Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Tim Wittler

Eine smarte LED-Anzeige zur Visualisierung von IoT-Ereignissen mit Apache Flink

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Tim Wittler

Eine smarte LED-Anzeige zur Visualisierung von IoT-Ereignissen mit Apache Flink

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. Olaf Zukunft
Zweitgutachter: Prof. Dr. Stefan Sarstedt

Eingereicht am: 29. Dezember 2017

Tim Wittler

Thema der Arbeit
Eine smarte LED-Anzeige zur Visualisierung von IoT-Ereignissen mit Apache Flink

Stichworte
Apache Flink, Komplexe Ereignisverarbeitung, Internet der Dinge, Visualisierung, Smart Home, LED, Arduino

Kurzzusammenfassung
Diese Arbeit beschäftigt sich kritisch mit dem Framework Apache Flink. Dazu wird eine Beispielanwendung mit Flink geplant und realisiert, um einen Eindruck von der Arbeit mit Flink zu bekommen. Die Beispielanwendung empfängt Ereignisse aus dem Internet of Things und visualisiert diese über eine smarte LED-Anzeige. Es wird dabei auf alle notwendigen Entwicklungsschritte von Soft- und Hardware eingegangen. Außerdem werden Anforderungen an Flink und an die Beispielanwendung entwickelt und kritisch überprüft. Dabei soll ein Eindruck über die Arbeit mit Flink entstehen und eventuelle Möglichkeiten zur Verbesserungen aufgezeigt werden.
Tim Wittler

Title of the paper
A smart LED display to visualise IoT events with Apache Flink

Keywords
Apache Flink, Complex Event Processing, Internet of Things, Visualisation, Smart Home, LED, Arduino

Abstract
This thesis critically evaluates the framework Apache Flink. A Flink example application will be planned and realized for this purpose. The example application receives events out of the Internet of Things and visualises them on a smart LED display. All essential development steps of soft- and hardware will be explained. Furthermore, this thesis will set up and check requirements to Flink and the example application. This should give an impression of developing with Flink and shows possible opportunities to improve Flink.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die Branche der Informatik befindet sich im stetigen Wandel. In den letzten Jahren ist das Thema Big-Data immer präsenter geworden. Big-Data beschreibt dabei hauptsächlich drei Eigenschaften von Daten: Die Menge der Daten hat massiv zugenommen (Volume), die Daten werden fortlaufend produziert (Velocity) und die Daten sind zumeist unstrukturiert (Varierty).

Dafür ist unter anderem die zunehmende Vernetzung von diversen Geräten verantwortlich, welche unter dem Stichwort Internet of Things (kurz: IoT) zusammengefasst wird. Seit dem Boom der Smartphones haben Besitzer die Möglichkeit, diverse Daten fortlaufend zu produzieren, zum Beispiel in Form von Fotos, Textnachrichten oder Live-Videos.

Die Informatik beschäftigt sich damit, wie mit den Eigenschaften von Big-Data umgegangen werden kann. Die Analyse und Verarbeitung der massiven Datenaufkommen in Echtzeit ist unter anderem ein großes Problem. Ein recht junger Ansatz ist es, die Daten direkt nach ihrer Entstehung zu verarbeiten, anstatt sie zunächst zu speichern, um sie später zu verarbeiten. Dazu werden die Daten als eine Vielzahl von Ereignissen betrachtet. Dieser Ansatz wird unter den Begriffen Event Processing oder Complex Event Processing zusammengefasst. Es wurden bereits Systeme entwickelt, welche eine ereignisorientiert Echtzeitverarbeitung von Daten ermöglichen sollen.

Ein weiterer Wandlungsprozess in der Informatik ist die Tendenz, dass sich Computer immer mehr in unseren Alltag integrieren. Mittlerweile gibt es viele Haushaltsgeräte, die einen integrierten Computer haben, ohne dass dies offensichtlich ist. Die integrierten Computer sorgen neben der klassischen Steuerung auch für eine Vernetzung mit anderen Geräten im Haushalt. Die Digitalisierung des Haushalts wird unter dem Begriff Smart Home geführt. Das Ziel von Smart Homes ist es, einen vollständig vernetzten Haushalt zu schaffen, der den Bewohnern durch Automatisierung mehr Komfort bietet.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist eine kritische Auseinandersetzung mit dem Framework Apache Flink. Hierfür sollen unter anderem Anforderungen an das Framework entwickelt und anschließend

überprüft werden. Außerdem soll eine Beispielanwendung mit Apache Flink geplant und entwickelt werden, um einen Eindruck über die Entwicklung mit Apache Flink zu bekommen.

Die Anwendung hat das Ziel IoT-Ereignisse aus verschiedenen Quellen über eine smarte LED-Anzeige zu visualisieren. Dafür sollen mehrere Szenarien einer solchen Anwendung aufgezeigt und realisiert werden.

Zur Entwicklung der Beispielanwendung soll diese Arbeit auf alle soft- und hardwaretechnischen Entwicklungsschritte (Analyse, Entwurf und Realisierung) eingehen und deren Besonderheiten aufzeigen. Der Fokus soll dabei in der softwaretechnischen Entwicklung mit Flink liegen.

Wie oben erwähnt findet die Entwicklung mithilfe des Frameworks Apache Flink statt. Zur kritischen Auseinandersetzung sollen, neben der Überprüfung der entwickelten Anforderungen, die eigenen Erfahrungen mit Apache Flink dokumentiert und eventuelle Verbesserungsvorschläge aufgezeigt werden.

1.2 Aufbau / Gliederung der Arbeit

Die Arbeit hat neben diesem noch sechs weitere Kapitel. Jedes Kapitel baut dabei auf dem erlangten Wissen aus den vorherigen Kapiteln auf.

Dieses Kapitel (*Einleitung*) hat sich mit der Relevanz und Zielsetzung der Arbeit selbst beschäftigt und gibt einen Ausblick auf die weitere Arbeit.

Im zweiten Kapitel (*Grundlagen*) werden umgebende Themen und Begriffe erläutert, die für die nachfolgenden Kapitel wichtig sind.

Kapitel 3 (*Analyse der Anforderungen*) analysiert anschließend alle Anforderungen an die zu entwickelnde Soft- und Hardware. Dazu wird zunächst das Ziel der Software anhand verschiedener Beispielszenarien erklärt. Außerdem werden dort – neben den Anforderungen an die Soft- und Hardware – die Anforderungen an das Framework Apache Flink aufgestellt.

Auf Basis der Anforderungen aus Kapitel 3 wird in Kapitel 4 (*Entwurf*) die Software entworfen. Dafür wird eine Architektur ausgewählt und die Software in Komponenten aufgeteilt. Außerdem wird dort auf die Kommunikation zwischen den Komponenten eingegangen.

Der Entwurf aus Kapitel 4 gibt den Rahmen vor, welche Komponenten entwickelt werden müssen. In Kapitel 5 (*Realisierung und Test*) werden die Komponenten realisiert beziehungsweise entwickelt. Die Realisierung orientiert sich zudem an den Anforderungen aus Kapitel 3. Kapitel 5 erläutert die Funktionsweise jedes Komponenten und geht auf dessen Besonderheiten

ein, außerdem wird erläutert, wie die Software getestet wird. Neben der softwaretechnischen Realisierung beinhaltet Kapitel 5 auch die Auswahl und Programmierung geeigneter Hardware.

Das sechste Kapitel (*Erfahrung und Kritik zu Apache Flink*) widmet sich der Auseinandersetzung mit dem Framework Apache Flink. Dort werden eigene Erfahrungen geschildert und die Anforderungen an das Framework überprüft.

Kapitel 7 (*Zusammenfassung und Ausblick*) ist der letzte Abschnitt dieser Arbeit. Es enthält neben einer kurze Zusammenfassung der Ergebnisse auch einen Ausblick auf eine mögliche Weiterentwicklung.

2 Grundlagen

2.1 Complex Event Processing

Das Thema Complex Event Processing (kurz: CEP; zu deutsch: komplexe Ereignisverarbeitung) erhält zurzeit eine große Aufmerksamkeit in der Informatik. Die Aufmerksamkeit ist darin begründet, dass die zu verarbeitenden Datenaufkommen immer größer werden, fortlaufend eintreffen und nicht strukturiert sind (Big-Data), es aber nur wenige Verfahren zur Verarbeitung und Analyse solcher Datenaufkommen gibt. CEP ist solch eine Technologie: Sie ermöglicht eine Echtzeit-Verarbeitung und -Analyse von heterogenen Datensätzen, die in einem massiven und hochfrequenten Datenstrom auftreten (Bruns und Dunkel, 2015; Hedtstück, 2017).

Ein Datenstrom besteht aus einer unbestimmten Anzahl von Datensätzen. Im Gegensatz zu Datenbeständen gibt es also weder Ende noch Anfang. Dies ist darin begründet, dass kontinuierlich neue Datensätze (Ereignisse) eintreffen können. Datenströme bringen deshalb einige Besonderheiten mit sich, die nur schwer mit der klassischen Datenverarbeitung gelöst werden können. Die nachfolgende Liste zeigt die zentralen Besonderheiten von Datenströmen (auf Basis von Bruns und Dunkel, 2015).

- Datenströme enthalten meist aktuelle Live-Daten, dabei repräsentiert ein Datensatz ein reales oder abstraktes Vorkommnis (Ereignis)
- Datenströme sind unbegrenzt, da fortlaufend neue Ereignisse eintreten können
- Die Ereignisse sind meist feingranular, sie lassen sich mit wenig Daten beschreiben
- Es ist mit einer hohen Eingangsrate zu rechnen, da in der Realität viele (einfache) Ereignisse auftreten können
- Die Reihenfolge und Auftrittszeit der Ereignisse spielen eine entscheidende Rolle für die Verarbeitung und Analyse
- Die Ereignisse können in impliziten Beziehungen zueinander stehen, welche sich in den Ereignisdetails verbergen und nicht direkt erkennbar sind (z.B. alle Ereignisse zweier benachbarter Sensoren)

Mit all diesen Besonderheiten versuchen CEP-Systeme umzugehen. Hierfür arbeiten CEP-Systeme nicht wie klassische Programme, die Schritt für Schritt eine vorgegebene Menge von

Anweisungen ausführen, sondern beurteilen anhand der Ereignisse laufend die Situation und reagieren entsprechend darauf (Hedtstück, 2017).

Daraus ergeben sich verschiedene Eigenschaften, die bei einem CEP-System wünschenswert wären. Diese werden ab Seite 8 erläutert.

Zunächst wird jedoch auf die wichtigsten Grundbegriffe des Themas CEP eingegangen, um ein besseres Verständnis zu bekommen. Dazu werden nachfolgend die Grundbegriffe *Ereignis*, *Mustererkennung* und *Sliding Windows* anhand von Abbildungen und Beispielen detaillierter erläutert.

Ereignis Ein Ereignis (event) kann verschiedenster Natur sein, z.B. die Änderung eines Messwertes bei einem Sensor (technisches Ereignis), das Bestellen eines Produktes im Online-Shop (Geschäftsereignis) oder der Eingang einer neuen E-Mail. Sie alle haben die Gemeinsamkeit, dass sie sich auf die Veränderung eines Zustandes beziehen, in der Regel ist das die Änderung einer Eigenschaft oder eines Wertes von realen oder virtuellen Objekten (nach Bruns und Dunkel, 2015).

Bei den Ereignissen unterscheidet man zwischen elementaren (auch „atomaren“) und komplexen Ereignissen: Die elementaren Ereignisse stellen eine Veränderung in der realen Welt dar. Sie befinden sich auf niedriger Abstraktionsebene, da diese einen direkter Bezug zur Realität haben (Bruns und Dunkel, 2015).

Die komplexen Ereignisse entstehen, wenn aus einer endlichen Menge von elementaren Ereignissen ein Muster identifiziert wird (Hedtstück, 2017). Sie befinden sich auf einer höheren Abstraktionsebene, als die elementaren Ereignisse, da sie ein Muster beziehungsweise eine Beziehung von elementaren Ereignissen zueinander repräsentieren (Bruns und Dunkel, 2015). Eine Veranschaulichung der Mustererkennung und Abstraktionsebenen findet sich in Abbildung 2.1 auf der nächsten Seite.

Diese Muster werden von dem eingesetzten CEP-System erkannt und lösen gegebenenfalls ein neues – komplexes – Ereignis aus.

Mustererkennung Das Erkennen von Mustern (pattern matching) im Ereignisstrom ist die zentrale Aufgabe eines CEP-Systems (Bruns und Dunkel, 2015). Es soll Muster im Strom von unterschiedlichen Ereignistypen erkennen und daraus neue Ereignisse generieren. Zum näheren Verständnis von Mustern und deren Erkennung folgt ein Beispiel.

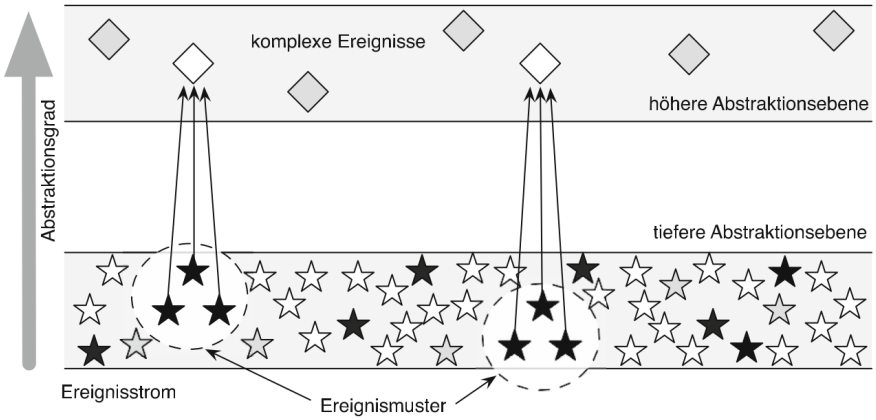


Abbildung 2.1: Abstraktionsebenen der Ereignisse und Mustererkennung (Bruns und Dunkel, 2015, S. 5)

In einem Smart Home gibt es zwei Sensoren, ein Sensor misst die Raumtemperatur (`Tp`) und der Andere erkennt den Status des Fensters (`Fe=z` für „Fenster ist zu“ beziehungsweise `Fe=a` für „Fenster ist auf“). Beide Sensoren leiten eine Änderung ihres Status direkt an das CEP-System weiter. Am Muster der Ereignisse soll erkannt werden, wenn die Raumtemperatur aufgrund eines offenen Fensters einen Schwellwert (threshold) – in diesem Beispiel 10 Grad – unterschreitet.

Das Muster besteht also aus den Ereignissen `Fe=a` („Fenster ist auf “) und `Tp<10` („Raumtemperatur ist unter 10 Grad“). Da das Fenster zwischenzeitlich schon wieder geschlossen sein könnte, muss dem Muster noch die Bedingung hinzugefügt werden, dass das Ereignis `Fe=z` („Fenster ist zu“) nicht zwischen den zwei Ereignissen eintreten darf.

Dieses Muster lässt sich vereinfacht mit folgendem Ausdruck darstellen `Fe=a; not Fe=z; Tp<10`. Hierbei stellt das Semikolon ; die zeitliche Unterteilung der Ereignisse dar. Folgt also auf das Ereignis `Fe=a` das Ereignis `Tp<10` und ist zwischenzeitlich das Ereignis `Fe=z` nicht eingetreten trifft das Muster zu.

Das CEP-System kann nun ein komplexes Ereignis produzieren, welches „Niedrige Raumtemperatur aufgrund offenen Fensters“ repräsentiert, auf dieses Ereignis könnte dann das Fenster reagieren und sich selbst schließen. Dieses Beispiel wird in der Abbildung 2.2 auf der nächsten Seite verdeutlicht, dabei treffen die Ereignisse `Tp=7` und `Fe=a` (blau umkreist) auf das Muster zu.

Ein weiteres Muster bei diesem Beispiel wäre `Fe=z; not Fe=a; Tp<10`, dieses erkennt, wenn die Temperatur trotz eines geschlossenen Fensters auf unter 10 Grad fällt. Dies

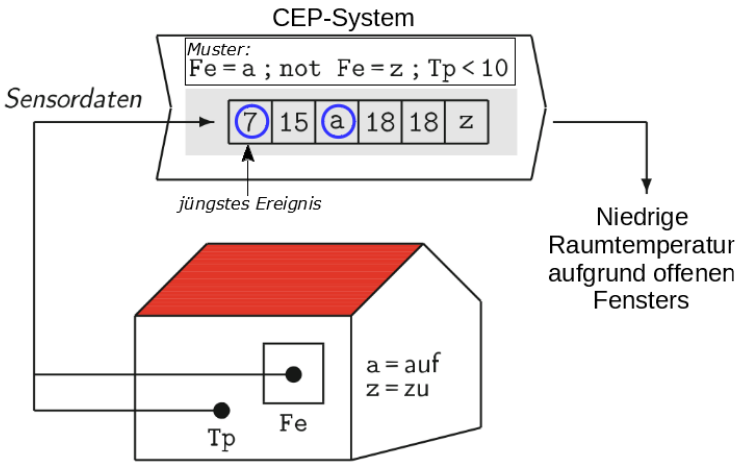


Abbildung 2.2: Mustererkennung: Niedrige Raumtemperatur aufgrund offenen Fensters (auf Basis von Hedtstück, 2017, S. 17)

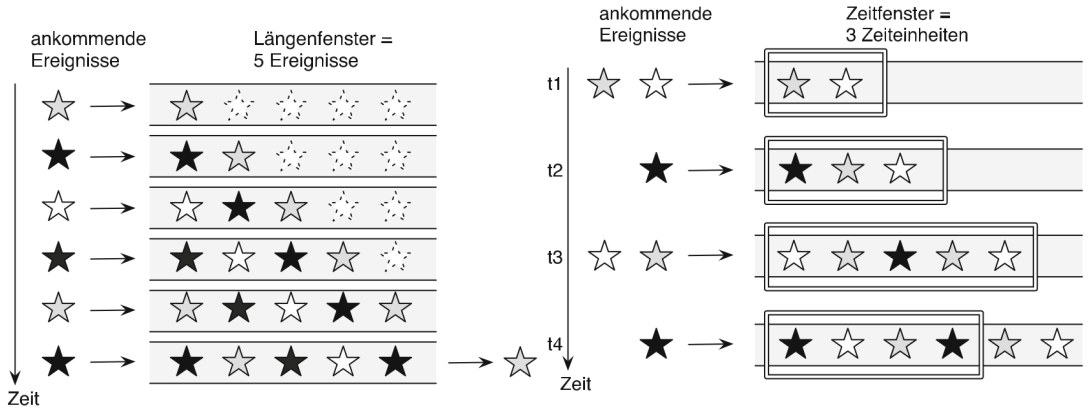
wäre dann ein Zeichen für eine defekte Heizung oder einen defekten Sensor (Beispiel nach Hedtstück, 2017).

Sliding Windows Da der Ereignisstrom in der Regel kein Ende hat, muss der Suchbereich für die Mustererkennung eingeschränkt werden, sodass nur ein Ausschnitt des Stroms betrachtet wird. Der Suchraum für die Mustererkennung wird also auf eine Teilmenge von Ereignissen beschränkt. Die Beschränkung findet durch sogenannte *Sliding Windows* (zu deutsch „gleitendes Fenster“) statt, dabei lässt sich zwischen gleitenden Längenfenstern und gleitenden Zeitfenstern unterscheiden (Bruns und Dunkel, 2015; Hedtstück, 2017). In Abbildung 2.3 auf der nächsten Seite werden beide Fenster grafisch dargestellt.

