HAW HAMBURG

BACHELORTHESIS
Henry Martens

Evaluierung und Erweiterung von EdgeX in einem Fog-Computing-Paradigma

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Henry Martens

Evaluierung und Erweiterung von EdgeX in einem Fog-Computing-Paradigma

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Angewandte Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr.-Ing. Marina Tropmann-Frick

Eingereicht am: 07.10.2020

Henry Martens

Thema der Arbeit

Evaluierung und Erweiterung von EdgeX in einem Fog-Computing-Paradigma

Stichworte

Fog Computing, Edge Computing, Cloud, Internet der Dinge, EdgeX, Framework, Evaluierung, Verarbeitung komplexer Ereignisse

Kurzzusammenfassung

Die Generierung immer größerer Datenmengen durch das Internet der Dinge zieht die Aufmerksamkeit von der Cloud auf neue Verarbeitungsparadigmen wie Edge- und Fog-Computing. In dieser Arbeit wird das Fog-Computing-Framework EdgeX auf die Tauglichkeit in einem solchen Szenario untersucht. Der zweite Teil der Arbeit befasst sich mit der Entwicklung eines Systems, um die Verarbeitung komplexer Ereignisse durch den Aufbau eines Event Processing Networks innerhalb von EdgeX zu unterstützen.

Henry Martens

Title of Thesis

Evaluation and extension of EdgeX in a fog computing paradigm

Keywords

Fog Computing, Edge Computing, Cloud, Internet of Things, EdgeX, Framework, Evaluation, Complex Event Processing

Abstract

The generation of large amounts of data through the Internet of Things is drawing attention from cloud solutions to new processing paradigms like edge and fog computing. In this thesis the fog computing framework EdgeX is examied for its suitability in such a scenario. The second part of the thesis deals with the development of a system to support complex event processing by building an event processing network within EdgeX.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungen

AES Advanced Encryption Standard.

AMQP Advanced Message Queuing Protocol.

API Application Programming Interface.

AWS Amazon Web Services.

CEP Complex Event Processing.

CLoC Comment Lines of Code.

DI Dependency Injection.

EDA Event Driven Architecture.

EPA Event Processing Agent.

EPN Event Processing Network.

GQM Goal Question Metric.

HDFS Hadoop Filesystem.

HTTP Hypertext Transfer Protocol.

IaaS Infrastructure as a Service.

IEEE Institute of Electrical and Electronics Engineers.

IIoT Industrial Internet of Things.

IoT Internet of Things.

JSON JavaScript Object Notation.

JWT JSON Web Token.

LoC Lines of Code.

MOM Message Oriented Middleware.

MQTT Message Queuing Telemetry Transport.

NoSQL Not only Structured Query Language.

PaaS Platform as a Service.

REST Representational State Transfer.

SaaS Software as a Service.

SDK Software Development Kit.

TLS Transport Layer Security.

XML Extensible Markup Language.

YAML YAML Ain't Markup Language.

1 Einleitung

In diesem Kapitel soll die zugrundeliegende Problemstellung bei der Verarbeitung großer IoT-Daten erläutert werden. Die daraus abgeleiteten Ziele dieser Arbeit werden im Abschnitt 1.2 definiert. Zuletzt wird in Abschnitt 1.3 der Aufbau der Arbeit aufgezeigt, indem die enthaltenden Kapitel kurz beschrieben werden.

1.1 Problemstellung und Motivation

Durch das Voranschreiten der Digitalisierung und die umfassende Vernetzung von Geräten im Rahmen des Internets der Dinge entstehen große Mengen von Daten. Diese Daten enthalten Informationen, welche sowohl für die Wissenschaft, als auch für die Wirtschaft und Industrie von großer Bedeutung sein können. Die Verarbeitung stellt Anwendungssysteme jedoch vor neue Herausforderungen. Der Begriff Big Data umfasst heute Methoden und Werkzeuge, um die wachsende Menge von Daten kontrollieren und analysieren zu können.

Cloud-Computing bietet eine Plattform, um skalierbare und flexible Big-Data-Systeme zu nutzen. Mit steigender Datenmenge und Heterogenität der Daten wird es für bestehende Verarbeitungsparadigmen zunehmend schwieriger den Anforderungen nach Echtzeitfähigkeit der Systeme gerecht zu werden. In den letzten Jahren rückten daher gerade Arbeiten mit einem dezentralisierten Ansatz in den Fokus. Dies führte zur Entwicklung des Edge- und Fog-Computing-Paradigmas. Hier werden die Daten auf sog. Fog-Knoten verarbeitet und die Ergebnisse gegebenenfalls in eine zentrale Cloud transportiert. Es gibt verschiedene Bemühungen Frameworks für die Unterstützung dieses Verarbeitungskonzeptes zu entwickeln.

Um die Latenz, Ausführungszeit und Menge der zu übertragenden Daten zu reduzieren, muss die Verarbeitung kontinuierlich in einem unendlichen Strom von Daten geschehen. Zu diesem Zweck kommen Techniken, wie das Complex Event Processing (CEP), zum

Einsatz. Dieses betrachtet die Daten der IoT-Geräte als Ereignisse und erlaubt die deklarative Definition von Regeln zur Erkennung von Ereignismustern. Bei der Identifizierung eines solchen Musters durch die Event Processing Engine können entsprechende Reaktionen abgeleitet werden.

1.2 Zielsetzung

Um die Anforderungen des Fog-Computing-Paradigmas zu erfüllen, sind in den letzten Jahren eine Reihe von Open-Source-Frameworks entstanden. EdgeX ist eines dieser Projekte. Es zielt auf den Einsatz in heterogenen IoT-Szenarien, um die Daten nahe der Datenquellen zu verarbeiten. Gegenstand dieser Arbeit ist eine kritische Betrachtung und Evaluierung der EdgeX Foundry nach dem GQM-Paradigma. Im zweiten Teil der Arbeit soll das EdgeX-System um eine weitere Komponente zum Aufbau eines Event Processing Networks erweitert werden. Dazu soll eine entsprechende Softwarearchitektur entworfen werden, welche die Anforderungen von Fog-Computing-Systemen im Allgemeinen, sowie die Ergebnisse der Evaluation berücksichtigt. Das implementierte System soll im Kontext von EdgeX in einem Fog-Computing-Umfeld getestet werden.

1.3 Aufbau der Arbeit

Diese Arbeit unterteilt sich in fünf Kapitel. In Kapitel 2 wird zunächst das Umfeld von Fog-Computing und die Ansätze für die Verarbeitung von Big Data beschrieben. Weiterhin werden die Techniken des Complex Event Processing erläutert und das GQM-Paradigma als Möglichkeit zur strukturierten Durchführung einer Evaluation vorgestellt. Das Kapitel 3 befasst sich mit der Evaluierung der EdgeX Foundry nach dem GQM-Paradigma. Hier soll die Eignung von EdgeX für den Einsatz in einem Fog-Computing-System innerhalb eines IoT-Umfelds untersucht werden. In Kapitel 4 wird ein System geplant und entwickelt, um ein Event Processing Network aufzubauen und eine Ereignisverarbeitung mit deklarativen Regeln zu unterstützen. Im Anschluss soll diese Erweiterung getestet und im Kontext von EdgeX evaluiert werden.

2 Grundlagen

Die Generierung von Big Data durch das Internet der Dinge, sowie der Umgang mit dieser speziellen Art von Daten haben bereits in der Vergangenheit zu der Notwendigkeit von neuen Technologien und Lösungen geführt. Im Teilabschnitt 2.1 wird auf die Eigenschaften des Internets der Dinge, sowie die Anforderungen an Softwarelösungen in diesem Bereich eingegangen. Der Abschnitt 2.2 beschäftigt sich mit den Eigenschaften von Big Data und welche Herausforderungen im Bezug auf den Umgang mit diesen Daten bestehen. Im Abschnitt 2.3 wird auf die Vorteile von Cloud Computing beim Umgang mit Big Data eingegangen und welche Probleme die Cloud bei der Verarbeitung von Daten aus dem Internet der Dinge aufweist. Der Teilabschnitt 2.4 befasst sich mit dem Paradigma des Edge- bzw. Fog-Computings und welche Lösungen es für die Probleme der Cloud bietet. Es wird auf die benötigten Eigenschaften von Fog-Computing-Systemen eingegangen und Ansätze zur Entwicklung einer Referenzarchitektur vorgestellt. Zusätzlich beschäftigt sich das Kapitel mit dem für diese Arbeit gewählten EdgeX-Framework und untersucht dessen Architektur und Arbeitsweise. Im Abschnitt 2.5 wird Complex Event Processing als Möglichkeit zur kontinuierlichen Auswertung von Datenströmen erläutert. Abschließend wird im Abschnitt 2.6 ein mögliches Vorgehen zur Evaluierung von Systemen vorgestellt.

2.1 Internet of Things (IoT)

Das Internet der Dinge (IoT) ist ein neuartiges Paradigma, dessen Vision es ist große Mengen von intelligenten Geräten über ein weltweites Netzwerk miteinander zu verbinden [Ast09]. Es wird eine vollständige Interoperabilität dieser verbundenen Geräte angestrebt, um somit eine Synergie durch den Austausch von Daten innerhalb des Netzwerkes zu schaffen [AIM10].

Einen Sonderfall stellt dabei das sog. Industrial Internet of Things (IIoT) dar, welches die Nutzung des IoT in einem industriellen Kontext beschreibt [MVM16]. Dabei werden große Mengen an Daten erzeugt, die einen hohen Wert für das Unternehmen haben. Diese Daten werden üblicherweise im Unternehmen oder bei einem Drittanbieter gespeichert und ausgewertet, um Informationen daraus abzuleiten. Gewonnene Informationen können nicht nur zur Steuerung der Geräte, sondern auch zur Optimierung der Prozesse genutzt werden. Die Hauptantriebskräfte für das IoT sind die sinkenden Kosten für Rechenleistung und Speicher, die Vermehrung von Sensoren und anderen Geräten des IoT sowie die Möglichkeit für elastische Big-Data-Analysen in der Cloud. Für den Einsatz gibt es ein hohes Potential und viele Anwendungsfälle. Zu den wohl bekanntesten Einsatzgebieten von IoT zählen E-Health, Smart Home, Smart City, sowie autonomes Fahren. IoT findet jedoch auch Anwendung in der erweiterten oder virtuellen Realität und in der Unterstützung und Steuerung der industriellen Fertigung [AIM10].

Die Geräte innerhalb des IoT reichen von simplen Sensoren und Aktuatoren, bis zu Wearables, mobilen Endgeräten und Einplatinencomputern. Diese erfassen bzw. steuern bestimmte Bereiche des alltäglichen Lebens oder der industriellen Fertigung. Die Geräte bekommen im IoT eine Identität und können über definierte Schnittstellen mit ihrer Umgebung kommunizieren [AIM10]. Sie stellen ihre Funktionen über ihre eigenen Dialekte und Protokolle zur Verfügung.

Der Einsatz von IoT-Geräten stellt Entwickler und Betreiber vor neue Herausforderungen, um die entstehenden Daten auswerten und nutzen zu können. Geräte des IoT werden durch eine geringe Rechen- und Energiekapazität charakterisiert. Daher muss neben den offensichtlichen Skalierungsproblemen, die durch die große Menge an weit verteilten Geräten und Daten verursacht werden, auch die Ressourceneffizienz beachtet werden [AIM10]. Anwendungen müssen zudem die Mobilität und geographische Verteilung von Geräten des IoT berücksichtigen, damit diese effizient genutzt werden können [BMZA12].

Durch die Erweiterung des IoT um zusätzliche Geräte entstehen große Mengen heterogener Daten, deren Verwaltung und Visualisierung eine der größten Herausforderungen für dieses Paradigma darstellt. Die relevanten Daten machen dabei nur einen verhältnismäßig geringen Teil der tatsächlich entstandenen Datenmenge aus [CML14]. Die Menge und Heterogenität der Daten führen dazu, dass diese mit herkömmlichen Lösungen nicht mehr effizient gespeichert und ausgewertet werden können. Für den Umgang mit Daten des IoT wird daher zunehmend auf Technologien und Verarbeitungskonzepte aus dem Bereich des Big Data zurückgegriffen.

2.2 Big Data

Neue Technologien wie zum Beispiel das Internet der Dinge oder auch soziale Netzwerke haben zu einem enormen Fluss an strukturierten und unstrukturierten Daten geführt [HYA+15]. Diese spezielle Art von Daten wird unter dem Begriff Big Data zusammengefasst. Herkömmliche Soft- und Hardwarelösungen können diese Daten nicht mehr in einer tolerierbaren Zeit verwalten und verarbeiten [CML14], weshalb neue Techniken und Paradigmen nötig sind. Big Data wird oftmals durch das 3V-Modell beschrieben, dieses umfasst [Lan01]:

- Volume: Großen Mengen von Daten
- Velocity: Hohe Erzeugungs- und Verarbeitungsgeschwindigkeit von Daten
- Variety: Starke Heterogenität von Daten ohne feste Struktur oder Schema

In modernen Definitionen von Big Data wird dieses Modell um den Begriff *Value* erweitert und bildet somit das 4V-Modell. Dadurch wird der Nutzen und der Wert der Informationen betont, die aus Big Data gewonnen werden können [CML14]. Diesen Wert zu generieren stellt eine große Herausforderung beim Umgang mit Big Data dar.

Auf Grund der natürlichen Grenzen von herkömmlichen Methoden und Techniken zur Analyse von Daten reichen diese für die Verarbeitung von Big Data nicht mehr aus, sodass spezielle Big-Data-Analysen notwendig werden. Durch die Anwendung von Big-Data-Analysen können im Datenbestand Muster erkannt, Korrelationen entdeckt und andere Einsichten gewährt werden [MJMRP19]. Sie können dabei helfen einen Nutzen aus Big Data zu generieren, um somit einen Wert für Unternehmen zu schaffen, indem sie die Entscheidungsfindung unterstützen und neue Arten von Produkten ermöglichen [MJMRP19]. Das IoT generiert große Mengen an Daten in einer hohen Frequenz. Diese Daten sind von Natur aus heterogen, unstrukturiert, verrauscht und redundant [CML14]. Sie entsprechen somit der Definition von Big Data nach dem 3V-Modell.

Die Verarbeitung und Analyse von Big Data kann auf unterschiedliche Weise durchgeführt werden. Analysen können über den gesamten Datenbestand (Batch-Mode) oder über einen kontinuierlichen Strom von Daten (Streamline-Mode) durchgeführt werden. Der Big-Data-Analyse-Prozess besteht dabei aus den Schritten Datenakquisition, Datenbereinigung und Feature-Extraktion, Datenintegration, Modellierung und Analyse sowie der Interpretation der Ergebnisse [MJMRP19]. Für die unterschiedlichen Phasen

werden passende Werkzeuge und Anwendungen verwendet. Klassische Ansätze, wie die Speicherung in relationalen Datenbanken, reichen auf Grund der Eigenschaften von Big Data oftmals nicht mehr aus. Hier wird vermehrt auf verteilte Konzepte zur Verarbeitung und Speicherung der Daten gesetzt. Dazu zählen zum Beispiel das verteilte Hadoop Filesystem (HDFS) oder NoSQL-Datenbanken. Die Flexibilität einer abstrahierten Cloud-Infrastruktur macht diese für den Einsatz im Big-Data-Analyse-Prozess besonders attraktiv.

2.3 Cloud Computing

Cloud Computing bezeichnet ein zentralisiertes Verarbeitungsparadigma, welches einen Fokus auf die dynamische Bereitstellung von Rechen- und Speicherressourcen legt. Diese Ressourcen können schnell bereitgestellt und freigegeben werden, was eine einfache und automatisierte Skalierung von Anwendungen und Daten erlaubt [MG11]. Die verfügbaren Ressourcen können dabei von heterogenen Clients über ein Netzwerk erreicht werden.

Die Bereitstellung der Ressourcen einer Cloud wird in drei Service-Modellen unterschieden [MG11]:

- Software as a Service (SaaS): Anwendungen werden direkt über die Infrastruktur des Cloud-Anbieters verfügbar gemacht und können von heterogenen Clients erreicht werden. Der Nutzer hat keinen Einfluss auf die unterliegende Infrastruktur oder die Kapazitäten der Anwendung.
- Platform as a Service (PaaS): Der Nutzer kann eigene Anwendungen auf die Cloud Infrastruktur verteilen und Programmiersprachen, Bibliotheken, sowie Laufzeitumgebungen vom Anbieter zu nutzen. Die physikalischen Server und das Netzwerk werden hier vom Cloud-Anbieter verwaltet, während der Nutzer die Anwendungen und Einstellungen der Anwendungsumgebung an die eigenen Bedürfnisse anpassen kann.
- Infrastructure as a Service (IaaS): Ressourcen der Cloud, wie Rechenleistung, Speicher oder Netzwerk werden bereitgestellt. Der Nutzer kann beliebige Software installieren und ausführen. Er hat die Kontrolle über die eingesetzten Betriebssysteme, Speicher, Anwendungen und eingeschränkte Kontrolle über Knoten im Netzwerk, wie zum Beispiel Proxies oder Firewalls.

Die Art, wie die Ressourcen der Cloud für die Nutzer bereitgestellt werden, wird in folgende Bereitstellungsmodelle unterteilt [MG11]:

- Public Cloud: Die Public Cloud stellt ihre Infrastruktur für eine allgemeine Nutzung über ein Netzwerk bereit. Sie wird meist von einem Unternehmen angeboten, welches die Ressourcen der Cloud verwaltet. Beispiele für eine Public Cloud sind die Google Cloud Plattform, Amazon Web Services (AWS) oder die Microsoft-Cloud Azure.
- Private Cloud: Eine Private Cloud stellt ihre Infrastruktur exklusiv für ein Unternehmen zur Nutzung bereit. Sie wird entweder von diesem Unternehmen selbst betrieben und verwaltet oder von einem externen Anbieter.
- Community Cloud: Die Community Cloud stellt ihre Infrastruktur einer Gruppe von Nutzern mit gemeinsamen Interessen zur Verfügung. Sie wird von einem oder mehreren Mitgliedern/Unternehmen dieser Gruppe betrieben und verwaltet. Alternativ kann die Community Cloud wie die Private Cloud von einem externen Anbieter betrieben und bereitgestellt werden.
- Hybrid Cloud: Die Hybrid Cloud ist ein Verbund von zwei oder mehreren Cloud-Infrastrukturen mit unterschiedlichen Bereitstellungsmodellen. Diese bleiben eigenständige Einheiten, bieten jedoch einen standardisierten Zugriff, sowie Daten- und Anwendungsportabilität.

Big Data und Cloud Computing sind eng miteinander verknüpft. Cloud Computing schafft die Infrastruktur für die Methoden von Big Data, um große und verteilte Datensätze auf eine effiziente und schnelle Weise auszuwerten [HYA+15]. Dafür werden die Daten in eine Cloud-Umgebung transportiert und in der verteilten Infrastruktur persistiert. Big Data schafft nun die Methoden und Werkzeuge, um mit diesen verteilten und heterogenen Daten arbeiten zu können und Auswertungen vorzunehmen. Ein Beispiel für einen solchen Aufbau findet sich in Abbildung 2.1. Der Nutzer kann über eine Schnittstelle auf die Daten und die Funktionen innerhalb der Cloud zugreifen. Auf Grund der Eigenschaften der Cloud, muss der Nutzer keine Entscheidungen über die interne Verwaltung der Server und des Netzwerks treffen und kann die Ressourcen über das Internet nutzen.

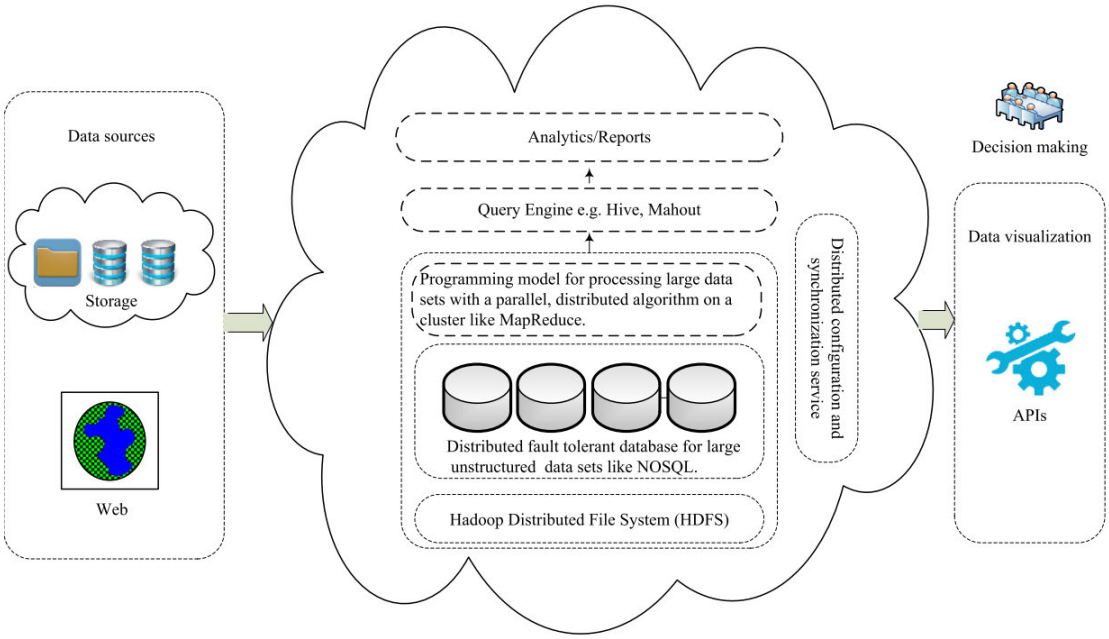

Abbildung 2.1: Big-Data-Lösungen in der Cloud [HYA+15]

Obwohl Cloud Computing die Ressourcennutzung abstrahiert und optimiert, stellt es keine effiziente Lösung für den Umgang mit Daten dar, die vom IoT generiert werden [DGC+16]. IoT-Geräte, wie zum Beispiel Sensoren, können Daten in großen Mengen und hohen Frequenzen generieren. Damit diese Daten in einer zentralisierten Cloud verarbeitet und analysiert werden können, müssen sie zunächst über das Netzwerk dorthin transportiert werden. Dieser Transport kostet Zeit, Durchsatz und Energie. Eine mögliche Entfernung zu den IoT-Geräten sorgt außerdem für eine Latenz, welche kritische Entscheidungen verzögert. Die Integration großer Mengen heterogener Daten kann unter Umständen aufwändig sein. Dieser Umstand wird dadurch erschwert, dass Daten aus dem IoT häufig Redundanzen und Rauschen enthalten und nur ein relativ geringer Teil der Daten tatsächlich für eine Entscheidungsfindung benötigt wird. Zudem sind gerade in einer Public Cloud besondere Strategien nötig, um die Sicherheit, Integrität und Privatsphäre der Daten sicherzustellen [HFRS17]. Diese Probleme führten zu Konzepten und Ideen über neuartige Verarbeitungsparadigmen, welche die Herausforderungen beim Umgang mit Big Data lösen sollen.

