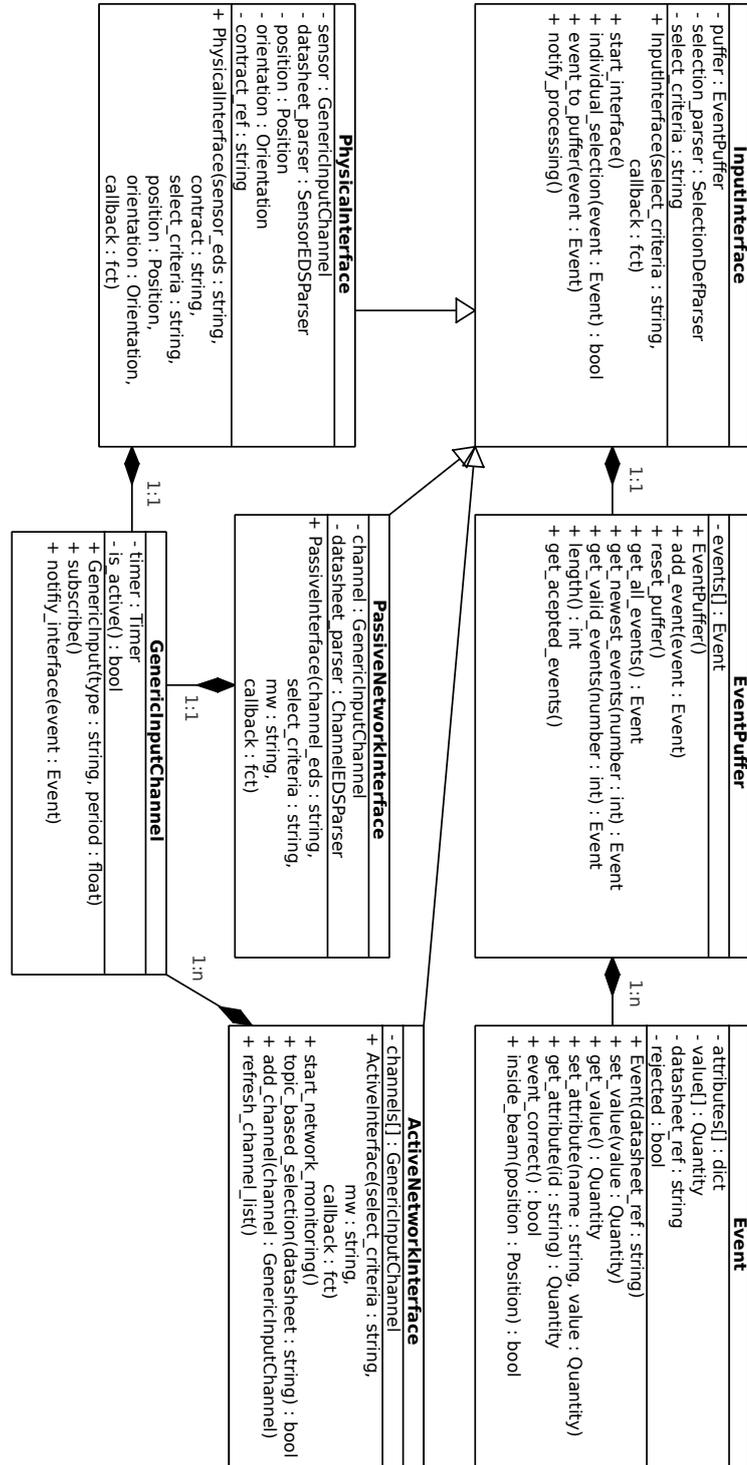


Abbildung 5.7: Klassenhierarchie der Eingangsschnittstellen

```

1 us_sensor=PassiveNetworkInterface(channel_eds="Dist_US_.xml",
2                                   select_criteria="SelectionA.sel",
3                                   mw="FAMOUSO",
4                                   callback=startProcessing)

```

Listing 5.1: Beispielhafte Definition einer passiven Netzwerkschnittstelle

```

1 <SelectionCriterium >
2   <Age Operator="<">
3     <value>1.0</value>
4     <Unit>
5       <Quantities>
6         <Seconds>1.0</Seconds>
7       </Quantities>
8     </Unit>
9   </Age>
10 </SelectionCriterium>

```

Listing 5.2: SelectionA.sel - Definition eines Kriteriums für die individuelle Selektion vor der Speicherung einer Nachricht

Aktive Netzwerkinterfaces im Sinne der Definitionen aus Kapitel 4 und Abbildung 4.1 sind durch die autonome und adaptive Datenakquise gekennzeichnet. Es wird kein konkretes Datenblatt, sondern wie Quelltextauszug 5.3 im Unterschied zu 5.1 zeigt, ausschließlich die Selektionsdefinition übergeben. Diese umfasst dann zwei Bestandteile. Zum einen werden die für die themenbasierte Selektion nötigen Informationen bereitgestellt, zum anderen aber auch die für die individuelle Selektion relevanten Kriterien benannt. Damit können, wie in Quelltextauszug 5.4 gezeigt, zunächst die bedeutsamen Kanäle identifiziert werden, um daraufhin die empfangen Ereignisse zu filtern. Zusätzlich ist für die Generierung der Netzwerkschnittstellen die Middleware anzugeben, auf die dieser Mechanismus aufbauen soll.

```

1 generic_dist=ActiveNetworkInterface(select_criteria="SelectionB.sel",
2                                    mw="ROS",
3                                    callback=[])

```

Listing 5.3: Beispielhafte Definition eines passiven Netzwerkinterfaces

Physische Schnittstellen setzen den Zugriff auf eine Messwertkarte um und werden über das Sensordatenblatt und die Konfiguration der Messkarte beschrieben. Letztgenannte bildet damit die in der Simulink-Implementierung als *contracts* bezeichnete Verknüpfung der Sensor- und Targetschnittstellen ab. So legt die Datei DAQ6402_

```

1 <SelectionCriterium >
2   <MeasurementUnit>
3     <Meters>1.0</Meters>
4   </MeasurementUnit>
5   <RPN Operator="<=">100</RPN>
6 </SelectionCriterium>

```

Listing 5.4: **SelektionB.xml** - Konfiguration eines aktiven Netzwerkinterfaces als generischen Distanzsensor (MeasurementUnit ist gleich Meter oder eine abgeleitete Einheit) mit einem RPN größer als 100 definiert

`config.xml` beispielsweise fest, welche Kanäle des ADC in welchem Modus zur Erfassung der Signale des Infrarotdistanzsensors verwendet werden. Damit wird die Systemspezifikation analog zur Simulink-Implementierung aus den Datenblättern der einzelnen Elemente entwickelt. Die Abbildung auf konkreten Schnittstellen erfolgt hier durch generische Klassen, die sich der jeweiligen Schnittstellendefinition zur Laufzeit anpassen. Da den verbreitenden Sensormesswerten eine Lokalisierung als Attribut beizufügen ist, sind die Position und die Orientierung bei der Definition mit anzugeben.

```

1 ir_sensor=PhysicalInterface ( sensor="GP120D_Datasheet.xml" ,
2                               contract="DAQ6402_config.xml" ,
3                               select_criteria = [] ,
4                               position=Position ,
5                               orientation=Orientation ,
6                               callback = [] )

```

Listing 5.5: Beispielhafte Definition eines physischen Interfaces

Die Nachrichten, die ein `PhysicalInterface`, `PassiveNetworkInterface` oder `ActiveNetworkInterface` empfängt, werden mit den Informationen des zugehörigen Datenblattes als eine Instanz der Klasse `Event` erfasst und nach der individuellen und kontextbezogenen Selektion in einem jedem Inputinterface zugeordneten Speicher abgelegt. Die Umsetzung dieses Konzeptes wird in der Klassenstruktur der `InputInterface`-Implementierung in Abbildung 5.7 deutlich. Die drei unterschiedlichen Inferfacetypen erben von einer gemeinsamen Basisklasse `InputInterface` zum einen das Handling der Ereignisse und deren Speicherung mittels Instanzen der Klassen `EventPuffer` und `Event` und zum anderen die Erfassung und Umsetzung der Selektionskriterien. Für letztere Aufgabe wird dem Konstruktor die XML-Datei `select_criteria` mit den Kriterien Selektion übergeben und während der Initialisierung über einen zugehörigen Parser `SelectionDefParser` ausgewertet. Zur Laufzeit lässt sich dann mit der Methode `individual_selection(event)` prüfen, ob ein eingetroffenes Ereignis den Anforderungen

genügt. Ist dies der Fall, wird das Ereignis mit der Methode `event_to_puffer()` im Ereignispuffer abgelegt und sofern eine Callbackfunktion definiert wurde der `ProcessingLayer` notifiziert. Die Klasse `InputInterface` ist eine abstrakte Basisklasse, die vom Anwendungsentwickler nicht unmittelbar genutzt wird.

Der Zugriff auf reale Sensoren wird durch die Klasse `PhysicalInterface` gekapselt. Sie integriert eine als `GenericInput` bezeichnete Klasse, deren Konstruktor als Fabrikmethode zur generischen Erzeugung von konkreten Klassen dient. Hinter der Instanz der Klasse `GenericInputChannel` kann sich somit eine beliebige vom Entwickler spezifizierte Inputklasse verbergen, die zum Beispiel den Zugriff auf eine DAQ-Karte, eine Netzwerkschnittstelle, eine Kamera usw. abwickelt. Die individuellen Parameter und Methoden der konkreten Schnittstelle werden durch Standardfunktionen gekapselt. Die Definition der Klasse erfolgt durch den Parameter `type` beim Aufruf des Konstruktors. Zur Überwachung der Schnittstelle kann ein Timer mit dem Aufruf des Konstruktors aktiviert werden, der die Periodizität der eingehenden Ereignisse prüft und bei deren Ausbleiben das Attribut `is_active` auf `False` setzt.

Analog zum `PhysicalInterface` der Sensoren bedient, greift das `PassiveNetworkInterface` auf eine konkrete Middlewareschnittstelle zu, die über den Parameter `mw` definiert wird. Der Kanal wird durch das Kanaldatenblatt `channel_ed` spezifiziert, für dessen Interpretation steht eine Instanz der Klasse `ChannelEDSParser` innerhalb des `PassiveNetworkInterface` bereit.

Die Klasse `ActiveNetworkInterface` benutzt, wie die Kardinalitäten zur Klasse `GenericInputChannel` zeigen, eine variable Anzahl von Kanälen, die abhängig von der Relevanz durch die kontinuierliche Überwachungsfunktion `start_network_monitoring()` aktiviert werden. Die auf die individuelle Ereignis Selektion ausgerichtete Methode `individual_selection(Event)` der Basisklasse wird durch eine weitere Funktion zur Analyse der Datenblätter ergänzt. `topic_based_selection(datasheet_ref)` signalisiert das Ergebnis des Vergleichs eines Datenblattes zum Beispiel mit den Anforderungen aus Quelltextauszug 5.4 durch einen booleschen Rückgabewert. Im Fall einer positiven Beurteilung wird mit der Methode `add_channel` ein neuer Ereigniskanal generiert und in die Liste `channels[]` eingefügt. Umgekehrt werden mit der Methode `refresh_channel_list()` diejenigen Kanäle aus der Liste entfernt, die für eine bestimmte Anzahl an Perioden ohne Ereignisempfang geblieben sind.

5.3.3 Verarbeitung

Die Umsetzung einer Verarbeitungskette lässt sich wie in der Beschreibung der Klasse `processing_layer` mit der Aneinanderreihung von Funktionsdefinitionen umsetzen. Die starre Vorgabe der Verarbeitungskette ist für einfache Messwerterfassungen mit nachgeordneter Filterung ein gangbarer Weg, für adaptive Anwendungen, die eine Verarbeitungskette in Abhängigkeit von den Daten, Validitätseinschätzungen usw. festlegen aber

nicht flexibel genug. Zudem behindert die Einbindung von Verarbeitungsfunktionalität in eine konkrete Kindklasse von `SmartX` die Wiederverwendbarkeit des Codes.

```
1 individual_validity={}  
2 puffer=[]  
3  
4 def mean_call(fn):  
5     def wrapped():  
6         fn()  
7         global puffer  
8         if len(puffer)>0:  
9             puffer=mean(puffer)  
10        else:  
11            global individual_validity  
12            individual_validity["mean"]=0  
13        return wrapped  
14  
15 def outlier_call(fn):  
16     def wrapped():  
17         fn()  
18         global individual_validity  
19         global puffer  
20         puffer = eliminate_outlier(puffer,9)  
21     return wrapped  
22  
23 def eliminate_outlier(values, limit):  
24     def f(x): return x < limit  
25     return filter(f, values)  
26  
27 def mean(values):  
28     return sum(values)/len(values)  
29  
30 @mean_call  
31 @outlier_call  
32 def start_processing():  
33     global puffer; puffer=[8,21,5]  
34  
35 start_processing()  
36 print puffer
```

Listing 5.6: Implementierung der Verarbeitungsfunktionalität als Dekorator-kette

Entsprechend wurde im Rahmen dieser Arbeit eine Möglichkeit zur Umsetzung flexibel steuerbarer Verarbeitungsketten gesucht. Wie Steup in seiner Diplomarbeit [Stel1] ausführlich anhand einer Analyse möglicher Verarbeitungsstrukturen zeigt, ist das Dekorator-muster eine geeignete Lösung für die adaptive Verknüpfung von einzelnen Verarbeitungsmodulen. Ein Dekorator erweitert die Funktionalität einer zugrundeliegenden Methode oder Klasse, ohne dass deren Code angepasst werden muss. Zudem erlaubt dieses

Entwurfsmuster die Konditionierung der Ausführung und damit die adaptive Komposition der Verarbeitungskette. Im Folgenden soll die Anwendung anhand des Quelltextauszug 5.6 illustriert werden, wobei das Beispiel aus dem *MOSAIC*-Framework zugunsten der Kompaktheit herausgelöst wurde.

Die globalen Variablen der Zeilen 1-2 dienen dem Datenaustausch während der Abarbeitung der Kette. Die Variable `individual_validity` bezeichnet die individuellen Validitätseinschätzungen der Verarbeitungsmethoden und ist außerhalb dieses vereinfachten Beispiels in die Klasse `ProcessingLayer` integriert. `puffer` umfasst die für die Verarbeitung bereitstehenden Ereignisse, im Quelltextauszug 5.6 eine einfache Liste aus skalaren Werten. Im Weiteren besteht der Beispielcode aus drei Bestandteilen, der Definition der Dekoratoren in den Zeilen 4-21, den einzubindenden Verarbeitungsfunktionen `mean()` und `eliminate_outliers()` und der durch die Dekoratoren erweiterten Basisfunktion `start_processing()`, die sonst ebenfalls in Klasse `ProcessingLayer` eingebettet ist. Letztere wird durch die Python-spezifische Dekoratornotation `@...` erweitert, sodass mit dem Aufruf der Methode `start_processing()` auch die über die Dekoratoren spezifizierte Verarbeitungskette von außen nach innen ausgeführt wird. Dazu wird den Dekoratorfunktionen jeweils die vorhergehende Funktion als Funktionsobjekt in `fn` übergeben. Im Ergebnis entsteht daraus eine Kette der Verarbeitungsfunktionen `mean(eliminate_outliers(start_processing()))`, die durch die Kontrollmechanismen der Dekoratorfunktionen gekoppelt sind. Die innere Dekoratorfunktion `outlier_call` bezieht sich mit `fn` auf die Basismethode `start_processing()`, auf deren Ausführung hin in Zeile 20 die Methode `eliminate_outliers()` gestartet wird. Diese manipuliert den Puffer und entfernt alle Werte größer als 9. Der Kette folgend wird danach geprüft, ob die Abarbeitung überhaupt fortzusetzen ist, indem die Zahl der Datensätze auf dem Puffer geprüft wird (Zeile 8). Gegeben falls wird die Funktion `mean()` aufgerufen und der Mittelwert gebildet. Wenn keine Daten mehr enthalten sind, wird die Validitätseinschätzung auf „0“ gesetzt. Dieser Parameter könnte wiederum Entscheidungsmerkmal für weitere Dekoratorfunktionen sein.

Anhand des Beispiels werden die Vorteile des Dekoratorkonzeptes, die Adaption der Verarbeitungskette zur Laufzeit, die Umsetzbarkeit von komplexen Verarbeitungsstrukturen mit mehreren Strängen und Abhängigkeiten sowie die konsequente Kapselung der Verarbeitungsfunktionen deutlich. Insbesondere im Zusammenspiel mit den listenbasierten Funktionsketten lassen sich damit komplexe und anpassungsfähige Verarbeitungsketten umsetzen.

5.3.4 Beispiel

Im Folgenden wird die Anwendung des *MOSAIC*-Frameworks unter Python anhand eines Beispiel-*Smart-X* dargestellt. Dabei soll ein aktiv wirkender Netzwerkknoten implementiert werden, der adaptiv alle relevanten Positionsmessungen erfasst, diese auf Ausreißer prüft und daraufhin periodisch einen Mittelwert der drei jüngsten Werte bestimmt. Da-

für wird die in Quelltextauszug 5.7 auszugsweise aufgeführte spezifische Kindklasse der Basisklasse `SmartX` realisiert. Der Konstruktor dieser neuen Klasse `MySmart_Position` umfasst unter anderem die Definition des entsprechenden generischen Schnittstellen `generic_position`. Diesem wird eine Selektionsbeschreibung analog zu Quelltextauszug 5.4 übergeben, die die physikalische Einheit der Messgröße der Zielstellung des Knotens entsprechend zu „Meter“ bestimmt. Mit der Integration des `generic_position` Interfaces werden der Puffer angelegt, die Überwachung der verfügbaren Kanäle gestartet und gegebenenfalls entsprechende Subscriber für das ROS-Netzwerk generiert. Mit dem Eintreffen einer Nachricht erfolgt die individuelle Filterung anhand der XML-Definitionsdatei `Position.sel` automatisch.

Die Verarbeitungskette besteht aus zwei Funktionen `outlier` und `mean`, die zur Verdeutlichung des prinzipiellen Aufbaus sehr einfach gehalten und ohne die Anwendung des Dekoratorkonzeptes implementiert wurden. Wie bereits an der fehlenden Callbackdefinition für das Interface (Zeile 4) deutlich geworden ist, arbeitet die Funktion zyklisch. Entsprechend der Zeile 9 startet die Abarbeitung der definierten Funktionen alle 100ms. Bevor allerdings diese aufgerufen werden, greifen die standardmäßigen Selektionsmethoden der kontextbasierten und alternativen Selektion. Die in der Klasse `SmartX` vorgesehenen abstrakten Methoden werden mit anwendungsspezifischen Funktionen überschrieben. `context_selection` prüft für die vorhandenen Ereignisse anhand der aktuellen Position `self.cur_pos` die Ausrichtung der zugehörigen Sensorkeulen und markiert außerhalb des Suchbereichs liegende Messungen als ungültig (Zeile 14-17). Der so bereits vorgefilterten Puffer wird mit der Methode `alternative_selection` auf die neuesten Messdaten reduziert, sofern mehr als drei Ereignisse enthalten sind. Überzählige Ereignisse werden wiederum mit `rejected=True` gekennzeichnet. Mit dem Abschluss der Vorfilterung werden die nunmehr als gültig erfassten Ereignisse in einer eigenen Liste `self.events` gespeichert und der Puffer des Inputinterfaces gelöscht (Zeile 22-23). Damit steht dieser auch parallel zur weiteren Verarbeitung wieder für die Aufnahme neuer Messwerte bereit. Die weitere Verarbeitung bezieht sich aber auf die Ereignisliste `self.events`.