Das gleitende Längenfenster ist in der Abbildung durch eine feste Anzahl von Ereignissen definiert, sodass nur die letzten fünf Ereignisse betrachtet werden. Übersteigt die Anzahl der Ereignisse die Fensterlänge, wird das älteste Ereignis aus dem Fenster entfernt, um Platz für das neue Ereignis zu schaffen (First-In-First-Out-Prinzip) (Bruns und Dunkel, 2015; Hedtstück, 2017).

Gleitende Zeitfenster hingegen sind durch eine feste Zeitdauer definiert. Das Zeitfenster wird auf der Zeitachse entlang geschoben und betrachtet so eine variable Menge von Ereignissen, die in dem Zeitdauer des Fensters sichtbar waren (Bruns und Dunkel, 2015; Hedtstück, 2017).

In Abbildung 2.3 auf der nächsten Seite hat das Zeitfenster eine Länge von drei Zeiteinheiten. Alle Ereignisse, die innerhalb der drei Zeiteinheiten stattgefunden haben, werden betrachtet. Die Anzahl der Ereignisse spielt dabei keine Rolle. Beim Zeitpunkt $t3$ werden beispielsweise



(a) Gleitendes Längenfenster (b) Gleitendes Zeitfenster

Abbildung 2.3: Gleitendes Längen- und Zeitfenster (Bruns und Dunkel, 2015, S.24+25)

fünf Ereignisse betrachtet, also alle Ereignisse, die in den letzten drei Zeiteinheiten stattgefunden haben (`t1=2`, `t2=1` und `t3=2`). Während beim Zeitpunkt `t4` nur vier Ereignisse betrachtet werden (`t2=1`, `t3=2` und `t4=1`), da Ereignisse aus Zeitpunkt `t1` zu Zeitpunkt `t4` nicht mehr im Zeitfenster sichtbar sind.

Für das vorherige Beispiel zur Mustererkennung (Fenster und Temperatur) wäre ein gleitendes Zeitfenster sinnvoll. So ließe sich definieren, dass die drei Bedingungen innerhalb eines Zeitfensters (z.B. von 30 Minuten) zutreffen sollen.

CEP-System Unter dem Begriff CEP-System (auch CEP-Software oder CEP-Framework) wird in dieser Arbeit eine Software oder Bibliothek verstanden, welche die Verarbeitung von Datenströmen ermöglicht und Werkzeuge zur komplexen Verarbeitung von Ereignissen bereitstellt.

Wie schon zu Beginn erwähnt, haben Datenströme einige Besonderheiten im Vergleich zu klassischen Datenbeständen. CEP-Systeme versuchen mit diesen Besonderheiten umzugehen. Dazu gibt es verschiedene Eigenschaften, die bei einem CEP-System wünschenswert wären. Einige dieser „Wünschenswerten Eigenschaften“ werden nachfolgend aufgelistet (auf Basis von Friedman und Tzoumas, 2016).

- Hoher Datendurchsatz (high throughput): Das System kann eine große Menge von Datensätzen (Ereignissen) verarbeiten.

- Niedrige Latenz / Verzögerungszeit (low latency): Ein Datensatz bleibt nicht lange im System, da er schnell verarbeitet wird. Wünschenswert ist die Echtzeitverarbeitung eines Datensatzes.
- Korrekte Zeit- und Zeitfenster-Semantik: Die Datensätze werden in der richtigen (zeitlichen) Reihenfolge verarbeitet. Das System bietet die Möglichkeit akkurate Zeitfenster zu definieren.
- Fehlertoleranz: Das System ist robust gegen Fehler (z.B. Ausfall eines Fremdsystems, fehlerhafte Daten) und arbeitet weiterhin korrekt. Andere Eigenschaften werden durch Fehler nicht beeinträchtigt, insbesondere die Reihenfolge bleibt erhalten.
- Benutzerfreundlich: Das System ist leicht zu bedienen und erfordert wenig Einarbeitung.
- Ausdrucksstarke Sprache (High-Level API): Die Programmierung des Systems ist mit wenig und übersichtlichem Quellcode möglich. Technische Details des Systems sind hinter der ausdrucksstarken Sprache versteckt.
- Verbreitung: Das System ist sehr verbreitet und wird produktiv eingesetzt. Es gibt viele Beispielprojekte und Erweiterungen.

Jede Eigenschaft für sich genommen, kann leicht von einem CEP-System erfüllt werden. Die Erfüllung aller Eigenschaften bringt jedoch Schwierigkeiten mit sich, denn einige Eigenschaften beschränken sich gegenseitig. So steht zum Beispiel das Einhalten der Datensatz-Reihenfolge (Korrekte Zeit-Semantik) der Fehlertoleranz im Weg.

Für die meisten CEP-Systeme bedeutet dies, dass sie einen Kompromiss zwischen den Eigenschaften eingehen müssen oder auf Eigenschaften verzichten müssen (Friedman und Tzoumas, 2016; Wingerath u. a., 2016a). Daraus resultiert meist ein Mehraufwand für den Entwickler, da dieser mit den Einschränkungen umgehen und gegebenenfalls selbst für die Einhaltung von bestimmten Eigenschaften sorgen muss. Das optimale CEP-System ist also schwierig zu realisieren. Nachfolgend wird unter Punkt 2.2 auf das CEP-System Apache Flink genauer eingegangen. Das CEP-System Flink kommt in dieser Arbeit zur Anwendung.

2.2 Apache Flink

Apache Flink (kurz: Flink) ist ein Open-Source Framework (Friedman und Tzoumas, 2016), welches als Complex Event Processing System genutzt werden kann. Im Kontext dieser Arbeit ist Apache Flink das zentrale System für die Verarbeitung und Analyse von IoT-Ereignissen (IoT steht für „Internet of things“, siehe 2.4). Unter Punkt 3.6 (ab Seite 35) werden die eigenen Erwartungen an Flink in Form von Anforderungen formuliert. In diesem Kapitel sollen die Grundlagen zu Apache Flink vermittelt werden.

Über Flink Flink war ursprünglich ein Teil des *Stratosphere*-Projekts, ein gemeinsames Forschungsprojekt von drei Berliner und weiteren europäischen Universitäten in den Jahren 2010 bis 2014 (Friedman und Tzoumas, 2016). Im April 2014 wurde das Projekt unter dem Namen Flink von der *Apache Software Foundation* übernommen (Friedman und Tzoumas, 2016; collaborative research project).

Der Name Flink entspricht nicht zufällig dem deutschen Adjektiv *flink*, das Framework wurde ganz bewusst nach diesem Adjektiv benannt, da dadurch seine Eigenschaften wie Schnelligkeit und Agilität gut beschrieben werden (Friedman und Tzoumas, 2016). Des Weiteren wurde auch ein Eichhörnchen als Logo passend zu diesen Eigenschaften gewählt.

Flink als CEP-System Wie schon in Punkt 2.1 ab Seite 8 erwähnt, gibt es verschiedene Eigenschaften, die bei einem CEP-System wünschenswert sind. Dabei müssen die meisten CEP-Systeme einen Kompromiss zwischen den Eigenschaften eingehen, da es schwierig ist, alle wünschenswerten Eigenschaften zu erfüllen.

Ein Beispiel für so einen Kompromiss zeigt das Framework *Apache Storm*: Storm weist eine sehr niedrige Latenz auf, arbeitet dafür im Fehlerfall nicht immer korrekt (Friedman und Tzoumas, 2016; Wingerath u. a., 2016a). Auch das Framework *Spark Streaming* geht einen Kompromiss ein: Die Stärke von Spark Streaming liegt bei der Fehlertoleranz und einem hohem Datendurchsatz, dafür ist die Latenz jedoch sehr hoch (Friedman und Tzoumas, 2016; Wingerath u. a., 2016a).

Flink hingegen hat den Anspruch, keinen Kompromiss zwischen den wünschenswerten Eigenschaften einzugehen und versucht alle Eigenschaften zu erfüllen (Friedman und Tzoumas, 2016). Lediglich die Eigenschaft einer hohen Verbreitung kann Flink im Vergleich zu anderen CEP-Systemen nicht vollständig erfüllen (siehe unten unter *Verbreitung*). Nachfolgend werden die Besonderheiten von Flink beschrieben.

Besonderheiten von Flink Eine große Besonderheit von Flink ist die Möglichkeit, neben Echtzeit-Datenströmen (nicht endlicher Datenstrom) auch statische, endliche Datenbestände zu verarbeiten. Dabei wird dieselbe Technologie verwendet (Friedman und Tzoumas, 2016; Wingerath u. a., 2016b), sodass es aus Sicht des Entwicklers kaum einen Unterschied macht, ob er mit statischen Daten oder Echtzeit-Daten arbeitet. Um dies zu erreichen, behandelt Flink statische Datenbestände als Sonderfall der Datenstrom-Verarbeitung. Dies ist weitgehend einzigartig unter den CEP-Systemen.

Weiter zeichnet sich Flink durch eine ausdrucksstarke Sprache (High-Level API) aus, welche den Prozess für die Verarbeitung und Analyse des Datenstroms generiert und in der

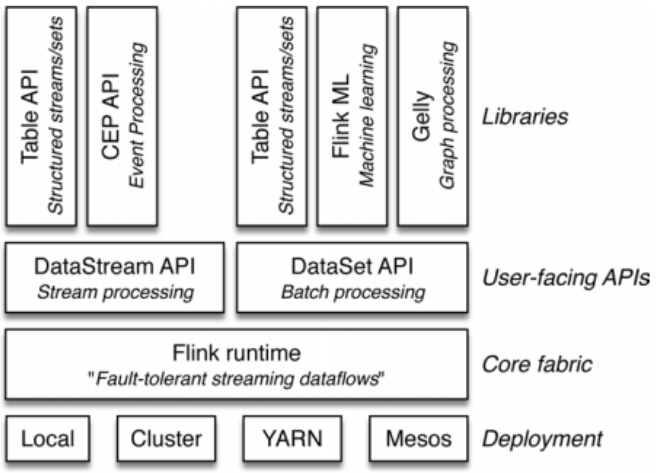


Abbildung 2.4: Komponenten von Flink (Friedman und Tzoumas, 2016)

Flink-Laufzeitumgebung ausführt (Friedman und Tzoumas, 2016). Abbildung 2.4 gibt eine detailliertere Übersicht über die Hauptkomponenten von Flink, die *DataStream API* und *DataSet API* sind die Schnittstellen für den Entwickler. *Flink runtime* ist die eigentliche Umgebung, in der die Prozesse ausgeführt werden.

Connectors Flink bietet eingebaute Verbindungen zu verschiedenen Fremdsystemen an. Das bedeutet, dass sich ohne eigenen Entwicklungsaufwand Ereignisse aus Fremdsystemen empfangen oder Ereignisse an diese senden lassen. Diese Verbindungen zu Fremdsystemen werden in Flink *Connector* genannt (Deshpande, 2017).

Zum Beispiel gibt es einen Connector, der Ereignisse aus Flink an das Messaging-System *Apache Kafka* (Apache Kafka wird unter 2.3 auf der nächsten Seite separat behandelt) sendet, sodass diese von anderen Systemen weiter verarbeitet werden können (Deshpande, 2017). Dies funktioniert auch andersherum: Über den Kafka-Connector können auch Ereignisse aus Apache Kafka empfangen und in Flink verarbeitet werden (Deshpande, 2017).

Der Kafka-Connector ist dabei nur ein Beispiel unter vielen weiteren. Flink bietet Connectoren zu weiteren Messaging-Systemen (zum Beispiel zu *RabbitMQ*) und auch zu Systemen an, die kein Messaging-System sind (zum Beispiel zu *Twitter*) (Deshpande, 2017).

Ausführung Flink kann sowohl lokal (auf einem einzelnen Computer) als auch in einem verteilten System (auf mehreren, verteilten Computern) ausgeführt werden (Friedman und Tzoumas, 2016; Wingerath u. a., 2016b). Die lokale Ausführung von Flink eignet sich besonders zum Testen und Debuggen. Bei der Ausführung in einem verteilten System übernimmt

Flink die korrekte Wiederherstellung der Ausführung nach einem Fehlerfall (Friedman und Tzoumas, 2016), der Entwickler muss sich also wenig Gedanken über die Verteilung seiner Flink-Anwendung machen.

Verbreitung In der Industrie ist Flink momentan vergleichsweise wenig verbreitet (Wingerath u. a., 2016b). In dem Wiki der Apache Software Foundation sind zurzeit 32 Unternehmen angegeben, die Flink offiziell verwenden (Stand: 20. Januar 2017, siehe Tzoumas u. a., 2017). Darunter befinden sich einige große Unternehmen wie zum Beispiel die *Alibaba Group* oder *Zalando* (Tzoumas u. a., 2017; Friedman und Tzoumas, 2016).

2.3 Apache Kafka

Apache Kafka (kurz: Kafka) ist ein Open-Source Software-Projekt der Apache Software Foundation, welches auf die Verteilung und Verarbeitung von Nachrichten (messages) und Datenströmen spezialisiert ist.

Dabei ist zu beachten, dass Nachrichten eine nicht näher spezifizierte Form von Daten sind. Bei einer Nachricht kann es sich zum Beispiel auch um ein Ereignis handeln.

Im Kontext dieser Arbeit soll Apache Kafka zur Kommunikation (Austausch von Nachrichten) zwischen unabhängigen Programmteilen der zu entwickelnden Anwendung verwendet werden. Apache Kafka eignet sich besonders gut für diese Arbeit, da Flink einen eingebauten Connector für Kafka anbietet. Über den Kafka-Connector lässt sich Flink ohne großen Entwicklungsaufwand mit Kafka verbinden (siehe 2.2 auf der vorherigen Seite).

Über Kafka Kafka wurde ursprünglich von *LinkedIn* entwickelt, um die Kommunikation zwischen einzelnen LinkedIn-Services (unabhängige Programmteile; auch *Komponenten*) zu optimieren (Dunning und Friedman, 2016; Garg, 2015). Seit 2012 ist es ein Projekt der Apache Software Foundation.

LinkedIns damalige Motivation zur Entwicklung eines solchen Messaging-Systems war die schnell anwachsende Menge von Daten und Services. Vor der Entwicklung von Kafka musste bei Änderungen an einem Service jede Kommunikationsschnittstelle aller abhängigen Services angepasst werden (Dunning und Friedman, 2016; Garg, 2015). Mit einem Messaging-System wie Kafka lassen sich die einzelnen Services besser voneinander entkoppeln, sodass es weniger Abhängigkeiten untereinander gibt und jeder Service für sich alleine stehen kann. Die Verbindung der Services untereinander findet nun über das Messaging-System statt:

Services können untereinander asynchrone Nachrichten austauschen und so miteinander kommunizieren.

Besonderheiten von Kafka Eine besondere Anforderung von LinkedIn war, dass die Nachrichten persistent (dauerhaft) gespeichert werden, sodass sie auch nach einem längeren Zeitraum verarbeitet werden können. Die konventionellen Messaging-Systeme waren aber nur auf eine kurze Speicherung der Nachrichten ausgelegt (Dunning und Friedman, 2016). Der konventionelle Ansatz war nämlich, dass die Nachrichten nur solange wie nötig gespeichert werden – bis alle Services die Nachricht empfangen haben.

Dies ist zugleich die große Besonderheit von Kafka: Alle Nachrichten können persistent gespeichert werden und dort – je nach Einstellung – mehrere Wochen verweilen (Dunning und Friedman, 2016; Garg, 2015; Estrada und Ruiz, 2016). Dies hat zum Beispiel den Vorteil, dass man nach einer Programmänderung einfach alle Nachrichten der letzten Tage erneut verarbeiten kann. Trotz der persistenten Speicherung garantiert Kafka eine konstante Lese-/Schreibzeit ($O(1)$), unabhängig davon, wie viele Nachrichten bereits gespeichert wurden (Garg, 2015; Estrada und Ruiz, 2016).

Publish-subscribe pattern Neben der persistenten Speicherung orientiert sich das grundlegende Konzept von Kafka an den konventionellen Messaging-Systemen. Kafka folgt – wie die meisten Messaging-Systeme – dem sogenannten *Publish-subscribe pattern* (Dunning und Friedman, 2016; Garg, 2015; Estrada und Ruiz, 2016). Dies ist ein Programmiermodell, welches beschreibt, wie Programme über Nachrichten asynchron kommunizieren können.

Das Programmiermodell funktioniert wie folgt: Ein oder mehrere Produzenten (englisch: publisher oder producer) senden neue Nachrichten zu einem bestimmten Thema (topic) an ein Messaging-System. Anhand des Themas werden die Nachrichten in Gruppen kategorisiert und unterteilt. Nachrichten einer Gruppe können von beliebig vielen Konsumenten (englisch: subscriber oder consumer) gelesen werden, dazu werden die Konsumenten über neue Nachrichten informiert. Das Modell wird in der Abbildung 2.5 auf der nächsten Seite grafisch veranschaulicht.

Der Vorteil dieses Modells ist, dass sich Produzent und Konsument nicht kennen müssen. So können beide unabhängig voneinander arbeiten, und es können leicht neue Konsumenten hinzugefügt werden. Wichtig ist nur, dass der Konsument die Nachricht des Produzenten lesen und verstehen kann.

Kafka Cluster und Broker Das eigentliche Messaging-System bei Kafka ist das *Kafka cluster*. Das Cluster besteht aus einem oder mehreren *Kafka brokern*. In einem Cluster können

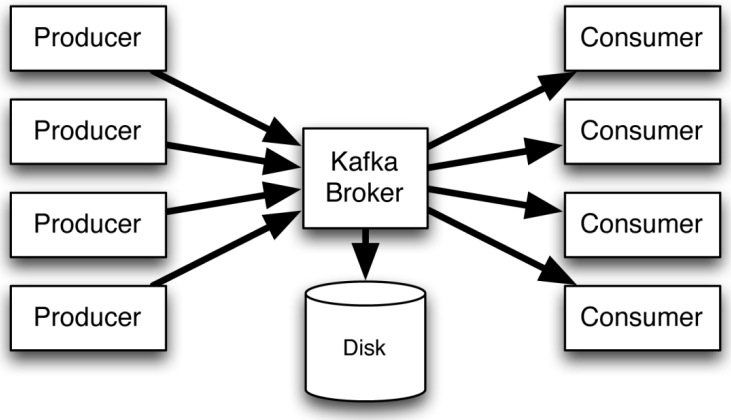


Abbildung 2.5: Produzenten und Konsumenten bei Kafka (Dunning und Friedman, 2016)

also mehrere Broker im Verbund arbeiten. Ein Broker ist das eigentliche Verbindungsglied zwischen Produzenten und Konsumenten (siehe Abbildung 2.5).

Das Cluster kann auf mehreren Servern verteilt werden. Ein Server des Clusters beinhaltet dabei einen oder mehrere Broker (Dunning und Friedman, 2016; Estrada und Ruiz, 2016).

Die Verwendung von einem Kafka cluster mit mehreren Kafka brokern eignet sich besonders für große verteilte Anwendungen. Da die – im Kontext dieser Arbeit zu entwickelnde Anwendung verhältnismäßig klein ist – wird die Verwendung von einem einzelnen Broker vermutlich ausreichen.

Produzent und Partitionierung Der Produzent bestimmt das Thema, zu welchem er Nachrichten an den Broker sendet. Dabei besteht die Möglichkeit ein Thema in verschiedene Partitionen aufzuteilen. Die Partitionierung sorgt dafür, dass ein Thema auf mehrere Broker verteilt wird. Eine Partition des Themas wird dabei von einem Broker verwaltet und enthält nur eine Teilmenge von Nachrichten zu dem Thema.