2.4 Edge- und Fog-Computing

Das Aufkommen der Cloud hatte einen großen Einfluss auf die Entwicklung neuer Anwendungen für den Umgang mit Big Data. Obwohl Cloud Computing in vielen Bereichen bereits etabliert ist und eingesetzt wird, gab es gerade in den letzten Jahren vermehrt Bemühungen einen dezentralisierten Ansatz zu entwickeln, um die Probleme der Cloud anzugehen. Zu diesen Problemen zählen unter anderem die hohen Latenzen für den Transport der Daten in die Cloud und fehlende Möglichkeiten zur Verteilung von Anwendungen an die Quellen der Daten. Diese Arbeiten haben zur Entwicklung des Edge- bzw. Fog-Computing-Paradigmas geführt.

2.4.1 Definition der Begriffe

Die Community hat sich noch nicht auf eine einheitliche Definition der Begriffe Edge- und Fog-Computing geeinigt. In vielen Arbeiten werden diese beiden Begriffe als Synonyme verwendet. In einer der ersten Arbeiten zu diesem Thema definiert Bonomi Fog-Computing als eine virtualisierte Plattform, welche Rechenleistung, Speicher und Netzwerkservices zwischen den Endgeräten und der Cloud bereitstellt [BMZA12]. Dieser Ansatz hat das Ziel die Distanz zwischen den Quellen der Daten (z.B. IoT-Geräte) und den Verarbeitungseinheiten zu minimieren. Diese Verarbeitungseinheiten werden Fog-Knoten genannt und bestehen aus den gleichen Komponenten wie die Cloud mit dem Unterschied, dass sie verteilt am Rande des Netzwerkes gelegen sind [MJMRP19]. Die Fog-Knoten dienen zur Vorverarbeitung und Filterung von Daten, bevor diese zu einer entfernten Cloud gesendet werden. Dadurch können die Latenz, Ausführungszeit und die Menge der über das Netzwerk übertragenen Daten reduziert werden. Fog Computing ermöglicht somit neue Arten von Anwendungen, sowie ein Zusammenspiel mit der Cloud, um die Verwaltung und Analyse von Daten auf eine möglichst effiziente Weise umzusetzen.

Die Abgrenzung zwischen den Begriffen Edge Computing und Fog Computing ist nicht klar definiert. In der folgenden Arbeit wird die Definition des OpenFog-Consortiums verwendet [Ope17]. Demnach besteht der Unterschied darin, dass Fog Computing einen hierarchischen Ansatz verfolgt, um Rechenleistung, Speicher und Netzwerkservices irgendwo zwischen der Cloud und den Geräten bereit zu stellen. Edge Computing ist dagegen limitiert auf eine Verwendung am Rande des Netzwerkes, um so nahe an den Geräten wie möglich zu agieren.

2.4.2 Eigenschaften von Fog Computing

Die wesentlichen Eigenschaften von Fog Computing wurden bereits 2012 von Bonomi in seiner wegweisenden Arbeit über die Verwendung von Fog Computing im Zusammenhang mit IoT definiert [BMZA12]:

- Geringe Latenzen: Endpunkte werden mit mächtigen Funktionalitäten unterstützt, um einen Teil der Verarbeitung nahe der Datenquellen zu erledigen. Dadurch sinkt die Latenz und kritische Entscheidungen sind schneller möglich.
- Geographische Verteilung: Fog-Anwendungen werden weitläufig im Netzwerk verteilt, um eine Nähe zu den Datenquellen zu schaffen.
- Mobilitätsunterstützung: Viele Fog-Anwendungen müssen mit mobilen Geräten kommunizieren und somit entsprechende Techniken unterstützen. Zu diesen Techniken gehören spezielle Protokolle und Algorithmen zur dynamischen Erkennung von Geräten.
- Interoperabilität: Fog Computing muss mit anderen Services und Anwendungen zusammenarbeiten können, dazu zählen unter anderem verschiedene Cloud-Anbieter.
- Heterogenität: Fog-Knoten müssen in unterschiedlichen Umgebungen lauffähig sein. Außerdem müssen Fog-Anwendungen mit einer Vielzahl von heterogenen Geräten und Datenquellen umgehen können.

Da Fog Computing nach wie vor ein aktuelles Forschungsthema ist und der Begriff nicht eindeutig definiert wurde, existieren verschiedene Interpretationen über die Eigenschaften von Fog-Computing-Systemen. In der folgenden Arbeit wird mit den hier genannten Kerneigenschaften gearbeitet, welche in den meisten Definitionen genannt werden. Das OpenFog-Consortium versucht in seinem Glossar die wichtigsten Begriffe zu erläutern, um eine Grundlage für zukünftige Arbeiten in diesem Bereich zu schaffen [Ope17]. Um die hier genannten Eigenschaften und Anforderungen umzusetzen, wurden in verschiedenen Arbeiten unterschiedliche Referenzarchitekturen für Fog-Anwendungen entworfen.

2.4.3 Referenzarchitekturen

Die Entwicklung einer Referenzarchitektur für Fog-Anwendungen ist ein aktuelles Forschungsthema. Die meisten vorgeschlagenen Architekturen sehen eine Aufteilung in Schich-

ten vor und unterscheiden sich im Bezug auf die Anzahl und Funktion dieser Schichten. Der ursprüngliche Ansatz sieht eine Aufteilung in drei solcher Schichten vor [BMZA12, NS17]. Die untere Schicht enthält dabei die verbundenen Geräte, wie zum Beispiel Sensoren oder Aktuatoren. Die darüber liegende Schicht besteht aus den Fog-Knoten, welche die Daten der Geräte vorverarbeiten und die gefilterten Daten an die dritte Schicht weiterleiten. Auf der dritten Schicht befindet sich die Cloud. Hier werden die Daten dauerhaft gespeichert und Ressourcen für komplexe Analysen auf diesen Daten bereitgestellt. Diese Architektur wurde in Abbildung 2.2 visualisiert.

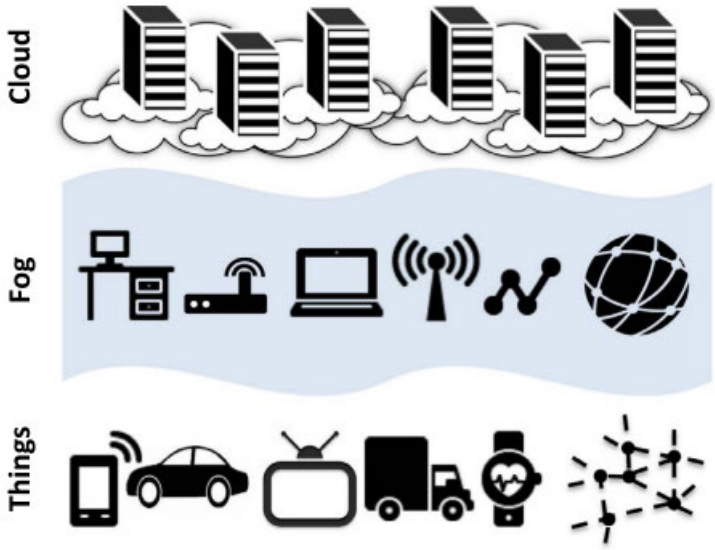

Abbildung 2.2: 3-Schichten-Architektur für Fog-Computing [ADP17]

Andere Arbeiten sehen die Verwendung von vier Schichten vor [ADP17]. Zu den bereits genannten drei Schichten der vorherigen Architekturen, gibt es hier noch eine zusätzliche vertikale Schicht. Diese Schicht gibt Nutzern die Möglichkeit mit den Services und Geräten auf allen drei Schichten zu interagieren. Im August 2018 hat das Institute of Electrical and Electronics Engineers (IEEE) die OpenFog-Architektur als Standard-Referenzarchitektur unter IEEE 1934-2018 bekannt gegeben [IEE18]. Diese Architektur folgt der Drei-Schichten-Architektur von Bonomi, wobei sich die Fog-Schicht aus vier Subschichten zusammensetzt [Ope17]. Diese sind Platform, Node Management and Software Backplane, Application Support und Application Services. Abbildung 2.3 zeigt eine Übersicht über diese Schichten.

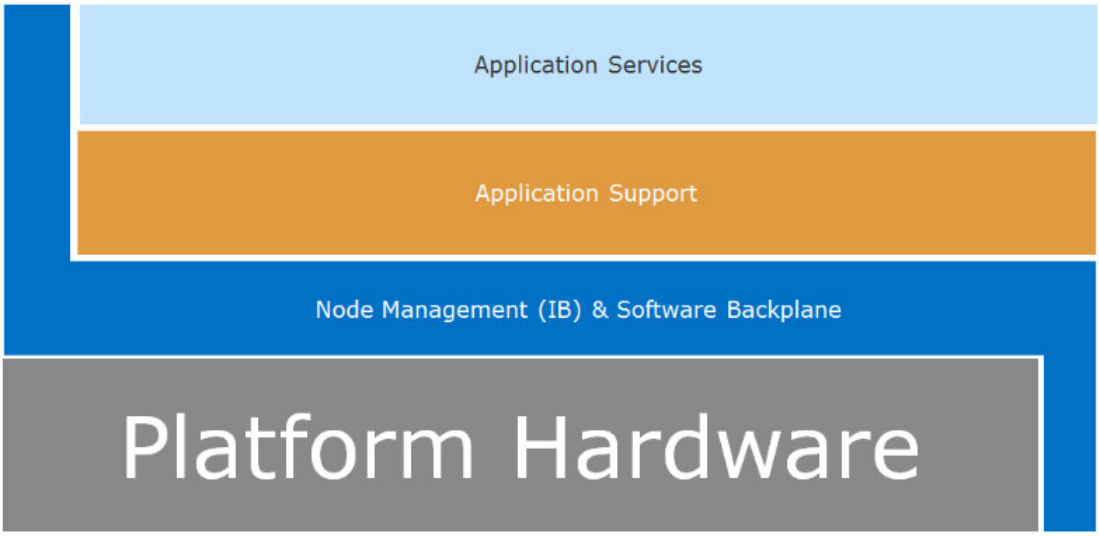

Abbildung 2.3: OpenFog-Referenzarchitektur [Ope17]

Die Platform-Hardware-Schicht enthält die physikalische Hardware des Fog-Knoten. Die darüberliegende Schicht ist für die allgemeine Verwaltung und Kommunikation der Knoten zuständig, dazu zählen sowohl Geräte als auch Cloud-Instanzen und andere Fog-Knoten. Die Application-Support-Schicht wurde als Sammlung von Microservices geplant, welche Funktionalitäten anbieten, die anwendungsneutral sind. Dazu zählen unterschiedliche Arten von Werkzeugen, sowie Datenbanken oder Sicherheitsmechanismen. Die Application-Services-Schicht bietet Funktionen und Schnittstellen für Anwendungen an, um die anwendungsspezifischen Ziele in einer Fog-Umgebung umzusetzen.

2.4.4 Existierende Frameworks

Edge- und Fog-Computing erregte gerade in den letzten Jahren vermehrt die Aufmerksamkeit in der Wissenschaft und der Industrie. In verschiedenen Projekten entstehen derzeit Frameworks, welche für einen Einsatz auf Fog-Knoten spezialisiert sind und die Eigenschaften des Paradigmas unterstützen sollen. Die meisten dieser Projekte stehen unter einer Open-Source-Lizenz und werden von einer Community entwickelt. Zu den wichtigsten Frameworks in diesem Bereich zählen:

- EdgeX Foundry: Die EdgeX Foundry ist ein von der Linux Foundation verwaltetes Open-Source-Projekt, welches ein anwendungsneutrales Framework für die Verar-

beitung von Daten auf Fog-Knoten ermöglicht. Das System kann über SDKs um eigene Microservices erweitert werden [Fou20d].

- FogFlow: FogFlow ist ein Open-Source-Framework zur Orchestrierung von Prozessen für die Datenverarbeitung über Knoten auf der Cloud- und Fog-Schicht. Dies geschieht automatisch auf Basis mehrerer Kriterien, wie System-Ressourcen, Metadaten oder QoS-Anforderungen (z.B. Latenz, Bandbreite) [Fog20].

- Apache Edgent: Edgent bietet die Möglichkeit Verarbeitungsvorschriften auf beliebige Knoten zu verteilen, um somit eine Analyse von kontinuierlichen Datenströmen in Echtzeit zu ermöglichen. Diese Verteilung wird durch den Nutzer gesteuert und dient der Vorverarbeitung von Daten nahe der IoT-Geräte [Fou20a].

- Microsoft IoT Edge: Azure IoT Edge verpackt Analysen und Geschäftslogik in Standardcontainer und kann diese auf verschiedene Geräte verteilen. Die Ergebnisse dieser Container können in die Cloud (MS Azure) übertragen werden und die gesamte Infrastruktur kann über die Cloud überwacht werden [Cor20].

- KubeEdge: KubeEdge erweitert die Orchestrierungsfunktionen von Kubernetes und dient zur Verteilung von Docker-Containern auf Edge- bzw. Fog-Knoten. Es bietet eine Infrastrukturunterstützung für Netzwerke, App-Deployment und die Synchronisation von Metadaten zwischen Cloud und Edge [Kub20].

- Eclipse ioFog: ioFog umfasst einen Agent, welcher auf jedem Edge-Knoten läuft und eine Plattform für die Ausführung von Microservices innerhalb eines Docker-Containers bietet. Der Controller bietet eine Möglichkeit zur Verwaltung der verteilten Microservices. Die Kommunikation zwischen den Microservices geschieht über Message-Broker [Fou20b].

- Eclipse Kura: Kura bietet eine Schnittstelle für den Zugriff auf Hardware in IoT-Gateways. Dafür unterstützt es wichtige Protokolle, wie z.B. Modbus oder S7. Es bietet außerdem die Möglichkeit der visuellen Datenflussprogrammierung, um Daten abzurufen, auf dem Edge-Knoten zu verarbeiten und per MQTT in die IoT-Cloud-Plattform eines Cloud-Anbieters zu schicken [Fou20c].

- Baetyl: Baetyl ist ein Open-Source-Framework, welches genau wie EdgeX von der Linux Foundation verwaltet wird. Es nutzt Kubernetes, um Anwendungen in Containern am Rand des Netzwerks auszuführen. Die Verwaltung von Knoten und Anwendungen, sowie die Konfiguration kann komplett in der Cloud erfolgen [Edg20].

- macchina.io: macchina.io bietet eine Umgebung für IoT-Systeme. Durch die Just-in-time-Kompilierung von JavaScript-Code können Daten auf den jeweiligen Edge- bzw. Fog-Knoten gesammelt, gefiltert und analysiert werden [Gmb20].

Diese Arbeit legt den Fokus auf eine kritische Betrachtung der EdgeX Foundry. EdgeX ist ein anwendungsneutrales, leichtgewichtiges Open-Source-Framework, welches von der Linux Foundation[1] bereitgestellt wird. Es unterstützt den Aufbau von Edge- und Fog-Computing-Anwendungen im Umfeld von IIoT. EdgeX besteht aus einer Menge von lose gekoppelten Microservices, welche mit Hilfe von SDKs um zusätzliche Services erweitert werden können [Fou20d]. Dies soll es ermöglichen verschiedene Arten von Anwendungen, wie zum Beispiel Machine-Learning-Algorithmen sowie Entscheidungen direkt am Rande des Netzwerks und somit nahe der Datenquellen auszuführen. EdgeX agiert dabei sowohl auf Edge- als auch auf Fog-Ebene und unterstützt einen hierarchischen Aufbau, wie in Abbildung 2.4 zu sehen ist.

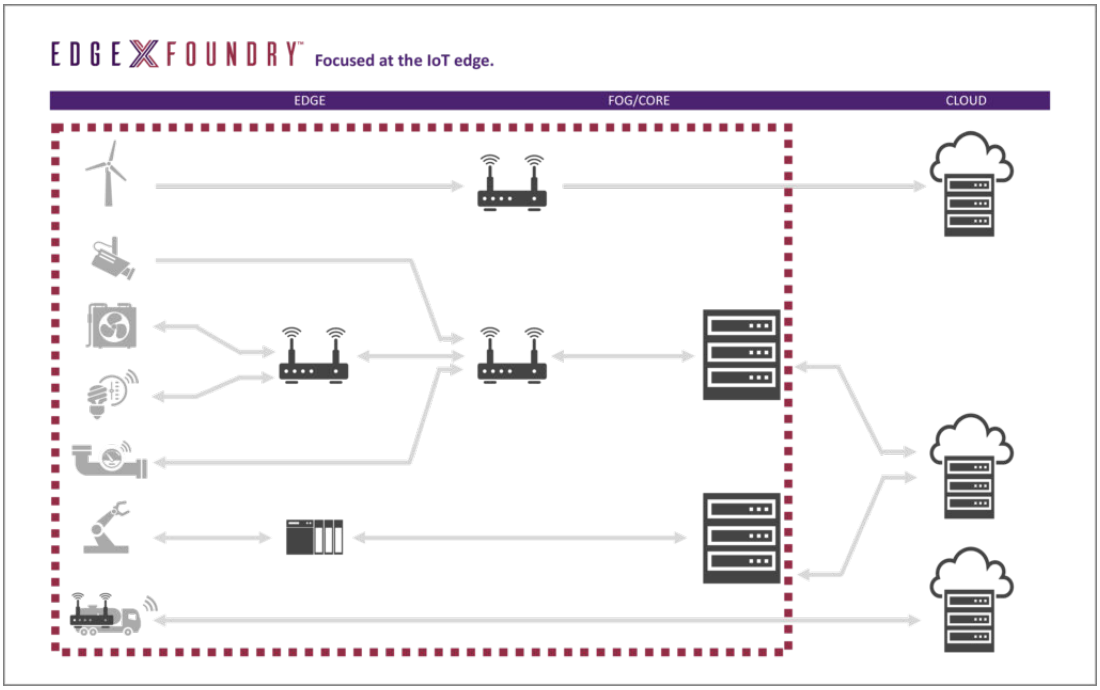

Abbildung 2.4: Nutzung von EdgeX auf Edge- und Fog-Ebene [Fou20d]

Durch seine leichtgewichtige Architektur soll EdgeX auf verschiedenen Arten von Edge- bzw. Fog-Knoten, wie zum Beispiel eingebetteten Rechnern oder Routern, ausführbar

[1] https://www.linuxfoundation.org/

sein. Dadurch soll die Anwendung auch über mehrere Knoten im Netzwerk verteilt werden können, um der geographischen Verteilung des IoT gerecht zu werden. Die Services sollen durch Virtualisierung unabhängig vom unterliegenden Betriebssystem (z.B. Linux, Windows, Mac OS) und der Prozessorarchitektur (z.B. ARM, x86) lauffähig sein.

Architektur von EdgeX

EdgeX Foundry besitzt eine lose gekoppelte Microservice-Architektur. Diese Microservices werden auf Basis ihrer jeweiligen Funktionalität in verschiedene Schichten unterteilt. Ein Überblick über die Schichten und Services kann in Abbildung 2.5 gesehen werden.

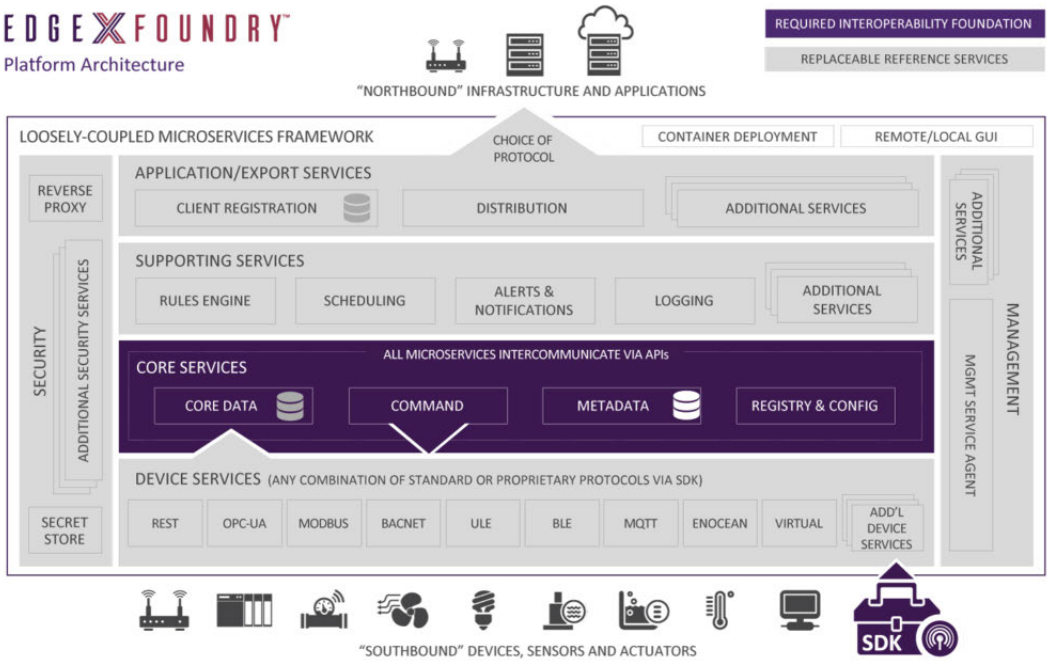

Abbildung 2.5: EdgeX-Foundry's Microservice-Architektur [Fou20d]

Die Architektur von EdgeX basiert auf der unter 2.4.3 vorgestellten Schichten-Architektur von Bonomi [BMZA12]. Die verbundenen Geräte befinden sich unterhalb (Southbound) der EdgeX-Architektur und bilden die Edge-Schicht. Die Cloud-Systeme oder übergreifenden Anwendungen befinden sich oberhalb (Northbound) der EdgeX-Architektur und bilden die Cloud-Schicht. EdgeX selbst befindet sich also in der Fog-Schicht der von Bonomi vorgestellten Referenzarchitektur.