Die Verarbeitung wird in der Reihenfolge der Funktionsdefinition durchgeführt. Entsprechend greift zunächst die Funktion `outlier` auf die verbliebenen Ereignisse im Puffer `self.puffer` zu und markiert signifikant abweichende Messungen als Ausreißer (Zeile 25-27 im Auszug). Für den Fall der Detektion eines fehlerhaften Wertes, werden alle Messwerte `self.events[]` mit der gegenwärtigen Schätzung `self.cur_pos` verglichen. Tritt ein Wert größer als `epsilon` auf, so wird von einem Fehler ausgegangen und die individuelle Validitätseinschätzung der Funktion `outlier` im Dictionary `self.processing_layer.individual_validity` auf den Wert „0“ gesetzt (Zeile 27). Der nachfolgende Mittelwert wird ähnlich der Bestimmung der maximalen Abweichung ermittelt und als neue Schätzung `self.cur_pos` festgehalten. Die Verarbeitungsfunktion `mean` generiert keine individuelle Validitätsschätzung, wie in der Definition der Funktion in Zeile 7 festgelegt wurde.

```

1 class MySmart_Position(SmartX):
2     def __init__(self):
3         ...
4         generic_pos=ActiveNetworkInterface("Position.sel", "ROS", [])
5         self.input_layer.add_interface(generic_pos)
6
7         self.processing_layer.add_function(self.outlier, True)
8         self.processing_layer.add_function(self.mean)
9         self.processing_layer.set_period(0.1)
10
11        mean_position=OutputInterface("meanPosition.xml", "ROS")
12        self.output_layer.add_output(mean_position)
13
14    def context_selection(self):
15        for event in self.input_layer.generic_pos.puffer.events:
16            if not event.inside_beam(self.cur_pos):
17                event.rejected=True
18
19    def alternative_selection(self):
20        if self.input_layer.generic_pos.puffer.length()>3:
21            self.input_layer.generic_pos.puffer.get_newest(3)
22            self.events=self.input_layer.generic_pos.get_accepted_events()
23            self.input_layer.generic_pos.reset_puffer()
24
25    def outlier(self):
26        if max([event.value-self.cur_pos for event in self.events])>
27            epsilon:
28                self.processing_layer.individual_validity["outlier"]=0
29
30    def mean(self):
31        self.cur_pos = sum([event.value for event in self.events])/len
32            (self.events)
33
34    def evaluation(self):
35        aux=self.processing_layer.individual_validity["outlier"]
36        if aux<1:
37            self.processing_layer.general_validity=aux
38
39    def build_output_event(self):
40        output=self.output_layer.mean_position.puffer.get_empty_event
41            ()
42        output.set_value(self.cur_pos)
43        output.set_attribute("Validity")=self.processing_layer.
44            general_validity
45        ...
46        self.output_layer.mean_position.puffer.add_event(output)
47
48    def output_selection(self):
49        for event in self.output_layer.mean_position.puffer.events:
50            if event.get_attribute("Validity")<0.5:
51                event.rejected=True

```

Listing 5.7: Implementierung eines adaptiven *Smart-X* zur Erfassung der Position

Die erste Stufe des `OutputLayers` stellt die Abbildung der Ergebnisse auf einem Ereignis dar. Dazu wird zunächst vom entsprechenden Puffer des Outputinterfaces `output_layer.mean_position.puffer` ein Ereignis angefordert (Zeile 38), um die Konsistenz der Formate sicherzustellen. Mit dem Wissen um den Aufbau des Ereignisses, die Attribute sowie deren Datentypen und Einheiten erfolgt eine kontinuierliche Prüfung der übergebenen Daten. Der Vorgang endet in Zeile 42, wenn das Ereignis an den Puffer zurückgegeben wird. Unmittelbar vor der Verbreitung besteht mit der obligatorischen letzten Selektionsstufe `output_selection` die Möglichkeit, die zu versendenden Ereignisse zu prüfen. Im Beispiel wird die Validität auf einen bestimmten Schwellwert hin untersucht, um zu vermeiden, dass unsichere Ergebnisse publiziert werden.

5.4 Zusammenfassung

In diesem Abschnitt wurden die Umsetzung des *MOSAIC*-Frameworks mittels Python und Simulink vorgestellt. Der Fokus der Implementierungen zielte jeweils auf unterschiedliche Anwendungsbeispiele ab. Während die Werkzeugkette unter Simulink auf eingebettete Sensorknoten und deren Entwicklung zugeschnitten ist, orientiert sich die Umsetzung in Python auf adaptive Fusions- und Filteranwendungen in dynamischen Umgebungen. Beide Konzepte können auf einen breiten Fundus an bestehenden Bibliotheken und Tools zurückgreifen, wobei auf der Seite der Simulink-Entwicklungsstrecke die inhärenten Methoden der Programmierumgebung eingesetzt werden, während für die Python-Implementierung im wesentlichen auf die Funktionen zur Speicherung, Analyse und Verarbeitung aus ROS aufgebaut wird.

Tabelle 5.1 stellt die durch die Implementierungen unterstützte Hardware und Middlewareinterfaces gegenüber. Die Simulink-Werkzeugkette ist auf die xPC-Umgebung und im Hinblick auf die Codegenerierung auf einen Mikrocontroller der Firma Atmel zugeschnitten. Für die sensorische Erfassung wurden bisher Schnittstellen für den ADC, das I²C-Interface und den Timer des AVR Mikrocontrollers integriert. Aus der xPC-Umgebung können Messdaten über unterschiedliche Messdatenerfassungskarten erfolgen, die zum Teil durch proprietäre Treiber unterstützt werden. Während der Validierung wurde die DAQ-Karte NI-6024, die über analoge und digitale Eingangs- und Ausgangsschnittstellen verfügt, eingesetzt. Die Python-Implementierung deckt über das ROS-Framework eine Vielzahl von robotikspezifischen Sensoren ab. Zusätzlich wurden entsprechende Inputklassen für Mikrofone, verschiedene NI-Messkarten und die mit umfangreicher Inertialsensorik ausgestatteten TI-Chronos Armbanduhr codiert.

Mit den bestehenden Implementierungen lassen sich die Konzepte von *MOSAIC* komfortabel umsetzen, wenngleich keine umfassende Programmierung über eine einzige Sprache für alle *Smart-X* möglich ist. Dieser Nachteil wird durch die uneingeschränkte Kombinationsfähigkeit von *Smart-X* unterschiedlicher Herkunft und die Umsetzung eines

Tabelle 5.1: Unterstützte Hardware und Middlewareinterfaces aus den Implementierungen des *MOSAIC*-Frameworks

	Simulink			Python
	Simulation	AVR	xPC	
Architektur	PC	AT90CAN128	PC	PC
Schnittstellen	-	ADC, I2C, Timer	NI-6024	NI-6024, NI-6032, ROS-Devices, Mikrofone, NI-Chronos
Middleware	-	FAMOUSO	FAMOUSO	FAMOUSO, ROS

übergreifenden Entwicklungsmodells aufgehoben. Das folgende Kapitel illustriert die Leistungsfähigkeit der Implementierungen anhand von Beispielszenarien.

6 Evaluation

Im folgenden Kapitel soll das Konzept der adaptiven Fusion auf der Basis des *MOSAIC*-Frameworks in roboterspezifischen Szenarien validiert werden. Ausgangspunkt ist ein Lokalisationsszenario mobiler Roboter, anhand dessen die Flexibilität der *Smart-X* dadurch gezeigt werden soll, dass sowohl die klassische, roboterzentrierte und statische Umsetzung als auch eine Implementierung mit einer dynamischen Sensorintegration vorgestellt wird. Zudem erfolgt eine Validierung der Fehlerdetektionsmechanismen in einer Simulink-basierten Simulation.

6.1 Einleitung

6.1.1 Einordnung

Die Lokalisation von Robotersystemen ist ein komplexes Problem, das sich aber mit zunehmender Rechenleistung und immer weiter verbesserter High-End-Sensorik auch in schwierigen Umgebungen meistern lässt. Diese reichen von Unterwasserapplikationen mit Tauchrobotern [Wil+00] über fliegende Systeme [CPR05] bis hin zu Bewegungsaufgaben in realistischen Alltagsumgebungen [Gro+02]. Die dabei zum Einsatz kommenden Sensoren, wie Laserscanner, (3D)-Kameras, Präzisionsgyroskope usw., garantieren in Verbindung mit entsprechend leistungsfähiger Signalverarbeitungshardware eine präzise Positionierungsaussage.

In vielen Szenarien stehen aber derlei hochgradig performante Komponenten nicht zur Verfügung. Grundlage dafür ist die Abwägung zwischen erreichbarer Genauigkeit und Güte der Positionsangabe zum erforderlichen Bauraum, notwendiger Energieversorgung und Verarbeitungskapazität sowie letztendlich den entstehenden Kosten. Sensornetze folgen diesem Ansatz konsequent und versuchen, trotz einer auf die minimale Energieaufnahme ausgerichteten, streng limitierten Hardware und Sensorik, ein übergreifendes Bild der Umgebung zu erfassen. In diesen Szenarien kann das Konzept des aktiven *Smart-X* seine Stärken ausspielen und in der Kombination aller verfügbaren Informationen unter Berücksichtigung von deren Validität eine größtmögliche Präzision erreichen.

Eine solche Anwendung ist der RoboCup-Junior Wettbewerb, ein Bestandteil des RoboCup, der in jährlichen Weltmeisterschaften ein Aushängeschild der aktuellen Roboterforschung ist. Im Gegensatz zu den professionellen und vor allem durch Universitäten getragenen Mannschaften der Midsized oder Humanoid League sind die Schülerwettbe-



Abbildung 6.1: Roboter des RoboCup Junior Turniers in einer Spielsituation, wobei einer der Roboter umgestoßen wurde. Dabei ist die häufig zum Einsatz kommende Lego NXT Box, die einen ARM7-Prozessor enthält, gut zuerkennen.

werbe im Hinblick auf die Hardware stark eingeschränkt. Diese Limitierungen betreffen zum einen aus dem Reglement heraus die Baugröße, das Gewicht usw. und aus Kostenerwägungen auch die Sensoren und Rechentechnik. Trotzdem hängt die Leistungsfähigkeit eines Roboterteams neben anderen Faktoren entscheidend von der Positioniergenauigkeit auf dem standardisierten Spielfeld ab. Diese Beschränkungen auf einfache, heterogene Sensoren, die unterschiedliche Fehlertypen aufweisen, prädestinieren diese Spielklasse als Anwendungsszenario für das *MOSAIC*-Framework.

Wie in Abbildung 6.1 dargestellt, besteht eine Mannschaft aus zwei Robotern, die darum „kämpfen“, einen ein infrarotes Signal ausstrahlenden Ball in das gegnerische Tor zu befördern. Das Spielfeld hat dabei bis auf die Ausbuchtungen der Tore eine rechteckige Form mit den Abmaßen 183x122 cm [Rob10]. Die Roboter dürfen, abhängig von der Klasse, in der sie antreten, einen Durchmesser von 22 cm und ein Gewicht von 2.5 kg nicht überschreiten. Die Bewegungsgeschwindigkeit erreichte in den letzten Jahren ein immer höheres Niveau und liegt bei etwa 30 cm/s. Die Energieversorgung ist so auszulegen, dass dieses Level über jeweils eine Halbzeit mit 10 min aufrecht erhalten werden kann. Kommunikation ist zwischen den Robotern via Bluetooth erlaubt, obwohl koordinierte Spielzüge zwischen den Robotern einer Mannschaft nur äußerst selten zu beobachten sind und eher als Zufallsprodukte gelten dürfen.

Diese fehlende Kooperation resultiert unter anderem aus dem Fehlen einer präzisen Lokalisation. Die meisten Teams unterscheiden zwischen Tormann und Stürmer, wobei Erstgenannter einen festen Bereich in Tornähe nicht verlässt und letzterer anhand einer Richtungsangabe auf das gegnerische Tor orientiert ist. Dafür arbeiten die meisten Systeme mit einer Kombination aus Infrarot- und Ultraschallsensoren, die durch die Roboterodometrie und elektronische Kompass ergänzt werden. Zusätzlich kann ein Graukeil auf dem Spielfeld oder die Verschiedenfarbigkeit der Tore zur Orientierung genutzt werden [OKM04].

Sensor			Messwert					Montage	
Herstellernfirma	Bezeichnung	Funktion	Distanz	Orientierung	Geschwindigkeit	Drehrate	Beschleunigung	Roboter	Intel. Umgebung
Sharp	GP2D120	IR-Sensor	x					x	x
Sharp	GP2Y0A	IR-Sensor	x						x
Devantech	SRF10	US-Sensor	x						x
Wenglor	YT87	Laser	x						x
Devantech	CMPS03	Kompass		x				x	
Sharp	GP1A038	Odometrie		x	x			x	
Analog Devices	ADXS150	Gyroskop				x		x	
Freescall	MMA7260	Beschl.-Sensor					x	x	
Sensirion	SHT11	Temperatur						x	
	SHT11	Feuchte						x	
Perkin Elmer	M9960	Licht						x	
Logitech	Pro9000	Webcam	x	x					x

Tabelle 6.1: Sensoren des Evaluationsszenarios

6.1.2 Verwendete Sensorik

Die Evaluationsszenarios integriert die in Tabelle 6.1 zusammengestellten und im Rahmen des RoboCups etablierten Sensortypen. Dabei können drei Kategorien unterschieden werden. Im ersten Block der Tabelle sind die für die Positionsbestimmung unmittelbar relevanten Sensoren enthalten, die in Abbildung 4.17 aufgabenbezogen als direkt bezeichnet werden. Der zweite Block umfasst die für Lokalisationsanwendungen indirekten Sensoren, die die Temperatur, Luftfeuchte usw. bestimmen, um darauf aufbauend den Einfluss dieser Umgebungsgrößen auf die eigentliche Messung zu reduzieren. Die letzte Kategorie wird allein durch die als Referenz dienende Kamera besetzt.

Für die gesamte Sensorik wurde der statische Validitätswert mit den in Abschnitt 4.2 beschriebenen Methoden bestimmt. Die Herleitung der charakteristischen Werte für den Infrarot-Distanzsensor ist in Abschnitt 4.4 aufgeführt, für die übrigen Daten ergibt sich

die Quantifizierung der Bewertung aus Schappheit [Sch11a]. Entsprechend sollen an dieser Stelle nur die Eckpunkte der FMEA-basierten Klassifikation vorgestellt werden.

Der GP2Y0A02YK Sensor [Sha01] zeigt ähnlich Fehlertypene wie der GP2D120, die jedoch insbesondere im Hinblick auf die Ausreißer und die Fremdlichtempfindlichkeit nicht so ausgeprägt sind. Folglich liegt der RPN unter dem des ersten Eintrages in der Tabelle 6.2. Die Fehlertypen des Ultraschallsensors SRF10 [Dev03] wurden im Zusammenhang mit Tabelle 4.3 diskutiert. Der RPN hängt für diesen Sensor entscheidend von der Durchführung einer Temperaturkompensation und der Ausreißerfilterung des Sensors ab. Der industriell eingesetzte Laserdistanzsensor YT87 der Firma Wenglor [Wen03] zeichnet sich durch eine hohe Verlässlichkeit aus. Lediglich spiegelnde und sehr raue Oberflächen rufen Messfehler mit geringer Amplitude hervor. Die in den Messungen des elektronischen Kompasses CMPS03 [Dev00] auftretenden Fehlertypen wurden intensiv in [Wer09] untersucht. Bedingt durch einen hohen Ausreißeranteil und Offsets, die sich aus elektromagnetischen Störungen der Umgebung ergeben, verbleibt der CMPS03 trotz der Integration eines Ausreißertests, einer Signalanalyse und eines Filters auf einem hohen Niveau. Erst durch die Kombination mit anderen Inertialsensoren, im Szenario dem Gyroskop ADXSR150 [Ana04] der Firma Analog Devices und dem Beschleunigungssensor MMA7260 [Fre08], lassen sich die Fehlereinflüsse aber nochmals deutlich reduzieren, wie zum Beispiel in [Lie08] beschrieben wurde.

Sensortyp	RPN ohne Detektion	RPN mit Detektion
GP2D120	720	72
GP2Y0A02YK	420	84
SFR08	460	40
YT87	90	18
CMPS03	720	640
GP1A038RBK	540	270
ADXSR150	490	245

Tabelle 6.2: Statische Fehlerbewertung der Sensoren des Evaluationsszenarios

Für die im Folgenden beschriebene Simulation wurden für die verwendeten Sensoren ausgehend von den Erkenntnissen der statischen Fehlerbewertung entsprechende Modelle entwickelt, die auf ein ideales Messsignal angewendet wurden.

Als Referenzmaß dient ein kamerabasiertes Erfassungssystem des Roboters, das auf der Basis der AR-Toolkit Bibliothek [Art] entwickelt wurde. Dazu trägt der Roboter, wie in Abbildung 6.16(a) gezeigt, einen Marker, der aus einem Abstand von etwa 2 m eine dreidimensionale Lokalisation und Ausrichtungsangabe ermöglicht. Die Ungenauigkeit dieser Schätzung beträgt positionsabhängig bis zu 1,5 cm.

6.2 Simulation der Fehlerdetektion

Bevor die Umsetzung der Ideen des *MOSAIC*-Frameworks in realen Applikationen vorgestellt wird, soll zunächst die Toleranz einer Positionsschätzung gegenüber den für die verwendeten Sensoren relevanten Fehlermethoden untersucht werden. Besonderes Augenmerk galt dabei der Lokalisation der Fehlerdetektionsmethoden und die Auswirkung einer Entscheidung darüber auf den Kommunikationsaufwand der verteilten Applikation.

6.2.1 Szenariobeschreibung

Eine Simulation bietet für diese Untersuchung eine geeignete Plattform, da auf diesem Weg bestimmte Fehler gezielt und reproduzierbar in das System indiziert werden können, um den Einfluss auf das System zu ermitteln. Die Simulationsumgebung bildet das eigentliche RoboCup-Evaluationsszenario im Sinne einer handhabbaren Komplexität mit zwei grundlegenden Vereinfachungen ab. Zum einen werden entsprechend der Abbildung 6.2 die Bewegungsmöglichkeiten des Roboters auf eine Dimension reduziert und eine kontinuierlich Bewegung (Vor- und Rückwärtsbewegung) entlang einer Linie angenommen. Im Weg-Zeit-Diagramm, wie in Abbildung 6.5 dargestellt, zeigt sich dies in einer sinusförmigen Linie, deren Form aus dem Dynamikmodell des Roboters und der implementierten restwegabhängigen Geschwindigkeitsregelung resultiert. Die zweite Einschränkung betrifft die Fehlertypen der Sensoren. Für jeden der vorhandenen Sensoren – die Ultraschallsensorik des Roboters und die externe Kamera sowie den Infrarotdistanzsensor – wurde jeweils der Fehler mit der größten Amplitude ausgewählt. Zusammen mit dem spezifischen Rauschen des Sensors wird dieser auf der Bewegung des Roboters abgebildet. Ein individuelles Sensormodell berücksichtigt dabei auch das Zeitverhalten der Messung. Die Entscheidung, welches Fehlermodell zu berücksichtigen ist und wie dessen Parameter definiert sind, wurde auf der Basis umfangreicher Messreihen und Analysen getroffen.