Der Produzent kann – bei Partitionierung eines Themas – entscheiden, ob er seine Nachrichten an eine bestimmte Partition sendet oder ob seine Nachrichten automatisch (im round-robin Verfahren) an wechselnde Partitionen gesendet werden (Dunning und Friedman, 2016).

Des Weiteren ist es auch möglich, Partitionen zu replizieren, sodass es mehrere Instanzen einer Partition gibt. Diese Redundanz sorgt bei Ausfall eines Brokers für eine beständige Verfügbarkeit der Nachrichten einer Partition.

Konsument Der Konsument kann ein Thema abonnieren und wird dadurch informiert, sobald es eine neue Nachricht gibt. Außerdem kann der Konsument auch nachträglich veraltete

Nachrichten lesen, dazu gibt er einen Startpunkt an, ab welchem er die Nachrichten lesen möchte. Dies ist möglich, da alle Nachrichten eine eindeutige Reihenfolge haben und in dieser persistiert wurden.

Mehrere Konsumenten können in Konsumenten-Gruppen (consumer groups) zusammengefasst werden. Die Nachrichten zu einem Thema werden anhand der Partitionierung innerhalb einer Konsumenten-Gruppe aufgeteilt (Dunning und Friedman, 2016). Dies hat den Vorteil, dass sich die Last bei hohem Nachrichtenaufkommen auf mehrere Konsumenten verteilt, sodass die Nachrichten schneller abgearbeitet werden können.

2.4 Internet of Things

Die genaue Definition des Begriffs *Internet of Things* („Internet der Dinge“) ist nicht möglich, da dieser Begriff je nach Kontext verschiedene Thematiken beschreibt und zunehmend in verschiedenen Sachverhalten verwendet wird. Daher kann dieser umfangreiche Begriff nachfolgend nur für den Kontext dieser Arbeit interpretiert werden.

Das Internet of Things beschreibt vorwiegend die zunehmende Vernetzung von Dingen (Things). Für diese Zunahme ist unter anderem die Miniaturisierung von elektronischen Bauteilen verantwortlich, welche auf die schnell fortschreitende Entwicklung der Smartphones zurückgeführt werden kann (Andelfinger und Hänisch, 2015; Braun, 2010; Mattern und Flörkemeier, 2010).

Nach dem Boom der Smartphones wird IoT als die nächste große Revolution in Wirtschaft und Industrie gehandelt. Schätzungen zufolge sind bis 2020 zwischen 200 und 250 Milliarden Dinge miteinander vernetzt (Andelfinger und Hänisch, 2015; Breur, 2015). Dazu zählen allerdings auch Dinge die mit einem RFID-Tag ausgestattet sind, obwohl RFID-Tags über keinen eigenen Internetanschluss verfügen.

RFID-Tags sind kleine Speichermedien, welche kontaktlos ausgelesen werden können. Sie können mit einem Barcode verglichen werden, mit dem Unterschied, dass sie Daten digital speichern und somit kein Sichtkontakt zum Auslesen notwendig ist. Außerdem kann ein Lesegerät mehrere Tags gleichzeitig auslesen. Sie werden im IoT unter anderem für Tracking-Zwecke verwendet (von Gagern, 2017). Zum Beispiel kann mit RFID-Tags ausgestattete Ware beim Eingang in ein Logistikzentrum getrackt werden und ein Ereignis in einem Warenwirtschaftssystem auslösen. Daher können die RFID-Tags trotz fehlendem Internetanschluss zum IoT gezählt werden.

Durch die zunehmende Vernetzung lässt sich auch eine deutliche Zunahme des Internet-Datenverkehrs erwarten. Dabei geht es vorwiegend um die Kommunikation zwischen den

Dingen untereinander, welche unter dem Begriff „Machine-to-Machine Communication“ (kurz: M2M) geführt wird (Breur, 2015; von Gagern, 2017).

M2M beschreibt die automatisierte Kommunikation zwischen Maschinen, ohne das Zutun von Menschen (von Gagern, 2017). Zum Beispiel das automatische nachbestellen oder einlagern Ware in einem automatisiertem Logistikzentrum.

Die Datenmengen die eine „Maschine“ produzieren variiert dabei stark. Der Messwert eines Temperatur-Sensors produziert zum Beispiel nur sehr kleine Datenmengen, je nachdem wie oft die Temperatur abgefragt wird. Hingegen eine Maschine, welche mehrere Sensoren vereint, kann gigantische Datenmengen produzieren, hierzu nachfolgend zwei Beispiele (nach Breur, 2015):

- Das autonom fahrende Auto von Google produziert circa 750 Megabit pro Sekunde an Daten.
- Die Sensoren eines Flugzeug produzieren bei der Überquerung des Atlantiks ungefähr 3 Terabyte an Daten.

Dabei ist zu beachten, dass hierbei nur die Rohdaten (konkrete, feingranulare Daten), wie z.B. die Messerwerte von Sensoren, berechnet wurden. Nach der Analyse der Rohdaten, in welcher weitere abstraktere Daten entstehen, ist die insgesamt produzierte Datenmenge deutlich höher (Breur, 2015). An dieser Stelle kann ein Bezug zum Thema Complex Event Processing (siehe Punkt 2.1) hergestellt werden, da dieses sich unter anderem mit der Analyse solcher Rohdaten beschäftigt.

Ein anderer Bezug kann zum Thema Smart Home hergestellt werden. Das Smart Home stellt ein prominentes Beispiel in der IoT-Welt dar (Andelfinger und Hänisch, 2015). In Punkt 2.5 wird das Thema Smart Home separat behandelt.

Außerdem ist das Thema IoT für diese Arbeit von Relevanz, da die smarte LED-Anzeige die Möglichkeit bieten soll diverse Ereignisse von Dingen (IoT-Ereignisse) zu visualisieren. Des weiteren ist die LED-Anzeige selbst ein „Ding mit Internet“ und wird so zu einem kleinen Teil der IoT-Welt.

2.5 Smart Home

Der Begriff „Smart Home“ überschneidet sich mit vielen anderen, ähnlichen Begriffen (Connected Home, Smart House, Intelligentes Wohnen u.a.). Momentan gibt es keinen anerkannten Begriff für diesen Bereich, der eine genaue Definition zulässt (Strese u. a., 2010; Andelfinger und Hänisch, 2015). Allerdings steht hinter den genannten Begriffen immer ein ähnliches Konzept.

Dieses Konzept bezeichnet die Vernetzung von elektronischen Gegenständen (z.B. Lampen, Haushaltsgeräten, Fernseher, Heizkörperthermostate) in einem „Zuhause“ (z.B. einer Wohnung). Die Gegenstände können untereinander kommunizieren und sollen sich so auf die Bedürfnisse und Gewohnheiten der Smart-Home-Bewohner intelligent einstellen. Der Gedanke ist, dass durch die Vernetzung ein Mehrwert für die Bewohner entsteht, der über die übliche Nutzung der Gegenstände hinausgeht (Strese u. a., 2010).

Die Idee eines Smart Homes ist schon viele Jahre alt (Andelfinger und Hänisch, 2015), dennoch besteht bis heute ein grundlegendes Problem: Die fehlende Standardisierung. Es sind weitgehend nur proprietäre Systeme oder Insellösungen verbreitet, welche den Nachteil haben, dass sie den Nutzer an einen Hersteller binden oder keine offenen Schnittstellen anbieten (Strese u. a., 2010; Strese, 2016; Andelfinger und Hänisch, 2015). Dies führt dazu, dass Smart-Home-Lösungen entweder sehr teuer sind oder nur einen Teilbereich im Smart Home abdecken (Strese u. a., 2010; Andelfinger und Hänisch, 2015).

Trotz der bestehenden Schwierigkeiten wird davon ausgegangen, dass das Thema Smart Home weiterhin ein großes Potential hat. Dafür spricht unter anderem die Integration von Multimediageräten in viele Haushalte (Andelfinger und Hänisch, 2015). Bei Multimediageräten (wie z.B. Smartphones und Smart-TVs) ist das gelungen, worauf Smart Home noch wartet: Eine standardisierte Vernetzung unterschiedlicher Geräte von unterschiedlichen Herstellern.

Mittlerweile ist es zum Beispiel recht problemlos möglich, ein mit dem Smartphone aufgenommenes Foto auf einem Fernseher mit Netzwerkanschluss wiederzugeben oder über den Fernseher auf die Videobibliothek des Heimrechners zuzugreifen (DLNA). Hier ist ein Trend zu fortschreitender Vernetzung innerhalb der Wohnung zu erkennbar.

Ein weiterer Antrieb könnte die zunehmend alternde Gesellschaft (demografischer Wandel) sein. Smart Home soll älteren Menschen ein selbstständiges und komfortables Leben zu Hause ermöglichen, sodass ein Umzug in eine geeignetere Wohnung oder Pflegeeinrichtung nicht notwendig ist. Dieser Teilbereich wird meist unter dem Begriff „Ambient Assisted Living“ geführt (Andelfinger und Hänisch, 2015).

Im Kontext dieser Arbeit ist das Thema Smart Home von Relevanz, da die Visualisierung von IoT-Ereignissen durch eine LED-Anzeige schon zu einem smarteren Wohnen beiträgt. Außerdem könnten Smart-Home-Geräte ihre Ereignisse über die LED-Anzeige visualisieren. Die LED-Anzeige könnte zum Beispiel die aktuelle Temperatur visualisieren oder anzeigen, dass der Kühlschrank leer ist. Des Weiteren könnte die LED-Anzeige auch einfach als smarte Beleuchtung genutzt werden, die sich an die Wünsche der Bewohner anpassen lässt.

2.6 LED

„LED“ ist die Abkürzung für Light-emitting diode (zu deutsch: licht-emittierende Diode), welche seit den 1970er-Jahren als Massenware produziert wird. Damals wurden die LEDs nur als Anzeige und nicht als Leuchtmittel genutzt, da ihre Lichtausbeute von unter $0.1\,\mathrm{lm/W}$ (Lumen pro Watt) viel zu gering war (Fördergemeinschaft Gutes Licht, 2010). Sie ermöglichten damals die ersten digitalen Armbanduhren und Taschenrechner (Schubert, 2006).

Die heutige Lichtausbeute von LEDs liegt mit circa $100\,\mathrm{lm/W}$ 1000-mal höher (Fördergemeinschaft Gutes Licht, 2010). Mit dieser Lichtausbeute eignen sich LEDs hervorragend als Leuchtmittel und haben schon Einzug in viele Haushalte gefunden, unter anderem, weil sie deutlich energieeffizienter sind als herkömmliche Glühbirnen.

Neben der Energieeffizienz haben LEDs weitere Eigenschaften, die sie für die Verwendung als Anzeige qualifizieren (nach Hager Vertriebsgesellschaft mbH, 2012, S. 31):

- 100% Betriebshelligkeit sofort nach dem Einschalten
- Lebensdauer ist unabhängig von der Schalthäufigkeit
- Stufenlos dimmbar, ohne Farbänderung und Effizienzverlust

Arten von LEDs

Es gibt verschiedene Arten von LEDs, welche sich durch ihre Funktionen und ihren Farbumfang voneinander abgrenzen. Die wichtigsten Arten und ihre Besonderheiten werden im Folgenden erläutert.

RGB-LED Eine „RGB-LED“ besteht aus drei einzelnen LEDs in den Farben Rot, Grün und Blau. Durch individuelle Ansteuerung der einzelnen LEDs kann mittels additiver Farbmischung jede Mischfarbe aus den drei Grundfarben erzeugt werden. Leuchten beispielsweise die Rote-, Grüne- und Blaue-LED mit gleicher Intensität entsteht ein weißliches Licht.

RGBW-LED Die „RGBW-LED“ enthält neben den LEDs in den drei Grundfarben noch eine weiße LED. Dies hat den Vorteil, dass sie reines weißes Licht anzeigen kann – ohne die Farbmischung zur Hilfe zu nehmen. Zudem ist das reine weiße Licht optisch deutlich wärmer als das weißliche Licht durch Farbmischung.

LED-Streifen Der „LED-Streifen“ fasst mehrere LEDs auf einer Leiterplatte zusammen, welche die Elektronik zur Steuerung und Energieversorgung enthält. Außerdem wird über die Leiterplatte entstehende Wärme abgeleitet. Häufig ist die Leiterplatte mit einem Klebestreifen auf der Unterseite ausgestattet, um die Installation zu erleichtern. Nachfolgend werden die Unterarten von mehrfarbigen LED-Streifen erläutert.

Unterarten von LED-Streifen

LED-Streifen können aus ein- oder mehrfarbigen LEDs bestehen. Mehrfarbige LED-Streifen lassen sich in zwei Unterarten kategorisieren, die sich in ihrer Form zur Steuerung der Farbe unterscheiden.

Nicht Adressierbar / analog Die Steuerung erfolgt über drei Adern (eine Ader je Farbe), dabei gibt die Spannungsstärke der Ader die Leuchtstärke der Farbe vor.

Adressierbar / digital Jede einzelne LED kann individuell über eine Ader gesteuert werden, dabei werden digitale Steuerbefehle über die Ader gesendet und von der entsprechenden LED ausgewertet.

Adressierbare LED-Streifen bieten im Vergleich zu nicht adressierbaren LED-Streifen eine deutlich größere Bandbreite an Anzeigemöglichkeiten, da durch die Adressierung auch die Anzahl und Lage der zu leuchtenden LEDs gesteuert werden kann. So ist es mit adressierbaren LED-Streifen zum Beispiel möglich, die eine Hälfte des Streifens blau und die andere Hälfte weiß leuchten zu lassen. Dafür sind adressierbare LED-Streifen im Vergleich zu nicht adressierbaren LED-Streifen preislich teurer, da für jede LED ein Steuerungschip in den Streifen integriert ist (Cooper, 2015).

3 Analyse der Anforderungen

In diesem und den folgenden Kapiteln wird der Name EventAlert als vorläufiger Name für die Anwendung verwendet, die es zu entwickeln gilt.

Das Ziel dieses Kapitels ist das Analysieren der Anforderungen an die Soft- und Hardware von EventAlert und an Apache Flink. Hierdurch soll deutlich werden, was die Hard- und Software von EventAlert konkret können soll. Außerdem soll ermittelt werden, inwiefern Flink die Entwicklung von EventAlert unterstützen kann, und was dafür von Flink gefordert wird.

Die Analyse basiert auf drei Szenarien (siehe 3.1), welche mögliche Anwendungen von EventAlert beschreiben. Anhand von diesen möglichen Anwendungen werden anschließend diverse Anforderungen an Soft- und Hardware aufgestellt.

Die Anforderungen an die Anwendung werden in funktionale und nicht funktionale Anforderungen aufgeteilt. Des Weiteren sind die Anforderung an Flink in einer eigenen Anforderungskategorie aufgeführt.

Die Anforderungen sind fortlaufend nummeriert und werden jeweils am Ende eines Absatzes präzisiert (erkennbar durch kursive Schrift). Zudem werden alle Anforderungen in einer Übersicht im Anhang zusammengefasst (siehe Anhang A, ab Seite 96).

3.1 Szenarien

Um eine Vorstellung und ein besseres Verständnis über das Ziel und den Nutzen von EventAlert zu bekommen, werden zunächst drei verschiedene Anwendungsszenarien aufgeführt.

Mithilfe dieser konkreten Beispielszenarien sollen später die Anforderungen an EventAlert und die umgebenden Systeme entwickelt werden.

E-Mail Szenario Der Anwender möchte informiert werden, wenn neue E-Mails in seinem Postfach ankommen. Allerdings möchte er nur informiert werden, wenn die E-Mail von der Absender-Adresse *erika@mustermann.de* stammt.

Dafür hat der Anwender zuvor die Login-Daten seines E-Mail-Kontos für EventAlert zur Verfügung gestellt.

Außerdem hat der Anwender Kriterien hinterlegt, die definieren über welche E-Mails er informiert werden möchte. Die Kriterien werden anhand von Parametern aus den Metadaten (zum Beispiel Empfangszeit oder Adresse des Absenders) und den Daten (Nachrichteninhalt) der E-Mail festgelegt.

Innerhalb dieses Szenarios würden die Kriterien den gewünschten Absender der E-Mail (*erika@mustermann.de*) und eine Identifikation des E-Mail Postfachs (falls der Anwender mehrere Postfächer besitzt) enthalten.

Des Weiteren wurde vom Anwender eine Reaktion festgelegt. Die Reaktion spezifiziert, wie auf eine E-Mail, die mit den Kriterien übereinstimmt, reagiert werden soll. Reaktionen haben dabei zumeist einen Einfluss auf die physische Umgebung des Anwenders, wie z.B. das Abspielen eines Tons oder das Leuchten eines LED-Streifens.

In diesem Szenario hat sich der Anwender für eine Reaktion entschieden, die einen LED-Streifen für 500 Millisekunden in Rot aufleuchten lässt.

Zukünftig wird der Anwender nun über neue E-Mails von *erika@mustermann.de* informiert, indem der LED-Streifen kurz in roter Farbe aufleuchtet.

Twitter Szenario Der Anwender möchte einen Twitter-Gefahrenmelder haben, welcher ihn über Gefahren in seiner Umgebung informiert.

Unter *seiner Umgebung* versteht der Anwender seine Heimatstadt Hamburg. Deshalb hat der Anwender EventAlert zunächst so konfiguriert, dass es alle Tweets mit dem Hashtag Hamburg (*#Hamburg*) empfängt.

Da der Anwender nicht über jedes Tweet mit dem Hashtag Hamburg informiert werden möchte, sondern nur, wenn es eine Gefahr gibt, hat der Anwender bestimmte Kriterien definiert. Die Kriterien basieren hier auf Parametern der Metadaten und Daten des Tweets (zum Beispiel Autor des Tweets oder Inhalt des Tweets).

Die Kriterien sollen in diesem Szenario also die „Gefahren-Tweets“ von den herkömmlichen Tweets unterscheiden. Dafür hat der Anwender Kriterien definiert, die erfüllt sind, wenn im Inhalt des Tweets eines der folgenden Worte vorkommt: „Gefahr“, „Katastrophe“, „Terror“, „Evakuierung“, „Anschlag“, „Attentat“ oder „Unfall“.

Der Anwender rechnet damit, dass seine Kriterien häufig erfüllt sein werden, auch wenn es gar keine Gefahr in Hamburg gibt. Deswegen schränkt der Anwender seine Kriterien weiter ein und definiert, dass innerhalb einer Stunde mehr als 100 Tweets mit einem der oben genannten Begriffe im Inhalt getwittert werden müssen.

Wenn also 100 „Gefahren-Tweets“ innerhalb einer Stunde getwittert werden, sind alle Kriterien erfüllt und der Anwender kann von einer ernsthaften Gefahr im Raum Hamburg ausgehen.

Als Reaktion – falls die Kriterien erfüllt sind – möchte der Anwender, dass ein Leuchteffekt auf einem LED-Streifen abgespielt wird. Daher hat der Anwender eine Reaktion definiert, die den LED-Streifen abwechselnd Rot und Blau leuchten lässt.

Falls der LED-Streifen des Anwenders nun abwechselnd Rot und Blau leuchtet, weiß dieser, dass es möglicherweise eine Gefahr in Hamburg gibt. Jedenfalls weist die hohe Anzahl von Tweets mit den einschlägigen Begriffen und dem Hashtag Hamburg darauf hin.