EdgeX selbst besteht aus vier Service-Schichten und zwei vertikalen Schichten für Sicherheit und Systemverwaltung. Die sog. Device Services kommunizieren mit den Geräten und IoT-Objekten. Sie können ein oder mehrere Geräte sowie Instanzen anderer Systeme verwalten, welche sich als Geräte verhalten. Die Device Services kommunizieren über unterschiedliche Protokolle und sind über ein SDK um weitere Services erweiterbar, um so verschiedene Arten von Geräten und Protokollen zu unterstützen. Die Core Services beinhalten wichtige Microservices und bilden den Kern der EdgeX-Anwendung. Hier findet sich der temporäre Speicher (Core Data) für die Daten von den Geräten, sowie ein Speicher für die Metadaten verbundener Geräte (Metadata). Über den Microservice Registry/Configuration können Microservices mit Informationen über andere Microservices versorgt werden. Jeder Service registriert sich hier und erhält seine benötigten Konfigurationsinformationen von diesem Service. Der Command-Service ist der zentrale Punkt, um Kommandos und Befehle an die angeschlossenen Geräte zu senden. Er nutzt dafür die vereinheitlichten Schnittstellen der Device-Services und funktioniert unabhängig von den Protokollen der Geräte. Die Supporting Services enthalten Microservices, die für Analysen am Rande des Netzwerks benötigt werden. Hier werden einige allgemeine Funktionalitäten, wie zum Beispiel Logging oder Datenbereinigung, angeboten. Die Supporting Services enthalten außerdem eine einfache Rules-Engine, welche es erlaubt Entscheidungen auf einem Edge- bzw. Fog-Knoten nahe der Datenquellen zu treffen. Die Export Services stellen die Verbindung zu den Northbound-Systemen (z.B. der Cloud) sicher. Zur Zeit registrieren sich Systeme über den Client-Registration-Microservice bei EdgeX und geben dabei an, wo und zu welchem Zeitpunkt die Daten der Geräte empfangen werden sollen. Zusätzlich können diese spezifizieren, welche Daten aus EdgeX exportiert werden sollen und in welches Format diese transformiert werden. Der Distribution-Microservice sendet anschließend die spezifizierten Daten zur angegebenen Zeit an die registrierten Northbound-Systeme. Wenn viele Northbound-Systeme an eine EdgeX-Instanz angeschlossen werden, stellen diese beiden Microservices einen potentiellen Flaschenhals dar. Daher sollen Northbound-Systeme in Zukunft über Application-Services angebunden werden. Die Application-Services basieren auf dem Prinzip von Funktionspipelines und dienen dazu Daten zu extrahieren, prozessieren, transformieren und über ein spezifisches Protokoll zu einem Endpunkt zu transportieren. Für die Erstellung einer solchen Pipeline dient das Application Functions SDK, welches vordefinierte Funktionen bereitstellt und um eigene Funktionen erweitert werden kann.

Funktionsweise von EdgeX

Geräte kommunizieren über unterschiedliche Protokolle und werden über die Device-Services an EdgeX angebunden. Die Spezifikation der Geräte und der Daten, die diese produzieren werden durch diesen Service als Device-Profile beim Metadata-Service registriert. Der Command-Microservice kann mit Hilfe eines solchen Profils Befehle unabhängig von dem verwendeten Protokoll an die jeweiligen Geräte senden. Jede Kommunikation mit den Geräten findet über den Command-Service und den jeweiligen Device-Service statt.

Die angeschlossenen Geräte senden ihre Daten nicht eigenständig an das EdgeX-System, sondern werden in einem konfigurierbaren Intervall von den Device-Services abgefragt. Die Daten werden über den Core-Data-Service in einer MongoDB gespeichert. Der Core-Data-Service verteilt die Daten über ein asynchrones Messaging an andere Microservices, wie den Distribution-Service oder die Rules-Engine. Dafür kommt ein ZeroMQ-Broker zum Einsatz. Alternativ kann das MQTT-Protokoll zur Verteilung der Daten genutzt werden. In diesem Fall wird ein zusätzlicher MQTT-Broker benötigt.

Für den Umgang mit sensiblen Daten, welche nicht persistiert werden dürfen, bietet der Core-Data-Service eine Streaming-Konfiguration an. Wenn diese Konfiguration aktiviert wird, werden die Daten nicht mehr zwischengespeichert, sondern direkt über den Message-Queue-Broker an die Application-Services verteilt. Durch diese Konfiguration können auch mögliche Latenzen und der Speicherbedarf des Fog-Knotens reduziert werden.

Der Scheduling-Service löscht in periodischen Abständen (standardmäßig alle 30 Minuten) die Daten aus EdgeX, die bereits exportiert wurden. Um eine klare Trennung der Zuständigkeiten zu wahren, stößt der Scheduling-Service die Löschung der Daten über die Schnittstelle des Core-Data-Service an, welcher die Daten eigenständig löscht. Durch dieses Vorgehen ist der Core-Data-Service die einzige Instanz, welche die gespeicherten Daten verwaltet. Eine alternative Konfiguration erlaubt es dem Scheduling-Service auch veraltete Daten zu löschen, welche noch nicht exportiert wurden.

Der Rules-Engine-Service bietet die Möglichkeit Regeln zu definieren und entsprechende Aktionen für die Geräte abzuleiten. Dafür registriert sich die Rules-Engine als Export-Client beim Export-Client-Registration-Service und erhält anschließend die Daten vom Distribution-Service. Durch eine alternative Konfiguration können die Daten auch direkt von der Message-Queue des Core-Data-Service bezogen werden. Dies ist besonders

für zeitkritische Anwendungen relevant. Für die Bedingungen werden entsprechende Aktionen definiert, welche von der Rules-Engine über den Command-Service ausgeführt werden. Dieser Microservice nutzt die Drools Rules Engine[2] als Kern für die eigene Implementierung. Regeln können über die REST-Schnittstelle registriert und entfernt werden.

Damit Northbound-Systeme, wie Analyse-Anwendungen oder Cloud-Services, Daten aus EdgeX empfangen können, müssen sich diese als Export-Client beim Export-Client-Service registrieren. Dieser speichert die Informationen, welche Daten exportiert werden, in welchem Format diese Daten exportiert werden und wohin die Daten verteilt werden sollen. Mögliche Exportziele sind beispielsweise REST-Endpunkte oder MQTT-Broker. Der Distribution-Service sorgt dafür, dass die Daten aus EdgeX in der spezifizierten Weise an die registrierten Northbound-Systeme exportiert werden. Dieser Mechanismus soll in Zukunft durch die Funktions-Pipelines der Applikation-Services abgelöst werden. Dadurch ist eine individualisierte Vorverarbeitung und Transformation für jeden Endpunkt möglich und ein Flaschenhals bei zu vielen Northbound-Systemen wird vermieden.

Neben den hier genannten Microservices gibt es weitere Services, welche die Zusammenarbeit, das Debugging und die Sicherheit des Systems sicherstellen. Der Logging-Service bietet eine zentralisierte Einheit für alle Logs innerhalb von EdgeX und bietet diese Informationen über seine REST-Schnittstelle an. Wichtige Meldungen, wie die Überschreitung eines Grenzwertes, werden von dem Notification-Service verteilt. So kann beispielsweise ein Alarm eingerichtet werden, welcher bei einem definierten Ereignis den Benutzer per E-Mail informiert. Der Nutzer kann dafür über eine REST-Schnittstelle bestimmte Meldungen abonnieren.

Die System-Management-Services bilden einen der zwei vertikalen Schichten von EdgeX. Sie dienen dazu Microservices zu starten, stoppen und neu zu starten. Hier können verschiedene Metriken über die Auslastung der einzelnen Services überwacht werden. Außerdem können die Konfigurationen der Microservices abgerufen werden. In zukünftigen Releases soll es zusätzlich auch möglich sein die Konfigurationen der Services direkt zu ändern.

Die zweite vertikale Schicht wird durch die Security-Services gebildet. Der Secret Store ist ein Microservice dieser Schicht und bietet einen zentralen Speicherort für geheime Informationen, wie Passwörter oder Zugangstokens. Diese Informationen können von anderen

[2] https://www.drools.org/

Microservices abgerufen werden, wobei die Kommunikation mittels TLS geschützt ist. Zusätzlich bietet der Service eine Funktionalität zum Verschlüsseln und Entschlüsseln von Daten an, ohne diese zu speichern. Das API Gateway stellt einen Broker dar und schirmt die EdgeX-Microservices von externen Clients ab. Sämtliche REST-Kommunikation wird somit über diesen Service gesteuert. Das Gateway prüft die Berechtigung der Clients und leitet die Anfragen entsprechend an die gewünschten Services weiter. Die Antworten der Services werden ebenfalls vom Gateway an den Client gesendet.

Durch die lose gekoppelte Microservice-Architektur von EdgeX kann das System problemlos um weitere Services erweitert werden. Für die Erstellung von Device-Services und Application-Services wird ein SDK für die Sprache GoLang[3] geboten.

2.5 Ereignisverarbeitung

Bei der Ereignisverarbeitung werden die Daten im Moment ihres Auftretens verarbeitet. Hier werden Ereignisse in einem kontinuierliche Strom erkannt und eine entsprechende Reaktion ausgelöst [BD10].

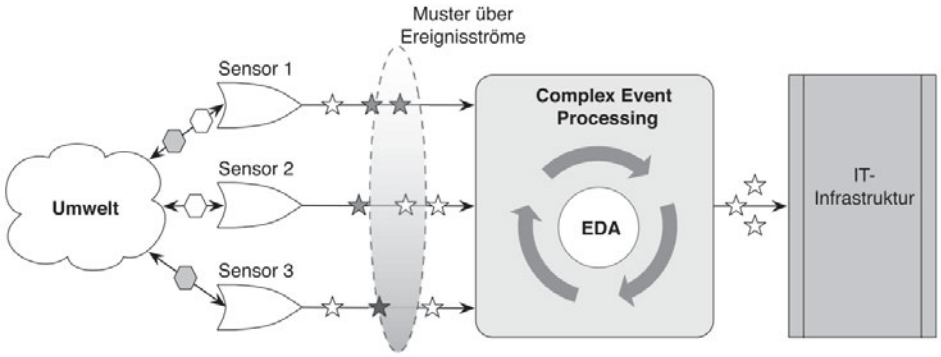

Abbildung 2.6: Nutzung von CEP in Sensornetzwerken [BD10]

Complex Event Processing (CEP) umfasst Methoden, Techniken und Werkzeuge, um diese Ereignisse zum Zeitpunkt ihres Auftretens zu verarbeiten [EB09]. Sie wird als Middleware zwischen der physikalischen Umwelt und der IT-Infrastruktur genutzt. Die Ereignisse werden also in das Zentrum der Softwarearchitektur gerückt [BD10].

[3] https://golang.org/

Wie in Abschnitt 2.1 beschrieben, generieren Sensornetzwerke große Mengen an Daten in kurzen Taktzyklen. Die Ereignisorientierung bietet die Möglichkeit die Auswertung dieser Daten agiler, reaktionsschneller und echtzeitfähiger zu machen, während konventionelle Informationssysteme nur eine retrospektive Sicht auf die Daten bieten [BD10]. Eine Anpassung der Geschäftsprozesse zur Laufzeit ist somit ohne Unterbrechung des Systems möglich. Aus den Ereignissen der physikalischen Umwelt lassen sich sog. komplexe Ereignisse ableiten, welche höheres, wertvolles Wissen repräsentieren [BD10]. Für Unternehmen führt eine Abstraktion und Verdichtung der Daten zu einer vereinfachten Entscheidungsfindung.

Bruns [BD10] beschreibt in seinem Buch insgesamt fünf Anforderungen, welche definieren, ob Ereignisverarbeitung für einen Geschäftsprozess geeignet ist:

1. Komplexe Fachlogik: Die Verarbeitungsprozesse sind komplex und es gibt Daten auf unterschiedlichen Abstraktionsebenen
2. Große Datenvolumina: Es gibt eine hohe Anzahl von Daten, die kontinuierlich mit einer hohen Eingangsrate eintreffen
3. Geringe Latenzen: Entscheidungen bzw. Reaktionen auf Ereignisse müssen schnell (in Echtzeit) getroffen werden
4. Skalierbarkeit: Das Daten- oder Transaktionsaufkommen steigt, wodurch das System gezwungen ist zu skalieren
5. Agilität: Die einfache Wartbarkeit und Änderbarkeit des Systems soll den Umgang mit der fachlichen Agilität sicherstellen

Diese Anforderungen decken sich mit den Anforderungen an Edge- bzw. Fog-Anwendungen, welche im Abschnitt 2.4.2 beschrieben wurden und durch die Eigenschaften des IoT bedingt sind.

2.5.1 Ereignisse

Ereignisse werden verschiedenen Abstraktionsebenen zugeordnet. Physikalische Ereignisse (Low Level Events) sind Beobachtungen von Vorkommnissen und werden meist kontinuierlich und hochfrequent erzeugt [BD10]. Anwendungsereignisse (High Level Events) repräsentieren dagegen einen fachlichen Sachverhalt auf einer höheren Abstraktionsebene. Diese sind Abstraktionen von einfachen Ereignissen und fassen diese zusammen oder

leiten eine fachliche Bedeutung ab [BD10]. Dieses Prinzip kann in Abbildung 2.7 gesehen werden.

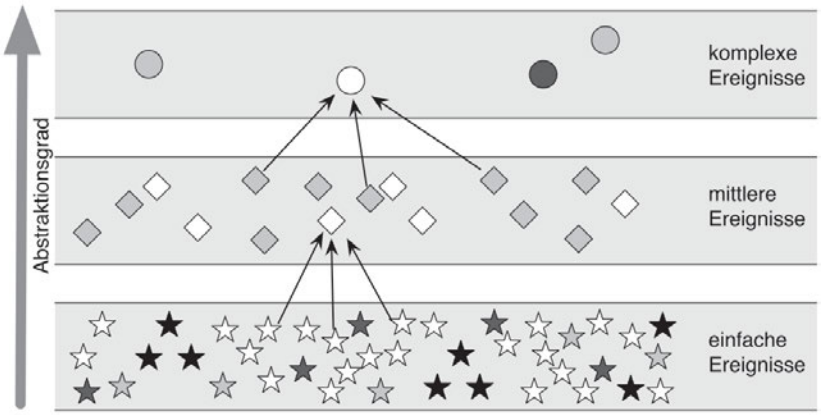

Abbildung 2.7: Abstraktion von Ereignissen [BD10]

Mit höheren Abstraktionsebenen reduziert sich die Anzahl der Ereignisse, während der Informationsgehalt eines einzelnen Ereignisses zunimmt. Jedes Ereignis bildet die Instanz eines definierten Ereignistyps [BD10]. Ein sog. Ereignismodell definiert die Ereignistypen, sowie deren Abhängigkeiten und Beziehungen untereinander. Ein Ereignis enthält alle Informationen, die für dessen Verarbeitung notwendig sind. Dazu gehören Metadaten, wie Ereignis-ID, Auftrittszeitpunkt, Ereignisquelle und Ereignistyp, aber auch die ereignisspezifischen Daten, welche den Kontext beschreiben [BD10]. Ein Strom von Ereignissen ist ein kontinuierlicher Fluss aus neuen Ereignisinstanzen mit unterschiedlichen Typen von verschiedenen Quellen.

2.5.2 Verarbeitung von Ereignisströmen

Der Grundzyklus von ereignisgesteuerten Systemen besteht aus drei Schritten [BD10]:

1. Erkennen: Erkennen von relevanten Ereignissen aus einem kontinuierlichen Ereignisstrom in Echtzeit
2. Verarbeiten: Analysieren der erkannten Ereignisse durch Aggregation, Filterung, Abstraktion oder Klassifikation
3. Reagieren: Ableiten von verschiedenen Reaktionen auf die erkannten Ereignismuster

Zusammenhängende Ereignisse bilden ein Muster (Event Pattern), welches in der Masse von Ereignissen erkannt werden muss [BD10]. Diese Muster können sich auf die Sequenz des Eintreffens, temporale oder räumliche Bedingungen beziehen.

Die Verarbeitung der Ereignisse erfolgt in einer zentralen CEP-Komponente [BD10]. Dafür werden Regeln (Event Processing Rules) definiert, welche von einem Regelinterpreter interpretiert und auf die Ereignisströme angewandt werden. Die Ereignisse treffen also kontinuierlich und dynamisch ein, während die Regeln persistent sind [EB09, BD10]. Eine solche Regel besteht aus einem Bedingungsteil und einem Aktionsteil. In den Aktionen können neue Ereignisse einer höheren Abstraktionsebene erzeugt oder Dienste angestoßen werden. Beim Erzeugen neuer Ereignisse können Informationen hinzugefügt werden (z.B. Anreicherung mit Kontextwissen). Mit sog. Regelsprachen (Event Processing Languages) werden Bedingungen und Reaktionen deklarativ beschrieben [BD10]. Da Ereignisströme kontinuierlich und unbegrenzt sind, ist für Operationen auf mehreren Ereignissen (z.B. Aggregation) die Definition von Längen- oder Zeitfenstern nötig.

2.5.3 Event-Driven Architecture

Ereignisgesteuerte Systeme bestehen aus einer Ereignisquelle und einer Ereignissenke [BD10]. Um die Schritte des Grundzyklus abzubilden wird eine eventgetriebene Architektur (EDA) in drei Schichten aufgeteilt: Ereignisquellen, Ereignisverarbeitung und Ereignisbehandlung.

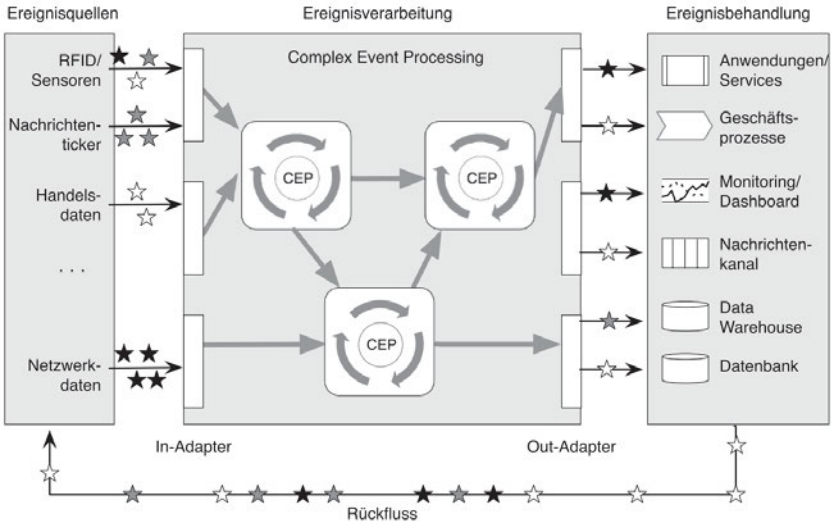

Abbildung 2.8: Aufbau einer Event-Driven Architecture [BD10]

Die In-Adapter transformieren die Ereignisse der Ereignisquellen in das von der Ereignisverarbeitung geforderte Format. Die Out-Adapter transformieren Ereignisse in das externe Format, welches von den Geräten und Systemen der Ereignisbehandlung benötigt wird [BD10].

Für die Ereignisverarbeitung ist das CEP ein integraler Bestandteil. Wenn ein Ereignismuster erkannt wird, gibt es verschiedene Arten, wie darauf reagiert werden kann. Zum einen können die Ereignisse kaskadiert werden, um komplexe Ereignisse zu erzeugen und diese als Eingabe weiter verarbeiten zu können, zum anderen kann ein Ereignis an die Behandlungsschicht weitergeleitet werden, um dort persistiert zu werden oder einen Geschäftsprozess anzustoßen [BD10]. Die Anwendungen dieser Schicht können weitere Ereignisse erzeugen, die wiederum als Ereignisquelle dienen [BD10, SE08].

Ein Event Processing Agent (EPA) ist ein Modul, welches Ereignismuster in einem kontinuierlichen Ereignisstrom erkennen kann und eine Verarbeitung durchführt. Bei Erkennung eines Musters werden entsprechende Reaktionen abgeleitet [BD10]. Ein EPA besteht aus einem Ereignismodell und Ereignisregeln, welche die zu erkennenden Muster definieren. Zudem besitzt jeder EPA eine Event Processing Engine zur Auswertung dieser Muster [LS08].

2.5.4 Event Processing Networks

Die Nutzung eines einzelnen EPA für alle Regeln erschwert das Verständnis des Gesamtsystems. Daher werden Netzwerke aus EPAs gebildet, die Ereignisse über Ereigniskanäle miteinander austauschen. Diese Netzwerke nennt man Event Processing Networks (EPN). Jeder EPA besitzt eigene Regeln und kann Ereignisse entweder server-intern oder plattformübergreifend austauschen [BD10]. Dadurch entsteht eine Kommunikation zwischen den EPAs und eine Modularisierung von Regeln ist möglich [SE08].

Ein Ereigniskanal ist eine technische Infrastruktur, welche dafür zuständig ist Ereignisse zwischen verschiedenen Komponenten des EPN auszutauschen [SE08]. Während der Laufzeit können EPAs erzeugt und freigegeben werden. Die dynamische Änderung von Ereigniskanälen ist zum Beispiel durch Content-based Routing möglich. Dabei wird anhand des Ereignistyps oder des Inhalts entschieden, an welchen EPA das Ereignis weitergeleitet wird [LS08].

Um eine lose Kopplung zwischen den Komponenten zu gewährleisten erfolgt die Kommunikation ausschließlich über Adapter und Schnittstellen. Adapter- und Schnittstellenklassen übernehmen dafür die technische Realisierung des Versendens und Empfangens von Ereignissen [BD10].

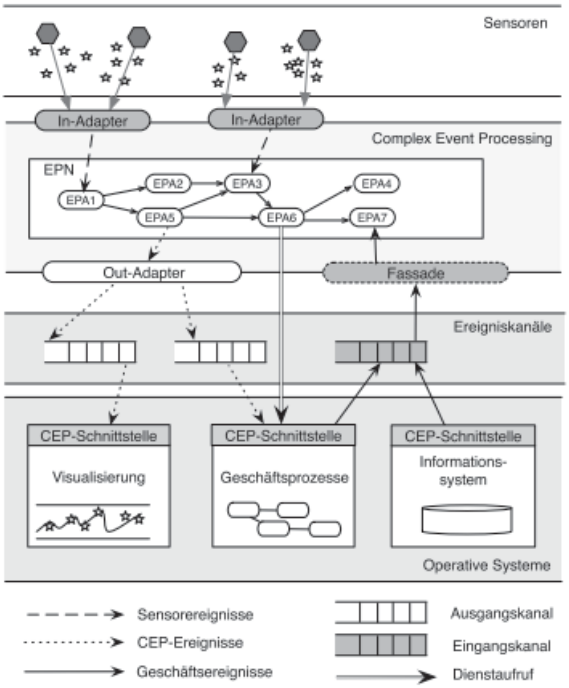

Abbildung 2.9: Aufbau eines Event Processing Networks [BD10]

Durch die somit erreichte Kapselung wird die Austauschbarkeit der Ereigniskanäle, sowie der Komponenten des Systems gewährleistet. Für die Realisierung eines Ereigniskanals bietet sich eine Message Oriented Middleware (MOM) an [BD10]. Diese bietet eine persistente, asynchrone Kommunikation zwischen den Komponenten und unterstützt eine lose Kopplung. Die Komponenten werden somit nicht blockiert, bis die Anfrage vom Empfänger verarbeitet wurde [vST17].