Für den Datenaustausch wurde eine heterogene Netzwerkstruktur angenommen, die mit der TrueTime-Toolbox [Tru] im Modell integriert ist. Damit lassen sich kommunikationsbedingte Einflüsse, wie Verzögerungen, Verbindungsverluste usw. berücksichtigen.

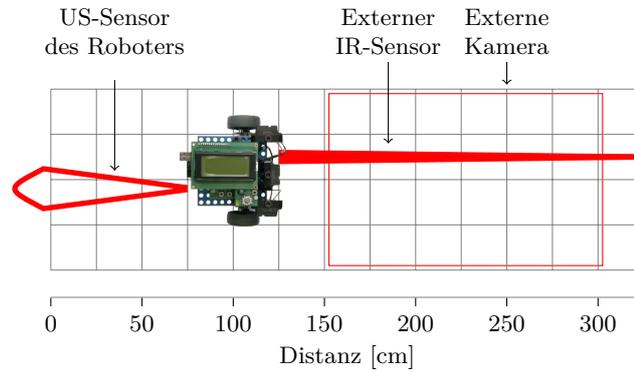


Abbildung 6.2: Das Simulationsszenario als Ausschnitt des Evaluationsszenarios, beschränkt auf eine eindimensionale Bewegung des Roboters und verschiedene partiell relevante Sensoren

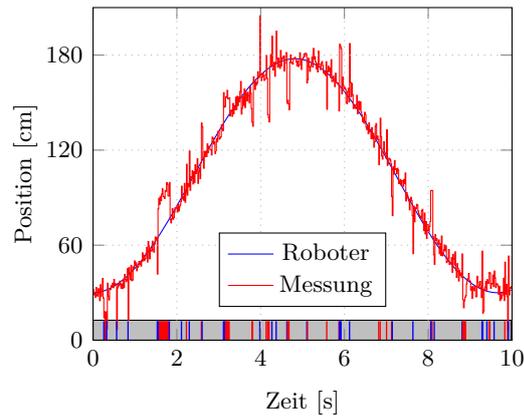


Abbildung 6.3: Ultraschall-Distanzmessungen mit stochastischen Störungen und deren Detektion

So werden die Daten auf dem Roboter über ein CAN Netzwerk ausgetauscht, die Informationen der stationären Sensoren sind für den Roboter via ZigBee verfügbar.

Abbildung 6.3 zeigt den Verlauf der Positionsmessung des Roboters mit dem simulierten SRF10-Ultraschallsensor. Der Roboter bewegt sich innerhalb von 10 Sekunden von der Position 0 nach 180 und kehrt wieder zurück. Der Sensor kann wegen seiner großen Reichweite während der gesamten Fahrt den Abstand zur Wand bestimmen. Die Periode der Ultraschallmessungen ergibt sich aus der maximal zu messenden Reichweite und der Schallgeschwindigkeit zu 0,017s. Die Ultraschallmessungen sind gekennzeichnet durch einen hohen normal verteilt angenommenen Rauschanteil und stochastische Ausreißer.

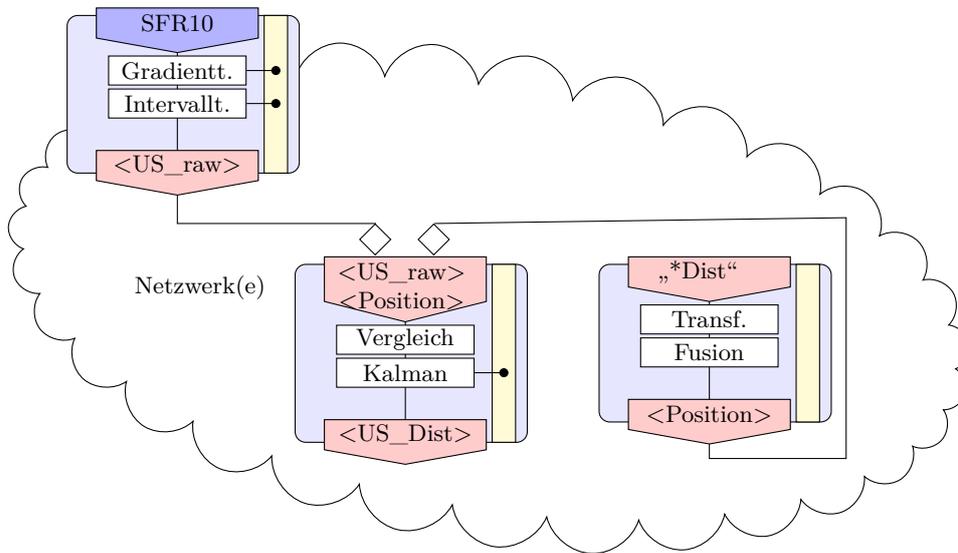


Abbildung 6.4: Simulierte Ultraschallmessungen und Struktur der Einbindung der Fehlerdetektionsmethoden

Im Sinne der Einteilung aus Abbildung 4.5 liegen also permanente und sporadische Überlagerungen vor, wobei die Parameter der erstgenannten Kategorie bekannt sind. Das hohe Messrauschen ist dabei typisch für Ultraschallsysteme und resultiert unter anderem aus dem großen Öffnungswinkel und der damit verbundenen höheren Störanfälligkeit der Sensoren dieses Wirkprinzips.

Die Abbildung 6.10(b) zeigt die mehrschichtige Fehlerdetektion in der schematischen *Smart-X*-Darstellung. Auf dem eigentlichen Sensorknoten laufen die beiden oberen *Smart-X*-Applikationen, während die zusätzlich auch alle anderen Distanzmessungen aggregierende Fusionsapplikation auf einem zentralen Knoten ausgeführt wird. Unmittelbar an den SRF10 Sensor ist eine Prüfung der Einhaltung der Intervallgrenzen und des Anstieges angefügt. Die aus dieser Untersuchung heraus als invalide Messungen erfassten Datensätze sind im Diagramm 6.3 innerhalb des grau unterlegten Bereich mit blauen, senkrechten Strichen markiert. Man erkennt, dass mit der komplexen Überlagerung verschiedener Fehler nur ein Teil der signifikant abweichenden Werte erkannt werden kann. Dies sind eben diejenigen Messungen, die entweder außerhalb des szenariospezifischen Messbereichs ($0 > d > 183 \text{ cm}$) liegen oder aber durch einen deutlichen Sprung im Signalverlauf wahrgenommen werden können.

Nutzt man, wie in Abbildung 6.10(b) gezeigt, aber einen weiteren *Smart-X*-Knoten, der die Messdaten mit den ebenfalls verfügbaren Ergebnissen der Fusion vergleicht, so kann, wie die roten Markierungen im unteren Teil des US-Sensor-Diagrammes zeigen, ein höherer Anteil an Datensätzen korrekt bewertet werden. Dieser Knoten setzt ein

Tracking der Sensormessungen auf Basis eines Kalman-Filters um, wobei nur Messdaten Berücksichtigung finden, die oberhalb eines global festgelegten Gültigkeitswertes liegen. Die Entscheidung darüber wird mit einem für stochastische Systeme vorgesehenen Nearest-Neighbor-Filter gemäß

$$d = (y_k - \hat{y}_k)^T (H^T P_k H^T + R)^{-1} (y_k - \hat{y}_k)$$

umgesetzt [Mit07]. Dabei wird der Abstand d zwischen zwei dem vektorisierten Schätzwert aus der Fusion \hat{y}_k und der aktuellen Sensormessung y_k bestimmt. Die jeweilige Unsicherheit wird über die Kovarianzmatrix der Schätzung P_k und die Messkovarianz R berücksichtigt. H definiert die sogenannte Messmatrix und damit die Abbildungsvorschrift zwischen dem Zustandsvektor und den Messungen. Die Abbildung des Abstandes d auf einem Validitätswert für die Ausgabe des Knotens wird nach der Dynamik des beobachteten Systems und dem erwarteten Vertrauenshorizont festgelegt. Die eigentliche Fusion bestimmt anhand aller eintreffenden Distanzmessungen eine Positionsschätzung über einen gewichteten Mittelwert, der sowohl den RPN und die Validitätsbewertung als auch das Rauschen des Tracks berücksichtigt.

Die für den Ultraschallsensor beschriebene Struktur wiederholt sich für die anderen Sensoren. Der Infrarotsensor, der die Position des Roboters von der rechten Seite her beobachtet, ist in seinem Verhalten dem Sharp GP2D120 [Sha07] nachgebildet. Die Reichweite beträgt 80 cm, wobei die Messungen mit einem entfernungsabhängigen Rauschen überlagert sind, das an der Verbreiterung des Amplitudenintervalls für $t < 4$ und $t > 6$ deutlich wird. Zudem reagiert der Sensortyp empfindlich auf externe Beleuchtung. Wie in Abbildung 6.5(a) dargestellt, ist dieser Einfluss ebenfalls entfernungsabhängig, wobei der Mittelwert der rauschbelasteten Messung sich von der wahren Position des Roboters nach oben bzw. unten entfernt. Die Störung des IR-Sensors durch externe Beleuchtung wurde in [DZK10] untersucht und die Detektionsmöglichkeiten anhand statistischer Tests evaluiert. Die im grau unterlegten Bereich angezeigten, sensorlokal bestimmten invaliden Messungen beschränken sich auf die extremen Abweichungen, die über eine Gradientenprüfung erfasst werden können. Der durch die externe Störung eintretende Offset ist sensornah mangels Referenz nicht detektierbar. Die Rückkopplung des Fusionsergebnisses stellt dabei eine geeignete Methode dar, auch diese Fehlervarianten auf dem Sensorknoten selbst zu erfassen. Die roten Markierungen in Abbildung 6.5(a) zeigen die korrekte Validitätseinschätzung mit diesem Ansatz.

Der dritte Sensor im System, die Kamera zeigt eine deutlich geringere Wiederholrate als die anderen Sensoren, da die Analyse des Videobildes über eine Verarbeitungsgeschwindigkeit von 15 Frames/s nicht hinauskommt. Der Überwachungsbereich der Kamera ist analog zu dem des Infrarotsensors beschränkt und deckt lediglich die rechte Seite des Roboteroperationsbereiches ab. Für die Kamera wird davon ausgegangen, dass der normal verteilte Schätzfehler eine Varianz von $var = (2,5\text{cm})^2$ beträgt. Allerdings wird im Szenario von einer schwankenden, nachteilig wirkenden Beleuchtung ausgegangen, die

sporadisch ein zusätzliches Rauschen in Höhe von $1,5 \cdot var$ bewirkt. Diese charakteristischen Größen ergeben sich aus den Messdaten des Kamerasystems des Versuchsaufbaus. Abbildung 6.5(b) zeigt die als fehlerhaft identifizierten Messungen an. Da das zusätzliche Rauschen nur gering über dem tatsächlichen liegt, ist die Zahl der detektierten Fehlmessungen gering.

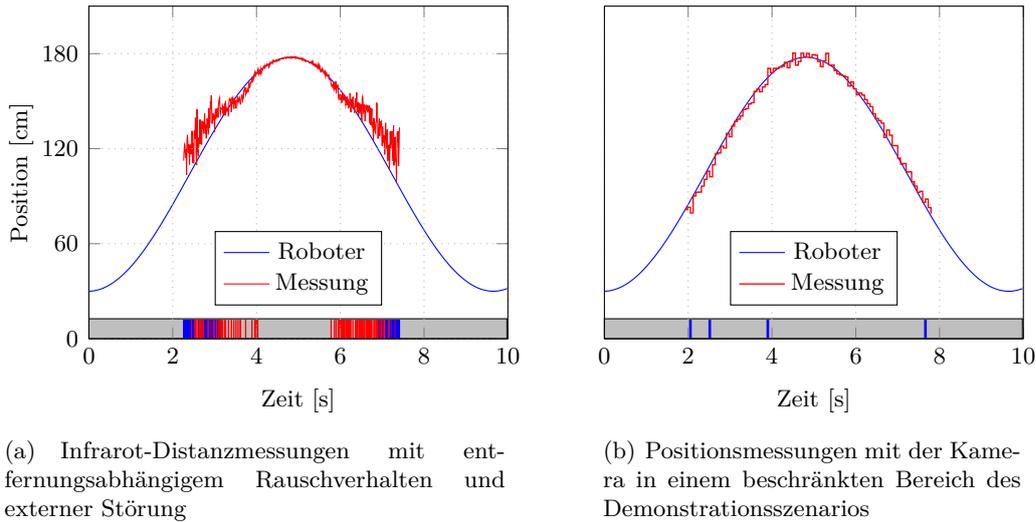


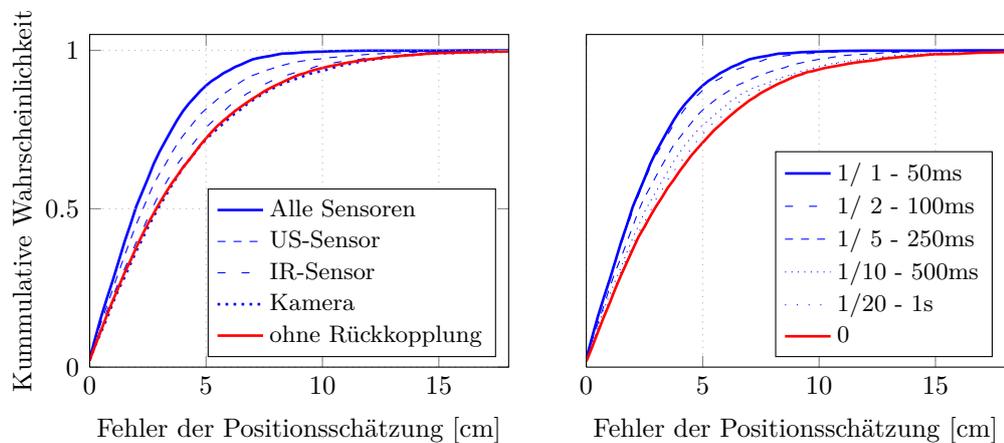
Abbildung 6.5: Vermessung der Position des Roboters mit fixen Umgebungssensoren

Die Kombination des preiswerten Ultraschallsensors am Roboter, der für seine einfache Bewegung des Roboters ausreichend ist, und der präziseren, aber nur selektiv wirkenden Sensortechnik im Bereich kritischer Positionen (Kreuzungsbereiche, Türzonen) bildet das adaptive Konzept von *MOSAIC* ab und stellt den Methoden zur Positionsbestimmung zu jedem Zeitpunkt eine situationspezifisch zugeschnittene Informationsauswahl zur Verfügung. Im Rahmen dieses Szenarios wird davon ausgegangen, dass im verteilten System auf eine globale Zeit zurückgegriffen werden kann.

6.2.2 Ergebnisse

Mit der korrekten Einschätzung der Gültigkeit der Messdaten lässt sich die Güte des fusionierten Positionsergebnisses steigern. Abbildung 6.6 zeigt mit der dicken blauen Linie den kumulativen Fehler der Schätzung an, wobei dem Graphen jeweils 100 simulierte Fahrten des Roboters zugrunde liegen. Für die Grafik wurden die Schätzfehler der Position als Verteilungsfunktion dargestellt. Am dicken blauen Graphen wird deutlich, dass der Fehler mit Einbeziehung der Rückkopplung maximal 7,4 cm beträgt. Ohne Rückkopplung der Fusionsergebnisse, markiert mit dem roten Graphen, steigt der Wert, bei dem

eine kumulative Wahrscheinlichkeit von 1 erreicht wird, auf 14,9 cm. Für die Fehlerverteilung zeigt sich ein analoges Bild. Abbildung 6.6(a) differenziert den Einfluss, den die einzelnen Sensoren auf diese Verbesserung haben. So wird deutlich, dass die Rückkopplung auf die Kamera, bedingt durch den geringen zusätzlichen Fehler, keinen Einfluss hat. Mit der zusätzlichen Validierung der IR-Messungen lässt sich die Positionsschätzung geringfügig verbessern. Für den Ultraschallsensor, markiert durch die gestrichelte Linie, ergibt sich die größte Steigerung. Für diese Verbesserung wurden die Fusionsergebnisse im Takt der Fusion 0,05 1/s verbreitet. Davon ausgehend betrug das zusätzliche Datenaufkommen 20 Nachrichten pro Sekunde mit 5 Byte Payload (2 Byte ganzzahlige Positionsangabe in cm, 2 Byte normierte Varianz der Schätzung, 1 Byte Validitätswert von 0 bis 100).



(a) Darstellung des kumulativen Schätzfehlers in Abhängigkeit der Rücksendung an einzelne Sensoren

(b) Darstellung des kumulativen Schätzfehlers für verschiedene Frequenzen der Transmission (anteilig von der Periode des Fusionsknotens)

Abbildung 6.6: Auswirkung der rückgekoppelten Messdatenvalidierung auf Positionsschätzungen des Roboters

Nachdem bei vorhergehenden Untersuchungen die Rückkopplung dem Takt der Fusion folgte, wurde in einer zweiten Versuchsreihe untersucht, wie sich eine selektive Rückkopplung der Schätzung hinsichtlich des Gesamtergebnisses auswirkt. Abbildung 6.6(b) illustriert den Einfluss unterschiedlicher Rücksendefrequenzen auf den kumulativen Fehler. Die Rücksendefrequenzen wurden dabei als Anteile des Taktes des Fusionsknotens angenommen. Wird nur jede zweite Nachricht (1/2 im Diagramm) an die *Smart-X*-Sensoren übersandt, verringert sich die Güte der Schätzung kaum. Wenn zumindest jedes fünfte Fusionsergebnis publiziert wird, reduziert sich der Fehler noch um die Hälfte. Mit

der weiteren Verringerung der Verbreitung nähert sich die Verteilung des Fehlers dem Graphen an, der die Schätzfehler ohne Rücksendung repräsentiert.