Telegram Szenario Telegram ist ein kostenloser, cloudbasierter *Instant Messaging*-Dienst (die Nachrichtenübermittlung erfolgt sofort) für das private Kommunizieren zwischen zwei oder mehreren Personen. Telegram Clients (Endbenutzer Anwendungen) stehen für unterschiedliche Geräte (Smartphone, PC, Tablet) und diverse Betriebssysteme zur Verfügung. Außerdem bietet Telegram eine offene API zur Entwicklung eigener Clients (Telegram Messenger LLP).

Der Anwender möchte benachrichtigt werden, wenn er eine neue Nachricht bei Telegram erhält. Dabei möchte er durch die Art der Benachrichtigung unterscheiden können, ob die Nachricht von seinem Lebensgefährten oder von einem anderen Kontakt kommt.

Hierzu hat der Anwender zunächst seinen Telegram-Account mit EventAlert verknüpft. Dazu musste er seine Handynummer in EventAlert hinterlegen und EventAlert über einen Authentifizierungscode für Telegram freigeben.

Anschließend wurden Kriterien vom Anwender in EventAlert definiert. Die Kriterien bestehen hier aus Parametern von Metadaten und Daten der Telegram-Nachricht (zum Beispiel Telegram-Name des Absenders oder Inhalt der Telegram-Nachricht).

Da der Anwender über jede neue Nachricht benachrichtigt werden möchte, definiert er zunächst Kriterien, die auf alle Nachrichten zutreffen. In diesem Szenario würde das Kriterium dann lediglich den Parameter Handynummer des Anwenders (Empfängers) enthalten, damit es auf alle Nachrichten zutrifft, die die Anwender-Handynummer über Telegram empfängt.

Zudem definiert der Anwender eine Reaktion zu diesem Kriterium: Er möchte mit einem hellblauen Leuchten seines LED-Streifens über neue Nachrichten informiert werden. Mit diesem Kriterium und dieser Reaktion wird der Anwender über alle neuen Telegram-Nachrichten benachrichtigt.

Jedoch will der Anwender – wie schon erwähnt –, dass Nachrichten von seinem Lebensgefährten besonders hervorgehoben werden. Dafür erstellt der Anwender ein weiteres Kriterium, welches zutrifft, wenn es sich bei dem Parameter Absender um seinen Lebensgefährten handelt. Für dieses Kriterium definiert der Anwender eine weitere Reaktion: Der LED-Streifen soll diesmal in zwei Farben gleichzeitig leuchten. Als Farben stellt der Anwender die Farbe Rot und Hellblau ein.

Leuchtet der LED-Streifen nun hellblau auf, weiß der Anwender, dass er eine neue Telegram-Nachricht hat. Leuchtet die LED-Streifen hingegen in den Farben Hellblau und Rot gleichzeitig, weiß der Anwender, dass er eine Nachricht von seinem Lebensgefährten erhalten hat.

3.2 EventAlert

Auf Basis der aufgestellten Szenarien soll nachfolgend die grundlegende Funktionalität von EventAlert abgeleitet werden. Des Weiteren soll analysiert werden, von welchen Bestandteilen EventAlert umgeben ist, beziehungsweise mit welchen Fremdsystemen EventAlert zusammenarbeiten muss. Auf Grundlage dieser Informationen soll abschließend ein technischer Anspruch an EventAlert formuliert werden.

Dabei werden – wie in der Einleitung von Kapitel 3 erwähnt – bereits Anforderungen an EventAlert aufgestellt.

Kernfunktionalität Die Szenarien geben schon einige Informationen über die gewünschte Funktionalität von EventAlert. Also über das, was EventAlert können soll. Aus den Szenarien lassen sich folgende Aussagen über die gewünschte Funktionalität von EventAlert ableiten:

- EventAlert soll IoT-Ereignisse aus verschiedenen Quellen (E-Mail, Twitter und Telegram) empfangen und auswerten können.
- Die Auswertung soll anhand von flexiblen, vom Anwender definierbaren Kriterien stattfinden können.
- Die Kriterien basieren auf den Parametern der Metadaten und Daten des jeweiligen IoT-Ereignisses.
- Falls die vom Anwender definierten Kriterien zutreffen, soll eine Reaktion ausgeführt werden.
- Die Reaktion soll flexibel vom Anwender definiert werden können.
- Die Reaktion hat eine Auswirkung auf die physische Umgebung des Anwenders.

Das Auswerten der IoT-Ereignisse soll dabei die Kernfunktion von EventAlert sein. Neben dieser Kernfunktion soll EventAlert Ereignisse aus verschiedenen Quellen empfangen und Reaktionen ausführen können.

Die Auswertung von EventAlert soll anhand von Kriterien analysieren, ob aus einem IoT-Ereignis eine Relevanz für den Anwender resultiert. Im Falle einer Relevanz für den Anwender muss analysiert werden, welche Reaktion ausgeführt werden soll.

Die Reaktion in den Szenarien ist immer die Visualisierung der Ereignisse über eine LED-Anzeige. Also das Leuchten eines LED-Streifens (siehe 2.6 auf Seite 19) mit einem vom Anwender definierten Leuchteffekt. Ein Leuchteffekt soll sowohl die Leuchtfarbe als auch Leuchtdauer und Anzahl der LEDs bestimmen können.

Auch wenn keine in den Szenarien vorkommen, sind weitere Reaktionen – neben der Visualisierung über eine LED-Anzeige – denkbar und sollten für eine spätere Weiterentwicklung von EventAlert bedacht werden.

Zur Umsetzung der Kriterien und Reaktionen würden sich in der Entwicklung Filterregeln anbieten. Anhand der Filterregeln soll der Anwender flexibel eigene Bedingungen definieren und auch die Reaktion bei Erfüllung festlegen können.

Die Filterregeln müssen dafür flexibel gestaltet werden, da die möglichen Bedingungen je nach Typ des Ereignisses wechseln. Zum Beispiel hat ein Telegram-Ereignis andere Parameter in seinen Daten als ein E-Mail-Ereignis.

Außerdem ist zu beachten, dass verschiedene Filterregeln gleichzeitig auf ein Ereignis zutreffen können, sodass zwei Reaktionen gleichzeitig hervorgerufen werden. Um dies einstellbar zu machen, soll es möglich sein, die Filterregeln zu priorisieren. Dies bietet dem Anwender die Möglichkeit, wichtige Reaktionen auf ein Ereignis bevorzugt zu behandeln.

Funktionale Anforderung (A1): Der Anwender kann individuelle Reaktionen (z.B. einen Leuchteffekt) für Ereignisse oder Ereignisfolgen festlegen. Die Festlegung erfolgt anhand von Filterregeln, welche auf die Daten und Metadaten des jeweiligen Ereignisses angewendet werden.

Funktionale Anforderung (A2): Der Anwender kann seine Filterregeln priorisieren. Die Reaktionen von Filterregeln mit hoher Priorität werden bevorzugt behandelt.

Umgebung Anhand der ersten Analyse der Kernfunktion von EventAlert und mithilfe der Szenarien lässt sich bereits erkennen, von welchen Bestandteilen EventAlert umgeben ist beziehungsweise mit welchen Fremdsystemen EventAlert zusammenarbeiten muss.

Demnach wird EventAlert von den zwei grundlegenden Bestandteilen *Ereignisquelle* und *Reaktionsempfänger* umgeben (siehe Abbildung 3.1 auf der nächsten Seite). Der Bestandteil

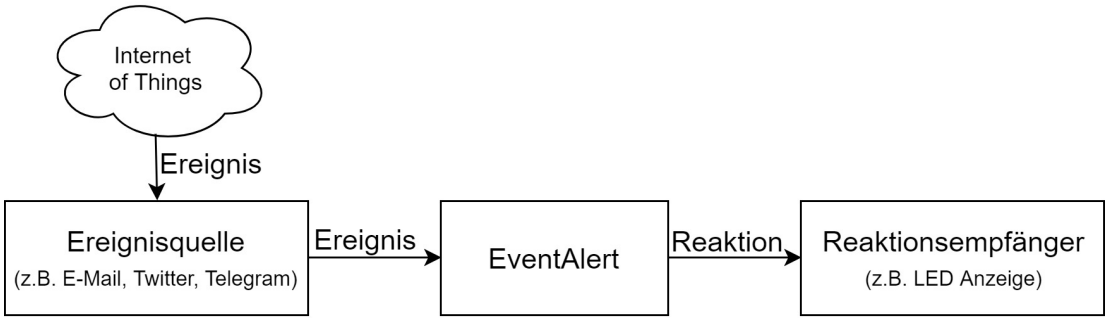


Abbildung 3.1: Umgebende Bestandteile von EventAlert

Ereignisquelle soll Ereignisse produzieren, damit diese von EventAlert verarbeitet werden können. Der Bestandteil Reaktionsempfänger soll Reaktionen aus EventAlert empfangen und verarbeitet beziehungsweise visualisieren.

Wie in Abbildung 3.1 zu sehen ist, stellt EventAlert also gewissermaßen das Verbindungsstück zwischen den zwei Bestandteilen Ereignisquelle und Reaktionsempfänger dar.

Die Ereignisquelle ist für jedes Szenario verschieden. Zum Beispiel ist die Ereignisquelle im E-Mail-Szenario das Postfach eines E-Mail-Kontos, während im Twitter-Szenario das Twitter-Portal die Ereignisquelle darstellt.

Anders ist dies hingegen bei dem Reaktionsempfänger, dieser ist in jedem Szenario gleich. Alle Szenarien haben zum Ziel, einen LED-Streifen zum leuchten zu bringen. Dennoch soll – wie bereits erwähnt – die Möglichkeit offen gehalten werden, später weitere Reaktionsempfänger hinzuzufügen.

Um eine möglichst große Flexibilität zu gewährleisten, sollen die Bestandteile unabhängig voneinander sein. Das bedeutet, dass die Ereignisquelle keine direkte Verbindung zum Reaktionsempfänger hat und nichts von dessen Existenz weiß, dies gilt genauso umgekehrt.

Die Unabhängigkeit der Bestandteile hat im Hinblick auf die Ausführung und Entwicklung mehrere Vorteile. Zum einen stellt die Unabhängigkeit der Bestandteile sicher, dass sich Fehler eines Bestandteils nicht auf andere Bestandteile auswirken können, zum anderen ist es möglich, einen ganzen Bestandteil auszutauschen oder einen neuen Bestandteil hinzuzufügen. Dadurch können später einfach weitere Reaktionsempfänger zu EventAlert hinzugefügt werden, dies könnte zum Beispiel ein Reaktionsempfänger sein, der ein akustisches – anstatt ein visuelles – Signal abspielt.

In der folgenden Anforderung wird der Begriff Komponente statt Bestandteil verwendet. Die Komponente ist der fachliche Ausdruck für den Bestandteil einer Anwendung ist. In diesem und in den nachfolgenden Kapiteln wird der Begriff Komponente anstatt Bestandteil verwendet.

Nicht funktionale Anforderung (A3): Die Komponenten von EventAlert sind voneinander vollständig unabhängig.

Im späteren Verlauf werden die Anforderungen an die beiden umgebenden Komponenten von EventAlert noch genauer analysiert (siehe hierzu unter Punkt 3.3 und 3.4).

Technischer Anspruch EventAlert soll auf eine spätere Weiterentwicklung ausgelegt und erweiterbar sein, sodass sich leicht weitere Komponenten, wie neue Ereignisquellen oder Reaktionsempfänger hinzufügen lassen.

Konkret soll das Hinzufügen einer neuen Ereignisquelle mit einem Aufwand von weniger als zwei Stunden erfolgen können.

Außerdem wäre es wünschenswert, wenn sich – ohne großen Programmieraufwand – weitere Reaktionsempfänger (z.B. für akustische Reaktionen) an EventAlert anschließen lassen. Für das Anbinden eines neuen Reaktionsempfängers gilt auch hier ein maximaler Aufwand von zwei Stunden.

Dies erfordert von EventAlert– neben unabhängigen Komponenten – eine offene Programmierung und die Bereitstellung von einfach zu benutzenden und gut dokumentierten Schnittstellen.

Nicht funktionale Anforderung (A4): Alle Schnittstellen, die zum Hinzufügen einer neuen Ereignisquelle benötigt werden, sind dokumentiert (JavaDoc).

Nicht funktionale Anforderung (A5): Der Aufwand, eine neue und nicht komplexe Ereignisquelle hinzuzufügen, beträgt weniger als zwei Stunden.

Nicht funktionale Anforderung (A6): Der Aufwand, einen neuen und nicht komplexen Reaktionsempfänger anzubinden, beträgt weniger als zwei Stunden.

3.3 Ereignisquellen

Die Ereignisquelle ist eine der beiden umgebenden Komponenten von EventAlert (siehe Abbildung 3.1 auf der vorherigen Seite). In diesem Abschnitt soll analysiert werden, was bei der Komponente Ereignisquelle zu beachten ist und wie sich dies konkret umsetzen lässt.

Die Aufgabe von Ereignisquellen ist das Zurverfügungstellen von Ereignissen für EventAlert. Eine Quelle kann dabei alles sein, woraus Ereignisse entstehen. Dies schließt im Prinzip jede Software oder jedes Gerät ein, das Daten produziert. Die einzige Aufgabe der Ereignisquelle ist es also, Ereignisse zu produzieren und für EventAlert zur Verfügung zu stellen.

Um die späteren Auswertungsmöglichkeiten durch EventAlert nicht im Voraus einzuschränken, soll die Ereignisquelle alle verfügbaren Ereignisse direkt an EventAlert ohne Einschränkung weiterleiten. Die Ereignisquelle soll also keine Vorauswahl über die Relevanz ihrer Ereignisse treffen, denn dies soll erst durch EventAlert geschehen. Für eine detaillierte Auswertung durch EventAlert ist es wichtig, dass eine möglichst große Menge an Ereignissen zur Verfügung steht und kein Ereignis „unter den Tisch fällt“.

Es gibt allerdings eine Ausnahme, bei der eine Vorauswahl und somit Einschränkungen von Ereignissen innerhalb der Quelle (vor der Übergabe an EventAlert) sinnvoll ist. Die Ausnahme gilt für Ereignisquellen, die hauptsächlich irrelevante Ereignisse produzieren, bei denen der Großteil der Ereignisse also keine Relevanz für den Anwender aufweist und auch in Zukunft keine Relevanz absehbar ist.

Ein Beispiel für so eine Ausnahme ist die Ereignisquelle des Twitter-Szenarios (siehe Punkt 3.1 auf Seite 21). Für dieses Szenario ist es sinnvoll, die Einschränkung nach dem Hashtag Hamburg bereits in der Twitter-Ereignisquelle selbst vorzunehmen. Ohne diese Vorauswahl durch die Twitter-Quelle würden mehrere tausend irrelevant Tweets pro Sekunde eintreffen. Diese große Menge an Ereignissen ließe sich sogar mit EventAlert verarbeiten, jedoch ist dies im Kontext des Szenarios nicht sinnvoll. Denn EventAlert benötigt für dieses Szenario lediglich Tweets mit dem Hashtag Hamburg und nicht alle Tweets die weltweit getwittert werden.

Ereignisquellen mit Flink

Das Anschließen einer Ereignisquelle an EventAlert erfolgt über sogenannte *Data Sources* (zu deutsch: „Daten Quellen“), welche von Apache Flink (siehe 2.2 auf Seite 9) zur Verfügung gestellt werden.

Flink bietet hierbei eingebaute Quellen an, welche das direkte Einlesen von Daten aus einer Datei oder von einem Server ermöglichen (Deshpande, 2017; Apache Software Foundation, b). Die eingebauten Quellen stellen dabei lediglich die Verbindung zu einem Server oder einer Datei her und lesen deren Zeichenketten zeilenweise ein, um sie in Flink zur Verfügung zu stellen. Da die Daten unstrukturiert eingelesen werden, muss zum Extrahieren von sinnvollen Ereignissen eigene Logik entwickelt werden.

Das Einlesen aus einer Datei eignet sich besonders für statische Datenbestände. Daten von einem Server (Programm, das Daten über eine Netzwerkschnittstelle bereitstellt) einzulesen

eignet sich hingegen eher für Echtzeit-Datenströme, dies hängt jedoch stark davon ab, welche Art Daten der Server bereitstellt.

Flink Connectors Weitere eingebaute Quellen bietet Flink als sogenannte *Connectors* an (siehe auch Punkt 2.2 auf Seite 11). Connectors sind Verbindungen zu externen Fremdsystemen (z.B. zu Apache Kafka oder zu Twitter), die sich direkt als Quelle in Flink einbinden lassen (Deshpande, 2017; Apache Software Foundation, f).

Über den Twitter-Connector kann beispielsweise direkt auf den Datenstrom von Twitter-Nachrichten (Tweets) zugegriffen werden – hierzu ist keine Entwicklungsarbeit notwendig. Allerdings überträgt der Twitter-Connector die Tweets lediglich als JSON-Objekte (strukturierter Text), die noch geparst werden müssen. Daher muss hierbei trotzdem eine eigene Logik entwickelt werden, die aus den JSON-Objekten sinnvolle Ereignisse extrahiert.

Eigene Quellen Des Weiteren lassen sich über ein – von Flink bereitgestelltes – Interface *SourceFunction* (zu deutsch: „Quellfunktion“) eigene Quellen entwickeln, welche Daten von anderen Systemen oder Dingen empfangen können (Apache Software Foundation, b).

Ein Interface ist eine softwaretechnische Schnittstelle, welche die Zugriffsweise auf ein Unterprogramm beschreibt. Über ein Interface kann ein bestehendes System mit selbst entwickelten Programmteilen kommunizieren. Dadurch wird das Einbinden von eigener Funktionalität in ein bestehendes System ermöglicht. In diesem Kontext ist das bestehende System Apache Flink, welches über das Interface SourceFunction mit selbst entwickelten Quellen erweitert werden kann.

Unterschied zwischen eingebauten und eigenen Quellen

Eigene – selbst entwickelte – SourceFunctions sind von den eingebauten Flink-SourceFunctions abzugrenzen. Eigene SourceFunctions sind mächtiger als die vordefinierten SourceFunctions von Flink, welche nur Zeichenketten zur Verfügung stellen können. Über selbst entwickelte SourceFunctions lassen sich Daten (Ereignisse) in Form von Objekten direkt an Flink übergeben. Objekte sind selbstdefinierte Datentypen (Datenstrukturen), die über bereitgestellte Funktionen den direkten Zugriff auf einzelne Datenbestandteile ermöglichen. Objekte sind eine Abstraktion der ursprünglichen Daten, die zum Beispiel als Text oder als Zahlen vorliegen. Das Übergeben von Objekten hat den Vorteil, dass die Daten schon strukturiert und aufbereitet in Flink ankommen, sodass keine weitere Logik notwendig ist, um ein sinnvolles Ereignis zu extrahieren. Die Logik zur Strukturierung und Aufbereitung verschiebt sich also in Richtung SourceFunction und findet im Gegensatz zu den eingebauten Flink-SourceFunctions bereits vor der Übergabe

an Flink statt. Die einzige Logik, die dann noch in Flink integriert werden muss, ist die Filterung anhand von einzelnen Datenbestandteilen (Parametern) und das Erkennen von Zusammenhängen zwischen Ereignissen.

Die Strukturierung der Daten innerhalb einer Quelle ist abzugrenzen von der Vorauswahl von Ereignissen. Die Quelle soll ihre Ereignisse lediglich als Objekt strukturiert übergeben, selbst aber keine Auswahl darüber treffen, welches Ereignis relevant ist.