Ein Message-Broker kann genutzt werden, um die Ereignisse an die korrekten EPAs des Netzwerks zu transportieren. Dazu bietet sich die Nutzung des Publish-Subscribe-Modells an. Die Sender schicken ihre Nachrichten zu einem bestimmten Topic an den Broker. Komponenten, welche diese Nachrichten empfangen möchten, können diese beim Broker abonnieren [vST17].

2.6 Evaluationsmethodik

Um bestehende Softwarelösungen und Frameworks, wie zum Beispiel EdgeX, bewerten zu können, sind systematische Methoden zur Erfassung und Auswertung von Informationen notwendig. Eine solche Bewertung sollte auf eine nachvollziehbare und transparente Weise durchgeführt werden.

2.6.1 Softwaremetriken

Metriken sind eine Möglichkeit, um Software zu bewerten und miteinander zu vergleichen. Sie stellen meist einen messbaren Wert oder eine mathematische Formel dar, welche eine bestimmte Eigenschaft der Software untersucht. Metriken sind also eine Methode, um die Entscheidungsfindung auf eine quantitative Weise zu unterstützen. Bei der Auswertung der Metriken besteht die Gefahr der Fehlinterpretation, sodass ein strukturiertes Vorgehen, wie das im Abschnitt 2.6.2 beschriebene Goal-Question-Metric-Verfahren, sinnvoll ist.

Eine der bekanntesten Softwaremetriken ist die McCabe-Metrik. Mit der McCabe-Metrik kann die zyklomatische Komplexität eines Software-Moduls gemessen werden [McC76]. Sie wird durch folgende Formel definiert:

$$C(G) = e - n + 2p$$

Die Metrik wird über den Kontrollflussgraphen G des Programms bestimmt. Dabei ist e die Anzahl der Kanten im Graph, n ist die Anzahl der Knoten im Graph und p ist die Anzahl der verbundenen Komponenten. Als obere Grenze für die zyklomatische Komplexität gibt McCabe den Wert 10 an. Für eine gute Wartbarkeit des Programms sollte die Komplexität möglichst gering sein.

2.6.2 Goal-Question-Metric-Ansatz

Um ein System evaluieren zu können ist ein zielgerichtetes Vorgehen und eine klare Struktur nötig. Messungen sind ein sinnvolles Werkzeug, um Feedback zu erhalten und eine Evaluierung durchzuführen [Bas92]. Damit Messungen interpretierbar sind, müssen

diese mit einem Top-Down-Ansatz definiert werden. Dazu werden diese zunächst durch Ziele fokussiert.

Das GQM-Paradigma ist ein Mechanismus zum Definieren und Evaluieren von Zielen durch Messungen. Dabei wird eine klare, baumartige Struktur verfolgt. Die definierten Ziele werden in quantifizierbare Fragen (Questions) verfeinert. Für diese Fragen wird eine spezifische Menge von Messungen (Metrics) und Daten gesammelt. Das GQM-Paradigma bietet ein Framework zur Interpretation der Messergebnisse zur Beantwortung der Fragen im Kontext der definierten Ziele. Durch diese Aufteilung entsteht ein gerichteter Graph, wie in Abbildung 2.10 dargestellt wird.

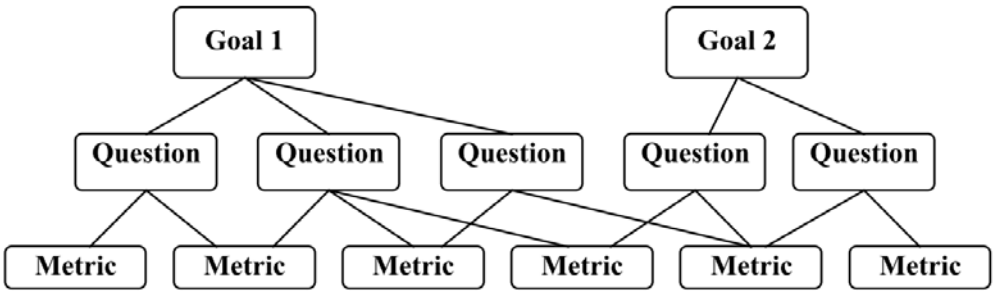

Abbildung 2.10: Struktur des GQM-Paradigmas als Graph [BCR94]

Eine Frage kann bei diesem Ansatz immer nur so umfassend beantwortet werden, wie die gewählten Metriken es zulassen. Die gleiche Frage kann genutzt werden, um mehrere Ziele zu definieren und gleichzeitig kann dieselbe Metrik verwendet werden, um mehrere Fragen zu beantworten. Fragen und Metriken werden auch aufgenommen, wenn diese noch nicht beantwortet oder gemessen werden können. Dies sorgt für ein vollständiges Modell und schafft die Möglichkeit Messungen zu einem späteren Zeitpunkt nachzuholen. Die verwendeten Metriken können entweder objektiv oder subjektiv sein. Subjektive Metriken werden in solchen Situationen genutzt, in denen keine exakte Messung möglich ist und eine Einschätzung von einem Benutzer bzw. Experten erfolgen muss [Bas92].

Der GQM-Prozess führt zu einer Evaluierungsstrategie und wird in insgesamt fünf Schritten durchgeführt [Bas92]:

1. Definition der Ziele: Zunächst werden die Ziele für die Evaluierung definiert und jeweils kurz beschrieben. Die Ziele werden immer für einen Betrachtungsgegenstand definiert. Dieses Objekt kann zum Beispiel das Produkt sein, aber auch spezifische

Komponenten oder Prozesse, welche evaluiert werden sollen. Ein Ziel verfolgt außerdem einen Zweck und hat einen Fokus auf ein bestimmtes Qualitätsmodell, wie zum Beispiel Zuverlässigkeit oder Wartbarkeit. Das Ziel wird für eine Umgebung und aus einem bestimmten Blickwinkel (z.B. Benutzer, Entwickler) definiert.

2. Aufstellen der Fragen: Nachdem die Ziele für die Evaluation bestimmt wurden, müssen quantifizierbare Fragen aufgestellt werden, welche diese Ziele so umfassend wie möglich definieren.

3. Definition von Messungen: Die entstandenen Fragen müssen durch Metriken messbar gemacht werden. Diese Metriken können objektiver und subjektiver Natur sein. Eine oder mehrere Metriken sollten die definierten Fragen möglichst umfassend beantworten können.

4. Erzeugen von Sammelmechanismen: Es müssen Mechanismen definiert werden, um die Daten für die Auswertung der im vorherigen Schritt definierten Metriken zu sammeln. Diese Mechanismen müssen klar definiert und wiederholbar sein.

5. Validieren und Analysieren: Die gesammelten Daten werden validiert und analysiert, dabei ist die Interpretation durch den Top-Down-Ansatz des GQM-Paradigmas vorgegeben. Die Metriken sind eindeutig mit den Fragen verknüpft, welche wiederum mit den entsprechenden Zielen der Evaluierung verknüpft sind.

3 Evaluierung von EdgeX

Das folgende Kapitel befasst sich mit der Planung, Durchführung und Auswertung einer Evaluierung der EdgeX Foundry. Zunächst erfolgt eine Definition der Ziele und Metriken, sowie deren Gewichtungen nach dem GQM-Paradigma. Anschließend wird im Abschnitt 3.2 die Konzeption eines Testszenarios beschrieben, um EdgeX im Umfeld des Fog-Computing-Paradigmas zu untersuchen. Der Abschnitt 3.3 beschreibt die für den Testaufbau verwendete Hard- und Software sowie die durchgeführten Schritte zum Aufbau des Szenarios. Im Anschluss werden die unter 3.1 definierten Bewertungskriterien im Abschnitt 3.4 erfasst und mit ihren jeweiligen Gewichtungen verrechnet. Abschließend wird aus den Bewertungen der Evaluationsziele im Abschnitt 3.5 eine Gesamtbewertung von EdgeX errechnet und interpretiert.

3.1 Definition der Bewertungskriterien

Der erste Schritt für eine Bewertung nach dem im Abschnitt 2.6.2 beschriebenen GQM-Paradigma ist es Ziele zu definieren. Diese Ziele werden durch ein Messobjekt, einen Zweck und Qualitätsfokus, sowie durch einen Blickwinkel und den Kontext bestimmt. Für die Evaluierung von EdgeX wurden insgesamt fünf solcher Ziele aufgestellt, welche zusammen mit ihren jeweiligen Fragestellungen und Metriken in diesem Abschnitt beschrieben werden sollen.

Für das Aufstellen eines Bewertungsschemas wurden Kennwerte von bekannten Open-Source-Frameworks im Bereich Edge- und Fog-Computing gesammelt. Dadurch ist der direkte Vergleich von Frameworks im Bezug auf diese Kennwerte möglich. Für jeden Wert wurden das Minimum, das untere Quartil, der Median, das obere Quartil, sowie das Maximum berechnet. Die jeweiligen Punkte wurden im Intervall zwischen diesen Werten festgelegt. Ein Beispiel hierfür findet sich Tabelle 3.1 für die Anzahl der Contributors bei GitHub.

Funktion	Funktionswert	Untere Grenze	Obere Grenze	Punkte
Minimum	8	8	17	0
Unteres Quartil	18	18	18	1
Median	19	19	60	2
Oberes Quartil	61	61	116	3
Maximum	116			

Tabelle 3.1: Beispiel für die Berechnung des Bewertungsschemas

Für den Vergleich wurden die im Abschnitt 2.4.4 vorgestellten Open-Source-Projekte herangezogen, diese gehören zu den bekanntesten Frameworks im Kontext von Edge- und Fog-Computing. Alle hier genannten Projekte werden über GitHub verwaltet.

3.1.1 Nachhaltigkeit und Akzeptanz

Die Nachhaltigkeit ist für den kommerziellen Einsatz einer Software entscheidend, da sie bestimmt wie lange ein Projekt gepflegt und weiterentwickelt wird. Lösungen, welche auf Projekten basieren, die nicht mehr unterstützt werden bedeuten einen erhöhten Wartungs- und Pflegeaufwand. Die Akzeptanz bestimmt auch, ob sich eine Community bilden kann. Communities sind für die Behebung von Fehlern, sowie für die Unterstützung beim Verständnis des Systems hilfreich.

Aspekt	Beschreibung
Messobjekt	EdgeX Foundry v1.1.0
Zweck	Evaluierung
Qualitätsfokus	Nachhaltigkeit/Akzeptanz
Blickwinkel	Softwarearchitekt/Projektleiter
Kontext	Projekt in einem Fog-Computing-Paradigma

Tabelle 3.2: Evaluationsziel - Nachhaltigkeit und Akzeptanz

Um das Ziel gemäß des GQM-Paradigmas zu definieren wurden die folgende Fragen formuliert. Die aufgestellten Fragestellungen dienen als Grundlage für die Definition von Metriken, welche durch Auswertung der transitiven Abhängigkeit von den Zielen direkt interpretiert werden können.

- Q1.1: Wie viele Personen arbeiten an dem Projekt?
- Q1.2: Wie bekannt ist das Projekt?

- Q1.3: Werden regelmäßig Releases veröffentlicht?
- Q1.4: Wie komplex ist das System?

Um diese Fragen zu beantworten wurden die in Tabelle 3.3 beschriebenen Metriken definiert und entsprechenden Gewichtungen zugeordnet. Alle hier definierten Metriken sind objektiv und können somit gemessen werden.

Nummer	Beschreibung	Gewichtung
M1.1	Anzahl der Contributors bei GitHub	15
M1.2	Verbreitungsgrad (Anzahl der Watches + Anzahl der Forks)	25
M1.3	Releasezyklus ($\frac{\text{Durchschnittliche Lebensdauer}}{\text{Anzahl der Releases}}$)	30
M1.4	Wartbarkeitsquotient ($\frac{\text{Lines of Code}}{\text{Anzahl der Contributors}}$)	30

Tabelle 3.3: Metriken - Nachhaltigkeit und Akzeptanz

Für die Bewertung der Metriken wurde ein vierstufiges Bewertungsschema verwendet. Die Bewertung mit 0 Punkten ist dabei am schlechtesten, während 3 Punkte die beste mögliche Bewertung darstellt. Die Punkte werden im Anschluss mit der in Tabelle 3.3 vergebenen Gewichtung verrechnet. Das Bewertungsschema für die angegebenen Metriken kann in Tabelle A.1 eingesehen werden.

3.1.2 Softwarequalität

Die Qualität der Software beeinflusst die Verständlichkeit und Wartbarkeit eines Systems und ist daher ein wichtiger Einflussfaktor bei der Auswahl von Frameworks. Sie wird durch die Komplexität des Projektes und die Art und Menge der Dokumentation beeinflusst.

Aspekt	Beschreibung
Messobjekt	EdgeX Foundry v1.1.0
Zweck	Evaluierung
Qualitätsfokus	Softwarequalität
Blickwinkel	Softwareentwickler
Kontext	Projekt in einem Fog-Computing-Paradigma

Tabelle 3.4: Evaluationsziel - Softwarequalität

Um das Ziel der Softwarequalität genau zu definieren, wurden auch hier Fragen formuliert. Diese Fragen befassen sich mit der Komplexität des Systems, sowie der Art und dem Umfang der Dokumentation:

- Q2.1: Gibt es Dokumentation für das Projekt?
- Q2.2: Wie umfangreich ist der Programmcode dokumentiert?
- Q2.3: Gibt es ein strukturiertes Issue-Tracking?
- Q2.4: Wie komplex ist das System?

Zur Beantwortung dieser Fragen wurden in Tabelle 3.5 einige Metriken definiert. Diesen Metriken wurden, wie im Abschnitt 3.1.1, Gewichtungen zugeordnet, sodass eine Bewertungsmatrix entsteht. Auch die hier definierten Metriken sind objektiv und können eindeutig bestimmt werden.

Nummer	Beschreibung	Gewichtung
M2.1	Kommentardichte	25
M2.2	Vorhandensein von API-Beschreibungen	25
M2.3	Vorhandensein eines Issue-Tracking-Systems	15
M2.4	Zyklomatische Komplexität	35

Tabelle 3.5: Metriken - Softwarequalität

Zur Bewertung dieser Metriken wurden die oberen und unteren Grenzen durch die Auswertung der Kennwerte der zuvor genannten Open-Source-Frameworks ermittelt. Da die Metriken M2.2 und M2.3 nur zwei Ergebnisse zulassen, werden für das Vorhandensein der Dokumentation entsprechend 3 Punkte vergeben, während das Fehlen der Dokumentation mit 0 Punkten bewertet wird. Für die Bewertung der zyklomatischen Komplexität wurde die von McCabe vorgeschlagene Obergrenze von 10 verwendet [McC76]. Eine Übersicht über das Bewertungsschema findet sich in Tabelle A.2.

3.1.3 Umgang mit anderen Systemen

Wie in Abschnitt 2.4.2 beschrieben, spielt die Zusammenarbeit mit anderen Systemen für das Fog-Computing eine wichtige Rolle. Dazu zählen zum einen die heterogenen Datenquellen, wie IoT-Geräte, aber auch die Interaktion mit Cloud-Services und Anwendungssystemen, wie Data-Lakes oder Analyse-Umgebungen.

Aspekt	Beschreibung
Messobjekt	EdgeX Foundry v1.1.0
Zweck	Evaluierung
Qualitätsfokus	Umgang mit anderen Systemen
Blickwinkel	Systembetreiber
Kontext	Projekt in einem Fog-Computing-Paradigma

Tabelle 3.6: Evaluationsziel - Umgang mit anderen Systemen

Um dieses Ziel genauer zu spezifizieren wurden einige Fragen bzgl. der Funktionalität von EdgeX aufgestellt:

- Q3.1: Können heterogene Datenquellen angebunden werden?
- Q3.2: Können Cloud-Dienste als Datensenke angebunden werden?
- Q3.3: Wird die Mobilität von Knoten unterstützt?
- Q3.4: Ist eine hierarchische Strukturierung möglich?

Diese Fragen beziehen sich auf die im Abschnitt 2.4.2 beschriebenen Eigenschaften eines Fog-Computing-Systems. Um die Fragen beantworten zu können, wurden die in Tabelle 3.7 aufgelisteten Metriken samt ihrer Bewertungen definiert. Die hier verwendeten Metriken sind subjektiv und werden durch einen im Abschnitt 3.2 beschriebenen Testaufbau vom Autor bewertet.

Nummer	Beschreibung	Gewichtung
M3.1	Anbindung von Geräten mit verschiedenen Protokollen	25
M3.2	Möglichkeit zur dynamischen Erkennung von Geräten	15
M3.3	Möglichkeit zur Anbindung verschiedener Cloud-Dienste	15
M3.4	Aufbau einer hierarchischen Struktur von EdgeX-Instanzen	20
M3.5	Anbinden von Geräten mit unterschiedlichen Datenformaten	25

Tabelle 3.7: Metriken - Umgang mit anderen Systemen

Die Auswertung dieser Metriken erfordert eine subjektive Einschätzung. Damit diese Einschätzung nachvollziehbar und vergleichbar ist, wurde im Abschnitt 3.2 ein Testszenario geplant. Für den Einsatz von EdgeX ist jedoch nicht nur relevant, ob eine Funktionalität grundlegend unterstützt wird, sondern auch, mit welchem Aufwand dies verbunden ist. Daher wurde das in Tabelle A.3 beschriebene Bewertungsschema mit Hinblick auf den benötigten Implementierungsaufwand entworfen. Eine einfache Konfiguration der

Services entspricht dabei der Bewertung 'Ohne Aufwand möglich', da hier keine Programmierkenntnisse benötigt werden und eine explizite Implementierung entfällt. Die Bewertung 'Mit wenig Aufwand möglich' bedeutet, dass Werkzeuge und Schnittstellen für eine einfache Erweiterung der Funktionalität von EdgeX zur Verfügung gestellt werden. Die Bewertung 'Mit viel Aufwand möglich' umfasst einen erheblichen Aufwand zur Implementierung, sowie ggf. eine Anpassung der bestehenden Services von EdgeX. Wenn eine Metrik mit der Bewertung 'Nicht möglich' bewertet wird, ist eine Umsetzung der Funktionalität nicht ohne eine tiefgreifende Veränderung der Funktionsweise bzw. des Verarbeitungsparadigmas von EdgeX möglich.

3.1.4 Verarbeitung von Daten

Der Vorteil von Edge- bzw. Fog-Computing ist die lokale Auswertung und Vorverarbeitung von Daten am Rande des Netzwerks. Wie in Abschnitt 2.4 beschrieben, soll durch eine Verarbeitung der Daten nahe der Datenquelle Bandbreite gespart und Latenzen vermieden werden. Es ist von entscheidender Bedeutung, dass ein Fog-Computing-Framework Möglichkeiten zur Vorverarbeitung und Filterung, sowie zur Entscheidungsfindung auf Fog-Knoten bietet.

Aspekt	Beschreibung
Messobjekt	EdgeX Foundry v1.1.0
Zweck	Evaluierung
Qualitätsfokus	Verarbeitung von Daten
Blickwinkel	Anwender
Kontext	Projekt in einem Fog-Computing-Paradigma

Tabelle 3.8: Evaluationsziel - Verarbeitung von Daten

Um dieses Evaluationsziel zu definieren wurden folgende Fragen bzgl. der Verarbeitung von Daten auf den Fog-Knoten aufgestellt:

- Q4.1: Können Entscheidungen am Rande des Netzwerks getroffen werden?
- Q4.2: Können Daten auf den Fog-Knoten gefiltert werden?
- Q4.3: Können Daten in andere Formate transformiert werden?
- Q4.4: Können komplexe Algorithmen auf den Fog-Knoten ausgeführt werden?

Zur Beantwortung der definierten Fragen werden die in Tabelle 3.9 beschriebenen Metriken genutzt. Wie in den vorherigen Abschnitten wurden die Metriken auch hier mit Gewichtungen für die abschließende Bewertung versehen.

Nummer	Beschreibung	Gewichtung
M4.1	Möglichkeit zum Festlegen von Regeln für Entscheidungen	25
M4.2	Möglichkeit zur Filterung vor dem Senden zu den Northbound-Systemen	30
M4.3	Möglichkeit zum Ausführen komplexer Algorithmen	15
M4.4	Möglichkeit zur Umwandlung in andere Datenformate	30

Tabelle 3.9: Metriken - Verarbeitung von Daten

Zur Filterung der Daten zählt die Fähigkeit Ereignisse nach definierten Kriterien zu verwerfen und komplexe Ereignisse aus den einfachen Ereignissen der IoT-Geräte zu erzeugen. Die Umwandlung umfasst die Transformation der Repräsentation der Daten in andere Formate, wie zum Beispiel JSON oder XML. Die Auswertung der Metriken erfolgt wie zuvor durch eine subjektive Einschätzung mit Unterstützung durch das im Abschnitt 3.2 beschriebene Testszenario. Das in Tabelle A.4 beschriebene Bewertungsschema berücksichtigt auch hier den Aufwand für die Nutzung der jeweiligen Funktionalität.

3.1.5 Sicherheit und Zuverlässigkeit

Die Daten innerhalb eines Fog-Computing-Systems sind unter Umständen sensibel und sollten somit vor einem unberechtigten Zugriff geschützt werden. Da Fog-Knoten, wie in Abschnitt 2.4 beschrieben oftmals in heterogenen und unzuverlässigen Infrastrukturen eingesetzt werden, muss ein Fog-Computing-Framework zuverlässig mit Ausfällen und Fehlern innerhalb des Systems umgehen können.

Aspekt	Beschreibung
Messobjekt	EdgeX Foundry v1.1.0
Zweck	Evaluierung
Qualitätsfokus	Sicherheit und Zuverlässigkeit
Blickwinkel	Systembetreiber
Kontext	Projekt in einem Fog-Computing-Paradigma

Tabelle 3.10: Evaluationsziel - Sicherheit und Zuverlässigkeit

Um dieses Evaluationsziel genauer zu definieren wurden folgende Fragen aufgestellt:

- Q5.1: Ist das Gesamtsystem nach Ausfall einer Instanz noch lauffähig?
- Q5.2: Ist eine Instanz nach dem Ausfall eines Microservice noch lauffähig?
- Q5.3: Ist eine verschlüsselte Übertragung zu den Northbound-Systemen möglich?
- Q5.4: Ist eine verschlüsselte Übertragung zwischen den Services möglich?
- Q5.5: Wird eine Authentifizierung beim Zugriff auf die Services unterstützt?