6.2.3 Zusammenfassung

Das vorliegende Beispiel illustriert die Möglichkeiten zur Umsetzung komplexer Fusions- und Filterstrategien auf der Basis der Fehlerdetektionsmechanismen, die in das Konzept der *Smart-X*-Knoten eingebettet sind. Mit dem Zusammenspiel der statischen und dynamischen Validierung gelingt es, die individuellen Sensoreigenschaften und insbesondere Fehlermodelle auf einer abstrakten Ebene objektiv zu bewerten. Das Evaluationsbeispiel zeigt dies in einer mehrschichtigen Fusion, die Messdaten unterschiedlicher Verarbeitungsniveaus validiert und verschmilzt. Die Simulation dieser Vorgänge ermöglicht eine optimale Auslegung der Parameter für die Validierung in Vorbereitung der konkreten Umsetzung. Die effiziente Kombination dieser beiden Entwicklungsschritte wird im folgenden Kapitel, anhand des Simulink-Frameworks für die Generierung von *Smart-X*-Komponenten beschrieben.

6.3 Entwurf eines roboterzentrierten *Smart-X*-Lokalisationssystems

Die im Folgenden beschriebene Entwicklungskette für ein einfaches Lokalisationssystem, das den Ansprüchen des RoboCup-Junior genügt, soll die Leistungsfähigkeit der *MOSAIC*-Implementierung unter Mathworks/Simulink unterstreichen und insbesondere die Möglichkeit des nahtlosen Überganges von der Simulation auf eine reale Zielplattform hervorheben.

6.3.1 Ansatz und Konfiguration

Laserscanner weisen in Roboterapplikationen eine Winkelauflösung von einem oder einem halben Grad auf. Vereinfacht man das Konzept der multiplen, polaren Entfernungsmessung, so gelangt man zu der Frage, welche minimale Auflösung und Reichweite ein Messsystem aufweisen muss, um trotzdem eine hinreichend genaue, fehlertolerante Positionierung zu ermöglichen. Zur Verdeutlichung der Problematik sei auf Abbildung 6.7 verwiesen. Die linke Darstellung 6.7(a) illustriert anhand zweier Roboter, die mit drei und vier Distanzsensoren ausgestattet sind, die nur partiell wirksame Lokalisation mit der jeweiligen Konfiguration. Der Roboter an der Position A kann seinen Abstand zu zwei orthogonalen Wänden der Spielfläche vermessen und darauf aufbauend seine Position (bei bekannter Orientierung) berechnen, für den Roboter B ist dies wegen der Unbestimmtheit in y -Richtung nicht möglich. Um die Gesamtabdeckung der Spielfläche mit einer Konfiguration quantitativ zu erfassen, wurde eine entsprechende Simulation implementiert. Mit dieser ließ sich ausgehend von einer definierten Sensorzahl und -reichweite

für alle Positionen und Orientierungen eine Darstellung wie in Abbildung 6.7(b) bestimmen. Die Grafik zeigt mit den weißen Bereichen die Areale des Spielfeldes an, in denen eine Lokalisation für alle Orientierungen des Roboters möglich ist. Die dunklen Abschnitte erlauben nur für wenige oder keine Orientierungen eine Lokalisation. Diese Untersuchung wurde für unterschiedlichste Sensorkonfigurationen ausgeführt. Wie in [Zug+11] dokumentiert, gelangte man mit dieser ersten Stufe der Entwicklung zu dem Ergebnis, dass 8 Sensoren mit einer Mindestreichweite von 105 cm eine kostenoptimierte Lösung darstellen.

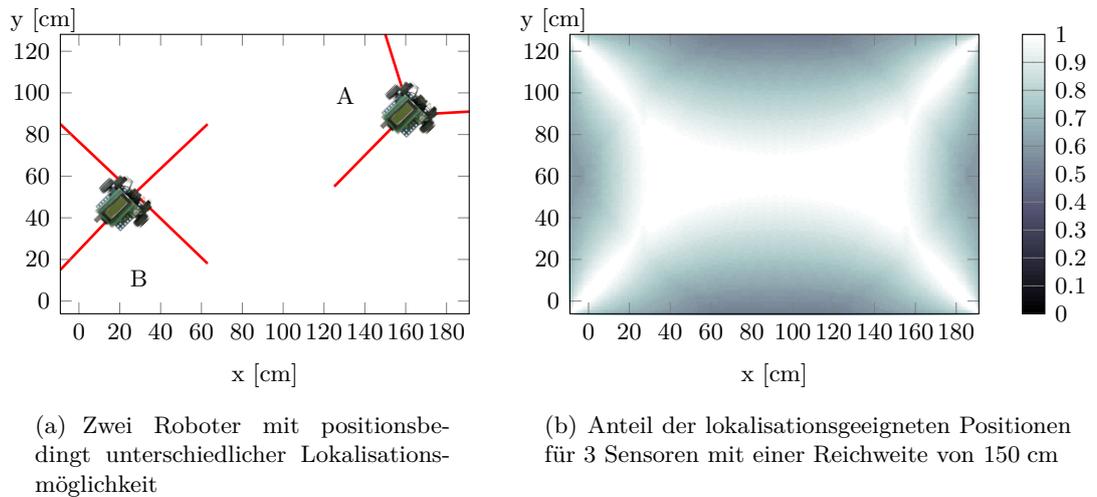


Abbildung 6.7: Konfiguration eines roboterlokalen Lokalisationssystems

6.3.2 Evaluation des Distanzsensors

Ausgehend von dieser Spezifikation wurde der GP2Y0A02YK [Sha01] Infrarot-Distanzsensoren ausgewählt und auf seine Eignung überprüft. Hierfür wurde die in Abschnitt 5.2 beschriebene Entwicklungskette konsequent ausgenutzt und ausgehend von den Datenblättern des Sensors und den einer DAQ 6032 Messerfassungskarte [Nat05] die entsprechenden Interfaces zur Untersuchung des Sensors generiert. Im Rahmen dieser Untersuchung wurden zwei Problemfelder untersucht. Zum einen weist der GP2Y0A02YK-Sensor eine starke Nichtlinearität der Spannungsausgabe über dem Messbereich von 10 cm bis 105 cm auf, die zudem mit einem starken Rauschen überlagert ist. Eine Beschreibung der Klassifikation des Rauschens findet sich in [DZK10]. Die Abbildung 6.8 illustriert die Abhängigkeit der Streuung von der Entfernung des Hindernisses bis zu einer Distanz von 110 cm, wobei die einzelnen Stufensegmente jeweils die Verteilung der Ausgaben angeben. Die helle Farbgebung deutet dabei eine Verteilungsspitze an. Vorteilhaft für die

Analyse dieses Verhaltens und die Herleitung eines entsprechenden Fehlermodells waren die in Simulink vorhandenen, domänenspezifischen Bibliotheken zur Approximation des Rauschverhaltens und der Nichtlinearität.

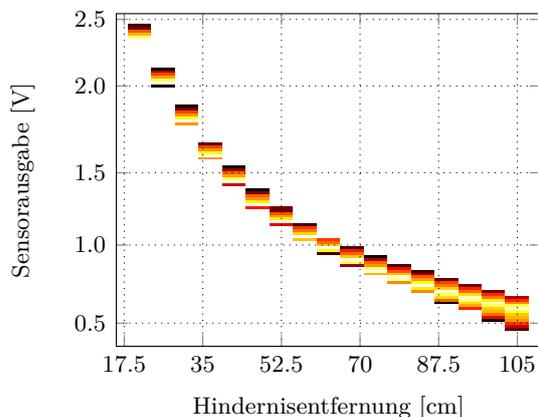


Abbildung 6.8: Verteilung der Messwerte des GP2Y0A02YK über verschiedene Distanzen

Ein zweiter Punkt ist die Validitätsbewertung ausgehend von der Winkellage des Sensors bezüglich des zu detektierenden Objektes. Abbildung 6.9 illustriert das Problem anhand zweier Kennlinien für einen Objektabstand von 18 cm und 50 cm unter verschiedenen Winkellagen. Man sieht deutlich, dass die Entfernungsmessungen, sofern der Winkelbereich zwischen 45° – 135° verlassen wird, einen erheblichen Fehler von bis zu 20 cm aufweisen. Entsprechend ist in die Verarbeitungskette ein Detektionsmechanismus auf der Basis eines Weltmodells zu integrieren, der ausgehend von diesem Wissen die Validität der Entfernungsmessung korrigiert.

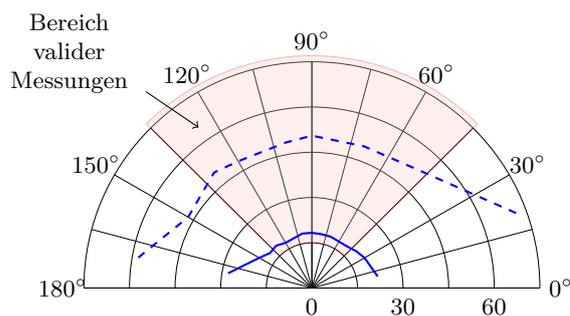


Abbildung 6.9: Distanzmessungen für unterschiedliche Winkellagen bei konstantem Objektabstand (durchgezogene Linie = 18 cm, gestrichelt = 50 cm)

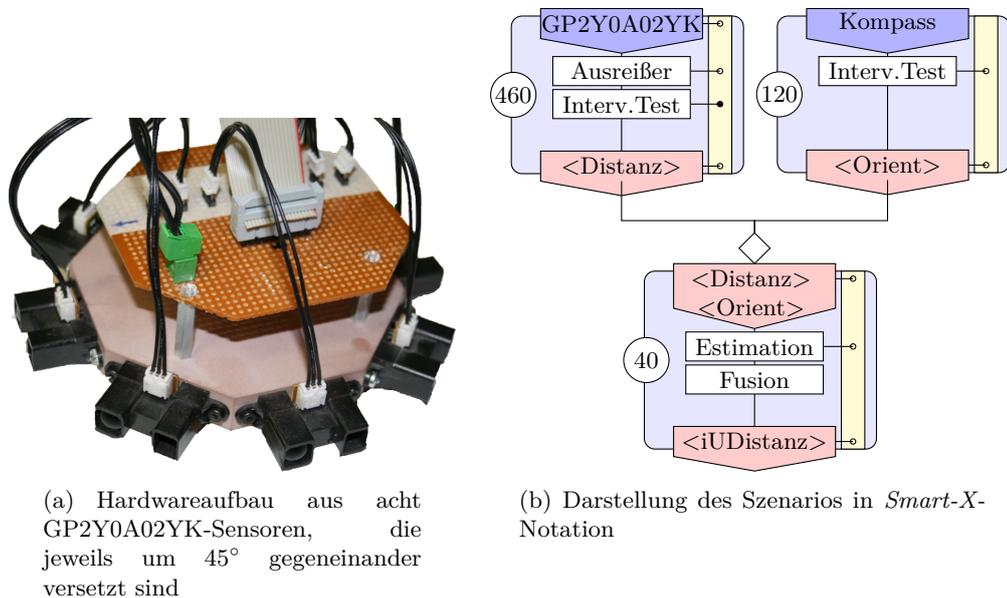


Abbildung 6.10: MEDUSA Lokalisationssystem

Wie in [Zug+11] beschrieben, wurde ausgehend von diesen Analysen eine Abschätzung der zu erwartenden Genauigkeit der Lokalisationsschätzung vorgenommen. Da diese für die Illustration der *MOSAIC*-Simulink-Entwicklungskette nicht von Bedeutung ist, wurde auf eine Darstellung an dieser Stelle verzichtet.

6.3.3 Implementierung

Basierend auf den Ergebnissen der theoretischen Auslegung und der praktischen Erprobung der Sensoren wurde entschieden, acht GP2Y0A02YK-Sensoren zum MEDUSA genannten Lokalisationssystem zusammenzufassen. In Abbildung 6.11 ist der ebenfalls zum System gehörende elektronische Kompass vom Typ CMPS03, nicht zu erkennen, da er an der Unterseite der Trägerplatine angebracht wurde. Die Leistungsaufnahme des Gesamtsystems beträgt bei einer Betriebsspannung von 5 V, etwa 1.1 W.

Der Algorithmus zur Positionsbestimmung baut dabei auf einem Kalman-Filter auf. Ausgehend von der Apriori-Position werden zunächst die Validitätseinschätzungen jeder Messung bestimmt und damit möglicherweise fehlerbehaftete Messungen anhand der oben genannten Grenzwinkellagen von der weiteren Verarbeitung ausgeschlossen. Danach werden in Abhängigkeit von der Performance der Ausführungsumgebung zufällige Positions- und Orientierungskombinationen gebildet und die dafür zu erwartenden Messdaten der Sensoren berechnet. Aus diesen Samples wird dann der am besten zu

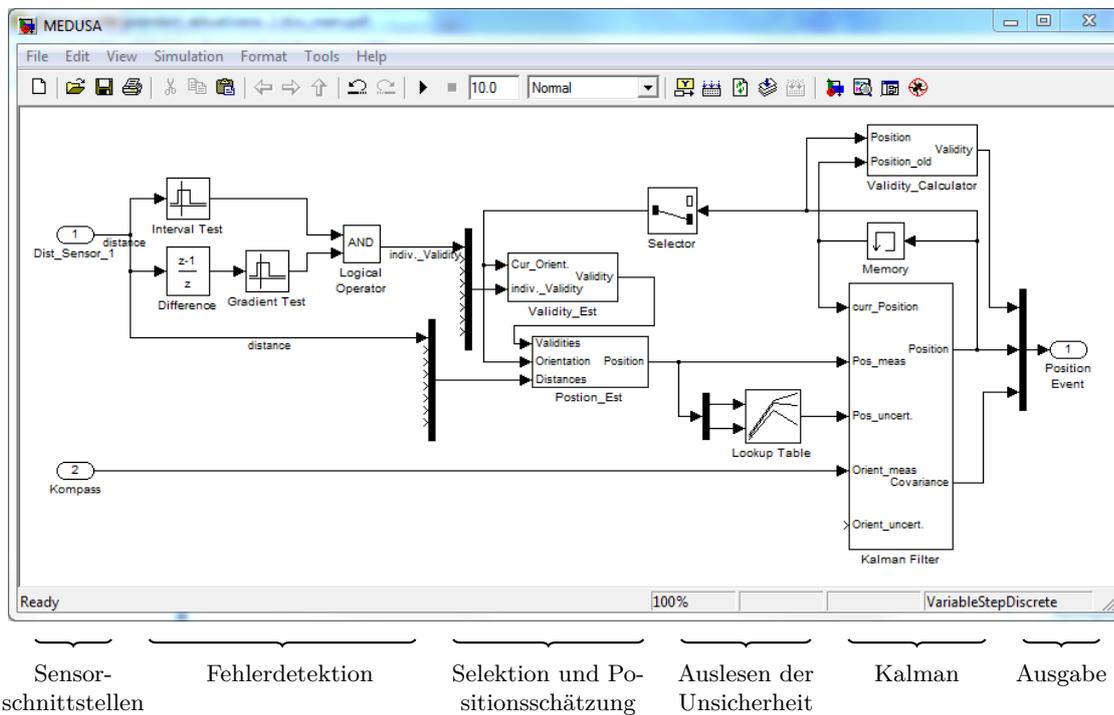


Abbildung 6.11: Auszug des MEDUSA Modells aus Simulink, wobei aus Platzgründen nur einer der acht Distanzsensoren mit der nachgeordneten Verarbeitungskette dargestellt ist

den realen Messungen passende Datensatz ausgewählt und zur Validierung der Apriori-Positionsschätzung genutzt.

Im Ergebnis hat sich die in den Diagrammen 6.12 gezeigte Standardabweichung bei den Positionsannahmen in x- und y-Richtung gegenüber der Referenzmessung mit der stationären Kamera eingestellt. Man erkennt, dass bedingt durch die Totbereiche in der Nähe der jeweiligen Ränder nur eine beschränkte Zahl an Sensoren zur Verfügung steht und damit ein ungenaueres Resultat zu erwarten ist. Mit zunehmender Entfernung von den Rändern (jeweils in x und in y-Richtung) steigt aber auch der Einfluss des Messrauschens, sodass sich unabhängig von der betrachteten Richtung in der Mitte des Spielfeldes ein Anstieg des Messfehlers ergibt. Dieser liegt in Abhängigkeit von der Position bei maximal $\pm 3\sigma = \pm(3 \cdot 2, 2)$ cm. Für eine Kooperation im Rahmen des RoboCup erscheint diese Ungenauigkeit zu groß, sollte aber durch weiterer Filtertechniken weiter reduzierbar sein. So könnte der Ausschluss von Messungen auf der Basis der Winkel-lage zugunsten einer entsprechenden Anpassungsfunktion des Distanzwertes verworfen werden.

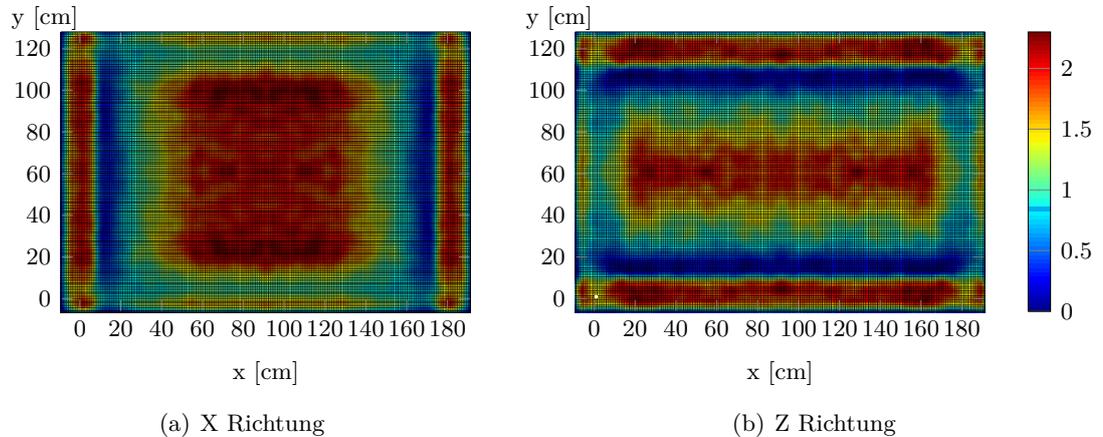


Abbildung 6.12: Verteilung der Standardabweichung der Positionsschätzung über dem Spielfeld

In die Steckverbindung auf der Oberseite, an die in Abbildung 6.11 ein Flachbandkabel angeschlossen ist, das zur Messwerterfassungskarte führt, lässt sich ebenso ein AT90CAN128-Mikrocontroller einhängen. Somit kann der im Rahmen der Simulationen entwickelte Code automatisiert auf den eingebetteten Knoten übertragen werden.

6.3.4 Diskussion

Die Entwicklungskette zur Umsetzung des MEDUSA-Lokalisationssystems zeigt die Leistungsfähigkeit der *MOSAIC*-Werkzeugkette unter Simulink. Gezeigt werden konnte der nahtlose Übergang von der Simulation, über die praktische Erprobung unter Einbeziehung der realen Messungen bis hin zur Codegenerierung für einen eingebetteten Controller. Dabei wurde mit der Integration der XML-basierten Datenblattkonzepte und der grafischen Konfiguration die Systemkomposition über die bestehenden Konzepte aus Simulink hinaus entscheidend vereinfacht. Mit den in Simulink bereitgestellten Bibliotheken war zudem die Umsetzung der Filtermethoden sowie die Integration einer effektiven Lookup-Table rasch möglich.