Beispiele in Bezug auf die Szenarien

E-Mail Das E-Mail-Szenario (siehe Punkt 3.1 auf Seite 20) stellt ein gutes Beispiel für die Verwendung einer eigenen SourceFunction dar. Die SourceFunction wäre – im Kontext des E-Mail-Szenarios – ein Programm, das sich mit einem IMAP-Server (E-Mail-Server) verbindet und ein Ereignis generiert, sobald sich auf dem E-Mail Konto etwas ändert. Das Ereignis würde hier nicht nur aus einer Zeichenkette bestehen sondern hätte eine Struktur, in der man zum Beispiel direkt auf den Absender oder den Betreff zugreifen könnte. Dadurch ließen sich E-Mail-Ereignisse leicht nach genau diesen Parametern filtern.

Telegram Auch für das Telegram-Szenario (siehe Punkt 3.1 auf Seite 22) muss vermutlich eine eigene SourceFunction entwickelt werden. Hinter der SourceFunction würde ein Programm stehen, welches alle Änderungen von Telegram mitbekommt und daraufhin ein Ereignis generiert. Hierbei ist zu beachten, dass der Anwender den Zugriff von EventAlert auf Telegram bestätigen muss, damit die Quelle auf die Telegram-Daten zugreifen kann.

Twitter Beim Twitter-Szenario (siehe Punkt 3.1 auf Seite 21) könnte der Twitter-Connector zum Einsatz kommen. Es muss für dieses Szenario also keine eigene SourceFunction programmiert werden. Allerdings wäre für das Szenario – wie schon erwähnt – eine Vorauswahl innerhalb der Quelle sinnvoll, damit nicht alle weltweiten Tweets an EventAlert übergeben werden. Außerdem müssen die JSON-Objekte, die der Connector liefert, geparst werden. Vorauswahl und Aufbereitung der Tweets erfordern eigene Entwicklungsarbeit. Der Hauptteil, also die Verbindung zu Twitter und das Abholen der Ereignisse, wird jedoch vom Twitter-Connector übernommen und muss nicht selbst entwickelt werden.

3.4 Ereignissenke

Der Begriff Senke kommt aus der Graphentheorie und stellt das Gegenstück einer Quelle dar. Umgangssprachlich könnte man sagen, dass die Senke der „Abfluss“ in Bezug auf den Daten- oder Ereignisfluss ist.

Im Kontext von EventAlert ist der *Reaktionsempfänger* (siehe Abbildung 3.1 auf Seite 25) eine Ereignissenke. Die Ereignissenke (der Reaktionsempfänger) ist – neben der Ereignisquelle – die andere umgebende Komponente von EventAlert. Nachfolgend wird ermittelt, welche Funktion diese Komponente hat, was es bei Ereignissenken zu beachten gibt und wie sich diese mit Flink umsetzen lassen.

Die Aufgabe der Ereignissenke in EventAlert ist die Endverarbeitung von Reaktionen. Sie nimmt Reaktionen von EventAlert entgegen und führt diese aus. Eine Reaktion kann dabei verschiedenster Natur sein. So kann zum Beispiel als Reaktion das Generieren eines neuen Ereignisses ausgelöst oder eine Aktion auf einem Fremdsystem ausgeführt werden (wie zum Beispiel die Visualisierung auf einer LED-Anzeige).

Dabei ist die Senke die letzte Stelle, die in der Ereignisverarbeitung von EventAlert erreicht wird. Nach dem Eintreffen der Reaktion in der Senke ist die Verarbeitung eines IoT-Ereignisses abgeschlossen.

Reaktionen werden nachfolgend auch als Ereignisse betrachtet. Dabei ist zu beachten, dass eine Reaktion kein IoT-Ereignis ist, sondern von EventAlert selbst generiert wurde, falls ein IoT-Ereignis eine Filterregel erfüllt hat. Daher findet in der Senke auch keine Filterung von Ereignissen statt. Sie nimmt lediglich Ereignisse entgegen und reagiert auf diese. Die Filterung und Analyse der Ereignisse soll vorher mittels der Filterregeln des Anwenders geschehen.

Ereignissenken mit Flink

Während Ereignisquellen, wie unter Punkt 3.3 erwähnt, über Data Sources angeschlossen werden, erfolgt der Anschluss von Ereignissenken über sogenannte *Data Sinks* (Daten Senken). Apache Flink (siehe 2.2 auf Seite 9) bietet eingebaute Senken an, die das Schreiben von Zeichenketten auf einen Server oder in eine Datei ermöglichen (Deshpande, 2017). Für die Nutzung dieser eingebauten Senken müssen die Ereignisse vom Datentyp *String* (Zeichenkette) sein oder es muss eine Anweisung übergeben werden, wie die Daten (Ereignisse) in Zeichenketten umgewandelt werden sollen. Wie schon bei den eingebauten Ereignisquellen stellen die eingebauten Senken lediglich die Verbindung zu einer Datei oder einem Server her. Flink weist in

seiner Dokumentation darauf hin, dass die eingebauten Senken hauptsächlich zum Debuggen und Testen vorgesehen sind (Apache Software Foundation, b).

Flink Connectors Neben den Connectors als Quelle gibt es auch eingebaute Connectors, die Senken zur Verfügung stellen (siehe auch 2.2 auf Seite 11). Ein Connector stellt auch hier eine Verbindung zu einem externen Fremdsystem her. Der Unterschied zu einer Connector-Quelle ist, dass die Connector-Senke keine Ereignisse produziert, sondern Ereignisse entgegen nimmt. Die Connectors, die Senken zu Verfügung stellen, sind unter anderem für die Speicherung und Archivierung von Ereignissen sinnvoll, um eine spätere Verarbeitung und Analyse der Ereignisse zu ermöglichen.

Flink bietet hier zum Beispiel den Kafka-Connector an, welcher eine Verbindung zu Apache Kafka aufbaut (Apache Software Foundation, f). Der Kafka-Connector stellt also neben einer Quelle auch eine Senke zur Verfügung. Im Gegensatz zu dem Connector als Quelle werden hier Ereignisse an Apache Kafka übergeben.

Eigene Senke Außerdem lassen sich Senken selbst programmieren. Dies funktioniert – wie bei den Quellen – über ein von Flink bereitgestelltes Interface. Das Interface für Senken heißt *SinkFunction* (zu deutsch: „Senkenfunktion“). Über dieses Interface lassen sich eigene Senken in Flink einbinden (Apache Software Foundation, b). Je nachdem, wie die Senke programmiert ist, können neben Zeichenketten auch Objekte an die eigene Senke übergeben werden.

Beispiel in Bezug auf die Szenarien

In Bezug auf die Szenarien (siehe Punkt 3.1 ab Seite 20) wird eine Ereignissenke benötigt, welche die generierten Reaktionen von EventAlert verarbeiten kann. In Szenarien ist die Reaktion immer das Leuchten eines LED-Streifens, also die Visualisierung der Ereignisse über eine LED-Anzeige. So eine Ereingissenke könnte über das SinkFunction-Interface von Flink entwickelt werden. Die selbst entwickelte SinkFunction müsste dann eine Verbindung zu einem LED-Streifen herstellen und ihn leuchten lassen, sobald ein Ereignis (eine Reaktion) eintrifft.

Im Twitter- und Telegram-Szenario möchte der Anwender, dass die LED-Anzeige komplexere Leuchteffekte anzeigt. Bei dem Twitter-Szenario ist dies ein wiederholtes Leuchten in wechselnden Farben und im Telegram-Szenario ein Leuchten in mehreren Farben gleichzeitig. Um dieses Szenario zu realisieren, soll der Anwender spezielle LED-Reaktionen definieren können.

Über die LED-Reaktion könnte der Anwender dann zum Beispiel mehrere Farben definieren oder eine Zeitdauer des Leuchtens definieren. Dabei ist zu beachten, dass die Ereingissenke so

erweitert werden müsste, dass sie Farben und Zeitdauer der LED-Reaktion auslesen kann und dementsprechende Befehle an den LED-Streifen sendet.

Des Weiteren ist zu beachten, dass die Ereignissenke auch für zukünftige Szenarien geeignet sein soll. Das bedeutet, dass die Ereignissenke zukünftig vielleicht auch andere Reaktionen – neben dem Leuchten eines LED-Streifens – verarbeiten können muss. Dazu wäre es von Vorteil, wenn die Ereignissenke flexibel entwickelt wird, sodass später leicht weitere Reaktionen hinzugefügt werden können.

3.5 LED-Streifen-Steuerung

In den Szenarien wird als Reaktion auf ein zutreffendes IoT-Ereignis das Leuchtens eines LED-Streifens gefordert.

Der LED-Streifen soll als Anzeige dienen und die Ereignisse nach Anwenderwünschen visualisieren. Der Anwender soll über eine LED-Reaktion eigene Leuchteffekte definieren können, welche von der Ereignissenke (Reaktionsempfänger) ausgewertet und anschließend an den LED-Streifen übertragen werden.

Um die Leucht-Wünsche aus den Szenarien – also eine große Bandbreite an Visualisierungsmöglichkeiten – umsetzen zu können, muss der LED-Streifen gesteuert werden (siehe Punkt 2.6 auf Seite 19).

Die Steuerung müsste dafür eine Verbindung zur Ereignissenke (zum Reaktionsempfänger) und zum LED-Streifen haben. Sie ist also das Bindeglied zwischen Ereignissenke und dem LED-Streifen. Ihre Aufgabe ist es, Befehle, die aus der Ereignissenke kommen, für den LED-Streifen zu übersetzen und an ihn zu übertragen.

In diesem Abschnitt soll analysiert werden, welche Komponenten für die Steuerung benötigt werden und wie diese Verbunden werden können. Außerdem sollen Anforderungen an die Steuerung aufgestellt werden.

Möglichkeiten der Visualisierung Um das E-Mail- und Twitter-Szenario (siehe 3.1 ab Seite 20) zu erfüllen, muss die Steuerung mindestens die Leuchtfarbe und -stärke des LED-Streifens bestimmen können.

Eine erweiterte Anforderung an die Steuerung wäre, dass die einzelnen LEDs des Streifens angesteuert werden können, sodass auch das Telegram-Szenario erfüllt werden kann (Leuchten des Streifens in mehreren Farben gleichzeitig).

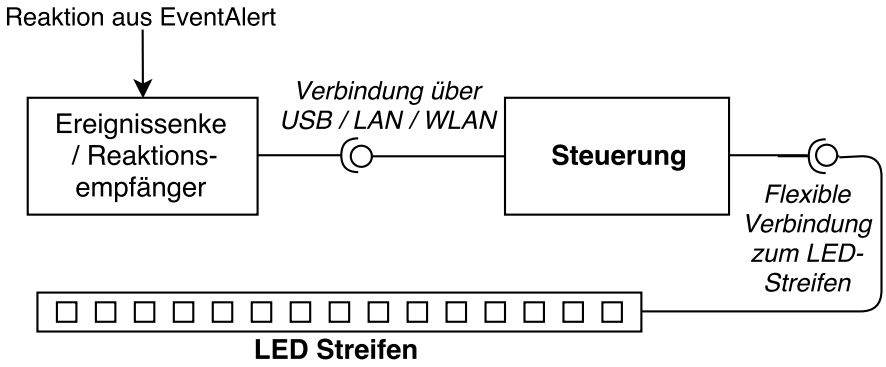


Abbildung 3.2: Umgebungskomponenten und Verbindungen der LED-Streifen-Steuerung

Zudem würde das individuelle Ansteuern einzelner LEDs die Bandbreite an Leuchteffekten und somit auch die Visualisierungsmöglichkeiten stark erhöhen, was auch für zukünftige Szenarien mehr Möglichkeiten bieten würde. Allerdings wird dafür ein adressierbarer LED-Streifen benötigt (siehe 2.6 auf Seite 19). Da die letztendliche Wahl des LED-Streifens in der Analyse offen bleiben soll, wird nur nachfolgende Anforderung an die Steuerung definiert.

Nicht funktionale Anforderung (A7): Die Steuerung kann Leuchtfarbe und -stärke des LED-Streifens steuern.

Hardwarekomponenten

Wie schon zu Beginn erwähnt, stellt die Steuerung eine Verbindung zwischen Ereignissenke und LED-Streifen her. Sie hat also mindestens zwei Verbindungen zu anderen Hardwarekomponenten. Die Anforderungen an diese Hardwarekomponenten und insbesondere an deren Verbindung werden im Folgenden erläutert. Die Ergebnisse werden zudem in der Abbildung 3.2 grafisch dargestellt.

Verbindung zwischen Steuerung und Ereignissenke Die Steuerung sollte sich über eine standardisierte Hardwareschnittstelle (z.B. USB, LAN oder WLAN) mit dem Computer, der die Ereignissenke ausführt, verbinden lassen. Die Verbindung soll direkt über USB oder indirekt über ein Netzwerk (LAN oder WLAN) erfolgen. Ein Anschluss über Adapter hätte den Nachteil, dass man neben LED-Streifen, Steuerung und Stromversorgung noch weitere Hardwarekomponenten benötigt.

Nicht funktionale Anforderung (A8): Die Steuerung wird über eine standardisierte Hardwareschnittstelle mit dem ausführenden Computer verbunden.

Verbindung zwischen Steuerung und LED-Streifen Des Weiteren sollte die Steuerung verschiedene LED-Streifen unterstützen können, hierzu wäre eine flexible Verbindung zwischen Steuerung und LED-Streifen sinnvoll. Flexibel bedeutet in diesem Kontext, dass die Verbindung zwischen LED-Streifen und Steuerung offen gestaltet ist und nicht über eine genormte Schnittstelle erfolgt. Die Steuerung soll also explizit keine standardisierte Hardwareschnittstelle besitzen.

Denkbar wäre beispielsweise eine Leiterplatte (Platine) auf welcher sich die Adern (Kabel) des LED-Streifens anschließen lassen. Diese Art von Verbindung hat den Vorzug, dass sie unabhängig vom Typ des LED-Streifens ist. Eventuell wäre es dadurch sogar möglich, neben dem LED-Streifen auch andere Geräte (zum Beispiel einen Lautsprecher) an die Steuerung anzuschließen – dies wäre im Hinblick auf die Möglichkeiten zur späteren Weiterentwicklung von großem Nutzen.

Nicht funktionale Anforderung (A9): Die Steuerung hat eine flexible Verbindung zum LED-Streifen, sodass sich verschiedene Arten von LED-Streifen anschließen lassen.

Stromversorgung Für die Stromversorgung von LED-Streifen und Steuerung sollen keine weiteren Hardwarekomponenten (wie z.B. ein Netzteil) notwendig sein. Optimal wäre eine Stromversorgung über die Verbindung zur Ereignissenke (beispielsweise über den Strom des USB-Anschlusses). Alternativ sollen LED-Streifen und Steuerung wenigstens eine gemeinsame Stromversorgung haben. Dies könnte zum Beispiel über ein gemeinsames Netzteil erfolgen.

Nicht funktionale Anforderung (A10): LED-Streifen und Steuerung haben eine gemeinsame Stromversorgung.

Programmierung

Die Steuerung soll nicht nur die Übersetzung von Befehlen in – für den LED-Streifen verständliche – Stromsignale vornehmen, sondern auch programmierbare Logik enthalten können.

Die Steuerung könnte beispielsweise so programmiert werden, dass sie komplexe Leuchteffekte, bei denen eine große Anzahl von LEDs individuell gesteuert werden, selbst berechnet.

Dies hat den Vorteil, dass aus der Ereignissenke nur ein Befehl zur Steuerung gesendet werden muss, um einen komplexen Leuchteffekt hervorzurufen. Der Leuchteffekt könnte dann von der Steuerung selbst berechnet und ausgeführt werden.

Die Alternative wäre, Befehle für jeden einzelnen Schritt des komplexen Leuchteffekts zu senden, was eine hohe Auslastung der Steuerung zur folge haben kann. Außerdem kann die

Latenzzeit der Verbindung zwischen Steuerung und LED-Streifen zu sichtbaren Verzögerungen des Leuchteffekts führen.

Die Verwendung einer programmierbaren Steuerung heißt allerdings nicht, dass die Leuchteffekte ausschließlich von der Steuerung berechnet werden sollen. Eine Berechnung durch die Ereignissenke wäre weiterhin möglich und ist gegebenenfalls auch sinnvoll.

Anforderungen Die Programmierung der Steuerung soll in einer hohen Programmiersprache erfolgen, sie soll mindestens eine objektorientierte Programmierung unterstützten. Außerdem sollen Bibliotheken für das Steuern von LED-Streifen vorhanden sein, sodass kein elektrotechnisches Wissen notwendig ist.

Des Weiteren sollen Bibliotheken vorhanden sein, welche die Kommunikation zwischen dem ausführenden Computer und der Steuerung ermöglichen. Es wird also eine softwaretechnische Kommunikationsschnittstelle erwartet, die sowohl von der Steuerung als auch von der Ereignissenke benutzt werden kann. Zudem wären Beispielprojekte zur Programmierung einer LED-Streifen-Steuerung von anderen Entwicklern wünschenswert.

Nicht funktionale Anforderung (A11): Die Steuerung des LED-Streifens ist programmierbar (enthält eigene Logik) und unterstützt eine objektorientierte Programmierung.

Nicht funktionale Anforderung (A12): Es sind Bibliotheken für die Steuerung zum Steuern eines LED-Streifens vorhanden. Für die Programmierung der Steuerung ist kein elektrotechnisches Wissen notwendig.

Nicht funktionale Anforderung (A13): Es sind Bibliotheken vorhanden, welche die Kommunikation zwischen dem ausführenden Computer und der Steuerung über eine softwaretechnische Kommunikationsschnittstelle ermöglichen.

3.6 Anforderungen an Apache Flink

Wie schon in der Einführung von Punkt 2.2 auf Seite 9 erwähnt, ist das Apache Flink Framework das zentrale System für die Verarbeitung und Analyse der Ereignisse in EventAlert. EventAlert basiert gewissermaßen auf Apache Flink.

Nachfolgend sollen Anforderungen an Flink formuliert werden, welche spezifizieren, was von Flink gefordert wird. Auf Basis dieser Anforderung soll sich später mit Flink kritisch auseinandergesetzt werden. Die Anforderungen an Flink sind dabei einer eigenen Anforderungskategorie zugeordnet.

Die grundsätzliche Erwartung an Flink ist, dass es eine gute Unterstützung bei der Entwicklung von EventAlert bietet. So wird erwartet, dass Flink das empfangen, filtern, analysieren und konvertieren von Ereignissen ermöglicht. Außerdem soll das Weiterleiten von Ereignissen an Fremdsysteme (z.B. LED-Anzeige) möglich sein.

Anforderung an Flink (A14): Flink kann Ereignisse empfangen, filtern, analysieren, konvertieren und an Fremdsysteme weiterleiten.

Korrektheit und fehlerfreie Ausführung Es wird erwartet, dass Flink während der Ausführung zu jeder Zeit korrekt und fehlerfrei arbeitet. Falls Fehler auftreten sollten, dürfen diese ausschließlich auf den eigenen Quellcode zurückzuführen sein (zum Beispiel falsche Nutzung von Flink-Funktionen oder Fehler in eigenen Ereignisquellen). Weiter wird erwartet, dass Flink robust ist: Ein Fehler im eigenen Quellcode darf nicht zum Absturz des Gesamtsystems führen, sondern maximal zum Absturz einzelner Komponenten (z.B. der Absturz einer Ereignisquelle).