Zur Beantwortung dieser Fragen wurden auch hier Metriken definiert, welche mit Hilfe des unter 3.2 beschriebenen Testaufbaus bewertet werden. Die Metriken sind mit ihren jeweiligen Gewichtungen in Tabelle 3.11 aufgeführt.

Nummer	Beschreibung	Gewichtung
M5.1	Ausfallsicherheit des Gesamtsystems	30
M5.2	Ausfallsicherheit einer Instanz	20
M5.3	Möglichkeit zur verschlüsselten Übertragung zu Northbound-Systemen	20
M5.4	Möglichkeit zur verschlüsselten Übertragung zwischen Services	10
M5.5	Möglichkeit zur Authentifizierung der Nutzer	20

Tabelle 3.11: Metriken - Sicherheit und Zuverlässigkeit

Die Bewertungen der Metriken bedürfen der Einschätzung und der Erprobung in einem Testszenario, dessen Planung und Aufbau in den folgenden Abschnitten beschrieben wird. Das Schema für die Bewertung der Metriken wurde vorab definiert und kann in Tabelle A.5 eingesehen werden.

3.2 Konzeption eines Testszenarios

Die Untersuchung der im Abschnitt 3.1 beschriebenen Metriken erfordert die Erprobung in einem Szenario. Daher wurde ein Testszenario geplant, welches eine typische Infrastruktur eines Fog-Computing-Systems abbildet. Die Anforderungen an diesen Testaufbau werden durch die Eigenschaften des Fog-Computing-Paradigmas beeinflusst. Um die unter 2.4.2 beschriebenen Eigenschaften abzudecken wurden die folgenden Anforderungen an ein Testszenario definiert:

- Mehrere Fog-Knoten: Die Geräte des IoT sind oftmals geographisch verteilt und müssen von mehreren Instanzen eines Fog-Computing-Frameworks auf unterschiedlichen Fog-Knoten verwaltet werden. Daher sollte es mindestens zwei unabhängige Fog-Knoten geben.
- Mobilität: Einige IoT-Geräte sind mobil und können somit ihren Zugangspunkt wechseln. Daher sollte es mindestens eine mobile Instanz einer Datenquelle geben, die den Zugangspunkt zum Fog-Computing-System wechselt.
- Heterogenität: IoT-Geräte kommunizieren über unterschiedliche Protokolle und senden ihre Daten über verschiedene Medien. Diese Eigenschaft sollte durch mehrere (u.U. simulierte) Geräte geprüft werden, welche mit dem System über die typischen Protokolle, wie REST und MQTT kommunizieren.
- Interaktion mit Northbound-System: Meist reicht die Speicherkapazität und Rechenleistung von Fog-Knoten nicht aus, um komplexe Analysen lokal durchzuführen zu können. Daher sollte eine Verbindung zu einem Northbound-System, wie der Cloud oder einem anderen Anwendungssystem, berücksichtigt werden.
- Lokale Verarbeitung: Damit Bandbreite und Latenz gespart werden kann, sollte es möglich sein, Entscheidungen auf einem Fog-Knoten zu treffen. Außerdem sollte eine Vorverarbeitung und Filterung möglich sein, bevor die Daten über das Netzwerk zu einem Northbound-System geschickt werden.
- Hierarchische Struktur: Es ist sinnvoll die Daten unterschiedlicher Fog-Knoten möglichst frühzeitig zu integrieren und zu analysieren, damit nicht alle Daten über das Internet zur Cloud transportiert werden müssen. Daher sollte eine hierarchische Struktur von Fog-Knoten unterstützt werden.

Als Referenz für den Aufbau dient das in Abbildung 3.1 abgebildete Verteilungsdiagramm. Hier wurden die Anforderungen an ein solches Testszenario berücksichtigt.

Die Daten werden durch die IoT-Geräte erzeugt und über die Device-Services zu ihrem verbundenen Fog-Knoten mit einer EdgeX-Instanz transportiert. Ein IoT-Gerät soll die Mobilitätsunterstützung überprüfen, indem es den Zugangspunkt wechselt und per Bluetooth kommuniziert. Die beiden unteren Fog-Knoten enthalten jeweils eine vollständige EdgeX-Instanz und können lokal Daten verarbeiten und filtern. Die Daten werden über das REST-Protokoll zu einer weiteren EdgeX-Instanz transportiert, welche die Daten integriert und weitere Verarbeitungs- und Filterungsschritte erlaubt. Viele Cloud-Anbieter

stellen eine MQTT-Schnittstelle zur Integration von IoT-Daten zur Verfügung. Daher soll ein entsprechendes System auch im Testszenario über MQTT angebunden werden. Dort können die Daten dauerhaft gespeichert und komplexe Analysen durchgeführt werden.

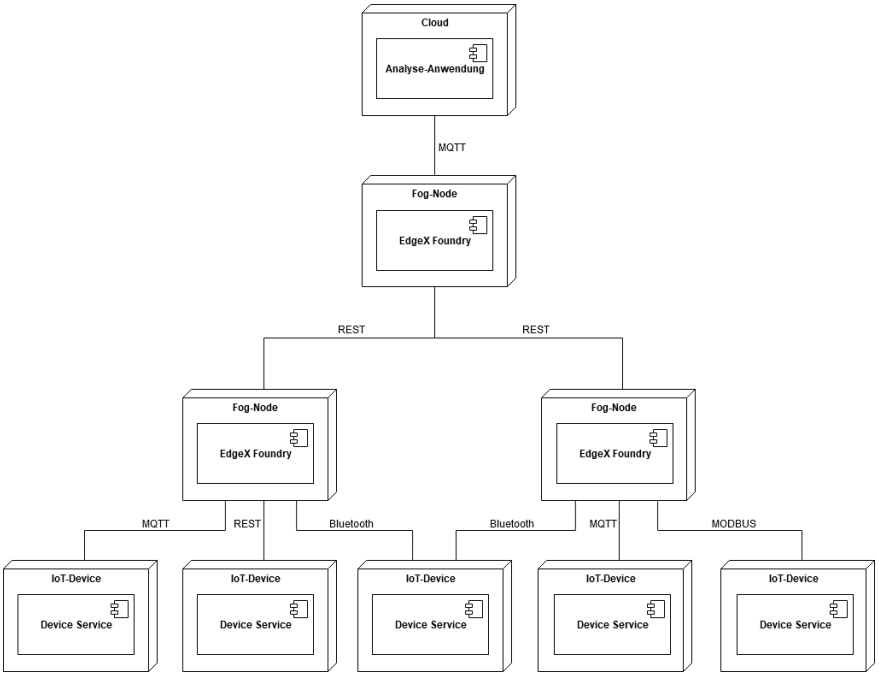

Abbildung 3.1: Testaufbau zur Evaluierung von Fog-Computing-Systemen

3.3 Aufbau und Konfiguration des Testszenarios

Im Folgenden soll die benötigte Soft- und Hardware für den Aufbau des Testszenarios beschrieben werden. Daneben werden auch die erforderlichen Erweiterungen der EdgeX Foundry und die vorgenommenen Konfigurationen kurz erläutert.

3.3.1 Verwendete Hardware

Für den Testaufbau werden zwei Raspberry Pis der vierten Generation verwendet. Diese fungieren als die beiden Fog-Knoten nahe der Datenquellen aus Abbildung 3.1. Auf beiden Geräten wurde die Linux-Distribution Ubuntu Core[1] in der 64-Bit-Version installiert. Um die Kosten und die Komplexität des Testszenarios zu senken wurden die IoT-Geräte

[1] https://ubuntu.com/download/raspberry-pi

simuliert. Diese Simulationen laufen in isolierten Docker-Containern auf den Raspberry Pis. Der Einsatz der Container-Software Docker hat den Vorteil, dass die simulierten IoT-Geräte unabhängig von EdgeX funktionieren und das System nicht ungewollt beeinflussen können. Außerdem können diese Container schnell gestartet und gestoppt werden, was eine Simulation von Ausfällen einfach ermöglicht. Effekte wie Rauschen und andere im Abschnitt 2.1 beschriebene Eigenschaften des IoT können ebenfalls simuliert werden. Als dritter Fog-Knoten dient ein Laptop mit der in der Tabelle 3.12 angegebenen Spezifikation. Hier wurde bewusst eine Maschine mit mehr Rechenleistung gewählt, damit auch komplexere Verarbeitungsschritte ermöglicht werden. Dieses System muss mit den Daten von allen angeschlossenen Quellen umgehen können.

Ressource	Spezifikation
CPU	i5-8265U CPU @ 1.60 GHz, 4 Kerne
RAM	16 GB DDR4
Festplatte	500 GB SSD
Betriebssystem	Windows 10 Professional

Tabelle 3.12: Fog-Knoten - Spezifikation

Zusätzlich zu den beschriebenen Fog-Knoten gibt es einen separaten Rechner, welcher als Datensenke fungiert. In einer realen Produktionsumgebung kann eine Cloud als Datensenke verwendet werden. Der hier genutzte Rechner hat die in Tabelle 3.13 beschriebene Spezifikation.

Ressource	Spezifikation
CPU	i7-8700K CPU @ 3.70 GHz, 6 Kerne
RAM	16 GB DDR4
Festplatte	250 GB SSD + 2 TB HDD
Betriebssystem	Windows 10 Professional

Tabelle 3.13: Cloud-Knoten - Spezifikation

3.3.2 Konfiguration von EdgeX

Alle Microservices von EdgeX werden mit Docker-Compose konfiguriert und gestartet. Dadurch können die Abhängigkeiten zwischen den Services klar definiert werden und die Verwaltung wird vereinfacht. Die Verwendung von Docker hat außerdem den Vorteil, dass einzelne Services isoliert ausgeführt werden und sich somit nicht ungewollt beeinflussen

können. Für den Aufbau des Testszenarios wurde für alle Microservices die Version 1.1.0 (Fuji-Release) verwendet.

Um verschiedene Situationen im Testszenario abbilden zu können, wurden die angeschlossenen IoT-Geräte simuliert. Dafür wurden Skripte in der Programmiersprache GoLang geschrieben, welche ihre Daten über entsprechende Protokolle bereitstellen. Diese Skripte laufen ebenfalls in isolierten Docker-Containern zusammen mit den EdgeX-Microservices auf den Fog-Knoten. Die Simulation der Geräte hat den Vorteil, dass Fehlerfälle, sowie Rauschen und Paketverluste, einfach simuliert werden können.

Die simulierten Geräte werden über Device-Services angebunden, welche das entsprechende Profil des Gerätes in EdgeX registrieren. Ein Geräteprofil beschreibt ein angeschlossenes Gerät und definiert das Schema der Daten, welche von diesem bezogen werden können. Für die Device-Services wurden die Referenzimplementierungen der EdgeX Foundry verwendet. Um die automatische Erkennung von Geräten zu testen wurde ein Device-Service für die Suche nach Bluetooth-Geräten implementiert. Dieser sucht in einem konfigurierbaren Intervall nach Bluetooth-Geräten, welche bestimmte Eigenschaften besitzen und registriert diese bei der EdgeX-Instanz.

Die Daten der Geräte werden auf allen Fog-Knoten vom Core-Data-Microservice zwischengespeichert. Für den Scheduling-Service wurde die Standardkonfiguration belassen, sodass dieser alle 30 Minuten die Löschung der exportierten Daten aus Core-Data anstößt. Core-Data kommuniziert die Daten der Geräte über den Message-Queue-Broker ZeroMQ.

Für die Entscheidungsfindung wurde die Referenzimplementierung der Rules-Engine verwendet. Diese wurde so konfiguriert, dass sie sich als Client bei den Export-Services registriert und somit die Daten nicht direkt von Core-Data bezieht. Dies entspricht der Standardkonfiguration der Rules-Engine. Es wurden einfache Produktionsregeln über die REST-API angelegt, welche auf unterschiedliche Ereignisse reagieren und entsprechende Aktionen ableiten.

Um einen hierarchischen Aufbau von EdgeX-Instanzen zu unterstützen, mussten zusätzliche Microservices mit Hilfe der von EdgeX bereitgestellten SDKs implementiert werden. Eine EdgeX-Instanz sendet ihre Daten über einen Application-Service an einen Northbound-Endpunkt, welcher eine andere EdgeX-Instanz darstellt. Dies geschieht über den synchronen Aufruf einer REST-Methode. Diese REST-Methode wurde mit Hilfe des

SDK in einem Device-Service implementiert. Die EdgeX-Instanz behandelt die andere Instanz also wie ein angeschlossenes Gerät. Diese Art der Konfiguration hat zur Folge, dass das Geräteprofil erneut angelegt werden muss. Durch die Verwendung der Funktionspipeline im Application-Service kann hier eine zusätzliche Filterung und Vorverarbeitung der Daten vorgenommen werden, bevor diese an die EdgeX-Instanz der höheren Ebene geschickt werden.

Als Northbound-System in der Cloud-Schicht wurde ein einfaches Anwendungssystem simuliert. Dieses nimmt die Daten von EdgeX im JSON-Format entgegen und persistiert sie in einer Mongo-Datenbank. Viele Cloud-Anbieter bieten die Möglichkeit Daten per MQTT in die Cloud zu senden. Daher wurde sich für eine Übertragung der Daten mittels MQTT entschieden. Der benötigte MQTT-Broker wird in einem Docker-Container auf dem Rechner des simulierten Northbound-Systems ausgeführt.

Das simulierte Northbound-System wird über einen entsprechenden Application-Service mit Daten versorgt. Die Verwendung von Funktionspipelines erlaubt es vor dem Senden der Daten eine Filterung und Vorverarbeitung, sowie eine Transformation der Daten durchzuführen.

3.4 Ergebnisse und Auswertung

Für die Erfassung der Daten der im Abschnitt 3.1 definierten Metriken wurden unterschiedliche Methoden und Quellen verwendet. Für die Erfassung der LoC und CLoC wurde das Kommandozeilenprogramm CLOC[2] verwendet. Kennwerte, wie die Anzahl der Contributor, konnten direkt über die API von GitHub ermittelt werden. Die Erfassung der Daten der subjektiven Metriken wurde durch eine Bewertung im Rahmen des im Abschnitt 3.2 beschriebenen Testszenarios durchgeführt. Für die Messung der zyklomatischen Komplexität nach McCabe [McC76] wurde das Open-Source-Tool gocyclo[3] verwendet.

Nachdem sämtliche Daten erfasst und EdgeX im Rahmen des Testszenarios ausgeführt wurde, konnten die aufgestellten Metriken bewertet werden. Durch die im Abschnitt 3.1 definierten Bewertungskriterien, sowie deren Gewichtungen konnten die erfassten Daten

[2] http://cloc.sourceforge.net/
[3] https://github.com/fzipp/gocyclo

ausgewertet werden. Das Vorgehen nach dem GQM-Paradigma schafft dabei eine eindeutige Zuordnung zu den jeweiligen Zielen der Messungen und ermöglicht somit eine einfache Interpretation der Werte.

Die Bewertungen für das Ziel 'Nachhaltigkeit und Akzeptanz' werden in Tabelle 3.14 gezeigt. Durch die im Abschnitt 3.1 definierten Gewichtungen wurde hier eine Gesamtwertung von 60% erreicht.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M1.1	Contributor-Anzahl	68	3	15%
M1.2	Verbreitungsgrad	345	3	25%
M1.3	Releasezyklus	15,13	0	0%
M1.4	Wartbarkeitsquotient	5280,41	2	20%
Summe der Bewertungen				60%

Tabelle 3.14: Nachhaltigkeit und Akzeptanz - Bewertung der Metriken

Die Kennwerte der objektiven Metriken für das Evaluationsziel 'Softwarequalität' wurden entsprechend erhoben und mit den Gewichtungen der einzelnen Metriken verrechnet. Dadurch entsteht die in der Tabelle 3.15 aufgeschlüsselte Gesamtbewertung von 75%.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M2.1	Kommentardichte	12%	0	0%
M2.2	API-Beschreibung	Ja	3	25%
M2.3	Issue-Tracking-System	Ja	3	15%
M2.4	Zyklomatische Komplexität	2,13	3	35%
Summe der Bewertungen				75%

Tabelle 3.15: Softwarequalität - Bewertung der Metriken

Um die subjektiven Metriken für das Evaluationsziel 'Umgang mit anderen Systemen' zu erfassen, wurde der beschriebene Testaufbau verwendet. Die Bewertung wurde dabei wie im Abschnitt 3.1.3 beschrieben vorgenommen. Die Ergebnisse der Bewertung wurden in Tabelle 3.16 aufgelistet und erreichen eine Gesamtbewertung von 68,33%.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M3.1	Anbindung von Geräten	Wenig Aufwand	2	16,67%
M3.2	Erkennung von Geräten	Wenig Aufwand	2	10%
M3.3	Anbindung von Cloud-Diensten	Wenig Aufwand	2	10%
M3.4	Hierarchische Struktur	Viel Aufwand	1	6,67%
M3.5	Unterschiedliche Datenformate	Kein Aufwand	3	25%
		Summe der Bewertungen		68,33%

Tabelle 3.16: Umgang mit anderen Systemen - Bewertung der Metriken

Externe Geräte werden mittels Device-Services an EdgeX angebunden. Diese Device-Services können mit geringem Aufwand mit Hilfe eines SDK in der Programmiersprache Go entwickelt werden. Dadurch ist es möglich mit Geräten über unterschiedliche Protokolle zu kommunizieren.

Die Erkennung von Geräten erfolgt ebenfalls mit Hilfe des SDK. Die Suche nach neuen Geräten kann entweder manuell über einen Endpunkt des Device-Services oder in einem konfigurierbaren Intervall ausgeführt werden. Der Device-Service implementiert dafür die protokollspezifische Kommunikation zum Erkennen der Geräte. Die Anbindung von Geräten mit unterschiedlichen Datenformaten ist durch die Definition von Geräteprofilen ohne Implementierungsaufwand möglich. Diese Profile werden mit Hilfe der Auszeichnungssprache YAML beschrieben und enthalten abstrahierte Informationen über die Geräte. Hier werden Ressourcen und Datentypen des Geräts beschrieben, sowie die Befehle definiert, über welche das Gerät angesteuert werden kann. Die konkrete Implementierung dieser Befehle mit Rücksicht auf das verwendete Protokoll wird von Device-Service übernommen. Diese Informationen werden als Metadaten im System gespeichert und dienen zur Validierung, Transformation und Interpretation der vom Gerät gesendeten Daten. Eine Zuordnung der erkannten Geräte zu einem Profil geschieht automatisch durch die Angabe eines Provision-Watchers. Dieser enthält eine Liste von identifizierenden Merkmalen, welche durch ein Whitelist- bzw. Blacklist-Verfahren den gefundenen Geräten zugeordnet werden.

Cloud-Dienste werden in EdgeX über die sog. Application-Services angebunden. Diese Services können ebenfalls mit Hilfe eines SDK in der Programmiersprache Go implementiert werden und enthalten eine Funktionspipeline aus Vorverarbeitungs-, Transformations- und Kommunikationsschritten. Das SDK bietet dabei bereits einige grundlegende Funktionen, wie das Senden der Daten mittels des MQTT-Protokolls an einen Northbound-

Endpunkt. Zusätzlich zu diesen bestehenden Grundfunktionen können eigene Funktionen implementiert und in der Pipeline verwendet werden.

Der hierarchische Aufbau von EdgeX-Instanzen wird nicht direkt unterstützt. Hier ist die Definition von eigenen Services notwendig. Für das im Abschnitt 3.3 beschriebene Testszenario wurde eine Kommunikation mittels HTTP-REST-Aufrufen realisiert. Ein Application-Service sendet die Daten aus einer EdgeX-Instanz zu einem REST-Endpunkt innerhalb eines Device-Services einer anderen Instanz. Dieses Vorgehen hat den Nachteil, dass das Profil für jedes angeschlossene Gerät hier erneut gespeichert werden und eine Transformation der Daten in das korrekte Datenformat durchgeführt werden muss.

Für die Bewertung der Metriken für das Evaluationsziel 'Verarbeitung von Daten' wurde ebenfalls der beschriebene Testaufbau verwendet. Die Bewertung wurde wie im Abschnitt 3.1.4 beschrieben vorgenommen. Die Ergebnisse der Bewertungen finden sich in Tabelle 3.17 und erreichen eine Gesamtbewertung von 43,33%.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M4.1	Entscheidungen über Regeln	Viel Aufwand	1	8,33%
M4.2	Filterung von Daten	Viel Aufwand	1	10%
M4.3	Komplexe Algorithmen	Viel Aufwand	1	5%
M4.4	Umwandlung von Datenformaten	Wenig Aufwand	2	20%
	Summe der Bewertungen			43,33%

Tabelle 3.17: Verarbeitung von Daten - Bewertung der Metriken

EdgeX implementiert eine simple Rules-Engine, welche es erlaubt Regeln über eine REST-Schnittstelle zu definieren. Die Rules-Engine bezieht ihre Daten von Core-Data und leitet aus den hinterlegten Regeln entsprechende Aktionen ab. Diese Aktionen werden über den Command-Service direkt an den Geräten ausgeführt. Das Treffen von Entscheidungen auf den Fog-Knoten wird somit grundlegend unterstützt, jedoch betrachtet die Rules-Engine nur einzelne Ereignisse. Die Auswertung von Ereignissen innerhalb eines Zeitfensters wird nicht unterstützt und komplexe Regeln mit booleschen Operatoren zur Verknüpfung der Bedingungen sind nicht möglich. Eine Interpretationslogik für komplexe Datentypen (z.B. JSON-Dokumente oder Bilder) kann über die verfügbaren Schnittstellen nicht konfiguriert werden. Die Verwendung dieser Datentypen innerhalb der Bedingungen wird somit nicht unterstützt und erfordert eine explizite Implementierung der benötigten Funktionalität und somit die erneute Kompilierung und Deployment der Rules-Engine. Somit wurde für die Bewertung dieser Metrik nur ein Punkt vergeben.

Die Daten können vor dem Senden zu den Northbound-Systemen mit Hilfe der Funktionspipelines der Application-Services gefiltert werden. EdgeX bietet vordefinierte Funktionen zur Filterung nach Gerätenamen und Art der Daten. Somit können Daten eines bestimmten Typs (z.B. Temperaturdaten) selektiert und zu einem Northbound-System transportiert werden. Weitere Filter müssen explizit über eigenständige Funktion implementiert und in der Funktionspipeline genutzt werden. Durch eine Konfigurationsdatei ist es möglich die Filter zu verändern, dies erfordert jedoch den Neustart des Microservices, sodass eine dynamische Veränderung zur Laufzeit nicht möglich ist. Die Definition von eigenen Filterfunktionen erfordert die erneute Kompilierung und Deployment des Application-Services mit der entsprechenden Funktionspipeline. Da die Pipeline nur die Betrachtung einzelner Ereignisse unterstützt, ist eine Aggregation von mehreren Werten innerhalb der Funktionspipeline nicht möglich und komplexe Ereignisse, welche sich aus dem gemeinsamen Auftreten mehrerer Ereignisse ergeben, lassen sich nicht ableiten. Aus diesen Gründen wurde sich für eine Bewertung mit einem Punkt entschieden.