Gegenüber den etablierten Lokalisationsverfahren bietet MEDUSA, wenn auch noch weitere Anpassungen und eine Verfeinerung des Schätzalgorithmus nötig sind, einen erheblichen Kostenvorteil. Im Hinblick auf die im kommenden Abschnitt beschriebene adaptive Positionsbestimmung weist der zentralistische Ansatz aber zwei grundsätzliche Schwächen auf.

Das System ist in starkem Maße abhängig von einer zumindest langzeitstabilen Orientierungserfassung, da die Abbildung der Entfernungsmessungen auf einer Position nur über eine hinreichend genaue Schätzung der Winkellage möglich ist. Eine permanente

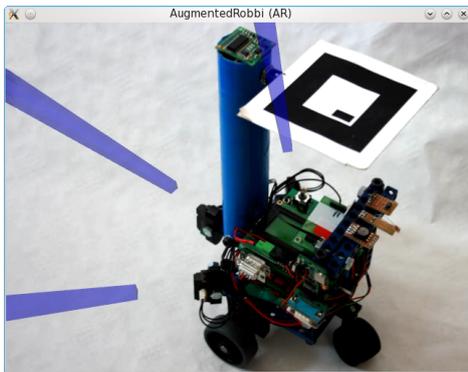
aktive oder passive Störung des Kompasses durch elektromagnetische Felder oder metallische Objekte in der Umgebung steigert die Ungenauigkeit erheblich. In kommenden Versionen des MEDUSA-Systems werden daher als zusätzliche Sensoren ein Gyroskop und die Odometrie des Roboters einbezogen.

Die Umsetzung baut auf die statisch angebundene Sensorik auf und lässt möglicherweise vorhandene, deutlich präzisere Systeme für die eigene Positionsbestimmung außer Acht. Zudem muss, eine Multi-Roboter-Applikation angenommen, jede Plattform mit einem eigenen MEDUSA-System ausgestattet werden, woraus sich gegenüber der (partiell) gemeinsamen Nutzung von Sensormesswerten der Umgebung rasch einen Kostennachteil herleiten lässt.

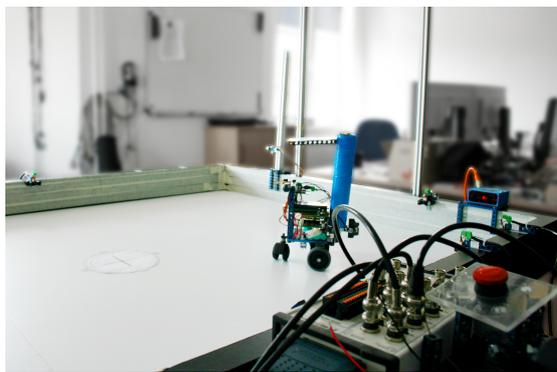
6.4 Adaptive Lokalisation

Kritiker können einwenden, dass aus dem Reglement des RoboCup heraus keine externe Sensorik erlaubt sei. An dieser Stelle soll daher betont werden, dass diese Kollision zwischen dem Erlaubtem und den Konzepten aus *MOSAIC* gut geeignet ist, die zumeist roboterzentriert orientierten Sensorkonzepte mobiler Systeme zu überdenken. Dies wird mit Sicherheit keine Änderung in den Ideen und Regeln des Roboterfußballs mit sich bringen, kann aber ausgehend von dem für die erste Evaluation des Ansatzes geeignete RoboCup-Umgebung auf andere Anwendungen übertragen werden.

6.4.1 Aufbau



(a) Roboter des Versuchsszenarios mit seinen in die Abbildung eingeblendeten Infrarot-Sensorkeulen



(b) Evaluationsumgebung mit Roboter und externen Sensoren

Abbildung 6.13: Roboter und Versuchsaufbau der adaptiven Positionsbestimmung

Abbildung 6.13 illustriert den praktischen Versuchsaufbau. Der mobile Roboter, der mit einer Vielzahl von Sensoren ausgestattet ist, bewegt sich in der Laborumgebung, die dem RoboCup-Szenario nachempfunden ist. Im Bild 6.16(a) sind die in Tabelle 6.1 aufgeführten Sensoren zumindest teilweise sichtbar. Der Sensor am Ende des turmartigen Aufbaus ist der CMPS03, ein elektronischer Kompass. Zur Minimierung des Einflusses der aktiven magnetischen Felder des Roboters (Antriebsmotoren) und der Verzerrung des Erdmagnetfeldes durch dessen metallisches Gehäuse war dieser Abstandhalter notwendig. An diesem wurde zudem der 8x8 cm große Marker für die grafische Erfassung der Position des Roboters befestigt. Im Heckbereich sind die Platinen der als indirekt bezeichneten Sensoren sichtbar, die für die Erfassung der Temperatur, Lichtstärke usw. verantwortlich sind. Mit den Mitteln der AR sind die drei Distanzsensoren vom Typ GP2D120 mit ihren Sensorkeulen hervorgehoben worden, um einen Eindruck von deren Konfiguration zu vermitteln. Diese werden einerseits benutzt, um autonome Bewegungen des Roboters umzusetzen und andererseits zur Kollisionsvermeidung zwischen Roboter und Spielfeldrand.

Die Sensoren der Umgebung sind entsprechend der Darstellung in Abbildung 6.13 verteilt. Am Rand der Spielfläche sind die unterschiedlichen Sensorknoten zu erkennen. Die Anordnung im Einzelnen wird in Abbildung 6.14 deutlicher. Diese Darstellung entstammt einer virtuellen Repräsentation des Szenarios, die zum einen basierend auf den Sensorinformationen aus *MOSAIC* und dem Visualisierungsmethoden von ROS umgesetzt wurde. Neben der allgemeinen Darstellung zusätzlicher Information wie Messwerten, Sensorkeulen usw. ist diese Darstellung insbesondere in Replay-Anwendungen von großer Hilfe. Dabei werden einmal aufgezeichnete Datensätze, zum Beispiel bei der Parameterbestimmung von Filtern immer wieder abgespielt. Die Visualisierung erlaubt dann einen Einblick in den Zustand des Systems.

Die Sensoren sind teilweise als *Smart-X* ausgelegt die über ein FAMOUSO-Netz und ein entsprechendes Gateway die Daten an die in ROS implementierten Verarbeitungsknoten senden. Andere werden über das im Vordergrund der Abbildung 6.13 sichtbare Messsystem erfasst, das als selbstständiger Knoten im ROS-Netz integriert wurde.

Das Muster der Verteilung der externen Sensoren zielte darauf ab, Bereiche mit einem hohen Abdeckungsgrad zu schaffen und auf der anderen Seite Areale zu definieren, für die keine zusätzlichen Sensorinformationen bereitstehen. Anhand Abbildung 6.14 lässt sich dieser positionsabhängig unterschiedliche Erwartungshorizont für die externe Sensorik illustrieren. Im Moment der Aufnahme erfassen der grün markierte Ultraschallsensor und der rot illustrierte Infrarotsensor eine relevante Distanz, die sich auf die Anwesenheit des Roboters zurückführen lässt. Mit der Breite der Keule wird der Ultraschallsensor den Roboter in jedem Fall erfassen, allerdings kann keine Positionsaussage mit einer hinreichenden Präzision erwartet werden. Im Unterschied dazu erlaubt der Infrarotsensor die Korrektur der Roboterlokalisationsannahme, wobei dies aber, wegen des kleinen Öffnungswinkels nur für wenige Datensätze möglich ist.

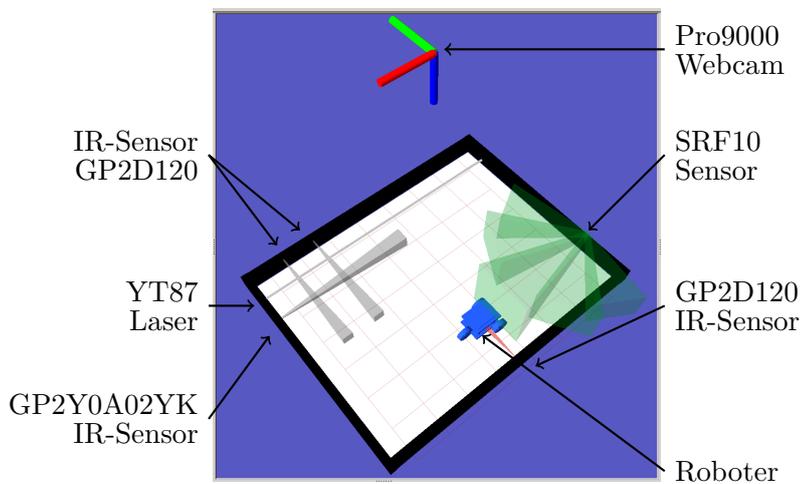
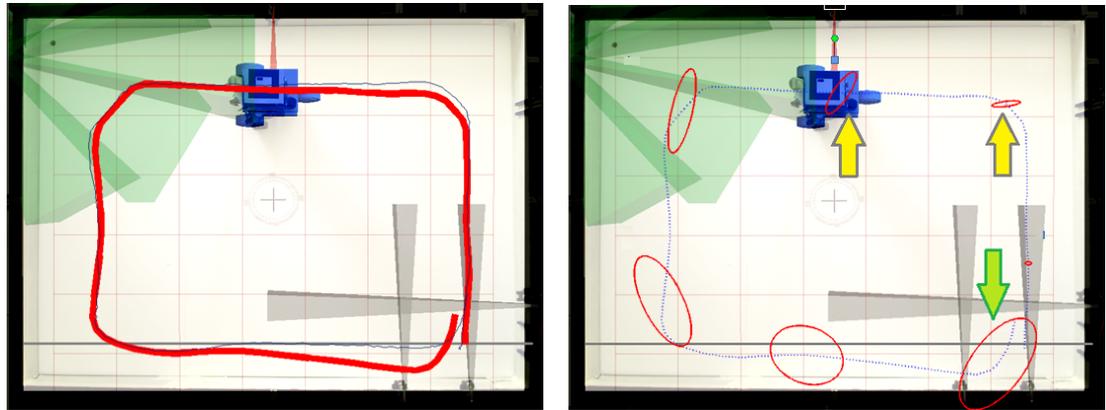


Abbildung 6.14: Darstellung der externen Sensoren in einer virtuellen Repräsentation des Szenarios (grün - Ultraschallsensor, grau - nicht relevante Sensoren, rot - Infrarotsensor)

6.4.2 Ergebnisse

Die Ergebnisse basieren auf der Aufzeichnung einer Fahrt des mobilen Roboters entlang der Außenbegrenzung des Demoaufbaus. Die Trajektorie beginnt dabei in der rechten unteren Ecke und folgt der Außenwand entgegen dem Uhrzeigersinn. Die beiden Illustrationen, Abbildung 6.15 und Abbildung 6.16, stellen dabei zwei Konzepte gegenüber. In Abbildung 6.15 wird die Bewegung eines statisch konfigurierten Systems illustriert, das seine Bewegung ausschließlich mit der Odometrie und dem Gyroskop erfasst. Im Unterschied dazu nutzt der Roboter in Abbildung 6.16 die umgebende Sensorik zusätzlich zu den Inertialmessdaten. Zugleich werden auf allen Sensoren und Messungen die Fehlertoleranzkonzepte dieser Arbeit und ein entsprechender adaptiver Fusionsalgorithmus angewendet.

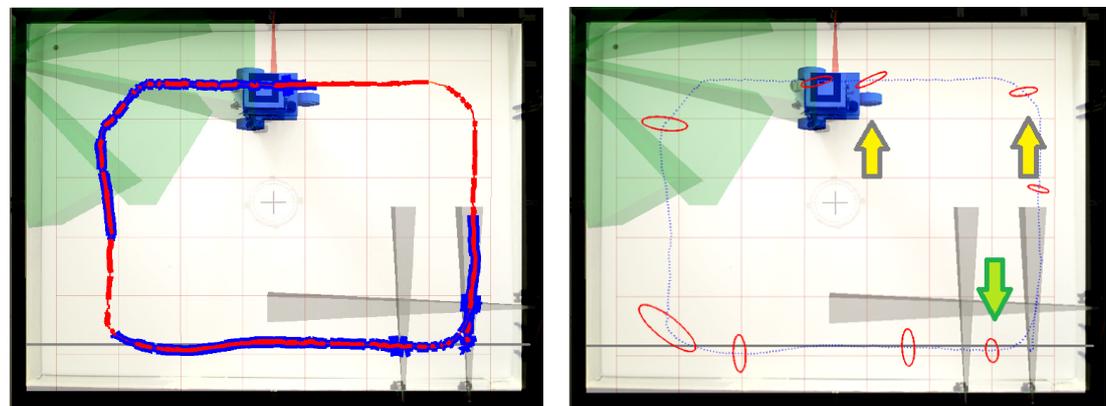
Beide Darstellungen, Abbildung 6.15 und Abbildung 6.16, stellen unter (a) die Positionsschätzung (rot) des Roboters der kamerabasierten Referenzposition (schwarz) gegenüber. Auf der Basis der externen Referenzierung wird die in Abbildung 6.15(a) erkennbare Abweichung der Trajektorie zur Referenzposition so weit reduziert, dass die schwarze Linie durch die Strichstärken vollständig verdeckt wird. Der positionsabhängige Gebrauch der Sensoren wird an der Liniendicke der Positionsschätzung in den Abbildung 6.15(a) und Abbildung 6.16(a) dargestellt. In Abbildung 6.15(a) ist dies eben eine gleichbleibende Stärke entsprechend der konstanten Zahl von Sensoren. Für das adaptive System wurde eine zusätzliche Kategorisierung eingeführt, die internen Sensoren werden analog zu Abbildung 6.15(a) rot und die externen blau visualisiert. Es ist deutlich zu



(a) Geschätzte Trajektorie mit Gyroskop und Odometrie ohne Fehlerbehandlung (rot) im Vergleich zu tatsächlichen Bahn (schwarz)

(b) Unsicherheiten der Positionsschätzungen ohne Fehlerbehandlung

Abbildung 6.15: Abbildung der Bewegung eines mobilen Roboters mit statischer Sensor-konfiguration und ohne Fehlerbehandlung



(a) Zahl der verfügbaren Sensoren in Abhängigkeit von der Position des Roboters

(b) Unsicherheiten der Positionsschätzungen

Abbildung 6.16: Abbildung der Bewegung eines mobilen Roboters in einer instrumentierten Umgebung

erkennen, dass sowohl die Zahl der inertialen Sensoren schwankt, als auch die Zahl der externen Positionsaufnehmer. Die permanent zur Verfügung stehenden Odometriedaten sind durch die dünne rote Linie sichtbar. Das Gyroskop ist wie zuvor angesprochen infolge von Sättigungseffekten nur für geringe Drehraten verwendbar, in den Kurvenbereichen und bei stochastischen Richtungsänderungen werden dessen Messdaten verworfen. Die variable Integration der externen Sensorik wird insbesondere in der rechten unteren Ecke deutlich. Hier stehen, wie in Abbildung 6.16(a) zu sehen, vier Sensoren als externe Referenz zur Verfügung, sodass in diesem Bereich die maximal verfügbare Zahl von Sensoren (zwei interne + zwei externe) erreicht wird.

In den mit (b) bezeichneten Darstellungen 6.15 und 6.16 werden für einige der Positionsschätzungen die jeweiligen Kovarianzen genormt auf eine Mahalanobisdistanz von $3 \cdot \sigma$ gezeigt. Die zu erwartende schleichende Abweichung der nur auf inertialen Sensoren basierenden Lokalisationsschätzung gegenüber der wahren Position geht mit einer wachsenden Unsicherheit der Schätzung einher. Für die Berechnung der Schätzung wurde der nichtlineare Kalman-Filter nach [SN04] implementiert. Dieser wurde für das adaptive System, dessen Unsicherheitsniveau in Abbildung 6.16(b) dargestellt ist, auf die veränderliche Zahl von Messdaten angepasst. Das Ergebnis wird mittels linearem Kalman-Filter mit den externen Positionsmessungen fusioniert. Im Vergleich der Abbildungen 6.15(b) und 6.16(b) werden zwei Ergebnisse deutlich. Zum einen steigt das Fehlerniveau für das adaptive System auch in einem Bereich, für den lediglich die Inertialsensoren zur Positionsschätzung bereitstehen, langsamer an, als für das statisch konfigurierte System. Die gelben Pfeile in beiden Abbildungen markieren diesen Übergang. Während die Unsicherheitsellipsen in beiden Abbildungen in der rechten oberen Ecke der Trajektorie in etwa gleich groß sind, steigt die Wert bis zur nächsten Darstellung der Unsicherheit in Abbildung 6.16(b) langsamer an. Dieser Effekt resultiert aus der Validierung der Messdaten des Inertialsystems und der Berücksichtigung dieser Einschätzung vor deren Fusion. Die zweite Verbesserung des Unsicherheitsniveaus folgt aus der externen Referenzierung. Damit stabilisiert sich die Unsicherheit unabhängig von der zurückgelegten Strecke, wie die Größe der letzten Ellipse auf dem Pfad zeigt, die mit einem grünen Pfeil markiert wurde. Der Einfluss der externen Sensoren auf die Unsicherheit tritt in der unteren linken Ecke von Abbildung 6.16(b) deutlich hervor. Mit dem Betreten des Überwachungsbereiches des Laser-Abstandssensors, dessen enge Keule durch die graue Linie illustriert ist, sinkt die Unsicherheit in horizontaler Richtung stark ab. In vertikaler Richtung lässt der Sensor über die Breite des Roboters keine dezidierte Aussage zu, die Unsicherheit in dieser Richtung verharret auf einem konstanten Wert. Mit dem Erreichen der IR-Sensoren in der rechten unteren Ecke wird auch dieser Anteil der Unsicherheit durch die orthogonal zum Laser wirkenden IR-Sensoren weiter reduziert.

6.4.3 Zusammenfassung

Im Demonstrationsszenario konnten zwei Vorteile des *MOSAIC*-Konzeptes gezeigt werden. Zum einen ist die nahtlose Integration von externen Sensoren in ein Szenario unter Zuhilfenahme der zuvor beschriebenen Mechanismen möglich. Im Beispiel profitiert die Positionsschätzung eines Roboters von dieser Erweiterung des roboterindividuellen Erfassungsbereiches. Mit der Umsetzung der Fehlertoleranzansätze des Frameworks steigt aber auch die Güte der roboterinhärenten Sensormessungen und damit die Validität der unmittelbar auf diesen Sensoren beruhenden Messungen.

Natürlich ließen sich auch auf dem Roboter selbst Abstandssensoren anbringen, um eine Referenzierung zur Umgebung zu gewährleisten und die kontinuierliche Integration des Positionierungsfehlers zu mildern. Allerdings sind deren Positionen selbst von der des Roboters abhängig und damit nur stochastisch beschreibbar. Ein Messwert dieser Sensoren ist somit mehrfachem Rauschen ausgesetzt. Im Unterschied dazu lässt sich die Position der externen Sensoren sehr exakt angeben, sodass die mit deren Messung verbundene Unsicherheit deutlich geringer ausfällt. Das *MOSAIC*-Framework erlaubt dem Entwickler an dieser Stelle eine beliebige Kombination aus interner und externer Sensorik, unabhängig von der für dieses Szenario zu Illustrationszwecken gewählten Konfiguration.