Anforderung an Flink (A15): Flink arbeitet zu jeder Zeit korrekt und fehlerfrei. Das Auftreten von Fehlern darf ausschließlich auf den eigenen Quellcode zurückzuführen sein.

Anforderung an Flink (A16): Die Ausführung von Flink ist robust. Ein Fehler im eigenen Quellcode darf nicht zum Absturz des Gesamtsystems führen.

Leistung und Geschwindigkeit Von Flink wird eine hohe Leistung bezüglich der Latenz und des Datendurchsatzes erwartet. Flink soll Ereignisse in Echtzeit verarbeiten können. Die Verarbeitung eines Ereignisses durch Flink soll nicht länger als $10\,\mathrm{ms}$ (Millisekunden) dauern. Auch bei erhöhtem Datenaufkommen soll Flink die Geschwindigkeit beibehalten. Flink muss einem Datenaufkommen von bis zu $10\,000$ Ereignissen pro Sekunde standhalten, ohne dass die Verarbeitungsgeschwindigkeit (unter $10\,\mathrm{ms}$ pro Ereignis) beeinträchtigt wird. Es sollen keine Datenstaus entstehen, die von Flink verursacht werden.

Anforderung an Flink (A17): Die Verarbeitung eines Ereignisses von Flink dauert nicht länger als 10 Millisekunden.

Anforderung an Flink (A18): Flink kann bis zu $10\,000$ Ereignisse pro Sekunde verarbeiten, ohne dass andere Anforderungen beeinträchtigt werden.

Handhabung Die Arbeit mit Flink soll leicht und verständlich sein. Erweiterte Kenntnisse über den Quellcode und die Arbeitsweise von Flink sollen nicht notwendig sein. Hierfür

sollen die technischen Details von Flink verborgen sein. Dadurch soll es möglich sein, einen funktionierenden EventAlert-Prototyp in weniger als fünf Tagen zu entwickeln.

Außerdem wird erwartet, dass mit Flink ein kurzer und eleganter Programmierstil möglich ist. Eine schlichte Verbindung von einer einfachen Ereignisquelle und -senke soll mit weniger als 20 LOC (Lines of Code; Code-Zeilen) möglich sein.

Die Bezeichnung von Methoden und Parametern sollen selbsterklärend sein, sodass sie leicht verständlich sind. Außerdem sollen allen Klassen, Methoden und Parameter dokumentiert sein.

Des Weiteren soll die Verwendung von Lambda-Ausdrücken möglich sein, dies gilt insbesondere für die Methoden zur Transformation und Filterung von Datenströmen in Flink. Lambda-Ausdrücke (*Lambda Expressions*) gelten als modernes Programmiersprachen-Feature, welches seit Version 8 in Java verfügbar ist (Günther und Lehmann, 2013).

Anforderung an Flink (A19): Ein funktionierender Prototyp von EventAlert lässt sich in weniger als fünf Tagen mithilfe von Flink entwickeln.

Anforderung an Flink (A20): Es sind weniger als 20 LOC notwendig, um eine schlichte Verbindung von einfacher Ereignisquelle und -senke herzustellen.

Anforderung an Flink (A21): Methoden, Parameter und Klassen von Flink sind dokumentiert.

Anforderung an Flink (A22): Flinks Methoden und deren Parameter sind selbsterklärend bezeichnet und leicht verständlich.

Anforderung an Flink (A23): Flink unterstützt die Verwendung von Lambda-Ausdrücken.

Kommunikation und Verbindung Selbst entwickelte Programmteile sollen fließend in Flink integriert werden können. Ein äußerlicher Unterschied zu Flink-eigenen Programmteilen soll im Quellcode nicht zu erkennen sein. Dafür wird erwartet, dass der Entwickler dieselben Interfaces benutzen kann, die auch von den eingebauten Komponenten, wie zum Beispiel den eingebauten Data Sinks (siehe Punkt 3.4 ab Seite 30), genutzt werden.

Außerdem soll die Kommunikation mit externen Systemen über Flink-Connectoren reibungslos funktionieren und einfach einzubinden sein. Es wird erwartet, dass keine Fachkenntnisse des Fremdsystems notwendig sind, um eine Verbindung über die Connectoren herzustellen.

Anforderung an Flink (A24): Der Entwickler benutzt dieselben Interfaces von Flink, die auch von in Flink eingebauten Komponenten verwendet werden.

Anforderung an Flink (A25): Das Einbinden einer Ereignisquelle über einen Flink-Connector erfordert keine Fachkenntnisse über das Fremdsystem hinter dem Connector.

Tests und Fehlerkorrektur Flink soll den Entwickler bei der Korrektur von Fehlern (Bug-fixing) unterstützen. Das Finden von Fehlern in selbst entwickelten Programmteilen soll nicht durch Flink behindert werden. Hierfür wird erwartet, dass das Debuggen von selbst entwickelten Programmteilen, die auf Flink basieren, möglich ist.

Außerdem soll Flink das Schreiben von Tests vollständig unterstützten. Von Flink wird hierfür erwartet, dass ein Test-Framework zur Verfügung gestellt wird, welches das Schreiben von Tests erleichtert. Das Test-Framework von Flink soll dabei mit anderen Test-Frameworks wie zum Beispiel dem JUnit-Framework kompatibel sein. Es soll außerdem eine Möglichkeit bieten, eine vollständige Verarbeitung der Ereignisse anhand von Testdaten zu simulieren.

Anforderung an Flink (A26): Das Debuggen von selbst entwickelten Programmteilen, die auf Flink basieren, ist möglich.

Anforderung an Flink (A27): Flink stellt ein Test-Framework zur Verfügung, welches mit anderen Test-Frameworks kompatibel ist.

Anforderung an Flink (A28): Das Test-Framework von Flink ermöglicht die Simulation einer vollständigen Ereignis-Verarbeitung anhand von Testdaten.

3.7 Fazit

In diesem Kapitel wurden EventAlerts Funktionen und Kernaufgaben anhand von Beispiel-Szenarien (siehe 3.1 ab Seite 20) erklärt. Es wurden diverse Anforderungen und Erwartungen an einzelne Hard- und Softwarekomponenten gestellt, außerdem wurden deren Aufgabenbereiche erklärt und abgesteckt.

Dafür wurde zunächst die Kernfunktionalität von EventAlert anhand der Szenarien spezifiziert. Außerdem wurden die umgebenden Komponenten von EventAlert ermittelt und ein technischer Anspruch formuliert (siehe Punkt 3.2 ab Seite 23).

Anschließend wurden die zentralen Komponenten Ereignisquelle und -senke (3.3 ab Seite 26 und 3.4 ab Seite 30) analysiert. Hierbei wurde speziell auf die Umsetzung mit Flink eingegangen und ein Bezug zu den Szenarien hergestellt.

Des Weiteren wurden Anforderungen an die Steuerung der LED-Anzeige formuliert (siehe Punkt 3.5 ab Seite 32), dabei wurde sowohl auf die Verbindungsmöglichkeiten als auch auf die Programmierbarkeit eingegangen. Entsprechende Anforderungen an Verbindungs- und Programmiermöglichkeiten wurden gestellt.

Zum Schluss wurde auf die Anforderung an Apache Flink eingegangen (siehe 3.6 ab Seite 35).

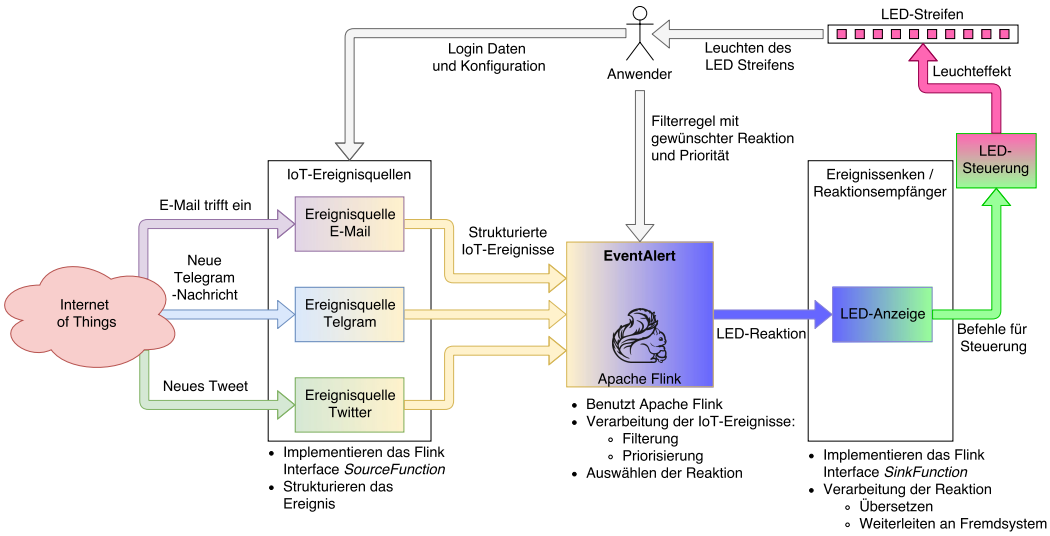


Abbildung 3.3: Überblick über die Komponenten der Anwendung mit ihren Funktionen

Alle in diesem Kapitel aufgestellten Anforderungen sind noch einmal in einer Übersicht in Anhang A ab Seite 96 zusammengefasst.

Außerdem ist – als Ergebnis der Analyse – ein grafischer Überblick über die Komponenten der Anwendung mit ihren Funktionen in Abbildung 3.3 dargestellt. Die Abbildung vereint die während Analyse erstellten Abbildungen 3.1 auf Seite 25 und 3.2 auf Seite 33.

In den folgenden Kapiteln wird anhand der Analyse und den Anforderungen zunächst ein Entwurf der Anwendung erstellt, welcher dann anschließend umgesetzt wird. Außerdem wird ein Test bezüglich der Anforderungen durchgeführt und die gewonnen Erfahrungen werden beschrieben.

4 Entwurf

In diesem Kapitel werden – basierend auf der Analyse der Anforderungen – die Software- und Hardwarekomponenten entworfen, bevor mit der tatsächlichen Umsetzung begonnen wird. Es wird verdeutlicht, wie die Komponenten im Detail funktionieren und welchen Auftrag sie in der Anwendung haben. Außerdem wird beschrieben, wie die Komponenten kommunizieren und zusammenarbeiten.

4.1 Architektur

Die Architektur des Gesamtsystems besteht aus mehreren Soft- und Hardwarekomponenten. In diesem Teil wird vorwiegend die Gesamtarchitektur des Systems behandelt. Einzelne Komponenten mit ihren Innensichten werden in Punkt 4.2 ab Seite 42 erläutert.

Ereignisgesteuerte Architektur Als Architekturmodell wird eine ereignisgesteuerte Architektur (englisch *Event-driven Architecture*) zugrunde gelegt. Das bedeutet, die Komponenten kommunizieren über Ereignisse (Nachrichten) miteinander (Bruns und Dunkel, 2010). Die Ereignisse werden über ein Netzwerk übermittelt und von einem Messaging-System verarbeitet und verteilt. Das Messaging-System bildet gewissermaßen die zentrale Anlaufstelle für die Kommunikation aller Komponenten. Als Messaging-System wird Apache Kafka verwendet, auf die Eigenschaften und Besonderheiten von Kafka wird in den Grundlagen unter Punkt 2.3 eingegangen.

Die Kommunikation über Ereignisse und ein Messaging-System ermöglicht eine Entkopplung der Komponenten, da die einzelnen Komponenten keine direkte Verbindung zueinander haben (Bruns und Dunkel, 2010). Dies wird anhand der Abbildung 4.1 auf der nächsten Seite deutlich, in welcher alle Hauptkomponenten der Anwendung abgebildet sind. In der Abbildung ist zu sehen, dass die Komponenten EventAlert und LED-Anzeige keine Verbindungen untereinander haben. Stattdessen hat jede Komponente mindestens eine Verbindung zum Messaging-System und kommuniziert dort über die Schnittstelle „Ereignis speichern“ und/oder „Ereignis abholen / abrufen“. Über diese Schnittstellen werden Ereignisse empfangen beziehungsweise versendet.

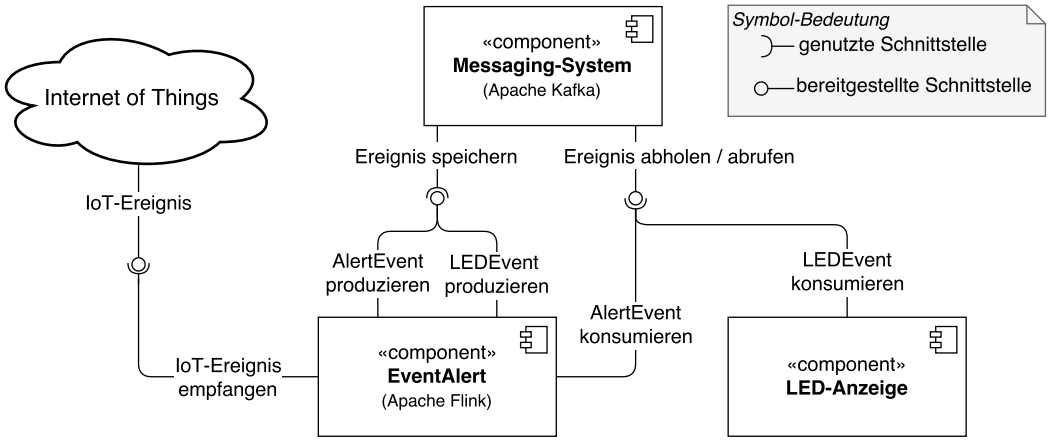


Abbildung 4.1: Komposition der Hauptkomponenten

Die Entkopplung erfüllt die nicht funktionale Anforderung, dass die Komponenten unabhängig voneinander sein sollen (siehe A3, ab Seite 96). Hierdurch wird eine offene und fehlertolerante Zusammenarbeit der Komponenten ermöglicht.

Ereignis-Typen Weiter zeigt die Abbildung 4.1, welche Ereignis-Typen die Komponenten zur Kommunikation verwenden. Eine Komponente kann nur bestimmte Ereignisse-Typen versenden und empfangen. In der Abbildung sind drei Typen von Ereignissen abgebildet:

- Typ IoT (IoT-Ereignis): Nicht genauer spezifizierte Ereignisse aus dem Internet der Dinge (zum Beispiel E-Mail-Ereignis).
- Typ Alert (AlertEvent): Eine Abstraktion der IoT-Ereignisse. Dabei enthält das AlertEvent alle für die Filterung relevanten Informationen des IoT-Ereignisses.
- Typ LED (LEDEvent): Abstrakte Ereignisse, die als Reaktion auf ein AlertEvent generiert werden. Ein LEDEvent enthält dabei alle Parameter, die zur Steuerung der LED-Anzeige benötigt werden.

Das Messaging-System kann – als zentrale Kommunikationsschnittstelle für alle Komponenten – Ereignisse jeden Typs empfangen und versenden. Der Ereignis-Typ für das Messaging-System irrelevant, da es nur zur Weiterleitung und Persistierung der Ereignisse dient und dafür weder Inhalt noch Typ des Ereignis kennen muss.

Anders ist dies hingegen bei den Komponenten EventAlert und LED-Anzeige, diese Komponenten können nur Ereignisse von bestimmten Typen versenden und empfangen. Die Komponente EventAlert kann nur Ereignisse der Typen IoT und AlertEvent empfangen und Er-

eignisse der Typen LEDEvent und AlertEvent versenden. Die Komponente LED-Anzeige kann lediglich Ereignisse vom Typ LEDEvent empfangen und generell keine Ereignisse versenden.

4.2 Komponenten

In diesem Teil wird genauer auf die Architektur und Funktionsweise der Hauptkomponenten eingegangen. Mithilfe der Komponenten-Innensicht (Abbildungen 4.2 auf der nächsten Seite und 4.4 auf Seite 46), werden die Details der jeweiligen Hauptkomponente deutlich. Außerdem zeigt die Innensicht, wofür die Schnittstellen aus der Komposition (Abbildung 4.1 auf der vorherigen Seite) innerhalb der Komponenten verwendet werden.

4.2.1 EventAlert

Die Komponente EventAlert enthält die zentrale Funktionalität der Anwendung. Ihre Aufgabe ist das Empfangen und Konvertieren von IoT-Ereignissen. Zudem übernimmt sie die Filterung der Ereignisse und generiert LEDEvents falls eine Filterregel zutrifft.

In Abbildung 4.2 auf der nächsten Seite ist zu sehen, dass die EventAlert-Komponente aus zwei Unterkomponenten besteht: *AlertEvent-Producer* und *AlertEvent-Consumer*.

Beide Unterkomponenten haben ihren individuellen Aufgabenbereich und arbeiten unabhängig voneinander. Daher könnten sie theoretisch auch Hauptkomponenten sein. Jedoch ist es sinnvoll, beide Komponenten als Unterkomponenten in der Hauptkomponente EventAlert zu vereinen, da EventAlert– in Bezug auf die Architektur des Gesamtsystems – eine Funktionseinheit darstellen soll. Außerdem nutzen beide Unterkomponenten Flink als Basis ihrer Funktionalität und haben somit ähnliche Abhängigkeiten.

AlertEvent-Producer

Die Unterkomponente AlertEvent-Producer nimmt IoT-Ereignisse aus verschiedenen Ereignisquellen entgegen. Jede Quelle benötigt eine eigene Schnittstelle, die nur spezielle IoT-Ereignisse entgegennimmt. In Abbildung 4.2 auf der nächsten Seite sind exemplarisch drei Schnittstellen (E-Mail, Telegram und Twitter) abgebildet. Jede dieser Schnittstellen stellt eine spezialisierte „IoT-Ereignis empfangen“-Schnittstelle aus Abbildung 4.1 auf der vorherigen Seite dar. Die spezialisierten Schnittstellen sind also jeweils für eine bestimmte Art von IoT-Ereignissen zuständig. Für Ereignisse aus Telegram wird zum Beispiel eine andere Schnittstelle benutzt, als für die Twitter-Ereignisse.

Nach der Entgegennahme eines IoT-Ereignisses besteht die weitere Aufgabe dieser Unterkomponente darin, das IoT-Ereignis in ein AlertEvent umzuwandeln und ans Messaging-System

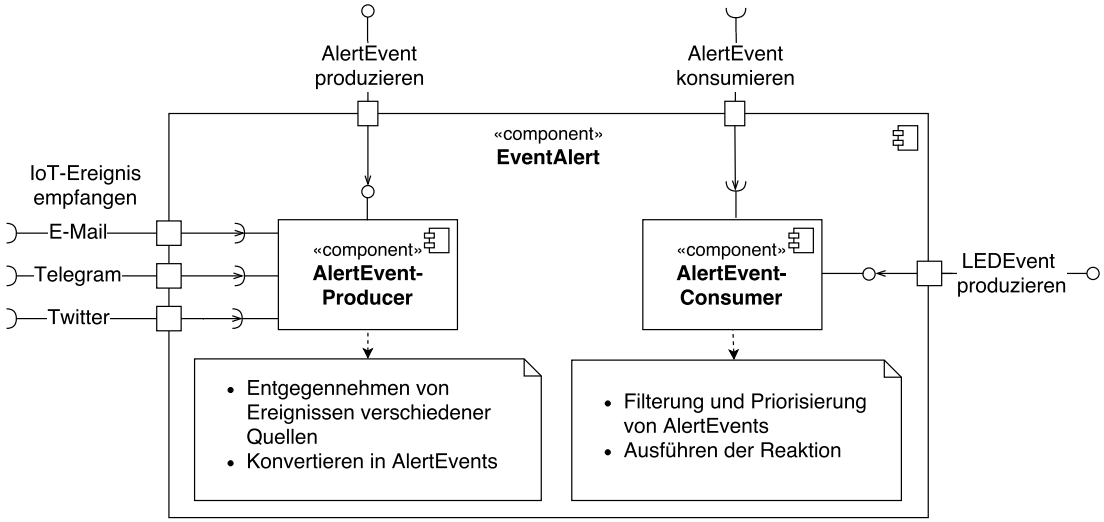


Abbildung 4.2: Innensicht der Komponente EventAlert

zu senden. Wie schon unter Punkt 4.1 erwähnt, ist ein AlertEvent eine Abstraktion eines IoT-Ereignis. Das AlertEvent hat im Gegensatz zum IoT-Ereignis eine vorgegebene Struktur und enthält nur die für die weitere Verarbeitung relevanten Informationen.