Die Nutzung komplexer Algorithmen entfällt auf Anwendungen außerhalb von EdgeX. Diese Anwendungen können als Northbound-System an EdgeX angeschlossen werden und somit die Daten der IoT-Geräte nutzen. Durch die modulare Microservice-Architektur von EdgeX ist es außerdem möglich das System um Services zu erweitern. Diese Services können unabhängig entwickelt und deployed werden, sodass Anwendungen in beliebigen Programmiersprachen implementiert werden können. Die Services können über das System-Management von EdgeX registriert werden und die Funktionen anderer Services über deren REST-Schnittstellen nutzen. Somit muss die Implementierung der Algorithmen zwar eigenständig geschehen, jedoch bietet EdgeX umfassende Möglichkeiten zur Anbindung und Integration.

Analog zu den Filterfunktionen, können in einer Funktionspipeline auch Umwandlungen in andere Datenformate vorgenommen werden. Die Application-Service-SDK bietet dafür die vordefinierte Umwandlung in das XML- und JSON-Format. Spezifische Umwandlungen können selbst implementiert und in der Funktionspipeline genutzt werden. Die Änderung der Funktionspipeline über eine Konfigurationsdatei erfordert den Neustart des Microservices. Die Implementierung von eigenen Umwandlungsfunktionen erfordert, wie bei den Filtern, eine erneute Kompilierung und Deployment des Application-Services.

Die Bewertung der Metriken für das letzte Evaluationsziel 'Sicherheit und Zuverlässigkeit' wurden ebenfalls mit Hilfe des Testaufbaus vorgenommen. Die Bewertung richtet sich dabei nach dem im Abschnitt 3.1.5 beschriebenen Bewertungsschema. Die einzelnen

Bewertungen wurden in Tabelle 3.18 aufgelistet und erreichen eine Gesamtbewertung von 60%.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M5.1	Ausfallsicherheit (Gesamtsystems)	Nicht erkannt	1	10%
M5.2	Ausfallsicherheit (Instanz)	Wird erkannt	2	13,33%
M5.3	Verschlüsselung (Northbound)	Wenig Aufwand	2	13,33%
M5.4	Verschlüsselung (Intern)	Viel Aufwand	1	3,33%
M5.5	Authentifizierung	Kein Aufwand	3	20%
	Summe der Bewertungen			60%

Tabelle 3.18: Sicherheit und Zuverlässigkeit - Bewertung der Metriken

Der Ausfall einer EdgeX-Instanz beeinflusst nicht die Verfügbarkeit anderer Instanzen. Wenn eine Instanz der unteren Ebene ausfällt, so können die Daten der jeweiligen IoT-Geräte nicht mehr erfasst werden, dadurch gehen u.U. Daten verloren, jedoch wird die Funktionsfähigkeit der anderen Instanzen vom Ausfall nicht beeinflusst. Daten können in den Instanzen jeder Ebene zwischengespeichert werden, sodass diese auch beim Ausfall einer Instanz der höheren Ebenen nicht verloren gehen. Der Scheduler-Service kann konfiguriert werden, sodass dieser nur solche Daten löscht, die bereits erfolgreich zu einem Northbound-System exportiert wurden. Da bei einem Ausfall der gesamten Instanz auch die Monitoring-Services von EdgeX nicht mehr verfügbar sind, muss die Erkennung des Ausfalls auf andere Weise geschehen. Andere EdgeX-Instanzen können diesen Ausfall nicht detektieren.

Durch die lose gekoppelte Microservice-Architektur von EdgeX führt der Ausfall eines Services nicht zum Ausfall weiterer Services. Dennoch wird dadurch die Funktionalität des Gesamtsystems gestört. Der System Management Agent von EdgeX bietet eine zentrale Schnittstelle für die Verwaltung der Microservices. Hier kann ein Health-Check durchgeführt und Services bei Bedarf neu gestartet werden. Dies geschieht zur Zeit jedoch noch nicht automatisch, weshalb eine weitere Anwendung zur dauerhaften Überwachung der Verfügbarkeit aller Services nötig wäre.

Eine verschlüsselte Übertragung von den Services zu den jeweiligen Clients wird durch ein API Gateway unterstützt. Dieses bietet eine Verschlüsselung der Kommunikation über TLS. Die Pipelines des Application-Service-SDK bieten die Möglichkeit Daten, welche zu einem Northbound-System transportiert werden sollen, symmetrisch mittels des AES zu verschlüsseln. Dafür muss jedoch zuvor ein geheimer Schlüssel über einen sicheren

Kanal ausgetauscht werden. Dieser Schlüssel kann im Secret Store von EdgeX gespeichert werden, welcher einen gesicherten Zugriff auf die gespeicherten Daten mittels TLS erlaubt.

Durch die Nutzung des Secret Stores kann auch die interne Kommunikation von EdgeX verschlüsselt werden. Neben dem Speicher für geheime Informationen bietet der Service die Möglichkeit Daten zu verschlüsseln und zu entschlüsseln, ohne diese zwischenzuspeichern. Die Verbindung zum Secret Store ist mittels TLS abgesichert. Die Microservices von EdgeX unterstützen die Verschlüsselung der internen Kommunikation jedoch nicht. Diese muss explizit implementiert werden, wodurch die Services auch neu kompiliert und deployed werden müssen.

Die Authentifizierung der Nutzer kann über das API Gateway von EdgeX konfiguriert werden. Dieses bietet einen zentralisierten Zugriffspunkt auf alle Services und unterstützt eine Authentifikation mittels JSON Web Token (JWT) oder OAuth2. Die Konfiguration wird dabei durch eine Datei vorgenommen, in welcher auch die Routen zu den entsprechenden Backend-Services angelegt werden können. Standardmäßig startet das Gateway mit einer vordefinierten Konfiguration, welche Routen zu den wichtigsten Services von EdgeX enthält. Die Kommunikation vom Client zu EdgeX findet anschließend nur noch über das Gateway statt und die konfigurierte Authentifikation muss bei jeder Anfrage angegeben werden.

Mit Hilfe der in diesem Abschnitt ermittelten Einzelbewertungen der Evaluationsziele kann nun eine Gesamtbewertung der EdgeX Foundry ermittelt werden. Durch das Vorgehen nach dem GQM-Paradigma können die Bewertungen aller Metriken direkt auf die spezifischen Evaluationsziele zurückgeführt werden, was die Interpretation der Ergebnisse erleichtert.

3.5 Bewertung von EdgeX

Um aus den Bewertungen der einzelnen Ziele aus Abschnitt 3.4 eine Gesamtbewertung von EdgeX zu errechnen, wurden diese in Tabelle 3.19 in einer Bewertungsmatrix zusammengefasst. In dieser Arbeit wurde angenommen, dass alle definierten Evaluationsziele für den Einsatz des Frameworks relevant sind und somit zu gleichen Teilen in die Bewertung einfließen sollten.

Ziel	Bewertung	Gewichtung	Gewichtete Bewertung
Nachhaltigkeit und Akzeptanz	60%	20%	12%
Softwarequalität	75%	20%	15%
Umgang mit anderen Systemen	68,33%	20%	13,67%
Verarbeitung von Daten	43,33%	20%	8,67%
Sicherheit und Zuverlässigkeit	60%	20%	12%

Tabelle 3.19: Bewertung der Evaluationsziele

Durch die Auswertung der Bewertungsmatrix ergibt sich eine Gesamtwertung von 61,33%. In den ersten zwei Evaluationszielen erreicht EdgeX hohe Bewertungen. Diese wurden im Vergleich zu anderen bekannten Edge- und Fog-Frameworks gemessen und können ein Indikator für eine wartbare und nachhaltige Lösung sein. EdgeX zeigt ein solides Verhalten beim Umgang mit unterschiedlichen Geräten, wie IoT-Geräte oder Cloud-Instanzen. Es bietet zudem SDKs zur einfachen Erweiterung und Unterstützung anderer Geräte. Eine weitere entscheidende Fähigkeit eines Fog-Computing-Frameworks ist die lokale Verarbeitung und Entscheidungsfindung. Hier zeigt EdgeX ein unterdurchschnittliches Verhalten, wie an der Bewertung von 43,33% erkannt werden kann.

Der größte Schwachpunkt scheint also in der Verarbeitung der Daten vor der Integration in ein Cloud- bzw. Northbound-System zu liegen. Die von EdgeX bereitgestellten Funktionen zur Vorverarbeitung der Daten innerhalb der Funktionspipelines weisen nicht die nötige Dynamik auf, um mit einer heterogenen und sich verändernden Systemlandschaft umgehen zu können. Anwendungsspezifische Filterfunktionen müssen explizit implementiert werden und Funktionen zur Betrachtung mehrerer Ereignisse in einem Längen- oder Zeitfenster werden nicht unterstützt. Aus diesen Gründen soll EdgeX im folgenden Kapitel um eine eventgetriebene Architektur unter Verwendung von Complex Event Processing (CEP) erweitert werden.

4 Erweiterung von EdgeX

Um den im Rahmen der Evaluierung erkannten Schwachpunkt bei der Verarbeitung von Daten zu verbessern, befasst sich das folgende Kapitel mit dem Entwurf und der Entwicklung eines Systems, um ein Event Processing Network innerhalb von EdgeX aufzubauen. Dazu wird in Abschnitt 4.1 zunächst eine Softwarearchitektur entworfen und beschrieben. Anschließend wird diese Architektur im Abschnitt 4.2 umgesetzt. Um den Effekt des entwickelten Systems zu überprüfen, wird dieses im letzten Abschnitt getestet und im Kontext von EdgeX evaluiert.

4.1 Erstellung eines Konzeptes

Bevor mit der Implementierung des Systems begonnen werden konnte, musste zunächst ein Konzept erstellt werden. Das Ergebnis der Konzeption ist eine Softwarearchitektur, welche als Referenz für die Umsetzung dient. Dafür mussten zunächst die Anforderungen an das System geklärt werden, dies geschieht in Abschnitt 4.1.1. Die Kontextabgrenzung aus Abschnitt 4.1.2 bietet eine Top-Down-Sicht auf das System mit seinen angrenzenden Nachbarsystemen. In der im Abschnitt 4.1.3 beschriebenen Bausteinsicht wird die statische Struktur der Architektur beschrieben. Auf Basis dieser Struktur werden in Abschnitt 4.1.4 die wichtigsten Anwendungsfälle erläutert. Die Verteilungssicht im Abschnitt 4.1.5 zeigt die Aufteilung der Komponenten des Systems auf die technische Infrastruktur.

4.1.1 Anforderungsanalyse

Die Anforderungsanalyse hat das Ziel die funktionalen und nicht-funktionalen Anforderungen an das zu entwickelnde System zu ermitteln. Die Anforderungen wurden vom Autor unter Berücksichtigung der im Abschnitt 2.5 beschriebenen Eigenschaften von CEP und der Evaluationsergebnisse aus Abschnitt 3.4 erhoben. Die Berücksichtigung weiterer Personen und Kontexten kann zu der Erhebung weiterer Anforderungen führen.

1. Deklarative Regeln: Der Anwender soll Regeln für die Erkennung von Ereignismustern und der Ableitung entsprechender Reaktionen auf eine deklarative Weise zur Laufzeit definieren können.

2. Erzeugen neuer Ereignistypen: Der Anwender soll neue Ereignistypen erzeugen können, um somit die Generierung komplexer Ereignisse zu unterstützen.

3. Automatische Übernahme von Ereignistypen: Das System soll die in EdgeX registrierten Ereignistypen automatisiert auslesen und verwenden können.

4. Sliding-Windows: Der Anwender soll Zeit- und Längenfenster definieren können, um die Aggregation von Ereignissen innerhalb dieses Fensters, sowie die Definition von zeitlichen Abhängigkeiten zu definieren.

5. Unterstützung von Application-Services: Der Anwender soll einen Export zu einem Application-Service definieren können, um die Ereignisse zu einem Northbound-System zu transportieren.

6. Definition des EPN: Die Struktur des EPN soll dynamisch anpassbar sein. Dafür sollen EPAs zur Laufzeit durch den Anwender registriert und entfernt werden können.

Der Anwender definiert die Regeln und konfiguriert das EPN, um die Fachlogik umzusetzen. Das System empfängt die Ereignisse von EdgeX und verarbeitet diese. Diese Interaktion ist in Abbildung 4.1 abgebildet.

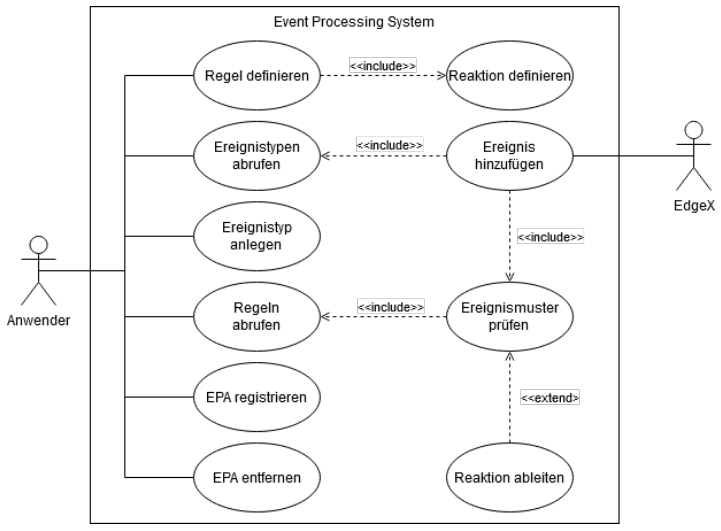

Abbildung 4.1: Anwendungsfalldiagramm für das Event Processing System

Neben den funktionalen Anforderungen sind für die Entwicklung des Systems auch die nicht-funktionalen Anforderungen relevant. Diese können die Entscheidungen über die Architektur und den Entwurf maßgeblich beeinflussen.

- Plattformunabhängigkeit: Das System muss auf Grund der heterogenen Systemlandschaft von Fog-Computing auf den selben Plattformen wie EdgeX lauffähig sein. Dazu gehören unterschiedliche Betriebssysteme (Linux, Windows, MacOS) sowie verschiedene Prozessorarchitekturen (x86, ARM).
- Erweiterbarkeit: Um unterschiedliche Arten von Reaktionen auf der Ereignisbehandlungsschicht umsetzen zu können, soll das System erweiterbar gehalten werden. Die Komponente zur Behandlung der erkannten Ereignismuster muss daher zur Laufzeit durch eine alternative Implementierung ersetzt werden können.
- Wiederherstellbarkeit: Da bei einem Einsatz des Systems auf heterogenen, unsicheren Fog-Knoten mit Ausfällen zu rechnen ist, sollte es möglich sein Daten wiederherzustellen. Bei einem Ausfall des Systems sollen die persistenten Daten (Regeln, Ereignistypen, registrierte Agenten) ohne Interaktion des Nutzers erneut angelegt werden.
- Skalierbarkeit: Bei einer hohen Datenlast durch eintreffende Ereignisse soll es möglich sein einzelne Komponenten des Systems (In-Adapter, Event Processing Agents, Out-Adapter) zu replizieren. Durch eine entsprechende Konfiguration der Services durch den Nutzer sollen die Daten entsprechend verteilt werden können.

Zur Erfüllung der Anforderungen wird im Folgenden eine Softwarearchitektur entworfen, welche durch die nicht-funktionalen Anforderungen beeinflusst wird. Dazu wird im nächsten Abschnitt zunächst der Kontext des zu entwickelnden Systems untersucht.

4.1.2 Kontextabgrenzung

Das System soll im Kontext von EdgeX eingesetzt werden und nutzt somit die von EdgeX zur Verfügung gestellten Services. Das in Abbildung 4.2 gezeigte Kontextdiagramm zeigt das System mit seinen Nachbarsystemen.

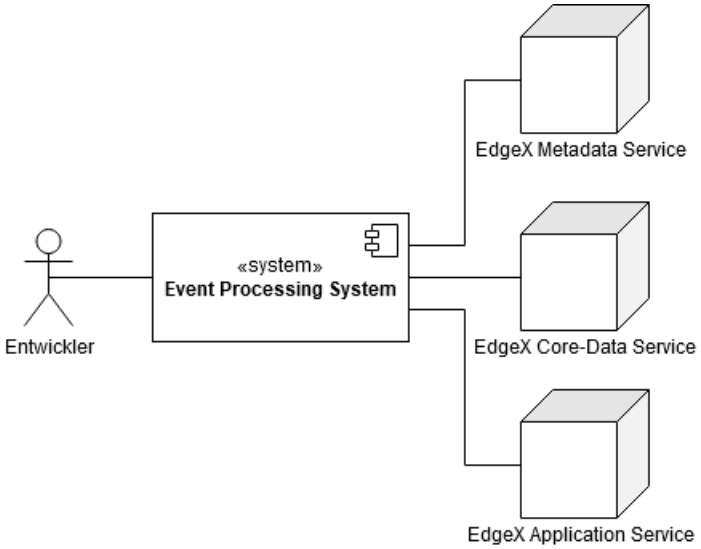

Abbildung 4.2: Kontextdiagramm

Für die Nutzung der in EdgeX registrierten Ereignistypen ist eine Kommunikation mit dem Metadata-Service notwendig. Die vom Core-Data-Service kommunizierten Ereignisse müssen diesen Ereignistypen zugeordnet werden und können anschließend für die Suche nach Ereignismustern verwendet werden. Damit die erzeugten Ereignisse als Reaktion auf das Vorhandensein eines Ereignismusters an ein Northbound-System gesendet werden können, muss ein entsprechender Application-Service genutzt werden. Das System bietet eine Schnittstelle an den Anwender, damit dieser die Regeln und Ereignistypen des Systems verwalten, sowie die Struktur des EPN anpassen kann. Auf diese Weise sollen fachliche Geschäftsprozesse auch zur Laufzeit implementierbar sein.

4.1.3 Bausteinsicht

Um die im Abschnitt 4.1.1 beschriebenen funktionalen und nicht-funktionalen Anforderungen abzudecken wurde eine Softwarearchitektur entworfen. Diese basiert auf der von Bruns [BD10] vorgeschlagenen Referenzarchitektur und unterteilt sich in drei Schichten. Die Referenzarchitektur ist in Abbildung 4.3 zu sehen.

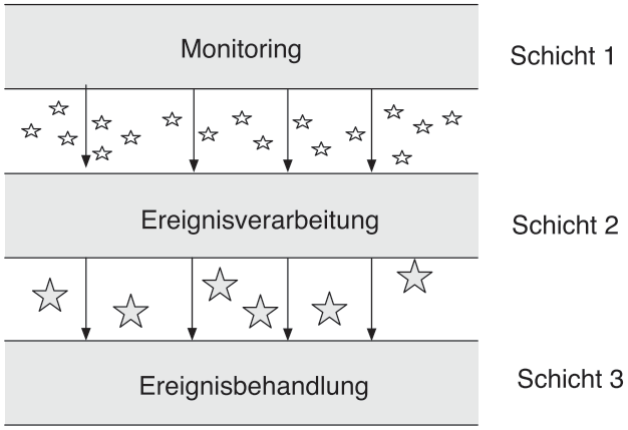

Abbildung 4.3: Schichten in einer eventgetriebenen Architektur [BD10]

Auf der Monitoring-Schicht werden die verschiedenen Ereignisquellen angebunden. Die empfangenen Informationen werden in das interne Format transformiert und über einen Ereigniskanal zur nächsten Schicht weitergeleitet. Die Ereignisverarbeitungsschicht führt die regelbasierte Verarbeitung der Ereignisse durch. Hier werden komplexe Ereignisse erzeugt und Reaktionen abgeleitet. Diese Reaktionen werden auf der Ereignisbehandlungsschicht implementiert und ausgeführt.

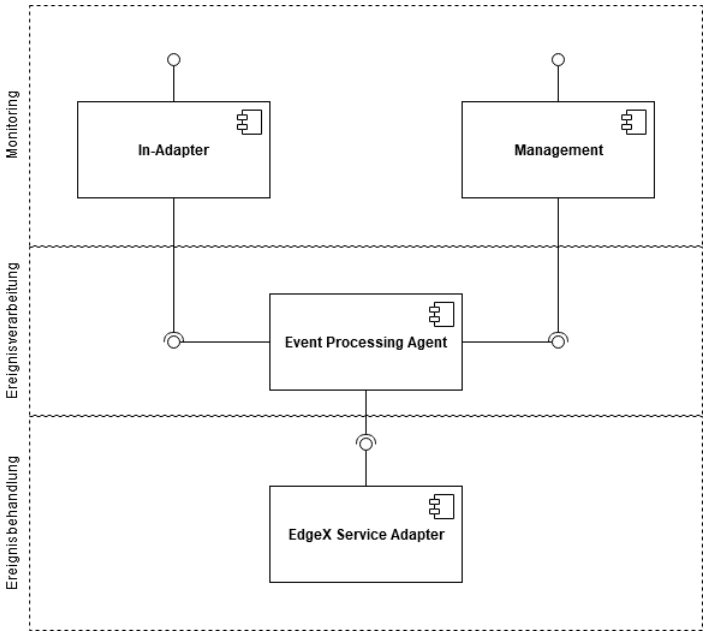

Abbildung 4.4: Komponentendiagramm

Die In-Adapter-Komponente empfängt die Ereignisse von EdgeX und transformiert diese in das interne Format. Die Management-Komponente bietet eine Fassade zur Verwaltung des Systems. Hier können EPAs registriert werden, um den Aufbau des EPN zu verwalten. Zudem können Ereignistypen und Regeln angelegt und über die registrierten EPAs verteilt werden. Die Event-Processing-Agent-Komponente ist für die Anwendung der Regeln zuständig und löst eine entsprechende Reaktion durch den Aufruf des EdgeX-Service-Adapters aus. Dieser transformiert die Ereignisse in die EdgeX-Repräsentation und führt die Kommunikation mit den Application-Services durch.