7 Zusammenfassung und Ausblick

Mit der vorliegenden Arbeit wurde die Idee der adaptiven und fehlertoleranten Verarbeitung von Messdaten in verteilten Sensor-Aktor-Anwendungen zunächst konzeptionell entwickelt, prototypisch implementiert und anhand von drei Szenarien evaluiert. In diesem Kapitel sollen die Beiträge der Arbeit anhand der Kernaufgabenstellungen des Motivationskapitels herausgearbeitet und über die Differenzierungen anhand des Anforderungskataloges evaluiert werden. Ausgehend von dieser Übersicht werden zukünftige Entwicklungen am *MOSAIC*-Framework dargestellt.

7.1 Beiträge der Arbeit

Vor diesem Hintergrund zielt die folgende Aufzählung darauf ab, die Fortschritte im Stand der Technik zu strukturieren, die mit dieser Arbeit erreicht wurden. Gleichzeitig werden diese den Elementen des Anforderungskataloges aus Kapitel 2 gegenübergestellt und deren Umsetzung geprüft.

1. Erweiterung bestehender Konzepte für (Sensor-)Datenblätter [Zug+10b, Die+10]

Bestehende Datenblattkonzepte dienen entweder dem Entwurf eines Systems, werden also zur Compilezeit benutzt, oder sie bilden die Grundlage der Knotenidentifikation und Nachrichteninterpretation. In *MOSAIC* wird ein beide Konzepte erfüllender Ansatz verfolgt. Die Kombination aus System- und Ereignisdatenblätter wird im Rahmen dieser Arbeit entwickelt und prototypisch umgesetzt.

Die Systemdatenblätter erlauben eine abstrakte Darstellung der Hardwareeigenschaften der verwendeten Baugruppen. Mit der Einteilung von Sensor-, Plattform- und Kommunikationsdatenblatt wird eine komponentenorientierte Betrachtungsweise verfolgt, die dem Nutzer eine weitgehende Freiheit hinsichtlich der Struktur seiner Baugruppen lässt. Um zu vermeiden, dass diese Variabilität zu einem erhöhten Fehleraufkommen führt, wird der Entwicklungsprozess durch geeignete Konsistenzprüfungen begleitet. Damit wird den Anforderungen hinsichtlich der Kapselung der Heterogenität der Hardware (Anforderung 4) innerhalb einer domänenspezifischen Sprache (Anforderung 3) entsprochen.

Gleichzeitig wurden die Inhalte der Ereignisdatenblätter auf die Anforderungen der adaptiven Selektion und Verarbeitung erweitert. Zum Beispiel sind nunmehr neben dem Rauschverhalten, den Fehlermodellen usw. die Überwachungsbereiche, die die

Verwendbarkeit von Messdaten dokumentieren, in der elektronischen Beschreibung eines Sensors enthalten. Die sich ergebenden Möglichkeiten sind vielfältig:

- Eine solche, mathematische Repräsentation erlaubt zum einen eine Entscheidung über die Relevanz des Sensors ausgehend von der Sensorposition.
- Neben der Entscheidung über die Sensorperspektive können diese Informationen, wie in [Die+10] vorgestellt, zur Visualisierung des Überwachungsbereiches, der gegenwärtigen Messwerte und Fehlerzustände genutzt werden. Diese Darstellungen in AR-Szenarien sind insbesondere bei der Entwicklung komplexer Mensch-Maschine-Applikationen hilfreich, um die Wahrnehmung eines Roboters zu illustrieren.

Die Ereignisdatenblätter werden neben der Datenidentifikation und -interpretation auch innerhalb des Entwicklungsprozesses für die automatisierte Konfiguration von *Smart-X*-Komponenten verwendet. So erfolgt die Parametrisierung eines entsprechenden Netzwerkinterfaces in der Python-Implementierung anhand der Informationen aus dem zugehörigen Ereignisdatenblatt automatisch. Das erweiterte Ereignisdatenblattkonzept genügt damit den Zielstellungen des Anforderungskataloges im Hinblick auf die Datenbeschreibung (Anforderung 6) vollständig.

Die vorliegende Arbeit identifiziert die für die adaptive Fusion und Validierung erforderlichen Parameter und leitet daraus ein mehrschichtiges Konzept für die Bereitstellung dieser Daten ab.

2. Integriertes Fehlertoleranzkonzept [DZK10, Zug+10a, ZD10, ZDK11b]

Zur Interpretation der Daten, die in einer verteilten Anwendung auf der Basis heterogener Sensoren gewonnen wurden, ist eine Einschätzung von deren Güte erforderlich. Anderenfalls würden hochgradig fehleranfällige IR-Sensormessungen und sehr verlässliche Laserdistanzdaten mit gleicher Gewichtung verarbeitet. Diese Arbeit beschreibt eine mehrschichtige Strategie zur Klassifikation der Güte von Sensordaten, die die Bewertung jedes Datensatzes mit einer Einschätzung der Validität eines Sensorknotens kombiniert. Der statische Anteil, der sich aus dem konkreten Sensorzuschnitt ergibt, fließt in das Datenblatt eines Ereignisses ein. Die Generierung des dynamischen Anteils, der sich auf einen konkreten Datensatz bezieht, ist als querschneidende Funktionalität in die *Smart-X*-Architektur eingebunden. Die Abbildung der verschiedenen Fehlermodelle der unterschiedlichen Sensoren auf zwei Kenngrößen gewährleistet die Übertragbarkeit des Konzepts und Vergleichbarkeit der Ergebnisse. Die Anforderungen des Punktes 7 werden damit in *MOSAIC* umgesetzt.

3. Adaptive Verarbeitung und Fusion [ZDK11a, ZK09, ZD10]

Gegenwärtige Programmierkonzepte für verteilte Anwendungen unterstützen die Anwendungsentwicklung, insbesondere bei der Verwendung einer unterschiedlichen

Zahl von Sensoren und Datensätzen pro Berechnungszyklus kaum. Die *Smart-X*-Architektur wurde auf diese Aufgabe zugeschnitten und kombiniert die Voraussetzungen für die Einbindung einer Datenselektion, adaptiven Verarbeitung und Fehlerdetektion. Für die Datenselektion wurde eine 5-Ebenen-Filterstruktur entwickelt, die zwischen Datenerfassung, Aufbereitung, Verarbeitung und Verbreitung eine differenzierte Selektion ermöglicht. Damit kann zum Beispiel neben der Prüfung der Validität und Relevanz ein ressourcensparender Ausschluss von Datenpaketen vor der Funkverbreitung vorgenommen werden.

Die variierende Zahl von Messungen erfordert ein entsprechend flexibles Verarbeitungsmanagement. Die Variabilität ergibt sich dabei in der vorliegenden Arbeit aus zwei Ansätzen. Zum einen kann die Verarbeitungskette eines *Smart-X* zur Laufzeit an die Datenlage angepasst werden. Zum anderen wurde die Anpassungsfähigkeit der einzelnen Verarbeitungsfunktionen untersucht. Das Hauptaugenmerk lag dabei auf stochastischen Filtern, wie dem Kalman-Filter, der in vielen Aufgabenfeldern der Robotik zum Einsatz kommt. Die Umsetzung der diesbezüglichen Anforderungen (8, 9) des Katalogs aus Kapitel 2 wurde im vorangegangenen Kapitel anhand eines Lokalisationsszenarios illustriert.

4. Architektur für *Smart-X*-Systeme [SZ08, Kai+08b, Zug+10a]

Die Arbeit des Softwareentwicklers im Bereich der verteilten eingebetteten Anwendungen besteht darin, die Sequenz – Messdatenerfassung, Verarbeitung, Ausgabe – aufgabenspezifisch zu adaptieren und zu kombinieren. Dazu wird eine Architektur vorgestellt, die die kommunikationsorientierte Programmierabstraktion des Sentient Objects um eine Verarbeitungsstruktur erweitert und abstrakte, uniforme Komponenten für die Sensorik und Aktorik, den Netzwerkzugriff sowie Schnittstellen für Verarbeitungsfunktionen bereitstellt. Die Integration der *Smart-X*-Architektur in domänenspezifische Entwicklungsumgebungen und Programmiersprachen erlaubt es dem Entwickler, sich auf die eigentliche Funktionalität des Knotens zu konzentrieren und die bestehenden Werkzeugketten dafür effektiv zu nutzen (Anforderungen 5 und 3)

Die im Anforderungskatalog beschriebene interne Modularität (Anforderung 1) wird, wie im Implementierungsbeispiel unter Python gezeigt, durch eine flexible Erweiterbarkeit der Art und Zahl der Schnittstellen sowie der Verarbeitung- und Filterfunktionen gewährleistet. Die externe Kombinierbarkeit der *Smart-x*-Knoten wird mit der Integration der Konzepte des Sentient Object und deren Umsetzungen in FAMOUSO und ROS sichergestellt (Anforderung 2).

Das *MOSAIC*-Framework gewährleistet bei der Umsetzung der Kernaufgaben die Integration der Anforderungen aus dem Kapitel 2. Die praktische Erprobung und Evaluation erfolgte an drei Szenarien, wodurch zum Beispiel die Wirksamkeit der Fehlerdetektionskonzepte unterstrichen wurde.

7.2 Zukünftige Arbeiten

Die Zielrichtungen zukünftiger Arbeiten richten sich sowohl auf die konzeptionelle Weiterentwicklung der Architektur und ihrer Teilabstraktionen wie auch den Zuschnitt bei einer konkreten Umsetzung in einer Programmiersprache.

- Die Abbildung von einfachen Abhängigkeiten wie zum Beispiel für die Kalibrierung des Ultraschallsensors anhand der aktuellen Umgebungstemperatur, die Berücksichtigung externen Lichteinfalls für Infrarotsensoren usw. muss bisher vom Entwickler auf der Ebene der Anwendung vorgegeben werden. Neben der Definition der hinzuzuziehenden Daten werden die Filter- oder Fusionsmethoden individuell implementiert. In zukünftigen Entwicklungen am Framework sollten diese unmittelbaren Beziehungen bereits Bestandteil der Datenblätter eines Sensors sein. Damit könnten Abhängigkeiten während der Initialisierung automatisch aufgelöst und die entsprechenden Kommunikationsschnittstellen generiert werden. Dies setzt aber eine allgemeingültige Abbildung der Zusammenhänge voraus.
- Die in den Abbildungen des Evaluationskapitels gezeigten Möglichkeiten der Visualisierung der Datenblattinformationen und Sensorzustände mit den Mitteln der erweiterten Realität öffnen ein weiteres Feld für die Nutzung des *MOSAIC*-Frameworks. Erste Konzepte zur Integration komplexer Systemzustände wurden im Hinblick auf die sichere Mensch-Maschine-Interaktion in [Die+10] untersucht. Die Informationen über Messbereiche, Sensorkennwerte usw. werden dabei genutzt, um eine auf die Funktion des Systemanwenders (Entwickler, Maschinenbediener, Wartungstechniker) zugeschnittene Visualisierung bereitzustellen. Die Darstellung der Informationen erfolgte in diesen Szenarien manuell, laufende Arbeiten bemühen sich im Hinblick auf die Darstellung der Sensorkeulen um einen generelleren Ansatz, der automatisiert auch komplexe Sensoren abbilden kann.
- Die Zusammenstellung der Informationen eines Ereignisdatenblattes ist in der bestehenden Implementierung vom Entwickler weitgehend manuell vorzunehmen. Im Hinblick auf die bereits in den Systemdatenblättern enthaltenden Daten, die in Tabelle 4.5 aufgelistet sind, sollte eine automatische Übernahme geprüft werden. Für einfache Beispiele, wie die Fehlermodelle und ihre Parameter, wurde dies umgesetzt, für komplexere Informationen fehlt dafür bisher eine Strategie.

Abkürzungsverzeichnis

ADC	Analog-to-Digital Converter
AR	Augmented Reality
CAN	Controller Area Network
COLLADA	COLLABorative Design Activity
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
DAQ	Data Acquisition
DCOM	Distributed Component Object Model
DSL	Domain Specific Language
FAMOUSO	Family of Adaptive Middleware for autonomOUS Sentient Objects
FDI	Fault Detection and Isolation
FMEA	Failure Modes and Effects Analysis
FPGA	Field Programmable Gate Array
I ² C	Inter-Integrated Circuit
IEEE	Institute of Electrical and Electronics Engineers
IFS	Interface File System
ILSS	Istrumented Logical Sensor Systems
JDL	Joint Directors of Laboratories
MOE	Measure of the Effectiveness
NCAP	Network Capable Application Processor
ODE	Open Dynamics Engine
OGC	Open Geospatial Consortium
OMG	Object Management Group
OOSM	Out of Sequence Measurement

Acronyms

OSGi	Open Services Gateway initiative
OSI	Open Systems Interconnection
ROI	Region of Interest
ROS	Robot Operating System
RPN	Risk Priority Number
SensorML	Sensor Model Language
SI	Système international d'unités
SiL	Software-in-the-Loop
SOAP	Simple Object Access Protocol
SPI	Serial Peripheral Interface
SQL	Structured Query Language
STD	Smart Transducer Descriptions
STIM	Smart Transducer Interface Modul
SWE	Sensor Web Enablement
TEDS	Transducer Electronic Data Sheet
TII	Transducer Independend Interface
TTP/A	Time-Triggered Protocol Class A
UART	Universal Asynchronous Receiver Transmitter
UDDI	Universal Description, Discovery and Integration
UML	Unified Modeling Language
US	Ultra Sonic
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WSDL	Web Services Description Language
XML	Extensible Markup Language
XSLT	Extensible Stylesheet Language Transformation

Literatur

- [AAM01] S. Alag, A. M. Agogino und M. Morjaria. „A Methodology for Intelligent Sensor Measurement, Validation, Fusion, and Fault Detection for Equipment Monitoring and Diagnostics“. In: *Artificial Intelligence for Engineering Design, Analysis and Manufacturing* 15 (4 2001), S. 307–320.
- [ALR01] A. Avizienis, J. Laprie und B. Randell. *Fundamental concepts of dependability*. Techn. Ber. University of Newcastle upon Tyne, 2001.
- [APB04] J. Artiola, I. Pepper und M. Brusseau. *Environmental monitoring and characterization*. Academic Press, 2004.
- [APR02] S. Adhikari, A. Paul und U. Ramachandran. „D-Stampede: Distributed Programming System for Ubiquitous Computing“. In: *ICDCS '02: Proceedings of the 22 nd International Conference on Distributed Computing Systems (ICDCS'02)*. Washington, DC, USA: IEEE Computer Society, 2002, S. 209.
- [Aga+02] B. Agarwalla, P. W. Hutto, A. Paul und U. Ramachandran. *Fusion Channels: A Multi-sensor Data Fusion Architecture*. Techn. Ber. 53. Georgia Tech College of Computing, 2002.
- [Aky+02] I. Akyildiz, W. Su, Y. Sankarasubramaniam und E. Cayirci. „Wireless sensor networks: a survey“. In: *Computer networks* 38.4 (2002), S. 393–422.
- [Ana04] Analog Devices. *Data sheet ADXSR150*. 2004. URL: http://www.analog.com/static/imported-files/data_sheets/ADXRS150.pdf.
- [Art] *ARToolKit*. <http://www.hitl.washington.edu/artoolkit/>. [(online), as at: 25.02. 2010]. 2007.
- [BK80] R. A. Breckenridge und S. J. Katzberg. „The status of smart sensors“. In: *Sensor Systems for the 80's*. Colorado Springs, USA, 1980, S. 41–46.
- [BPOARLS07] H. Benítez Pérez, J. Ortega Arjona und G. Reza Latif Shabgahi. „Definition and Empirical Evaluation of Voters for Redundant Smart Sensor Systems“. en. In: *Computación y Sistemas* 11 (Sep. 2007), S. 39 –60.

- [BR06] M. Botts und A. Robin. *OpenGIS sensor model language (SensorML) implementation specification*. Techn. Ber. Open Geospatial Consortium Inc, 2006.
- [BS02] Y. Bar-Shalom. „Update with out-of-sequence measurements in tracking: Exact solution“. In: *IEEE Transactions on Aerospace and Electronic Systems* 38.3 (2002), S. 769–777.
- [BW08] F. J. Brunner und K. W. Wagner. *Taschenbuch Qualitätsmanagement: Leitfaden für Studium und Praxis*. 4. Auflage. Carl Hanser Verlag GmbH & CO. KG, 2008.
- [Bar07] H. Bartsch. *Taschenbuch mathematischer Formeln*. Hanser Verlag, 2007.
- [Bel+02] T. Bellwood u. a. *UDDI Version 3.0*. Hrsg. von Organization for the Advancement of Structured Information Standards (OASIS). 2002.
- [Ben02] A. Benaskeur. *Sensor Management in Command & Control*. 2002.
- [Bib06] Bibliografisches Institut und F.A. Brockhaus AG. *Brockhaus Enzyklopädie in 30 Bänden*. 21. Aufl. F.A. Brockhaus GmbH, 2006.
- [Bie04] G. Biegel. „A programming model for mobile, context-aware applications“. Phd Thesis. University of Dublin, Trinity College, 2004.
- [Bla+03] M. Blanke, M. Kinnaert, J. Schröder, J. Lunze und M. Staroswiecki. *Diagnosis and fault-tolerant control*. Heidelberg: Springer, 2003.
- [Bos+07] R. Bose, A. Helal, V. Sivakumar und S. Lim. „Virtual sensors for service oriented intelligent environments“. In: *Proceedings of the third conference on IASTED International Conference: Advances in Computer Science and Technology*. Phuket, Thailand: ACTA Press, 2007, S. 165–170.
- [Bra09] T. Brade. „Codegeneration aus Simulink/Embedded Real Time Workshop Modellen am Beispiel eines AVR Targets,“ Studienarbeit. Otto-von-Guericke Universität Magdeburg, 2009.
- [Bra+10] T. Brade, M. Schulze, S. Zug und J. Kaiser. „Model-Driven Development of Embedded Systems“. In: *12th Brazilian Workshop on Real-Time and Embedded Systems (WTR)*. Gramado, Brazil: Brazilian Computer Society, Mai 2010.
- [Bra11] T. Brade. „Simulink-Framework für die Entwicklung von Smart Sensors“. Diplomarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2011.
- [CAN05] CAN in Automation, Hrsg. *CiA 306 DS V1.3: Electronic Data Sheet Specification for CANopen*. CiA, CANopen, 2005.