Die IoT-Ereignis-Verarbeitung endet also mit dem Produzieren eines AlertEvents, das anschließend über die Schnittstelle „AlertEvent produzieren“ an das Messaging-System übertragen wird.

AlertEvent-Consumer

Der AlertEvent-Consumer ist die andere Unterkomponente von EventAlert. Sie ist für die weitere Verarbeitung der AlertEvents zuständig, welche zuvor vom AlertEvent-Producer produziert wurden. Über die Schnittstelle „AlertEvent konsumieren“ werden die AlertEvents vom AlertEvent-Consumer aus dem Messaging-System entgegengenommen.

Die Komponente benötigt keine Konvertierung der Ereignisse, da diese bereits als strukturierte AlertEvents vorliegen. Der AlertEvent-Consumer kann also umgehend mit der Filterung und Priorisierung der AlertEvent beginnen werden, worin auch die Aufgabe dieser Unterkomponente besteht.

Filterung und Priorisierung Zur Filterung liest der AlertEvent-Consumer zunächst den Typ des AlertEvents aus. „Typ“ bedeutet ist in diesem Zusammenhang die Herkunft des

zugrunde liegenden IoT-Ereignisses. Der Typ definiert zum Beispiel, ob das Ereignis aus Telegram oder aus einem E-Mail-Konto stammt. Anhand des AlertEvent-Typs kann eine Vorauswahl über die Relevanz des AlertEvents getroffen werden, indem überprüft wird, ob der Anwender Filterregeln für diesen Typ definiert hat.

Nach der Vorauswahl erfolgt die genaueren Filterung nach einfachen Bedingungen und eine anschließende Priorisierung. Hierzu enthält die Filterregel enthält neben dem AlertEvent-Typ und der Bedingung drei weitere Informationen:

AlertEvent-Typ Typ des AlertEvent (z.B. E-Mail oder Telegram), für welchen der Filter vorgesehen ist.

Feldname Name des Feldes, auf welches die Bedingung zutreffen kann, zum Beispiel das Feld „Absender“ bei E-Mail-AlertEvents.

Bedingung Bedingung, die zutreffen muss, damit die individuelle Visualisierung ausgeführt wird. Die Bedingung besteht dabei aus einem Bedingungstyp und einem Muster.

Prioritätswert Wert, der die Priorität der Filterregel angibt (je höher der Wert, desto höher die Priorität).

Reaktion Reaktion, die ausgeführt werden soll, falls die Bedingung zutrifft.

Der geplante Ablauf der Filterung und der Priorisierung wird anhand des Aktivitätsdiagramms in Abbildung 4.3 auf der nächsten Seite deutlich. Der Ablauf endet entweder mit der Entscheidung, dass das Ereignis irrelevant für den Anwender ist oder dem Ausführen einer Reaktion. Vorerst soll es nur die LED-Reaktion geben, welche ein LEDEvent über die Schnittstelle „LEDEvent produzieren“ an das Messaging System sendet, sodass es von der LED-Anzeige visualisiert werden kann.

Der Entwurf des Ablaufs der Filterung und Priorisierung erfüllt bereits alle funktionalen Anforderungen an EventAlert (siehe A1 und A2 in Anhang A, ab Seite 96). Diese forderten die Festlegung und von individuellen Reaktionen auf ein Ereignis anhand von Filterregeln. Außerdem wurde auch die Anforderung an die Priorisierung erfüllt.

Mit dem Ausführen der Reaktion endet der Zuständigkeitsbereich dieser Unterkomponente und zugleich auch der Zuständigkeitsbereich der Hauptkomponente EventAlert.

Fazit

Die Hauptkomponente EventAlert besteht aus zwei Unterkomponenten: Der AlertEvent-Producer nimmt IoT-Ereignisse aus verschieden IoT-Quellen entgegen und konvertiert diese zu AlertEvents. Anschließend filtert und priorisiert der AlertEvent-Consumer alle AlertEvents an-

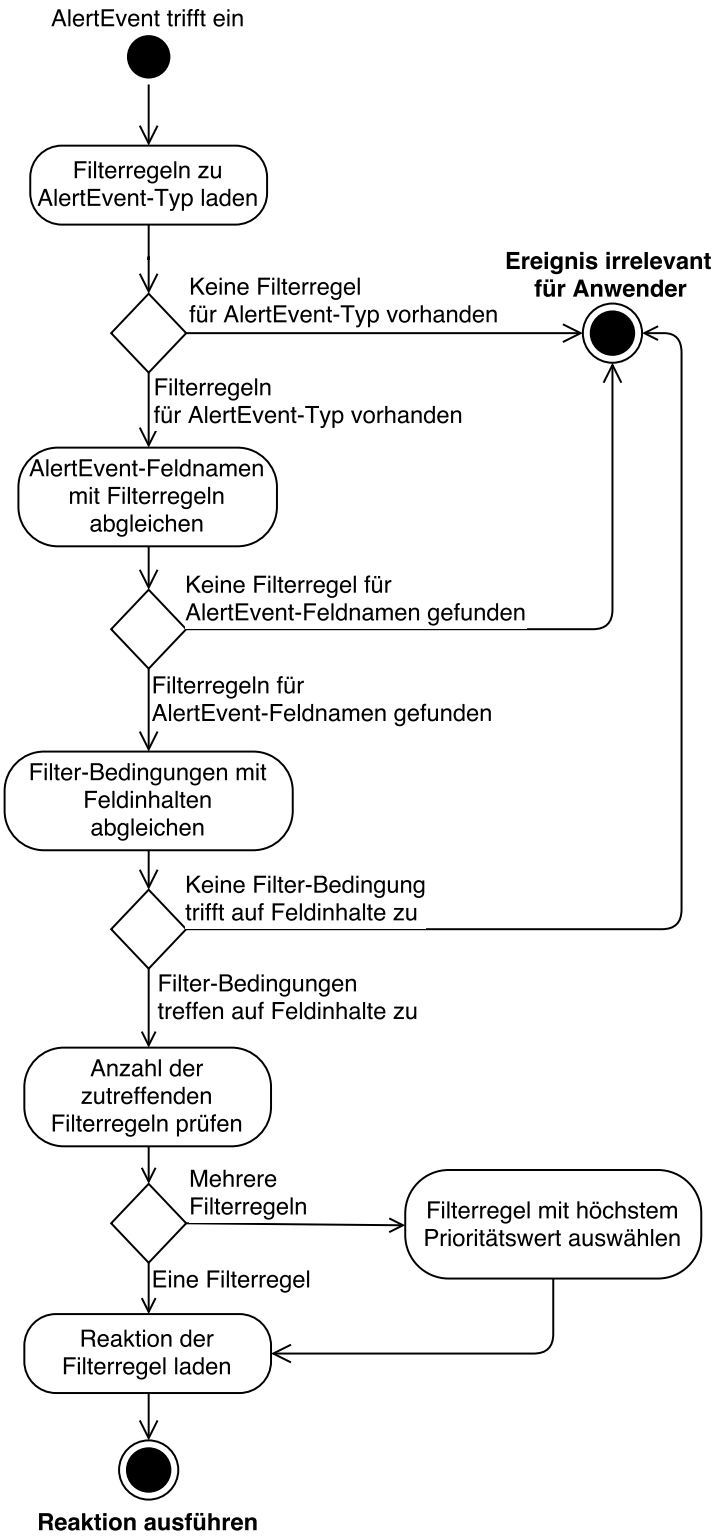


Abbildung 4.3: Aktivitätsdiagramm: Ablauf der Filterung und Priorisierung von AlertEvents

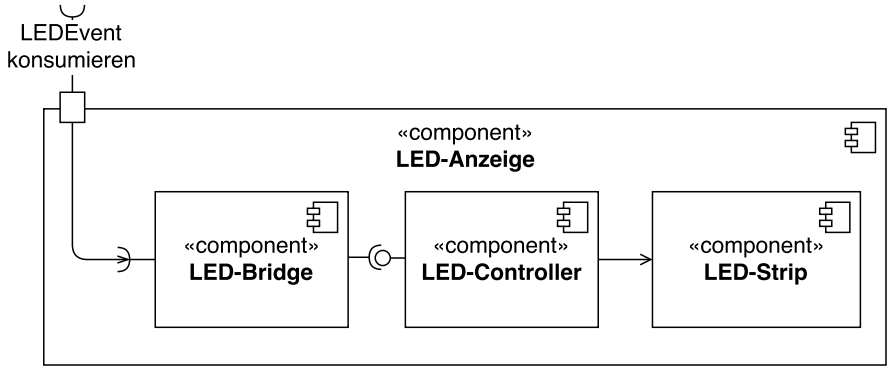


Abbildung 4.4: Innensicht der Komponente LED-Anzeige

hand von Filterregeln, die der Anwender definiert hat, und führt die entsprechende Reaktionen aus. Bei einer LED-Reaktion ist dies das Übergeben eines LEDEvents an das Messaging-System.

Die Architektur dieser Hauptkomponente wirkt möglicherweise komplizierter als sie sein müsste. Theoretisch könnte auf das Produzieren (Übergeben) und Konsumieren (Entgegennehmen) von AlertEvents über eine externe Komponente (Messaging-System) verzichtet werden. Stattdessen könnten der AlertEvent-Producer und AlertEvent-Consumer direkt miteinander verbunden werden. Auf die direkte Verbindung wurde jedoch zugunsten einer „offenen“ Architektur bewusst verzichtet, denn der „Umweg“ über ein externes Messaging-System zur Kommunikation schafft die Möglichkeit, dass weitere Komponenten von den abstrakten AlertEvents profitieren können.

Es können so zum Beispiel neue Komponenten erstellt werden, welche AlertEvents für ihre Zwecke benutzen oder selbst AlertEvents an das Messaging-System übergeben, ohne dass dafür eine Änderung der bestehenden Komponenten notwendig ist.

4.2.2 LED-Anzeige

Die Komponente LED-Anzeige ist für die Visualisierung der Ereignisse zuständig. Ihre Aufgabe ist es die LEDEvents über ihre Schnittstelle „LEDEvent konsumieren“ entgegen zu nehmen, um sie anschließend zu verarbeiten und den LED-Streifen mit dem gewünschten Leuchteffekt zu leuchten zu bringen.

Auch diese Komponente besteht aus mehreren Unterkomponenten. Die Unterkomponenten heißen *LED-Bridge*, *LED-Controller* und *LED-Strip* (siehe Abbildung 4.4). Dabei ist zu beachten, dass die Unterkomponenten LED-Controller (LED-Steuerung) und LED-Strip (LED-Streifen)

in einem Hardwaresystem vereint sind, also eine Einheit bilden. Nachfolgend werden die einzelnen Unterkomponenten und ihre Funktionalität genauer erläutert.

Die Entscheidung für eine Hardware von LED-Controller und LED-Strip bleibt im Entwurf vorerst offen, da die Software-Architektur der LED-Anzeige für jede Form von LED-Steuerung und -Streifen ausgelegt sein soll. Die Hardwareauswahl erfolgt erst im Kapitel Realisierung und Test unter Punkt 5.2 ab Seite 69.

LED-Bridge Die Unterkomponente LED-Bridge ist die erste Anlaufstelle in der LED-Anzeige. Sie verarbeitet die LEDEvents, welche über die Schnittstelle „LEDEvent konsumieren“ entgegengenommen werden. „Verarbeiten“ heißt in diesem Zusammenhang, dass der Leuchteffekt aus dem LEDEvent extrahiert und für die nachfolgende Unterkomponente (LED-Controller) übersetzt wird. Sie bildet sozusagen eine Brücke zwischen Messaging-System und LED-Steuerung (LED-Controller). Diese Brücke ist notwendig, da davon ausgegangen wird, dass die LED-Steuerung nicht direkt auf das Messaging-System zugreifen kann.

LED-Controller Die Unterkomponente LED-Controller wandelt die Leuchteffekte in elektrotechnische Signale um und sendet diese zum LED-Streifen (LED-Strip).

Für die Umwandlung in elektrotechnische Signale stellt der LED-Controller einige vorgefertigte Befehle zu Verfügung, die von der LED-Bridge aufgerufen werden können. Dies hat den Vorteil, dass die LED-Bridge keine Informationen darüber braucht, welcher LED-Strip an dem LED-Controller angeschlossen ist, sondern lediglich die verfügbaren Befehle aufrufen muss. Um die korrekte Übersetzung für den angeschlossenen LED-Strip kümmert sich der LED-Controller selbst.

Ein Beispiel-Befehl, den der LED-Controller zur Verfügung stellen könnte, wäre „Alle LEDs in voller Stärke an; Farbe rot“.

LED-Strip Die Unterkomponente LED-Strip empfängt die elektrotechnischen Signale des LED-Controllers und leuchtet entsprechend. Sie enthält keinerlei programmierbare Logik und ist direkt mit dem LED-Controller verbunden.

Fazit Die Hauptkomponente LED-Anzeige besteht aus drei Unterkomponenten. Die Hauptaufgabe der Unterkomponenten ist das Übersetzen von Befehlen für die jeweils nachfolgende Unterkomponente, sodass aus einem abstrakten LEDEvent ein konkretes elektrotechnisches Signal für den LED-Strip wird. Der Befehl wird mit jedem Übersetzungsschritt konkreter und technischer.

Nachfolgend ist der Datenfluss von einem LEDEvent zu einem konkreten elektrotechnischem Signal kurz zusammen gefasst.

- Das LEDEvent wird von der Unterkomponente LED-Bridge entgegengenommen. Außerdem wird der Leuchteffekt ausgelesen.
- Anhand dieses Leuchteffekts ruft die LED-Bridge den entsprechenden Befehl von der Unterkomponente LED-Controller auf.
- Der LED-Controller übersetzt den aufgerufenen Befehl in elektrotechnische Signale für die Unterkomponente LED-Strip.
- Der LED-Strip leuchtet schlussendlich in der entsprechenden Farbe und Stärke.

Mit der Visualisierung des Ereignisses, dem Leuchten des LED-Streifen, endet auch der Aufgabenbereich der Komponente LED-Anzeige.

4.3 Kommunikation

In diesem Punkt wird die Kommunikation der Komponenten und Unterkomponenten genauer erläutert. Außerdem werden Beispiele für einzelne Nachrichten gezeigt.

Es ist zu beachten, dass im Kontext dieser Arbeit eine Nachricht ein Ereignis darstellt, sodass Nachricht und Ereignis als Synonym verwendet werden können.

Hauptkomponenten

Wie schon unter Punkt 4.1 erwähnt, kommunizieren die Hauptkomponenten (EventAlert und LED-Anzeige) mithilfe eines Messaging-Systems. Über dieses können die Komponenten Nachrichten miteinander austauschen.

Asynchrone Kommunikation Die Hauptkomponenten kommunizieren asynchron miteinander. Das bedeutet, dass eine Komponente ihre Nachricht versendet (Quellkomponente) und nicht auf eine Antwort wartet, bevor sie weitere Nachrichten versendet (Bruns und Dunkel, 2010). Sie wird also nicht von der Verfügbarkeit der Zielkomponente (empfangende Komponente) eingeschränkt. Lediglich das Messaging-System muss den Erhalt der Nachricht bestätigen. Ist die Nachricht erst einmal im Messaging-System angelangt, muss sich die Quellkomponente nicht mit der Weiterverarbeitung der Nachricht befassen – dies erledigt das Messaging-System.

Aufbau eines Ereignisses Ein Ereignis beziehungsweise eine Nachricht soll mindestens alle Informationen enthalten, die für die Zielkomponente von Relevanz sind. Ohne die relevanten

Informationen könnte die Zielkomponente das Ereignis nicht verarbeiten, da ihr Informationen fehlen. Außerdem muss das Datenformat des Ereignisses (Struktur der Daten) bekannt sein, sodass alle Komponenten wissen, wie das Ereignis einlesen beziehungsweise erstellen können.

Hierfür bietet sich die *JavaScript Object Notation* (kurz: JSON) an. JSON hat eine sehr kompakte und leichtgewichtige Struktur und eignet sich für den Austausch von Daten (Ereignissen) zwischen verschiedenen Systemen. Obwohl die Programmiersprache JavaScript im Namen auftaucht, wird JSON von diversen Programmiersprachen und Frameworks unterstützt und ist unabhängig von der Programmiersprachen (Bassett, 2015).

Durch Sprachunabhängigkeit des JSON-Datenformats könnten sogar Programme einer anderen Programmiersprache mit EventAlert kommunizieren, dies gibt erneut mehr Möglichkeiten für eine spätere Weiterentwicklung. Als Beispiel für die JSON-Struktur wird nachfolgend ein exemplarisches Ereignis des Typs AlertEvent gezeigt:

```
1  {
2     "eventType":"email",
3     "eventData":{
4        "imapAccountId":4211,
5        "from":"Erika Muster <erika.muster@example.com>",
6        "to":"Max Muster <max.muster@example.com>",
7        "replyTo":"Erika Muster <erika.muster@example.com>",
8        "cc":null,
9        "bcc":null,
10       "sendTime":1504008109000,
11       "receivedTime":1504008112000,
12       "subject":"Betreffzeile",
13       "content":"Nachrichteninhalt"
14    }
15 }
```

Quellcode 4.1: AlertEvent im JSON Format

Dabei ist zu beachten, dass die Daten unter dem Schlüssel *eventData* (Zeile 4–13 in Quellcode 4.1) variabel gestaltet sind. Die dort stehenden Parameter stammen aus dem ursprünglichen IoT-Ereignis (im Quellcode 4.1 sind dies alle E-Mail-Parameter).

Ein *LEDEvent* ist strukturell gleich aufgebaut wie ein *AlertEvent*, enthält aber Informationen über den Leuchteffekt und nicht über das *AlertEvent*.

Unterkomponenten

Die Unterkomponenten innerhalb von EventAlert kommunizieren genauso wie ihre Hauptkomponente, also über ein Messaging-System mithilfe von asynchronen Nachrichten.

Besonderheit serielle Kommunikation Anders ist dies bei den Unterkomponenten der LED-Anzeige. Dort werden zwar LEDEvents über die Unterkomponente LED-Bridge vom Messaging-System entgegengenommen, doch findet die weitere Kommunikation – zwischen den anderen Unterkomponenten – nicht über das Messaging-System statt.