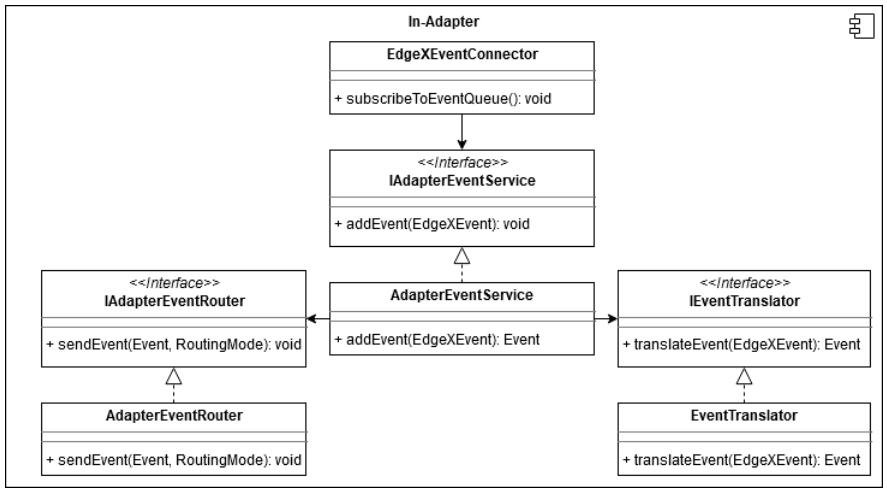

Abbildung 4.5: Klassendiagramm - In-Adapter

Der strukturelle Aufbau der In-Adapter-Komponente kann in Abbildung 4.5 gesehen werden. Der *EdgeXEventConnector* empfängt die Ereignisse von der Message-Queue der EdgeX-Instanz und leitet diese an den *AdapterEventService* weiter. Die Klasse *EventTranslator* ist für die Transformation der Ereignisse in das interne Format zuständig. Über eine zusätzliche Klasse *AdapterEventRouter* können verschiedene Arten des Routing implementiert werden, wie das in Abschnitt 2.5.4 beschriebene Content-based Routing, um somit eine Verteilung der Ereignisse im EPN zu realisieren.

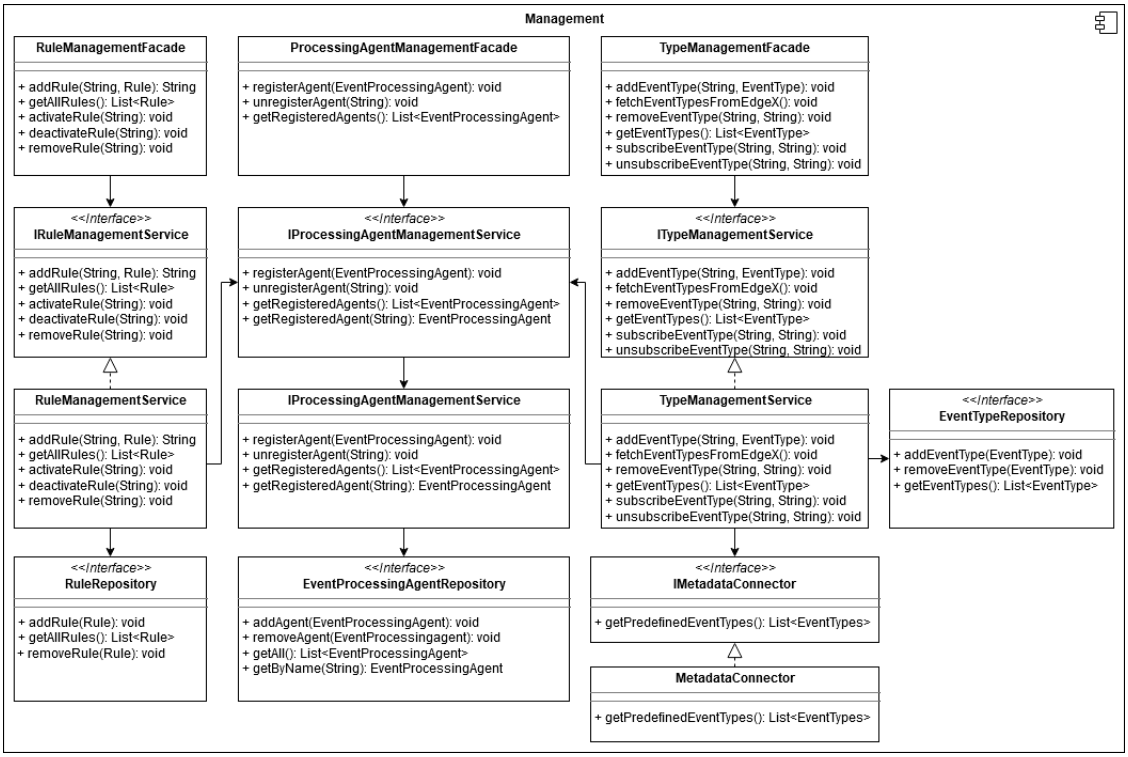

Abbildung 4.6: Klassendiagramm - Management

Die Management-Komponente versteckt die interne Struktur des EPN und bietet einen einheitlichen Zugriff auf das Netzwerk. Die Service-Klassen kommunizieren jeweils mit den EPAs der Ereignisverarbeitungsschicht. Zusätzlich werden die Konfigurationen der EPAs persistiert, um das System bei einem Ausfall mit allen Konfigurationen erneut starten zu können. Zu diesen Konfigurationen gehören die registrierten Ereignistypen sowie die Regeln. Die registrierten EPAs des Netzwerks werden ebenfalls persistiert und können bei einem Neustart der Management-Komponente abgerufen werden. Über den *MetadataConnector* werden die Ereignistypen von EdgeX geladen und anschließend im System gespeichert. So soll die Integration des Systems in die EdgeX-Infrastruktur erleichtert werden und die Ereignisse der Geräte einfach nutzbar sein.

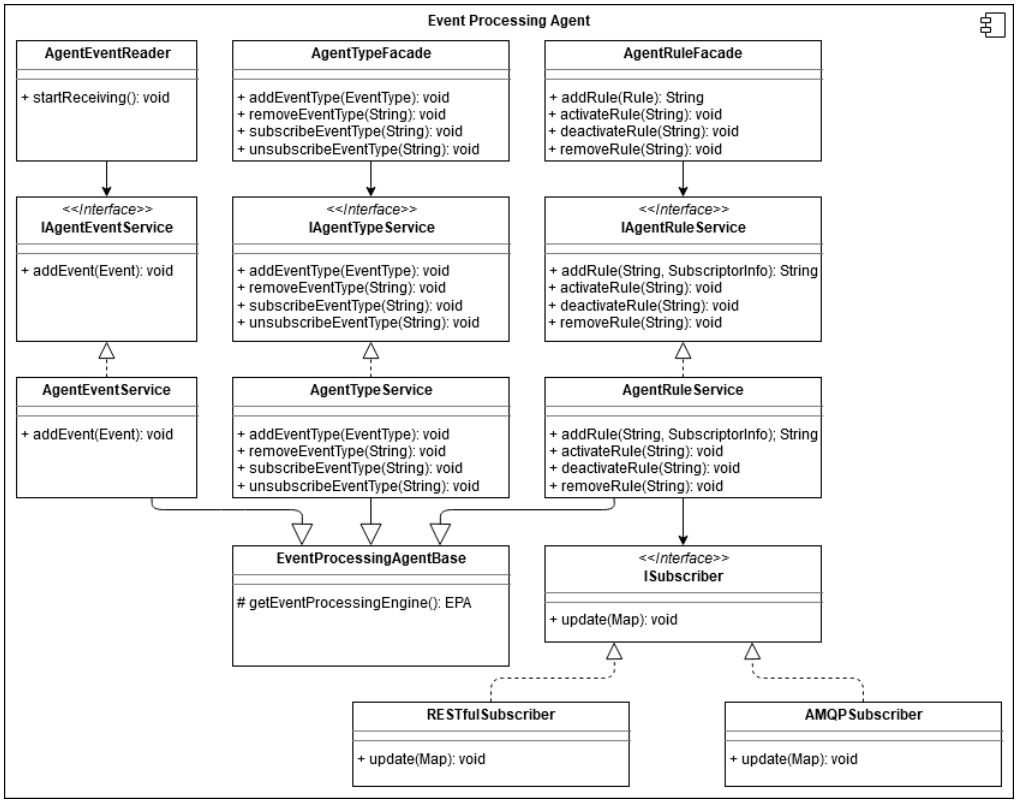

Abbildung 4.7: Klassendiagramm - Event Processing Agent

Der Event Processing Agent befindet sich auf der Ereignisverarbeitungsschicht und bietet zwei Fassaden für Ereignistypen und Regeln. Der *AgentEventReader* verbindet sich mit einem Message-Queue-Broker, um die Ereignisse der In-Adapter-Komponente zu empfangen. Für den Zugriff auf die Event Processing Engine werden insgesamt drei Schnittstellen angeboten. Die Implementierungen dieser Schnittstellen nutzen eine gemeinsame Basis für die Verbindung zur Event Processing Engine. Zur Anbindung verschiedener Out-Adapter auf der Ereignisbehandlungsschicht dient die Schnittstelle *ISubscriber*. Diese leitet die erzeugten Ereignisse je nach Implementierung über unterschiedliche Protokolle an die nächste Schicht weiter. Alternativ können die abgeleiteten Ereignisse auch an andere EPAs des Netzwerks geschickt werden, um somit eine hierarchische Verarbeitung zu ermöglichen. Die Konfigurationsdaten der Verbindung erhält die jeweilige Klasse bei der Initialisierung und werden vom Nutzer beim Anlegen einer Regeln definiert.

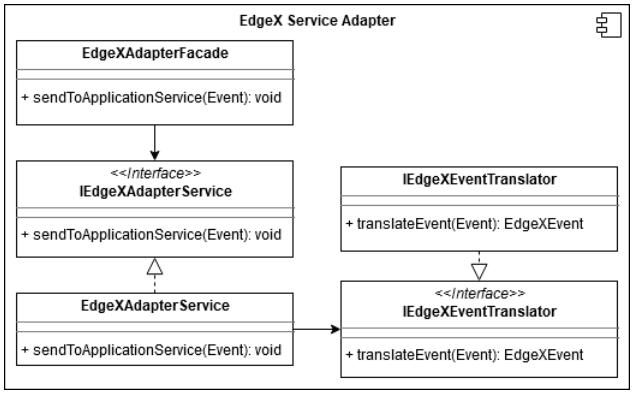

Abbildung 4.8: Klassendiagramm - EdgeX Service Adapter

In Abbildung 4.8 ist der interne Aufbau des Out-Adapters aufgezeigt. Ereignisse der höheren Schicht werden an die Schnittstelle der *EdgeXAdapterFacade* geschickt. Der *EdgeXAdapterService* steuert die Verarbeitung und den Transport zum entsprechenden Applikation-Service. Der *EdgeXEventTranslator* wird verwendet, um die Ereignisse in das Format von EdgeX zu transformieren.

4.1.4 Laufzeitsicht

Im Folgenden sollen die beiden wichtigsten Interaktionen mit dem System aufgezeigt werden. Abbildung 4.9 zeigt die Logik zum Hinzufügen einer neuen Regel durch den Anwender.

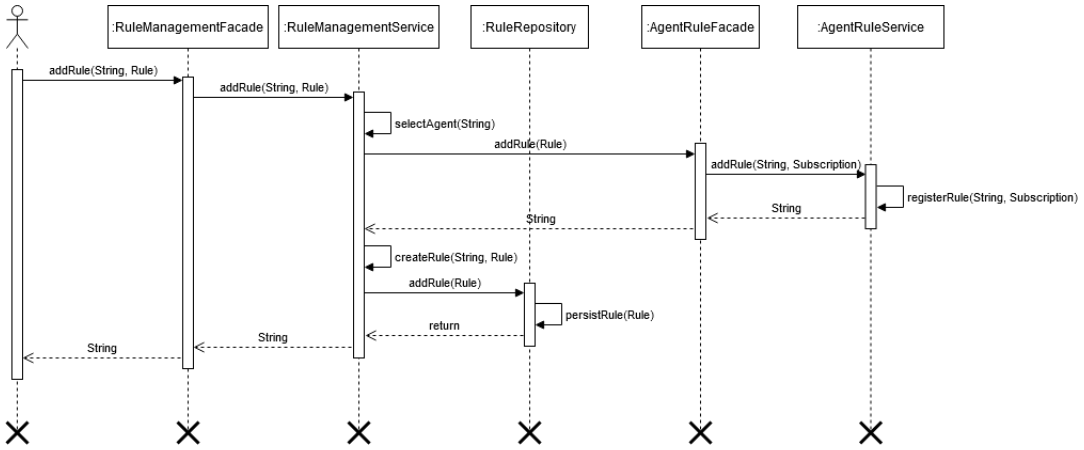

Abbildung 4.9: Sequenzdiagramm - Hinzufügen einer Regel

Das Hinzufügen einer Regel geschieht über die einheitliche Fassade auf der Monitoring-Schicht. Hier wird der korrekte Event Processing Agent gesucht und die Anfrage an diesen weitergeleitet. Der EPA fügt die Regel hinzu und liefert einen eindeutigen Namen. Wenn die Regel korrekt hinzugefügt wurde, wird sie persistiert um den Zustand des Systems nach einem Fehler wieder herstellen zu können. Die dabei erzeugte ID wird zur Verwaltung der Regel durch den Anwender genutzt.

Ein weiterer Anwendungsfall ist das Hinzufügen von Ereignissen durch die EdgeX Foundry. Die Ablauflogik dafür kann in Abbildung 4.10 gesehen werden.

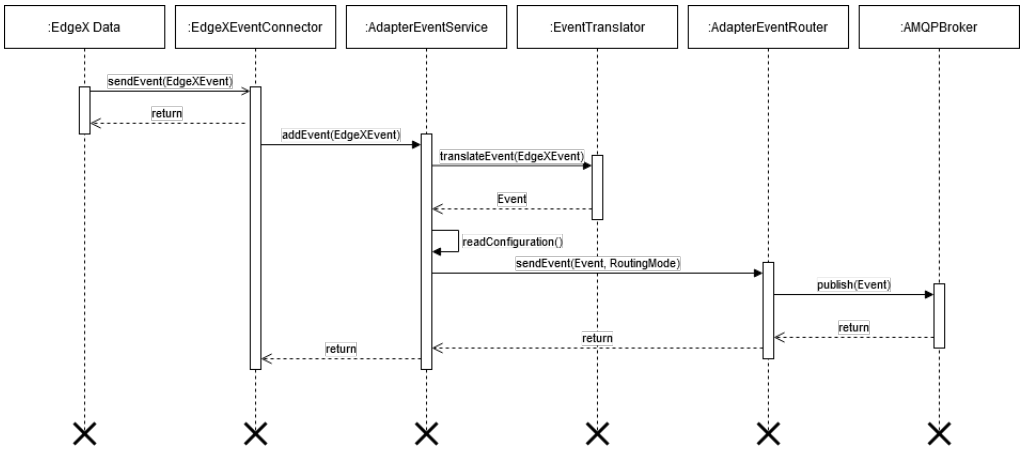

Abbildung 4.10: Sequenzdiagramm - Hinzufügen eines Ereignisses

Die Ereignisse von EdgeX werden über den In-Adapter zum System hinzugefügt. Dazu werden die Ereignisse vom *EdgeXEventConnector* empfangen und an den *AdapterEventService* weitergeleitet. Um die Ereignisse von EdgeX in das interne Format zu überführen wird der *EventTranslator* genutzt. Anschließend wird die Konfiguration abgerufen, um zu ermitteln, auf welche Weise die Ereignisse an die EPAs der Ereignisverarbeitungsschicht weitergeleitet werden sollen. Diese Information wird zusammen mit dem Ereignis an den *AdapterEventRouter* gegeben. Dieser nutzt einen Message-Queue-Broker als Ereigniskanal, um das Content-based Routing umzusetzen.

4.1.5 Verteilungssicht

Um die Skalierbarkeit des Systems zu gewährleisten, werden die in der Abbildung 4.4 beschriebenen Komponenten als eigenständige Services konzipiert. Dadurch ist es möglich

einen Service bei Bedarf zu replizieren. Durch die Betrachtung der EPAs als eigenständige Services kann ein EPN gebildet werden. Eine entsprechendes Verteilungsdiagramm findet sich in Abbildung 4.11.

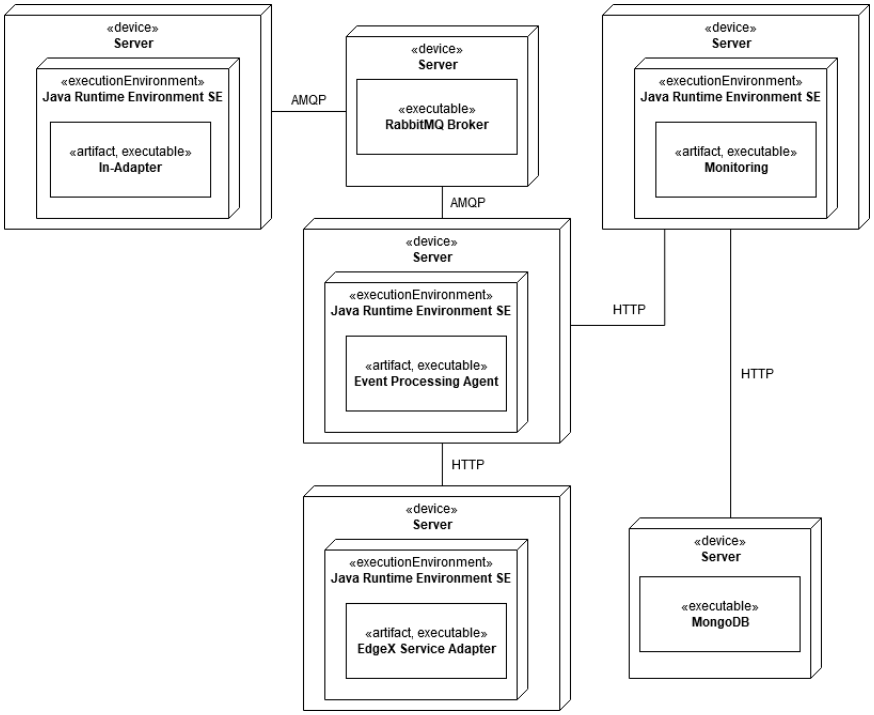

Abbildung 4.11: Verteilungsdiagramm

Über einen Message-Queue-Broker wird eine Message Oriented Middleware (MOM) umgesetzt. Dies ermöglicht eine lose Kopplung zwischen den EPAs. Über unterschiedliche Topics kann ein Content-based Routing realisiert werden. Für diese Arbeit wurde RabbitMQ[1] als leichtgewichtiger Message-Broker gewählt, welcher per Advanced Message Queuing Protocol (AMQP) angebunden wird. Für die Persistenz der Regeln und Ereignistypen wurde MongoDB[2] gewählt. MongoDB erlaubt eine performante Speicherung der Daten im JSON-Format. Ein weiterer Vorteil von MongoDB ist, dass die Struktur der Daten nicht feststehen muss. Da die genaue Struktur der Ereignistypen erst zur Laufzeit feststeht, ist diese Eigenschaft für diese besonders nützlich.

[1] https://www.rabbitmq.com/
[2] https://www.mongodb.com/

4.2 Implementierung der Komponenten

Im Folgenden sollen die wichtigsten Entscheidungen während der Implementierung des Systems aufgezeigt werden. Dies umfasst die Wahl einer geeigneten Event Processing Engine für die EPA-Komponente, die Kommunikation zwischen den Services des Systems und den Microservices von EdgeX sowie die Art des Deployments und die Dokumentation des Systems und seiner Schnittstellen.

4.2.1 Auswahl einer Event Processing Engine

Für die Auswahl einer geeigneten Event Processing Engine stehen mehrere Lösungen zur Verfügung. Da EdgeX auf Open-Source-Projekte setzt, wurde dies bei der Wahl einer Engine berücksichtigt. Die bekanntesten Vertreter dieser Art sind Apache Storm, Apache Spark, Apache Flink, Drools Fusion und Esper. Alle Produkte wurden für die Realisierung der Event Processing Engine innerhalb der EPA-Komponente in Betracht gezogen.

Ein wichtiges Kriterium ist die Möglichkeit Regeln auf eine deklarative Weise zu definieren. Alle genannten CEP-Produkte außer Drools unterstützen eine an SQL angelehnte Event Processing Language, welche für die einfache Benutzbarkeit durch den Anwender bevorzugt wird. Drools implementiert eine eigene Regelsprache, über welche Bedingungen und Aktionen innerhalb einer Datei definiert werden können. Die SQL-ähnliche Abfragesprache von Apache Storm ist zum aktuellen Zeitpunkt nur experimentell verfügbar. Apache Spark hat ein auf Batch-Verarbeitung spezialisiertes Verarbeitungskonzept. Beim Structured Streaming werden neue Daten kontinuierlich an eine Tabelle angehängt. Abfragen können wie auf statischen Tabellen ausgeführt werden. Apache Flink erlaubt die Nutzung einer an SQL angelehnten Abfragesprache durch die Verwendung des Schlüsselwortes *MATCH_RECOGNIZE*, dabei wird jedoch nur ein eingeschränkter Teil der Funktionalität angeboten. Esper bietet eine SQL-ähnliche Regelsprache über welche Ereignismuster bestimmt und komplexe Ereignisse abgeleitet werden können.

Zusätzlich soll die eingesetzte Engine die Definition von Sliding-Windows erlauben, um somit eine Aggregation von Ereignissen innerhalb dieses Fensters zu ermöglichen. Bei Storm wird ein Windowing nicht über die angebotene SQL-Schnittstelle unterstützt, während Flink dies nur über die sog. Table-API anbietet, jedoch nicht über die CEP-Schnittstelle und SQL-Abfragesprache. Drools ermöglicht die Definition von Zeit- und

Längenfenstern über die eigene Abfragesprache. Spark und Esper unterstützen Zeit- und Längenfenster über ihre SQL-ähnliche Sprache. Esper bietet darüber hinaus auch sortierte und gruppierte Fenster an.

Für die Umsetzung der Event-Processing-Agent-Komponente wurde sich für Esper[3] als Event Processing Engine entschieden. Esper hat auf Grund der an den SQL-Standard angelehnten Event Processing Language und der umfassenden Möglichkeiten zur Definition von Sliding-Windows einen Vorteil gegenüber den anderen Produkten. Bei der Änderung von Anforderungen oder dem Hinzukommen oder Weiterentwicklung der Produkte sollte eine erneute Evaluierung durchgeführt werden. Die im Abschnitt 4.1 beschriebene Architektur wurde modular entworfen, sodass die Implementierung der Event Processing Engine substituiert werden kann.

Esper bietet eine gute Integration mit der Programmiersprache Java, sodass diese für sämtliche Services verwendet wurde. Mit Hilfe des Build-Management-Tools Gradle können die Abhängigkeiten der Services auf einfache Weise definiert und die Projektumgebung schnell auf anderen Systemen aufgesetzt werden. Java unterstützt zudem die Nutzung des Spring-Frameworks. Dieses bietet eine einfache Möglichkeit ein Dependency Injection (DI) umzusetzen. Zudem unterstützt Spring beim Aufbau von REST-Fassaden und der Anbindung von AMQP-Brokern, welche für die Kommunikation zwischen den Services und den Anwendern genutzt werden.