-
- [CD93] J. Crowley und Y. Demazeau. *Principles and Techniques for Sensor Data Fusion*. 1993.
- [CH+02] M. Chu, H. Haussecker u. a. „Scalable information-driven sensor querying and routing for ad hoc heterogeneous sensor networks“. In: *International Journal of High Performance Computing Applications* 16.3 (2002), S. 293.
- [CH05] W.-P. Chen und J. C. Hou. „Data Gathering and Fusion in Sensor networks“. In: *Handbook of sensor networks: Algorithms and architectures*. Hrsg. von I. Stojmenovi c. Wiley, 2005.
- [CJ11] D. Chris und D. Jan. *Janus Webseite*. 2011. URL: <http://www.ros.org/wiki/Janus>.
- [CKV04] A. Casimiro, J. Kaiser und P. Verissimo. „An Architectural Framework and a Middleware for Cooperating Smart Components“. In: *ACM Computing Frontiers conference (CF'04)*. Ischia, Italy, Apr. 2004, S. 28–39. URL: <http://portal.acm.org/citation.cfm?id=977098>.
- [CPR05] P. Corke, R. Peterson und D. Rus. „Networked robots: Flying robot navigation using a sensor net“. In: *Robotics Research* (2005), S. 234–243.
- [CRS00] L. C mara, O. Ruiz und J. Samitier. „Complete IEEE-1451 node, STIM and NCAP, implemented for a CAN network“. In: *Instrumentation and Measurement Technology Conference, 2000. IMTC 2000. Proceedings of the 17th IEEE*. Bd. 2. IEEE. 2000, S. 541–545.
- [C m+04] L. C mara, O. Ruiz, A. Herms, J. Samitier und J. Bosch. „Automatic generation of intelligent instruments for IEEE 1451“. In: *Measurement* 35.1 (2004), S. 3–9.
- [DEE03] M. Delavai, U. Eisenmann und W. E. Elmenreich. *A Generic Architecture for Integrated Smart Transducers*. Techn. Ber. Technische Universit t Wien, Juni 2003.
- [DFP98] S. Doebling, C. Farrar und M. Prime. „A summary review of vibration-based damage identification methods“. In: *Shock and Vibration Digest* 30.2 (1998), S. 91–105.
- [DH98] M. Dekhil und T. C. Henderson. „Instrumented logical sensor systems-practice“. In: *Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on*. Bd. 4. IEEE. 1998, S. 3103–3108.

- [DLH11] C. Diedrich, A. Lüder und L. Hundt. „Bedeutung der Interoperabilität bei Entwurf und Nutzung von automatisierten Produktionssystemen“. In: *at - Automatisierungstechnik* 59.7 (Juli 2011), S. 426–438. URL: <http://www.oldenbourg-link.com/doi/abs/10.1524/auto.2011.0937>.
- [DMC00] L. Drolet, F. Michaud und J. Côté. „Adaptable sensor fusion using multiple Kalman filters“. In: *Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings. 2000 IEEE/RSJ International Conference on*. Bd. 2. IEEE. 2000, S. 1434–1439.
- [DS04] David L. Hall und Sonya A. H. McMullen. *Mathematical Techniques in Multisensor Data Fusion*. 2. Aufl. Artech House Inc, 2004.
- [DW88] H. F. Durrant-Whyte. „Sensor models and multisensor integration“. In: *The International Journal of Robotics Research* 7.6 (1988), S. 97.
- [DZK10] A. Dietrich, S. Zug und J. Kaiser. „Detecting External Measurement Disturbances Based on Statistical Analysis for Smart Sensors“. In: *Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE)*. Bari, Italy, Juli 2010, S. 2067–2072.
- [DZK11a] A. Dietrich, S. Zug und J. Kaiser. „Model based Decoupling of Perception and Processing“. In: *ERCIM/EWICS/Cyberphysical Systems Workshop, Resilient Systems, Robotics, Systems-of-Systems Challenges in Design, Validation & Verification and Certification*. Naples, Italy, Sep. 2011.
- [DZK11b] A. Dietrich, S. Zug und J. Kaiser. „Modelbasierte Fehlerdetektion in verteilten Sensor-Aktor-Systemen“. In: *11./12. Forschungskolloquium am Fraunhofer IFF*. Fraunhofer Institut für Fabrikbetrieb und Automatisierung (IFF), 2011.
- [Das10] Dassault Systemes. *Dymola - Webseite*. 2010. URL: <http://www.3ds.com/products/catia/portfolio/dymola>.
- [Deu95] Deutsches Institut für Normung. *DIN 1319-1, Grundlagen der Messtechnik*. Norm. 1995.
- [Dev00] Devantech. *Data sheet CMPS03*. 2000. URL: <http://www.robot-electronics.co.uk/htm/cmeps3tech.htm>.
- [Dev03] Devantech. *Data sheet SRF10*. 2003. URL: <http://www.robot-electronics.co.uk/htm/srf10tech.htm>.
- [Dia10] R. Diankov. „Automated Construction of Robotic Manipulation Programs“. Doktorarbeit. Carnegie Mellon University: The Robotics Institute, 2010.

-
- [Die+10] A. Dietrich, M. Schulze, S. Zug und J. Kaiser. „Visualization of Robot’s Awareness and Perception“. In: *First International Workshop on Digital Engineering (IWDE)*. Magdeburg, Germany: ACM Press New York, NY, USA, Juni 2010.
- [Dow97] M. Dowson. „The Ariane 5 software failure“. In: *ACM SIGSOFT Software Engineering Notes* 22 (2 März 1997). URL: <http://doi.acm.org/10.1145/251880.251992>.
- [EAM02] P. Escamilla-Ambrosio und N. Mort. „Multi-sensor data fusion architecture based on adaptive Kalman filters and fuzzy logic performance assessment“. In: *Information Fusion, 2002. Proceedings of the Fifth International Conference on*. Bd. 2. IEEE. 2002, S. 1542–1549.
- [EJC01] E. Euler, S. Jolly und H. Curtis. „The failures of the mars climate orbiter and mars polar lander: A perspective from the people involved“. In: *Proceedings of Guidance and Control*. Hrsg. von M. Maurer und C. Stiller. Springer, 2001, S. 59–89.
- [EPS04] W. Elmenreich, S. Pitzek und M. Schlager. „Modeling Distributed Embedded Applications on an Interface File System“. In: *Proceedings of the Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC’04)*. Vienna, Austria, 2004, S. 175–182.
- [Ehr07] M. Ehrenstraßer. „Sensoreinsatz in der telepräsenten Mikromontage“. Diss. Technische Universität München, 2007.
- [Elm02] W. Elmenreich. „Sensor Fusion in Time-Triggered Systems“. Diss. Technische Universität Wien, Okt. 2002.
- [Elm+02] W. Elmenreich u. a. *TTP/A smart transducer programming-a beginner’s guide*. Techn. Ber. Technische Universität Wien, 2002.
- [Enc07] Encyclopaedia Britannica. *Encyclopaedia Britannica online. Academic edition*. http://www.bibliothek.uni-regensburg.de/dbinfo/frontdoor.php?titel_id=5389. Chicago, Ill., 2007.
- [FKS97] P. Frank und B. Köppen-Seliger. „Fuzzy logic and neural network applications to fault diagnosis“. In: *International Journal of Approximate Reasoning* 16.1 (1997), S. 67–88.
- [Fit+02] A. Fitzpatrick, G. Biegel, S. Clarke und V. Cahill. „Towards a sentient object model“. In: *Workshop on Engineering Context-Aware Object Oriented Systems and Environments (ECOOSE)*. Citeseer. 2002.
- [Fra00] R. Frank. *Understanding smart sensors*. Bd. 11. IOP Publishing, 2000, S. 1830.

- [Fra90] Frank, P.M. „Fault diagnosis in dynamic systems using analytical and knowledge-based redundancy“. In: *Automatica* 26.3 (1990), S. 459–474.
- [Fra93] P. Frank. „Advances in observer-based fault diagnosis“. In: *International Conference on Fault Diagnosis (TOOLDIAG)*. Toulouse, France, 1993.
- [Fre08] Freescale Semiconductors. *Data sheet MMA7260*. 2008. URL: http://www.freescale.com/files/sensors/doc/data_sheet/MMA7260QT.pdf.
- [GA00] K. Goebel und A. Agogino. „Fuzzy belief nets“. In: *International Journal of Uncertainty Fuzziness and Knowledge based Systems* 8.4 (2000), S. 453–470.
- [GB97] Geir E. Hoveland und Brennan J. McCarragher. „Dynamic Sensor Selection for Robotic Systems“. In: *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*. Albuquerque: IEEE Computer Society, 1997.
- [GNU11] GNU R Project. *The R Project for Statistical Computing - Webseite*. verfügbar unter <http://www.r-project.org/> am 11.01.2011. 2011.
- [GY03] G. Gupta und M. Younis. „Fault-tolerant clustering of wireless sensor networks“. In: *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*. Bd. 3. Ieee. 2003, S. 1579–1584.
- [GZ04] L. Guibas und F. Zhao. *Wireless Sensor Networks: An Information Processing Approach (Morgan Kaufmann Series in Networking): An Information Processing Approach*. Morgan Kaufmann, 2004.
- [Gro+02] H. Gross, A. Koenig, H. Boehme und C. Schroeter. „Vision-based monte carlo self-localization for a mobile service robot acting as shopping assistant in a home store“. In: *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*. Bd. 1. IEEE. 2002, S. 256–262.
- [HAR10] HART Communication Foundation. *HART Device Description Language*. 2010.
- [HD98] T. C. Henderson und M. Dekhil. „Instrumented Sensor System Architecture“. In: *The International Journal of Robotics Research* 17.4 (1998), S. 402–417.
- [HKU01] B. Hardekopf, K. Kwiat und S. Upadhyaya. „Secure and fault-tolerant voting in distributed systems“. In: *Proceedings of IEEE Aerospace Conference*. Bd. 3. 2001.

-
- [HS84] T. C. Henderson und E. Shilcrat. „Logical Sensor Systems“. In: *Journal of Robotic Systems* 6.2 (1984), S. 169–193.
- [Her+08] G. Heredia, A. Ollero, A. Bejar und R. Mahtani. „Sensor and actuator fault detection in small autonomous helicopters“. In: *Mechatronics* 18.2 (2008), S. 90–99.
- [Hu+03] C. Hu, W. Chen, Y. Chen und D. Liu. „Adaptive Kalman filtering for vehicle navigation“. In: *Journal of Global Positioning Systems* 2.1 (2003), S. 42–47.
- [Hun+10] L. Hundt, A. Lüder, R. Drath und B. Grimm. „Verhaltensbeschreibung mit PLCopen XML“. In: *Datenaustausch in der Anlagenplanung mit AutomationML* (2010), S. 135–193.
- [IEE97] IEEE Standards Association. *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators (IEEE 1451.2)*. 1997. URL: <http://ieeexplore.ieee.org/xpl/standards.jsp?findtitle=1451&letter=1451>.
- [Int09] International Frequency Sensor Association. *Internetumfrage - What does it mean „smart sensor“?* available at http://www.sensorsportal.com/HTML/DIGEST/E_27.htm (8.04.2011). 2009.
- [Ise05] R. Isermann. „Model-based fault-detection and diagnosis—status and applications“. In: *Annual Reviews in control* 29.1 (2005), S. 71–85.
- [Ise06] R. Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Verlag, 2006.
- [Jay94] D. Jayasimha. „Fault tolerance in a multisensor environment“. In: *Reliable Distributed Systems, 1994. Proceedings., 13th Symposium on*. 1994, S. 2–11.
- [KAN05] Klaus Dietmayer, Alexander Kirchner und Nico Kämpchen. „Fusionsarchitekturen zur Umfeldwahrnehmung für zukünftige Fahrerassistenzsysteme“. In: *Fahrerassistenzsysteme mit maschineller Wahrnehmung*. Springer, 2005, S. 59–88. URL: http://dx.doi.org/10.1007/3-540-27137-6_4.
- [KHE00] H. Kopetz, M. Holzmann und W. Elmenreich. „A universal smart transducer interface: TTP/A“. In: *Third IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*. Published by the IEEE Computer Society. Newport Beach, California, März 2000, S. 16.
- [KK90] H. Kopetz und K. Kim. „Temporal uncertainties in interactions among real-time objects“. In: *Reliable Distributed Systems, 1990. Proceedings., Ninth Symposium on*. IEEE. 1990, S. 165–174.

- [KP06] J. Kaiser und H. Piontek. „CODES: Supporting the development process in a publish/subscribe system“. In: *Proceedings of the fourth Workshop on Intelligent Solutions in Embedded Systems WISES 06*. ISBN: 3-902463-06-6. Vienna, Juni 2006, S. 1–12.
- [KPJ06] S. Kabadayi, A. Pridgen und C. Julien, Hrsg. *Virtual sensors: Abstracting data from physical sensors*. 2006.
- [KPSV03] F. Koushanfar, M. Potkonjak und A. Sangiovanni-Vincentelli. „On-line fault detection of sensor measurements“. In: *Proceedings of IEEE Sensors*. Bd. 2. 2003, S. 974–979.
- [Kah+98] H. Kahn, V. Hsu, J. Kahn und K. Pister. „Wireless communications for smart dust“. In: *Electronics Research Laboratory Technical Memorandum M98/2*. Citeseer. 1998.
- [Kai+03] J. Kaiser, C. Brudna, C. Mitidieri und C. Pereira. „COSMIC: A middleware for event-based interaction on CAN“. In: *9th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2003)*. Lisbon, Portugal, Sep. 2003.
- [Kai+08a] J. Kaiser, S. Zug, M. Schulze und H. Piontek. „Exploiting self-descriptions for checking interoperations between embedded components“. In: *International Workshop on Dependable Network Computing and Mobile Systems (DNCMS 08)*. Napoli, Italy, Okt. 2008, S. 41–45.
- [Kai+08b] J. Kaiser, M. Schulze, S. Zug, C. Cardeira und F. Carreira. „Sentient Objects for Designing and Controlling Service Robots“. In: *Proceedings of IFAC’08*. Bd. 17th International Federation of Automatic Control World Congress. Seoul, Korea, Juli 2008, S. 8315–8320.
- [Kai+09] J. Kaiser, L. B. Becker, S. Zug und M. Schulze. „Supporting independent development, deployment and co-operation of autonomous objects in distributed control systems“. In: *9th International Symposium on Autonomous Decentralized Systems (ISADS’ 09)*. Athens, Greece: IEEE Computer Society Washington, DC, USA, März 2009, S. 195–200.
- [Kie+96] R. Kieburtz u. a. „A software engineering experiment in software component generation“. In: *icse*. Published by the IEEE Computer Society. 1996, S. 542.
- [Kir+02] N. V. Kirianaki, S. Y. Yurish, N. Shpak und V. P. Deynega. *Data acquisition and signal processing for smart sensors*. Wiley, 2002.
- [Kle98] R. Kleger. *Sensorik für Praktiker*. VDE-Verl., 1998.

-
- [Kum+03] R. Kumar u. a. „DFuse: a framework for distributed data fusion“. In: *Proceedings of the 1st international conference on Embedded networked sensor systems*. Los Angeles, California, USA: ACM, 2003, S. 114–125.
- [Kum+08] R. Kumar, J. Shin, L. Iftode und U. Ramachandran. „Mobile Virtual Sensors: A Scalable Programming and Execution Framework for Smart Surveillance (Position Paper)“. In: *The Fifth Workshop on Embedded Networked Sensors, HotEmNets'08*. ACM, 2008.
- [L. +93] L. Prasad, S.S. Iyengar, R. Rao und R.L. Kashyap. „Fault-tolerant integration of abstract sensor estimates using multiresolution decomposition“. In: *Systems, Man and Cybernetics, 1993. 'Systems Engineering in the Service of Humans', Conference Proceedings., International Conference on. 1993*, 171–176 vol.5.
- [LK95] R. Luo und M. Kay. *Multisensor integration and fusion for intelligent machines and systems*. Ablex Pub, 1995.
- [LSP82] L. Lamport, R. Shostak und M. Pease. „The Byzantine generals problem“. In: *ACM Transactions on Programming Languages and Systems (TOPLAS) 4.3 (1982)*, S. 382–401.
- [Lap85] J. Laprie. „Dependable Computing and Fault-Tolerance“. In: *Digest of Papers FTCS-15 (1985)*, S. 2–11.
- [Lar+98] T. Larsen, N. Andersen, O. Ravn und N. Poulsen. „Incorporation of time delayed measurements in a discrete-time Kalman filter“. In: *Decision and Control, 1998. Proceedings of the 37th IEEE Conference on*. Bd. 4. IEEE. 1998, S. 3972–3977.
- [Lie08] J. Liebig. „Entwurf und Implementierung eines Inertialnavigationssystems bestehend aus einem Gyroskop und Beschleunigungssensoren“. Laborpraktikum. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2008.
- [Loe+04] D. Loebis, R. Sutton, J. Chudley und W. Naeem. „Adaptive tuning of a Kalman filter via fuzzy logic for an intelligent AUV navigation system“. In: *Control engineering practice 12.12 (2004)*, S. 1531–1539.
- [Mö9] M. Mörbe. „Fahrdynamik-Sensoren für FAS“. In: *Handbuch Fahrerassistenzsysteme*. Hrsg. von H. Winner, S. Hakuli und G. Wolf. Vieweg+Teubner, 2009, S. 94–109.
- [MB01] Marc van de Wal und Bram de Jager. „A review of methods for input/output selection“. In: *Automatica 37 (2001)*, S. 487–510.
- [MEU07] Marco F. Huber, Eric Stiegeler und Uwe D. Hanebeck, Hrsg. *On Sensor Scheduling in Case of Unreliable Communication*. Bremen, Germany, 2007.

- [MFH03] S. Madden, M. J. Franklin und J. M. Hellerstein. „The Design of an Accuisional Query Processor for Sensor Networks“. In: *ACM SIGMOD Conference*. 2003.
- [MM03] M. Mallick und A. Marrs. „Comparison of the KF and particle filter based out-of-sequence measurement filtering algorithms“. In: *6th International Conference on Information Fusion*. 2003, S. 422–430.
- [MZ05] M. Meo und G. Zumpano. „On the optimal sensor placement techniques for a bridge structure“. In: *Engineering Structures* 27.10 (2005), S. 1488–1497.
- [Mad+05] S. Madden, M. Franklin, J. Hellerstein und W. Hong. „TinyDB: an acquisitional query processing system for sensor networks“. In: *ACM Transactions on Database Systems (TODS)* 30.1 (2005), S. 122–173.
- [Mad08] S. Madden. *Declarative Database for Sensor Networks*. Webseite. verfügbar unter <http://telegraph.cs.berkeley.edu/tinydb/index.htm> 10.05.2011. 2008.
- [Mar90] K. Marzullo. „Tolerating Failures of Continuous-Valued Sensors“. In: *ACM Transactions on Computer Systems (TOCS)* 8.4 (Nov. 1990), S. 284–304.
- [Mat10a] T. MathWorks. *Real Time Workshop - User's Guide*. 2010. URL: [\url{http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf}](http://www.mathworks.com/access/helpdesk/help/pdf_doc/rtw/rtw_ug.pdf).
- [Mat10b] T. MathWorks. *Real Time Workshop Embedded - User's Guide*. Webseite. verfügbar unter <http://www.mathworks.de/access/helpdesk/help/toolbox/ecoder> 10.02.2010. 2010.
- [Mau+06] M. Mauthner, W. Elmenreich, A. Kirchner und D. Boesel. „Outof-sequence measurements treatment in sensor fusion applications: Buffering versus advanced algorithms“. In: *Proceedings of the 4. Workshop Fahrerassistenzsysteme Hoesslinsuelz/Loewenstein*. 2006.
- [Meu04] M. van der Meulen. „On the use of smart sensors, common cause failure and the need for diversity“. In: *6th International Symposium of Programmable Electronic Systems in Safety Related Applications*. Cologne, Germany, Mai 2004.
- [Mic11] Microsoft Corporation. *Microsoft Robotics Studio*. online, <http://msdn.microsoft.com/en-gb/library/bb881626.aspx>. 2011.
- [Mit07] H. B. Mitchell. *Multi-sensor data fusion: An introduction*. 1. Aufl. Berlin, Germany: Springer, 2007. URL: <http://www.loc.gov/catdir/enhancements/fy0825/2007926176-d.html>.