4.2.2 Kommunikation innerhalb des Systems

Neben den REST-Schnittstellen wird ein Ereigniskanal benötigt, über welchen die Ereignisse der EdgeX Foundry, sowie die abgeleiteten, komplexen Ereignisse an entsprechende EPAs im EPN kommuniziert werden können. Für die Implementierung wurde sich für eine nachrichtenorientierte Middleware (MOM) entschieden. Dadurch entsteht eine lose Kopplung zwischen dem In-Adapter und den EPAs sowie zwischen den EPAs des Netzwerks. Es ist somit möglich dem Netzwerk weitere EPAs zur Laufzeit hinzuzufügen. Wie in Abschnitt 2.5.4 beschrieben, kann ein entsprechender Message-Queue-Broker die Verteilung der Ereignisse auf die Services übernehmen und dadurch ein Content-based Routing implementieren. Die Nutzung von AMQP als Standard für die Kommunikation in einer MOM erlaubt es die Realisierung des Message-Brokers gegen einen anderen AMQP-Broker auszutauschen.

[3] http://www.espertech.com/esper/

Damit die EPAs unabhängig von Veränderungen der Ereignis-Repräsentation bleiben, wird zwischen einem internen und einem externen Format unterschieden. Die Transformation zwischen den Formaten findet zum einen im In-Adapter und umgekehrt im EdgeX Service Adapter statt. Für die Transformation werden Metadaten über die Geräte und Geräteprofile von EdgeX benötigt. Diese Informationen können durch REST-Aufrufe vom Metadata-Service bezogen werden. Der EdgeX Service Adapter nimmt die erzeugten Ereignisse des EPN entgegen, transformiert diese in das EdgeX-Format und sendet sie zu einem Application-Service. Die Application-Services von EdgeX bieten die Möglichkeit eine Funktions-Pipeline durch einen REST-Aufruf zu starten. Dieser Mechanismus wird genutzt, um die erzeugten, komplexen Ereignisse in EdgeX zu integrieren. Für bestehende Application-Services muss somit nur die Konfiguration geändert werden, sodass diese ihre Daten nicht mehr direkt beziehen. Weitere Anpassungen der Funktions-Pipelines sind nicht nötig.

4.2.3 Konfiguration und Deployment

Um die im Abschnitt 4.1.1 spezifizierte Plattformunabhängigkeit zu erreichen, werden für die Bereitstellung der Services Docker-Container genutzt. Diese bieten eine Virtualisierung für Linux-, Windows- und MacOS-Betriebssysteme. Zudem wird eine Ausführung auf unterschiedlichen Prozessorarchitekturen unterstützt. Um eine Integration des Systems in die EdgeX-Architektur zu erlangen, wurde die Docker-Compose-Konfiguration von EdgeX erweitert. Die Konfiguration der Services wird durch entsprechende Umgebungsvariablen vorgegeben. Eine Änderung dieser Konfiguration erfordert den Neustart des jeweiligen Services.

Um sowohl den Anwender, als auch die Entwicklung eigener Systeme zu unterstützen, welche die Schnittstellen der Services nutzen, wurde eine Entwicklerdokumentation geführt. Diese Dokumentation besteht zum einen aus JavaDoc-Kommentaren, um die Wartung und Erweiterung der Services zu erleichtern. Zum anderen wurde für alle REST-Schnittstellen eine Swagger-Dokumentation[4] erzeugt. Diese beschreibt die Schnittstellen mit ihren erwarteten Parametern sowie Statuscodes und Rückgabewerten.

[4] https://swagger.io/

4.3 Evaluierung der Erweiterung

Die Tabelle 4.1 zeigt die Abdeckung der Anforderungen durch das entwickelte System. Alle funktionalen Anforderungen, welche im Abschnitt 4.1.1 erhoben wurden, konnten durch die beschriebene Softwarearchitektur konzeptionell geplant und später implementiert werden. Zur Überprüfung der Funktionalität wurden die einzelnen Komponenten des Systems isoliert mit Hilfe von automatisierten Tests getestet. Andere Komponenten, sowie Schnittstellen zu anderen Services wurden dabei durch Mock-Objekte simuliert.

Anforderung	Konzeptioniert	Implementiert	Getestet
1. Deklarative Regeln	Ja	Ja	Ja
2. Erzeugen neuer Ereignistypen	Ja	Ja	Ja
3. Übernahme von Ereignistypen	Ja	Ja	Ja
4. Sliding-Windows	Ja	Ja	Ja
5. Unterstützung von Application-Services	Ja	Ja	Ja
6. Definition des EPN	Ja	Ja	Ja

Tabelle 4.1: Abdeckung der spezifizierten Anforderungen

Um die Integration des Systems in die EdgeX-Infrastruktur zu testen, wurde dieses in das im Abschnitt 3.2 beschriebene Testszenario eingebunden. Dafür wurden die Services des Systems zur Docker-Compose-Konfiguration hinzugefügt und werden zusammen mit den anderen Services von EdgeX gestartet. Dies erlaubt eine erneute Bewertung des im Abschnitt 3.1.4 beschriebenen Evaluationsziels.

Das Treffen von Entscheidungen über Regeln wird durch das entwickelte System vereinfacht. Esper erlaubt die Definition von komplexen Ereignismustern und Zeitfenstern, sodass mehrere Ereignisse betrachtet werden können. Komplexe Ereignisse, welche von einem EPA erzeugt werden, können zur Erkennung weiterer Ereignismuster genutzt werden. Dies ermöglicht eine hierarchische Anordnung von Verarbeitungsschritten. Die Umsetzung der Reaktionen geschieht durch entsprechende Out-Adapter, wie den EdgeX Service Adapter. Durch die lose Kopplung der Microservice-Architektur ist es möglich eigene Services zu entwickeln, welche die gewünschten Reaktionen implementieren.

Die Definition von deklarativen Regeln ermöglicht eine dynamische Filterung der eingehenden Sensordaten zur Laufzeit. Diese Regeln können über die im Management-Service bereitgestellte Schnittstelle angelegt und verwaltet werden und erfordern nicht den Neustart einzelner Services. Die Betrachtung von Ereignissen innerhalb eines Zeit- bzw. Län-

genfensters ist durch die Verwendung von Esper als Event Processing Engine ebenfalls möglich. Dadurch kann auch die Aggregation von Werten innerhalb der Regeln erfolgen.

Komplexe Algorithmen können als Reaktion auf gefundene Ereignismuster im Out-Adapter umgesetzt werden. Dies erfordert jedoch noch immer die gesonderte Implementierung und Deployment des Services. Die Anbindung an das Event Processing Network ist durch die lose Kopplung der Services jedoch problemlos auch zur Laufzeit möglich. Das entwickelte System vereinfacht somit nicht die Nutzung komplexer Algorithmen.

Durch die deklarativen Regeln können komplexe Ereignisse erzeugt werden. Diese stellen Ereignisse einer höheren Abstraktionsebene dar und können Werte mehrerer Ereignisse aggregieren. Die Umwandlung in andere Datenformaten kann auf zwei Wegen geschehen. Zum einen kann die Transformation in den Application-Services von EdgeX umgesetzt werden, wie es im Abschnitt 3.4 beschrieben wurde. Des Weiteren kann eine Transformation in ein anderes Datenformat auch in einem Out-Adapter implementiert werden. Das Hinzufügen von neuen Umwandlungsschritten erfordert jedoch auch hier das erneute Deployment des Services. Daher konnte auch für dieses Bewertungskriterium keine Verbesserung erreicht werden.

Eine Übersicht über die aktualisierten Bewertungsergebnisse vom Evaluationsziel 'Verarbeitung von Daten' kann in Tabelle 4.2 gesehen werden.

Metrik	Kurzbeschreibung	Kennwert	Punkte	Bewertung
M4.1	Entscheidungen über Regeln	Wenig Aufwand	2	16,67%
M4.2	Filterung von Daten	Ohne Aufwand	3	30%
M4.3	Komplexe Algorithmen	Viel Aufwand	1	5%
M4.4	Umwandlung von Datenformaten	Wenig Aufwand	2	20%
	Summe der Bewertungen			71,67%

Tabelle 4.2: Verarbeitung von Daten - Bewertung der Metriken nach Erweiterung

Unter der Verwendung des entwickelten Systems wird für dieses Evaluationsziel eine Bewertung von 71,67% erreicht. Dies ist eine signifikante Verbesserung gegenüber der vorherigen Bewertung von 43,33%. Durch die Verwendung dieses Wertes für die im Abschnitt 3.5 beschriebene abschließende Bewertung von EdgeX wird eine Gesamtwertung von 67% erreicht. Das Evaluationsziel 'Verarbeitung von Daten' besitzt somit die zweithöchste Bewertung nach der 'Softwarequalität'.

5 Fazit

Im folgenden Kapitel werden die Ergebnisse dieser Arbeit kurz zusammengefasst. Im Abschnitt 5.2 werden einige Möglichkeiten genannt, auf welche Weise diese Ergebnisse für weiterführenden Arbeiten genutzt werden können.

5.1 Zusammenfassung

Ziel dieser Arbeit war die Überprüfung der Praxistauglichkeit von EdgeX in einem Fog-Computing-System. Dazu wurden Bewertungskriterien auf Basis der in der Literatur genannten Eigenschaften eines IoT- und Fog-Computing-Systems aufgestellt. Um eine zielgerichtete Evaluation durchführen zu können, wurden die Bewertungskriterien nach dem GQM-Paradigma strukturiert. Die so entstandenen Metriken wurden durch Messungen und Auswertungen in einem Testszenario bewertet. Diese Evaluation hat ergeben, dass EdgeX ein solides Verhalten bei der Kommunikation mit anderen Systemen und der Anbindung von heterogenen Datenquellen zeigt. Der entscheidende Schwachpunkt liegt in der Verarbeitung dieser Daten und der Anpassung der Filterung und Transformation zur Laufzeit.

Im zweiten Teil dieser Arbeit wurde ein System entworfen und implementiert, um einen Ausgleich für diesen Schwachpunkt zu schaffen. Das entwickelte System dient zur dynamischen Konfiguration eines Event Processing Networks und der kontinuierlichen Anwendung deklarativer Regeln auf die von EdgeX erfassten Daten. Dazu wurde eine Softwarearchitektur auf Basis der von Bruns [BD10] vorgeschlagenen Schichtenarchitektur entworfen. Diese Architektur wurde anschließend implementiert und getestet. Abschließend wurde eine erneute Auswertung der EdgeX Foundry in Zusammenarbeit mit dem entwickelten System durchgeführt. Hierfür wurden die in der Evaluation definierten Bewertungskriterien verwendet, um einen direkten Vergleich zu dem vorherigen Ergebnis zu schaffen. Das Ergebnis dieser Auswertung zeigt, dass eine signifikante Verbesserung bei der Filterung und Entscheidungsfindung erreicht werden konnte.

5.2 Ausblick

Die Bewertungskriterien dieser Arbeit beziehen sich vor allem auf die Funktionalität eines typischen Fog-Computing-Systems. Um eine weitreichende Bewertung der EdgeX Foundry zu ermitteln, kann die Evaluation um weitere Bewertungskriterien ergänzt werden. Beispielsweise ist die Performanz in der Praxis ein entscheidendes Kriterium. Vergangene Arbeiten haben bereits den zeitlichen Vorteil von hybriden Lösungen von Cloud- und Fog-Computing aufgezeigt [ARMP17]. Eine solche Untersuchung könnte dabei helfen EdgeX als Lösung für zeitkritische Systeme zu bewerten.

Zusätzlich zu der alleinstehenden Bewertung von EdgeX kann auch eine vergleichende Evaluation von zwei oder mehreren der in Abschnitt 2.4.4 genannten Edge- bzw. Fog-Frameworks vorgenommen werden. Dazu können die in dieser Arbeit vorgestellten Bewertungskriterien verwendet werden. Ziel eines solchen Vergleiches wäre es die Unterschiede zwischen den Frameworks herauszustellen, um somit die geeigneten Lösungen für verschiedene Anwendungsfälle zu ermitteln.

Eine sinnvolle Erweiterung des entwickelten Systems ist die Ablösung der bestehenden Referenzimplementierung der Rules-Engine. Dazu wurden die Out-Adapter auf der Ereignisbehandlungsschicht der in Abschnitt 4.1 vorgestellten Architektur bereits als austauschbare Services entworfen. Die Entwicklung eines eigenen Out-Adapters würde die bestehende Rules-Engine somit ablösen können und mächtige Auswertungen durch deklarative Regeln erlauben, um entsprechende Reaktionen abzuleiten.

Des Weiteren kann der entworfene Event Processing Agent um weitere Reaktionen auf die Erkennung von Ereignismustern erweitert werden. Eine mögliche Ergänzung ist beispielsweise die dynamische Anreicherung von Ereignissen mit Kontextwissen. Durch Contraints wäre es möglich Regeln zu definieren, welche die eingehenden Ereignisse erfüllen müssen. Bei einer Verletzung dieser Regeln könnten diese Ereignisse verworfen oder mit externem Wissen angereichert werden.

Literaturverzeichnis

[ADP17] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[ARMP17] Badraddin Alturki, Stephan Reiff-Marganiec, and Charith Perera. A Hybrid Approach for Data Analytics for Internet of Things. *Proceedings of the Seventh International Conference on the Internet of Things (IoT '17)*, pages 189–190, 2017.

[Ast09] Kevin Asthon. That ' Internet of Things ' Thing. *RFID Journal*, 2009.

[Bas92] Victor R Basili. Software modeling and measurement: the Goal/Question/Metric paradigm, 1992.

[BCR94] Victor R Basili, Gianluigi Caldiera, and H Dieter Rombach. The goal question metric approach. *Encyclopedia of Software Engineering*, 2:528–532, 1994.

[BD10] Ralf Bruns and Jürgen Dunkel. *Event-Driven Architecture*. Springer Berlin Heidelberg, 2010.

[BMZA12] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *MCC'12 - Proceedings of the 1st ACM Mobile Cloud Computing Workshop*, pages 13–16, 2012.

[CML14] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile Networks and Applications*, 19(2):171–209, 2014.

[Cor20] Microsoft Corporation. Azure iot edge documentation. https://docs.microsoft.com/de-de/azure/iot-edge/, 2020. Accessed: 2020-06-13.

[DGC+16] Amir Vahid Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and Rajkumar Buyya. Fog Computing: Principles, architectures, and applications. In *Internet of Things: Principles and Paradigms*, chapter 4, pages 61–75. Morgan Kaufmann, 2016.

[EB09] Michael Eckert and François Bry. Aktuelles Schlagwort 'Complex Event Processing (CEP)'. *Informatik-Spektrum*, 32(2):163–167, 2009.

[Edg20] Linux Foundation Edge. Beatyl documentation. https://baetyl.readthedocs.io/en/1.0.0/overview/Whatis.html, 2020. Accessed: 2020-09-02.

[Fog20] FogFlow. Fogflow documentation. https://fogflow.readthedocs.io/en/latest/, 2020. Accessed: 2020-06-13.

[Fou20a] Apache Software Foundation. Apache edgent documentation. https://edgent.incubator.apache.org/docs/home, 2020. Accessed: 2020-05-13.

[Fou20b] Eclipse Foundation. Eclipse iofog documentation. https://iofog.org/docs/2/getting-started/whats-new.html, 2020. Accessed: 2020-09-02.

[Fou20c] Eclipse Foundation. Eclipse kura documentation. https://www.eclipse.org/kura/, 2020. Accessed: 2020-09-02.

[Fou20d] EdgeX Foundry. Edgex foundry documentation. https://fuji-docs.edgexfoundry.org/index.html, 2020. Accessed: 2020-05-14.

[Gmb20] Applied Informatics Software Engineering GmbH. macchina.io documentation. https://macchina.io/index.html, 2020. Accessed: 2020-09-02.

[HFRS17] Amin Hosseinian-Far, Muthu Ramachandran, and Charlotte Lilly Slack. Emerging trends in cloud computing, big data, fog computing, IoT and smart living. In *Technology for Smart Futures*, chapter 2, pages 29–40. Springer, 2017.

[HYA+15] Ibrahim Abaker Targio Hashem, Ibrar Yaqoob, Nor Badrul Anuar, Salimah Mokhtar, Abdullah Gani, and Samee Ullah Khan. The rise of 'big data' on cloud computing: Review and open research issues. *Information Systems*, 47:98–115, 2015.

[IEE18] IEEE. Ieee standard for adoption of openfog reference architecture for fog computing. https://standards.ieee.org/standard/1934-2018.html, 2018. Accessed: 2020-05-13.

[Kub20] KubeEdge. Kubeedge documentation. https://docs.kubeedge.io/en/latest/, 2020. Accessed: 2020-09-02.

[Lan01] Doug Laney. 3D Data Management: Controlling Data Volume, Velocity, and Variety. *META Group Research Note*, 2001.

[LS08] David Luckham and Roy Schulte. Event Processing Glossary. 1.1:1–19, 2008.

[McC76] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976.

[MG11] Peter M. Mell and Timothy Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 2011.

[MJMRP19] Farhad Mehdipour, Bahman Javadi, Aniket Mahanti, and Guillermo Ramirez-Prado. Fog Computing Realization for Big Data Analytics. In Srirama Buyya, editor, *Fog and Edge Computing: Principles and Paradigms*, chapter 11, pages 261–290. Wiley STM, 2019.

[MVM16] D. Mourtzis, E. Vlachou, and N. Milas. Industrial Big Data as a Result of IoT Adoption in Manufacturing. *Procedia CIRP*, 55:290–295, 2016.

[NS17] Muhammad Aamir Nadeem and Muhammad Anwaar Saeed. Fog computing: An emerging paradigm. *2016 6th International Conference on Innovative Computing Technology, INTECH 2016*, pages 83–86, 2017.

[Ope17] OpenFogConsortium. Openfog reference architecture for fog computing. https://www.iiconsortium.org/pdf/OpenFog_Reference_Architecture_2_09_17.pdf, 2017. Accessed: 2020-05-13.

[SE08] Guy Sharon and Opher Etzion. Event-processing network model and implementation. *IBM Systems Journal*, 47(2):321–334, 2008.

[vST17] Maarten van Steen and Andrew S. Tanenbaum. *Distributed Systems.* Pearson Education, 2017.

A Anhang

A.1 Bewertungsschemen

Metrik	Bewertungsschema	Punkte	Faktor
M1.1	$8 \leq x \leq 17$	0	0
	$18 \leq x \leq 18$	1	1/3
	$19 \leq x \leq 60$	2	2/3
	$61 \leq x$	3	1
M1.2	$40 \leq x \leq 187$	0	0
	$188 \leq x \leq 296$	1	1/3
	$297 \leq x \leq 344$	2	2/3
	$345 \leq x$	3	1
M1.3	$15{,}13 \leq x$	0	0
	$9{,}78 \leq x \leq 15{,}12$	1	1/3
	$5{,}73 \leq x \leq 9{,}77$	2	2/3
	$1{,}6 \leq x \leq 5{,}72$	3	1
M1.4	$18101{,}61 \leq x$	0	0
	$7911{,}67 \leq x \leq 18101{,}6$	1	1/3
	$2320{,}06 \leq x \leq 7911{,}66$	2	2/3
	$319{,}95 \leq x \leq 2320{,}05$	3	1

Tabelle A.1: Metrik-Bewertung - Nachhaltigkeit und Akzeptanz

Metrik	Bewertungsschema	Punkte	Faktor
M2.1	$0 \leq x \leq 24$	0	0
	$25 \leq x \leq 49$	1	1/3
	$50 \leq x \leq 74$	2	2/3
	$75 \leq x$	3	1
M2.2	Nein	0	0
	Ja	3	1
M2.3	Nein	0	0
	Ja	3	1
M2.4	$10 < x$	0	0
	$10 \geq x$	3	1

Tabelle A.2: Metrik-Bewertung - Softwarequalität

Metrik	Bewertungsschema	Punkte	Faktor
M3.1	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M3.2	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M3.3	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M3.4	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M3.5	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1

Tabelle A.3: Metrik-Bewertung - Umgang mit anderen Systemen

Metrik	Bewertungsschema	Punkte	Faktor
M4.1	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M4.2	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M4.3	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1
M4.4	Nicht möglich	0	0
	Mit viel Aufwand möglich	1	1/3
	Mit wenig Aufwand möglich	2	2/3
	Ohne Aufwand möglich	3	1

Tabelle A.4: Metrik-Bewertung - Verarbeitung von Daten

Metrik	Bewertungsschema	Punkte	Faktor
M5.1	Ausfall führt zu weiteren Ausfällen	0	0
	Ausfall wird nicht erkannt	1	1/3
	Ausfall wird erkannt	2	2/3
	Ausfall wird erkannt und behoben	3	1
M5.2	Ausfall führt zu weiteren Ausfällen	0	0
	Ausfall wird nicht erkannt	1	1/3
	Ausfall wird erkannt	2	2/3
	Ausfall wird erkannt und behoben	3	1
M5.3	Verschlüsselung ist nicht möglich	0	0
	Verschlüsselung ist mit viel Aufwand möglich	1	1/3
	Verschlüsselung ist mit wenig Aufwand möglich	2	2/3
	Verschlüsselung ist ohne Aufwand möglich	3	1
M5.4	Verschlüsselung ist nicht möglich	0	0
	Verschlüsselung ist mit viel Aufwand möglich	1	1/3
	Verschlüsselung ist mit wenig Aufwand möglich	2	2/3
	Verschlüsselung ist ohne Aufwand möglich	3	1
M5.5	Authentifikation ist nicht möglich	0	0
	Authentifikation ist mit viel Aufwand möglich	1	1/3
	Authentifikation ist mit wenig Aufwand möglich	2	2/3
	Authentifikation ist ohne Aufwand möglich	3	1

Tabelle A.5: Metrik-Bewertung - Sicherheit und Zuverlässigkeit

A.2 Inhalt des elektronischen Anhangs

Die beiliegende CD enthält folgende Dateien und Verzeichnisse:

- Bachelorthesis.pdf
- Erweiterung
  - edgex-service-adapter
  - event-processing-agent
  - in-adapter

  - management

- Evaluierung
  - bluetooth-device-service
  - cloud-mqtt-connector
  - cloud-service-mock
  - connector-application-service
  - hierarchical-device-service
  - modbus-device-mock
  - mqtt-device-mock
  - rest-device-mock

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Evaluierung und Erweiterung von EdgeX in einem Fog-Computing-Paradigma

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original