- [Mor+06] J. J. Moreau, R. Chinnici, A. Ryman und S. Weerawarana. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. Candidate Recommendation. W3C, März 2006.
- [NGM98] E. Nett, M. Gergeleit und M. Mock. „An adaptive approach to object-oriented real-time computing“. In: *First International Symposium on Object-Oriented Real-time Distributed Computing (ISORC)*. IEEE. 1998, S. 342–349.
- [NL80] P. J. Nahin und P. J. L. „NCTR Plus Sensor Fusion Equals IFFN or Can Two Plus Two Equal Five?“ In: *IEEE Transactions on Aerospace and Electronic Systems* 16.3 (1980), S. 320–327.
- [Nai00] M. D. Naish. „ELSA: A Multisensor Integration Architecture for Industrial Grading Tasks“. Magisterarb. University of British Columbia, 2000.
- [Nat05] National Instruments. *NI DAQCard-6032 Data Sheet*. 2005. URL: <http://sine.ni.com/nips/cds/print/p/lang/de/nid/11914>.
- [Nat09] National Instruments. *LabVIEW 2009 - Herstellerwebseite*. verfügbar unter <http://www.ni.com/labview/d/> am 20.02.2011. 2009.
- [Nel90] V. P. Nelson. „Fault-tolerant computing: Fundamental concepts“. In: *IEEE Computer* 23.7 (Juli 1990), S. 19–25.
- [Ni+09] K. Ni u. a. „Sensor network data fault types“. In: *ACM Transactions on Sensor Networks (TOSN)* 5.3 (2009), S. 1–29.
- [OKM04] T. Oelkers, B. Koch und D. Möller. „Field Patterns for the RobocupJunior League? - A Car-Park Problem with LEGO Mindstorms Robots“. In: *18th European Simulation Multiconference - Networked Simulations and Simulated Networks*. Hrsg. von G. Horton. The Society for Modeling and Simulation International, Juni 2004, S. 419–424.
- [Obj03] Object Management Group (OMG). *Smart Transducer Interface Specification*. Juli 2003.
- [Ope07] Open Geospatial Consortium Inc. *OpenGIS Sensor Model Language (SensorML) Implementation Specification Version 1.0.0*. 2007.
- [Ope11] Open Source Modelica Consortium OSMC. *OpenModelica - Webseite*. verfügbar unter <http://www.openmodelica.org/> am 11.01.2011. 2011.
- [PH00] S. Park und C. Han. „A nonlinear soft sensor based on multivariate smoothing procedure for quality estimation in distillation columns“. In: *Computers and Chemical Engineering* 24.2-7 (2000), S. 871–877.

- [Pat97] R. Patton. „Fault-tolerant control systems: The 1997 situation“. In: *IFAC symposium on fault detection supervision and safety for technical processes*. Bd. 3. 1997, S. 1033–1054.
- [Pep06] W. Pepels. *Produktmanagement: Produktinnovation - Markenpolitik - Programmplanung - Prozessorganisation*. 5. Aufl. Oldenbourg Wissenschaftsverlag, Mai 2006.
- [Pfa11] T. Pfahl. „Konzeption und Umsetzung eines adaptiven Fusionsalgorithmus für variierende Sensorkonfigurationen am Beispiel einer Roboterlokalisierung“. Diplomarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2011.
- [Pik+04] L. Pike, J. Maddalon, P. Miner und A. Geser. „Abstractions for fault-tolerant distributed system verification“. In: *Theorem Proving in Higher Order Logics* (2004), S. 185–193.
- [Pio07] H. Piontek. „Self-description mechanisms for embedded components in cooperative systems“. Doktorarbeit. Universität Ulm: Fakultät für Informatik, 2007.
- [Pit02] S. Pitzek. „Description mechanisms supporting the configuration and management of TTP/A fieldbus systems“. Magisterarb. Technische Universität Wien, Vienna, Austria: Institut für Technische Informatik, 2002.
- [Pow92] D. Powell. „Failure mode assumptions and assumption coverage“. In: *Fault-Tolerant Computing, 1992. FTCS-22. Digest of Papers., Twenty-Second International Symposium on*. IEEE. 1992, S. 386–395.
- [QIC01] H. Qi, S. Iyengar und K. Chakrabarty. „Distributed sensor networks - a review of recent research“. In: *Journal of the Franklin Institute* 338.6 (2001), S. 655–668.
- [QM10] M. Quinlan und R. Middleton. „Multiple model kalman filters: A localization technique for robocup soccer“. In: *RoboCup 2009: Robot Soccer World Cup XIII* (2010), S. 276–287.
- [Qui+09] M. Quigley u. a. „ROS: an open-source Robot Operating System“. In: *ICRA Workshop on Open Source Software*. 2009.
- [RB07] F. Rhéaume und A. Benaskeur. *Out-of-sequence measurements filtering using forward prediction*. Techn. Ber. Technical Report TR 2005-484, Defence R & D Canada Valcartier, 2007.
- [RN04] G. L. Rogova und V. Nimier. „Reliability in information fusion: literature survey“. In: *In Proceedings fo the 7th international Conference on Information Fusion*. 2004, S. 1158–1165.

- [RZB10] M. Rossdeutscher, M. Zuern und U. Berger. „Virtual robot program development for assembly processes using rigid-body simulation“. In: *2010 14th International Conference on Computer Supported Cooperative Work in Design (CSCWD)*. IEEE, Apr. 2010, S. 417–422.
- [Rob06] A. Robin. *Using Sensor ML to describe a Complete Weather Station*. University of Alabama, Earth System Science Center - NSSTC. Feb. 2006.
- [Rob10] RoboCupJunior Technical Committee. *RoboCupJunior Soccer Rules 2011*. online, <http://rcj.robocup.org/rcj2011/soccer2011.pdf>. 2010.
- [Ros+09] S. Rossi u. a. „Open and Standardized Resources for Smart Transducer Networking“. In: *Instrumentation and Measurement, IEEE Transactions on* 58.10 (2009), S. 3754–3761.
- [SF11] M. Schulze und M. Förster. „Ressourcengewahres Framework für Kontextinformationen in eingebetteten verteilten Systemen“. In: *Electronic Communications of the EASST (ISSN: 1863-2122)* 37 (März 2011). Hrsg. von T. Margaria, J. Padberg und G. Taentzer. Proceedings of the Workshops der wissenschaftlichen Konferenz Kommunikation in Verteilten Systemen 2011 (WowKiVS 2011), S. 12.
- [SF88] Y. Shalom und T. Fortmann. *Tracking and data association*. Bd. 179. Mathematics in science and engineering. Academic Press, Boston, 1988.
- [SGG07] A. Sharma, L. Golubchik und R. Govindan. „On the prevalence of sensor faults in real-world deployments“. In: *Sensor, Mesh and Ad Hoc Communications and Networks, 2007. SECON'07. 4th Annual IEEE Communications Society Conference on*. IEEE. 2007, S. 213–222.
- [SL08] E. Song und K. Lee. „Understanding IEEE 1451-Networked smart transducer interface standard-What is a smart transducer?“. In: *Instrumentation & Measurement Magazine, IEEE* 11.2 (2008), S. 11–17.
- [SN04] R. Siegwart und I. Nourbakhsh. *Introduction to autonomous mobile robots*. The MIT Press, 2004.
- [SPH84] E. Shilcrat, P. Panangaden und T. Henderson. *Implementing Multi-sensor Systems in a Functional Language*. Techn. Ber. Tech. Rep. UUCS-84-001, The University of Utah, 1984.
- [SS06] D. Smith und S. Singh. „Approaches to multisensor data fusion in target tracking: A survey“. In: *Knowledge and Data Engineering, IEEE Transactions on* 18.12 (2006), S. 1696–1710.
- [SS07] S. Sumathi und P. Surekha. *LabVIEW based advanced instrumentation systems*. Springer Verlag, 2007.

- [ST97] S. Joe Qin und Thomas A. Badgwell. „An Overview Of Industrial Model Predictive Control Technology“. In: *Fifth International Conference on Chemical Process Control (CACHE)*. Hrsg. von J. C. Kantor, C. E. GarcemHa und B. Carnahan. 1997, S. 232–256.
- [SY04] F. Sivrikaya und B. Yener. „Time synchronization in sensor networks: a survey“. In: *Network, IEEE* 18.4 (2004), S. 45–50.
- [SZ08] M. Schulze und S. Zug. „A Middleware based Framework for Multi-Robot Development“. In: *Proceedings of the 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC)*. Zurich, Switzerland, Okt. 2008, S. 29–31.
- [San07] V. Sankaranarayanan. *How Smart is a Sensor*. Webseite <http://www.frost.com/prod/servlet/market-insight-top.pag?docid=94884718> (12.04.2011). 2007.
- [Sch+04] J. Schilp, M. Ehrenstrasser, S. Clarke, B. Petzold und M. Zaeh. „Smart sensor application in teleoperated microassembly systems“. In: *Proceedings of SPIE*. Bd. 5263. 2004, S. 38.
- [Sch08] M. Schulze. *FAMOUSO Project Website*. <http://famouso.sourceforge.net>. [(online), as at: 25.02.2010]. 2008–2011.
- [Sch11a] M. Schappeit. „Entwicklung und Implementierung eines Fehlerdetektionskonzepts für Sensoren in verteilten Anwendungen“. Diplomarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2011.
- [Sch11b] M. Schulze. „Adaptierbare ereignisbasierte Middleware für ressourcenbeschränkte Systeme“. Doktorarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2011.
- [Sch75] R. Schneidermann. „Smart Cameras Clicking with Electronic Functions“. In: 48.17 (1975), S. 74–81.
- [Sha01] Sharp Cooperation. *GP2Y0A02YK Data Sheet*. 2001.
- [Sha07] Sharp Cooperation. *GP2D120 Data Sheet*. 2007.
- [Sin01] I. Sinclair. *Sensors and transducers*. Newnes, 2001.
- [Sou07] L. Sa de Souza. „FT-CoWiseNets: A fault tolerance framework for wireless sensor networks“. In: *International Conference on Sensor Technologies and Applications (SensorComm07)*. IEEE. 2007, S. 289–294.
- [Ste11] C. Steup. „Entwicklung eines Sensorframeworks für Systeme mit beschränkten Ressourcen“. Diplomarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2011.

- [Suk+] S. Sukumar, H. Bozdogan, D. Page, A. Koschan und M. Abidi. „Sensor Selection Using Information Complexity for Multi-sensor Mobile Robot Localization“. In: *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, S. 4158–4163.
- [Sze+04] R. Szewczyk u. a. „Habitat monitoring with sensor networks“. In: *Communications of the ACM* 47.6 (Juni 2004), S. 34–40.
- [TIB11] TIBCO Spotfire. *S+ – Herstellerwebseite*. verfügbar unter <http://spotfire.tibco.com/products/s-plus> am 11.01.2011. 2011.
- [TM03] T. Tietjen und D. H. Müller. *FMEA-Praxis*. Hanser Verlag, Aug. 2003.
- [TO98] H. Tränkler und E. Obermeier. *Sensortechnik*. Springer, 1998.
- [The10] The MathWorks. *Matlab – Herstellerwebseite*. Webseite. verfübar unter <http://www.matworks.com> am 20.02.2011. 2010.
- [The11] The Mathworks. *Simulink - Webseite*. 2011. URL: [\url{http://www.mathworks.de/products/simulink/}](http://www.mathworks.de/products/simulink/).
- [Tru] *TrueTime: Simulation of Networked and Embedded Control Systems*. [Online, as at: 1.12.2009]. Available: <http://www.control.lth.se/truetime/>. 2009.
- [VDKV00] A. Van Deursen, P. Klint und J. Visser. „Domain-specific languages: An annotated bibliography“. In: *ACM Sigplan Notices* 35.6 (2000), S. 26–36.
- [VK93] P. Veríssimo und H. Kopetz. „Design of distributed real-time systems“. In: *Distributed systems (2nd Ed.)* ACM Press/Addison-Wesley Publishing Co. 1993, S. 511–530.
- [VKC03] P. Verissimo, J. Kaiser und A. Casimiro. „An architecture to support interaction via generic events“. In: *Work in Progress Proceedings of the 24th IEEE Real-time Systems Symposium*. Cancun, Mexico, Dez. 2003.
- [VW+00] A. Vande Wouwer, N. Point, S. Porteman und M. Remy. „An approach to the selection of optimal sensor locations in distributed parameter systems“. In: *Journal of Process Control* 10.4 (2000), S. 291–300.
- [Vac+06] G. Vachtsevanos, F. Lewis, M. Roemer, A. Hess und B. Wu. *Intelligent fault diagnosis and prognosis for engineering systems*. Wiley, 2006.
- [Ver+02] P. Veríssimo u. a. „CORTEX: Towards Supporting Autonomous and Cooperating Sentient Entities“. In: *Proceedings of European Wireless 2002*. Florence, Italy, Feb. 2002, S. 595–601.
- [Ver96] Verband der Automobilindustrie e.V. *Qualitätsmanagement in der Automobilindustrie - Sicherung der Qualität vor Serieneinsatz*. 1996.

- [Vij+06] Vijay Gupta, Timothy H. Chung, Babak Hassibi und Richard M. Murray. „On a Stochastic Sensor Selection Algorithm with Applications in Sensor Scheduling and Dynamic Sensor Coverage“. In: *Automatica* 42.2 (2006), S. 251–260.
- [WMF00] K. Worden, G. Manson und N. Fieller. „Damage Detection Using Outlier Analysis“. In: *Journal of Sound and Vibration* 229 (2000), S. 647–667.
- [WPI] K. Wetzelsberger, T. Pfannschmidt und T. Ihme. „Real-time image processing for motion planning based on realistic sensor data“. In: *Robotic and Sensors Environments (ROSE), 2010 IEEE International Workshop on*. IEEE, S. 1–6.
- [Wan+05] Y. Wang u. a. „The NDIR CO₂ monitor with smart interface for global networking“. In: *Instrumentation and Measurement, IEEE Transactions on* 54.4 (2005), S. 1634–1639.
- [Wei01] D. Weiler. „Selbsttest und Fehlertoleranz mit zugelassener milder Degradation in integrierten CMOS-Sensorsystemen“. Diss. Universität Duisburg, 2001.
- [Wen03] Wenglor Sensoric. *Data sheet YT87MGV80*. 2003. URL: <http://www.q-tech.hu/pdf/Wenglor/Photoelectric20sensors/YT87MGV80.pdf>.
- [Wer09] J. Werner. „Fusion von Messungen elektronischer Kompass für Indooranwendungen“. Studienarbeit. Otto-von-Guericke Universität Magdeburg: Fakultät für Informatik, 2009.
- [Wer11] M. Werdich. *Neue FMEA Risiko Bewertung*. Webseite http://www.werdichengineering.de/images/7_Werdich_optimierte_Methoden_der_Risikoabschaetzung.pdf (12.07.2011). 2011.
- [Wil+00] S. Williams, P. Newman, G. Dissanayake und H. Durrant-Whyte. „Autonomous underwater simultaneous localisation and map building“. In: *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*. Bd. 2. IEEE. 2000, S. 1793–1798.
- [Wil] WillowGarage. *ROS Project Website*. <http://www.ros.org>. [(online), as at: 25.02.2010].
- [Wyc03] M. Wycisk. „Hochintegrierte Drucksensor für $\frac{1}{2}$ r Automotive Anwendungsbereiche“. In: *DESIGN & VERIFICATION* (Juni 2003). Hrsg. von publish industry. URL: <http://www.eue24.net/pi/index.php?StoryID=253&articleID=9464>.
- [XS02] N. Xiong und P. Svensson. „Multi-sensor management for information fusion: issues and approaches“. In: *Information fusion* 3.2 (2002), S. 163–186.

-
- [YS00] Yael Edan und Shimon Y. Nof. „Sensor economy principles and selection procedures“. In: *IIE Transactions* 32 (2000), S. 195–203.
- [ZD10] S. Zug und A. Dietrich. „Examination of Fusion Result Feedback for Fault-Tolerant and Distributed Sensor Systems“. In: *IEEE International Workshop on Robotic and Sensors Environments (ROSE 2010)*. Phoenix, AZ, USA, 2010.
- [ZDK11a] S. Zug, A. Dietrich und J. Kaiser. „An Architecture for a Dependable Distributed Sensor System“. In: *IEEE Transactions on Instrumentation and Measurement* 60 Issue 2 (Feb. 2011), S. 408–419.
- [ZDK11b] S. Zug, A. Dietrich und J. Kaiser. „Fault-Handling in Networked Sensor Systems“. In: *Fault Diagnosis in Robotic and Industrial Systems*. Hrsg. von G. Rigatos. accepted. St. Franklin, AUS: Concept Press Ltd., 2011.
- [ZJJ02] F. Zhao, S. J. und R. J. „Information-Driven Dynamic Sensor Collaboration for Target Tracking“. In: *IEEE Signal Processing Magazine* 19.2 (2002), S. 61–72.
- [ZK09] S. Zug und J. Kaiser. „An approach towards smart fault-tolerant sensors“. In: *Proceedings of the International Workshop on Robotics and Sensors Environments (ROSE 2009)*. Lecco, Italy, Nov. 2009.
- [Zig03] ZigBee Alliance. *ZigBee Specification - IEEE 802.15.4*. 2003.
- [Zil08] Zila Elektronik. *Data sheet acceleration sensor ZBS-03*. 2008. URL: www.zila.de/attachments/ZBS_03.pdf.
- [Zug+10a] S. Zug, M. Schulze, A. Dietrich und J. Kaiser. „Programming abstractions and middleware for building control systems as networks of smart sensors and actuators“. In: *Proceedings of Emerging Technologies in Factory Automation (ETFA '10)*. Bilbao, Spain, Sep. 2010.
- [Zug+10b] S. Zug, M. Schulze, A. Dietrich und J. Kaiser. „Reliable Fault-Tolerant Sensors for Distributed Systems“. In: *Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS '10)*. Cambridge, United Kingdom: ACM Press New York, NY, USA, Juli 2010, S. 105–106.
- [Zug+11] S. Zug, C. Steup, A. Dietrich und K. Brezhnyev. „Design and Implementation of a Small Size Robot Localization System“. In: *IEEE International Symposium on Robotic and Sensors Environments (ROSE 2011)*. Montreal, Quebec, Canada, Sep. 2011.