Die LED-Bridge kommuniziert stattdessen über eine serielle Schnittstelle mit dem LED-Controller. Die Besonderheit bei der seriellen Datenübertragung ist, dass die Nachrichten bitweise übertragen werden (jedes Bit einer Nachricht wird einzeln übertragen). Durch die bitweise Übertragung ist der Datendurchsatz bei serieller Übertragung relativ gering. Aufgrund dieser Einschränkung sind die Nachrichten möglichst kurz zu halten. Deswegen wird bei der Seriellen-Kommunikation auf ein Datenformat wie JSON verzichtet. Es werden lediglich einzeilige, URI-ähnliche Nachrichten übertragen. Nachfolgend ein Beispiel für eine Nachricht, die seriell übertragen wird.

```
colr/255,0,0,0/?id=123
```

Der Beginn der Nachricht `colr/` gibt den gewünschten Befehl an. Danach folgen die Parameter für diesen Befehl, im Beispiel sind das die Farbwerte für Rot, Grün, Blau und Weiß (`255,0,0,0/`). Am Schluss wird noch eine Identifikationsnummer (ID) mitgegeben (`?id=123`), damit die Nachricht eindeutig identifiziert werden kann. Anhand der ID kann der LED-Controller den Empfang der Nachricht bestätigen. Dies bedeutet allerdings nicht, dass die LED-Bridge zwingend auf die Empfangsbestätigung durch den LED-Controller warten muss.

4.4 Fazit

Diese Kapitel hat sich mit der Entwurf und der Vorplanung dieser Anwendung befasst. Zunächst wurde ein Architekturmodell ausgewählt, nach welchem eine Komposition der Hauptkomponenten (Abbildung 4.1 auf Seite 41) erstellt wurde. Außerdem wurden die verschiedenen Ereignis-Typen im Kontext der Anwendung erläutert.

Auf Basis dieser Komposition wurden die Hauptkomponenten EventAlert und LED-Anzeige entworfen. Hierzu wurde jeweils eine Innensicht für beide Hauptkomponenten erstellt (Abbildungen 4.2 auf Seite 43 und 4.4 auf Seite 46). Die Innensicht zeigt die Unterkomponenten der jeweiligen Hauptkomponente.

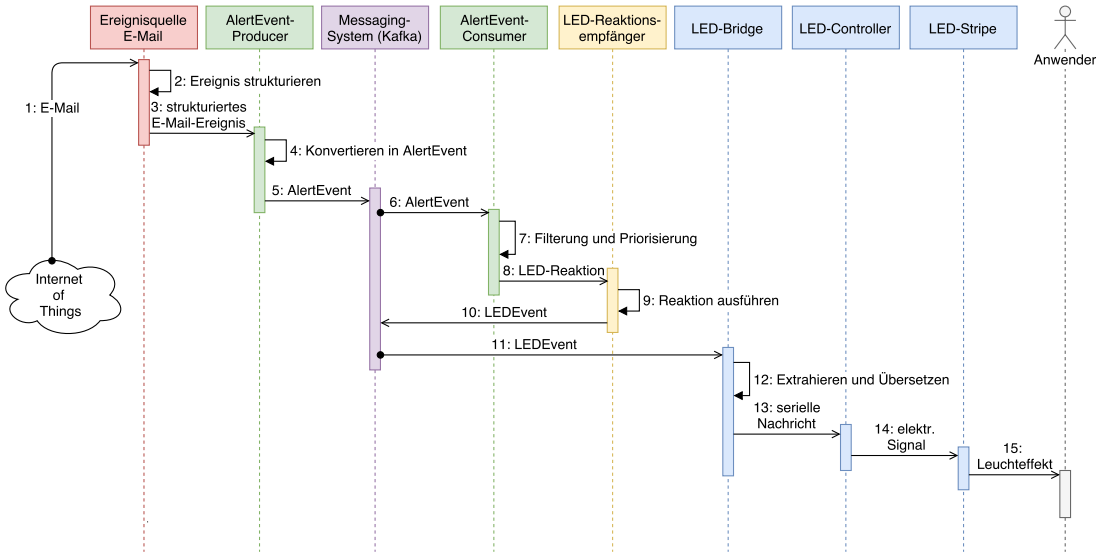


Abbildung 4.5: Sequenzdiagramm: Exemplarischer Ablauf einer Ereignis-Verarbeitung

Des weiteren wurde der Aufgabenbereich der Unterkomponenten abgesteckt und die geplante Funktionsweise erläutert. Für die Unterkomponente AlertEvent-Consumer wurde außerdem ein Aktivitätsdiagramm (Abbildung 4.3 auf Seite 45) erstellt, welches den Ablauf der Filterung und Priorisierung präzisiert.

Zum Abschluss des Kapitels wurde auf die Details der Kommunikation der Haupt- und Unterkomponenten eingegangen. Es wurde beispielhaft der Aufbau eines AlertEvents gezeigt (Quellcode 4.1 auf Seite 49) und auf die Besonderheiten der seriellen Kommunikation eingegangen.

Um die Ergebnisse des Entwurfs übersichtlich und zusammenfassend festzuhalten wurde ein Sequenzdiagramm erstellt (Abbildung 4.5), welches den exemplarischen Ablauf einer vollständigen Ereignisverarbeitung zeigt. Außerdem zeigt die Abbildung 4.6 auf der nächsten Seite die geplante Pakethierarchie der Anwendung.

Auf Grundlage der Diagramme, Abbildungen und Funktionsbeschreibungen wird im nächsten Kapitel die Anwendung realisiert beziehungsweise entwickelt.

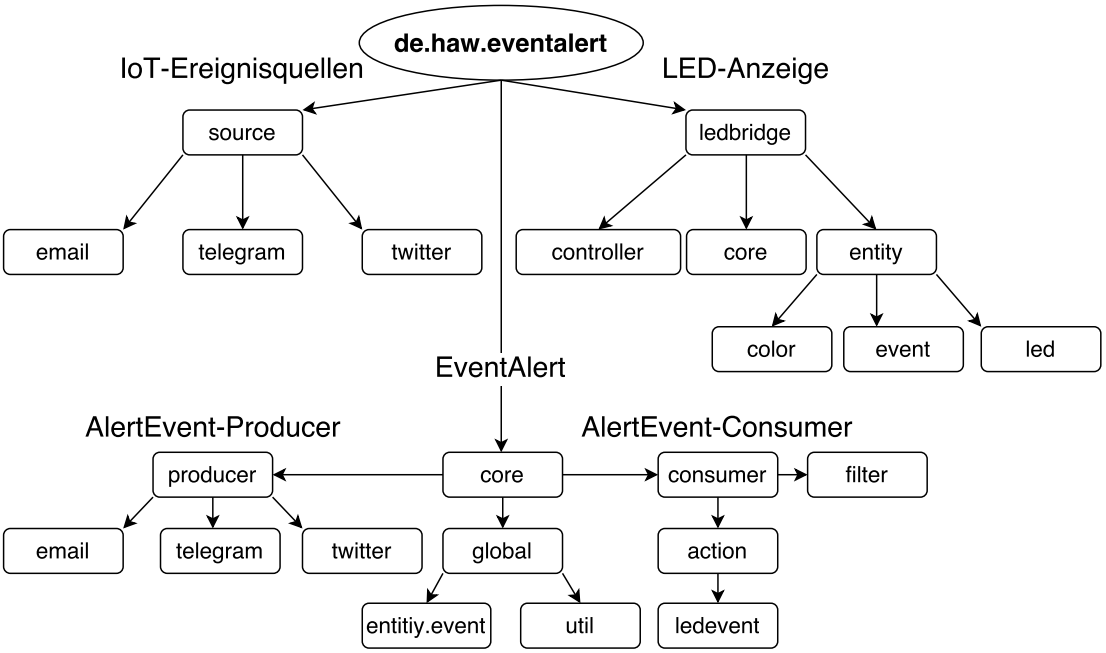


Abbildung 4.6: Pakethierarchie der entwickelten Komponenten

5 Realisierung und Test

In diesem Kapitel wird anhand des Entwurfs (Kapitel 4) die Realisierung des Projekts und insbesondere der Software EventAlert behandelt. Dabei wird versucht, die Anforderungen aus Kapitel 3 zu erfüllen.

Die Realisierung ist in die Bereiche softwaretechnische und hardwaretechnische Umsetzung unterteilt. Die softwaretechnische Umsetzung behandelt die Entwicklung der beiden Hauptkomponenten EventAlert und LED-Anzeige. Für jede dieser Komponenten wird zudem auf das Testen der Komponenten eingegangen. Die hardwaretechnische Umsetzung beschäftigt sich mit der Auswahl von geeigneter Hardware, die für die LED-Anzeige benötigt wird.

5.1 Softwaretechnische Umsetzung

Die softwaretechnische Umsetzung orientiert sich an dem Entwurf (Kapitel 4), da dieser als Entwicklungsplan für die Programmierung dient. Durch den Entwurf wird die Komposition der Komponenten und deren Kommunikation vorgegeben. Des weiteren gibt der Entwurf den Ablauf der Anwendung und die grobe Funktionsweise der Komponenten vor.

Auf Basis dieser Vorgaben gilt es nun die Hauptkomponenten mit ihren Unterkomponenten zu entwickeln oder – falls es sich um fertige Komponenten handelt – zu integrieren, wie es bei dem Messaging-System der Fall ist. Das Messaging-System muss mit den Hauptkomponenten (EventAlert und LED-Anzeige) verbunden werden, damit eine Kommunikation zwischen den Komponenten stattfinden kann.

Innerhalb der Hauptkomponenten muss die Kommunikation zwischen den Unterkomponenten hergestellt werden. Besondere Aufmerksamkeit ist der LED-Anzeige zu widmen, da deren Unterkomponenten nicht über das einheitliche Messaging-System kommunizieren (siehe 4.3 auf Seite 50).

5.1.1 EventAlert

Die Hauptkomponente EventAlert besteht aus zwei Unterkomponenten (siehe Abbildung 4.2 auf Seite 43).

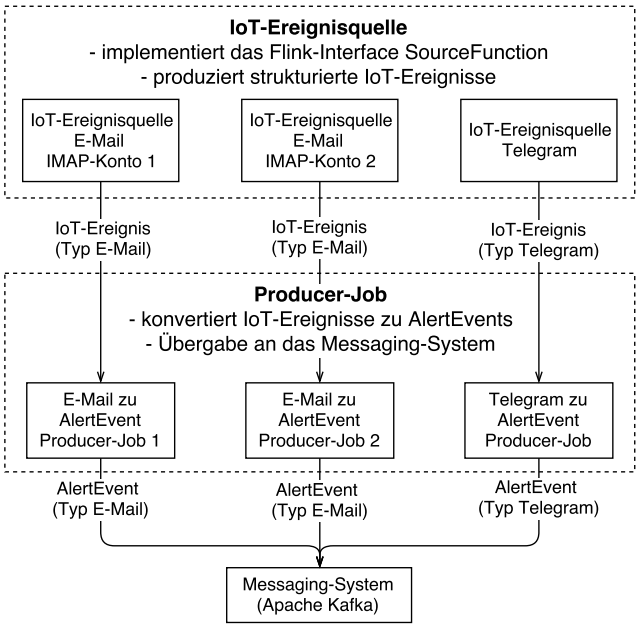


Abbildung 5.1: Ablauf und Funktionsweise des AlertEvent-Producers anhand exemplarischer IoT-Ereignisquellen

Die Unterkomponente AlertEvent-Producer erhält Ereignisse aus unterschiedlichen IoT-Ereignisquellen und konvertiert diese zu AlertEvents. Die Unterkomponente AlertEvent-Consumer nimmt diese produzierten AlertEvents entgegen, filtert und priorisiert sie anhand von Filterregeln des Anwenders und führt eine Reaktion aus, falls passende Filterregeln gefunden wurden. Nachfolgend wird die Realisierung der beiden Unterkomponenten erläutert.

AlertEvent-Producer

Der AlertEvent-Producer besteht aus verschiedenen, heterogenen IoT-Ereignisquellen. Eine IoT-Ereignisquelle nimmt jeweils nur einen bestimmten Typ von IoT-Ereignis entgegen und stellt dieses für den AlertEvent-Producer zur Verfügung.

Die E-Mail-Ereignisquelle stellt beispielsweise Ereignisse zur Verfügung, wenn eine neue E-Mail auf dem IMAP-Konto eingeht. Es kann mehrere IoT-Ereignisquellen geben, die denselben Typ von IoT-Ereignis zur Verfügung stellen. Zum Beispiel gibt es für jedes IMAP-Konto eine eigene E-Mail-Ereignisquelle, die jeweils nur die Ereignisse dieses Kontos zur Verfügung stellt.

In Abbildung 5.1 ist der Ablauf und die Funktionsweise des AlertEvent-Producers grafisch dargestellt, dabei sind die dort aufgeführten IoT-Ereignisquellen nur exemplarisch. Die einzelnen Bestandteile der Abbildung werden nachfolgend näher erläutert.

IoT-Ereignisquelle Die IoT-Ereignisquelle implementiert in jedem Fall das Flink-Interface SourceFunction (siehe Punk 3.3 auf Seite 28), über welches die Ereignisse für den AlertEvent-Producer zur Verfügung gestellt werden. Darüber hinaus kann jede IoT-Ereignisquelle beliebig viele weitere Klassen (Programmteile) oder Module enthalten, die für das Empfangen von Daten aus einem Fremdsystem notwendig sind.

Beispielsweise hat die E-Mail-Ereignisquelle neben der Klasse *EMailSource*, welche das Interface SourceFunction implementiert, noch weitere Klassen (siehe 5.2 auf der nächsten Seite). Zum Beispiel stellt die Klasse *EMailImapClient* eine Verbindung zu einem IMAP-Konto her und hält diese aufrecht. Sobald eine E-Mail eintrifft, wird die EMailSource informiert.

Producer-Job Nach dem Eingang eines IoT-Ereignisses in der IoT-Ereignisquelle soll die Konvertierung in ein *AlertEvent* stattfinden. Hierfür hat jede IoT-Ereignisquelle einen eigenen „Job“. Da es hier um das Produzieren von AlertEvent geht, wird dieser Job „Producer-Job“ genannt.

Ein Producer-Job ist gewissermaßen ein dauerhafter Auftrag, welcher vorgibt, wie mit den Ereignissen aus einer IoT-Ereignisquelle umgegangen wird. Er beginnt immer mit dem Einbinden einer Ereignisquelle und endet mit der Übergabe an eine Ereignissenke, in diesem Fall ist die Senke das Messaging-System.

Der Auftrag des Producer-Jobs ist das Entgegennehmen von IoT-Ereignissen aus der jeweiligen Ereignisquelle. Anschießend konvertiert der Job das Ereignis in ein AlertEvent und stellt dieses an der Schnittstelle „AlertEvent produzieren“ (siehe Abbildung 4.2 auf Seite 43) zur Verfügung, sodass es an das Messaging-System übergeben werden kann.

Der Producer-Job hat mehrere Abhängigkeiten zu externen und internen Klassen, die je nach IoT-Ereignisquelle variieren. Das Klassendiagramm in Abbildung 5.2 auf der nächsten Seite zeigt diese Abhängigkeiten exemplarisch für eine E-Mail-Ereignisquelle.

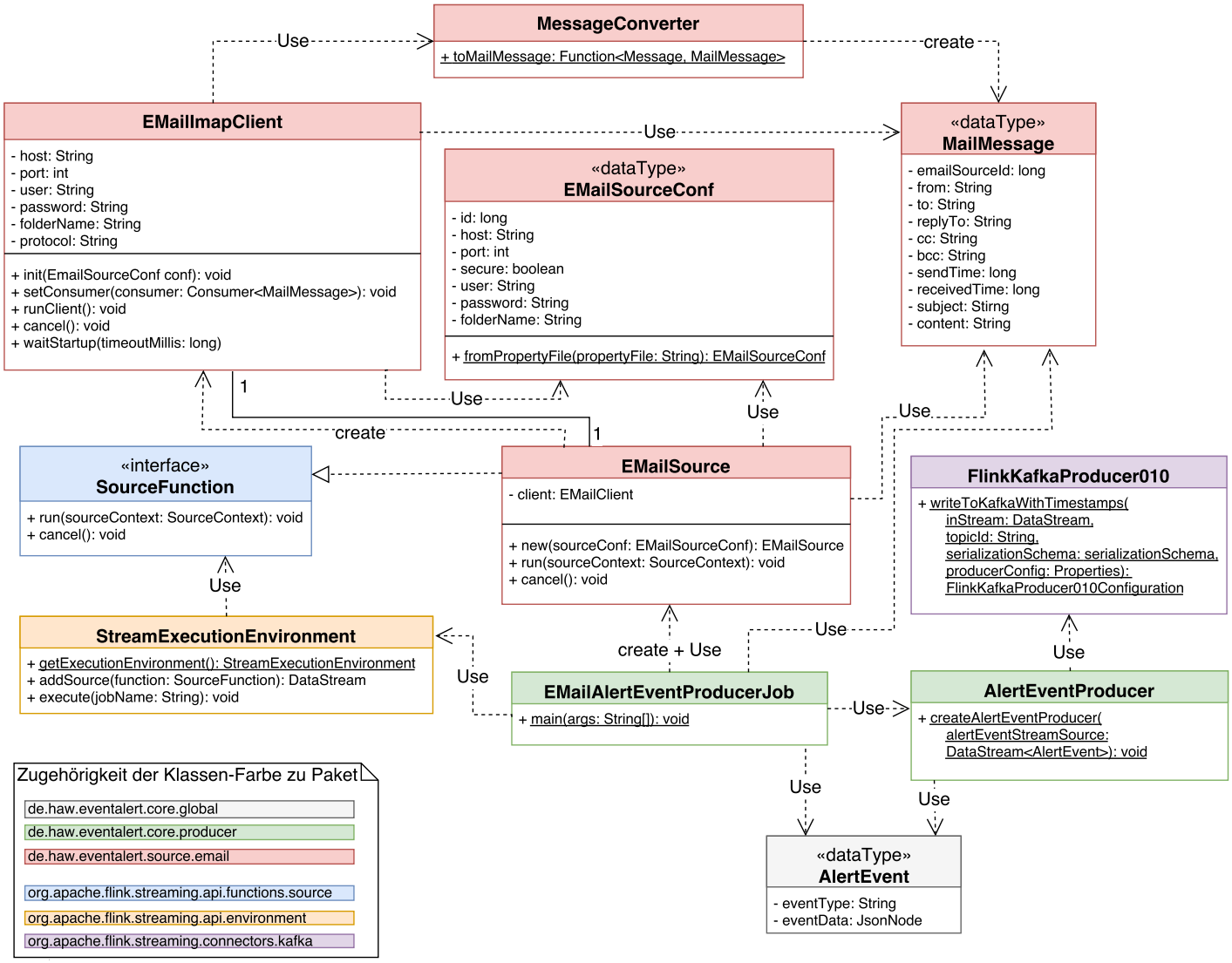


Abbildung 5.2: Klassendiagramm: Exemplarische Abhängigkeiten eines Producer-Jobs mit einer E-Mail-Ereignisquelle

Konvertierung zum AlertEvent Die Konvertierung zu einem AlertEvent muss einmalig für jeden Typ von IoT-Ereignis implementiert werden. Eine individuelle Konvertierung pro IoT-Ereignis-Typ ist notwendig, weil sich die Struktur der Typen teilweise stark unterscheidet, sodass eine einheitliche Konvertierung nicht möglich ist.

Allerdings gibt es für Ereignisquellen, die einfache Datenobjekte produzieren, eine einheitliche Konvertierungsmöglichkeit. Objekte, die nur Daten erhalten, können über eine zentrale Funktion in ein AlertEvent konvertiert werden. Die Funktion wandelt das Objekt lediglich in JSON um und fügt den ursprünglichen Ereignis-Typ hinzu.

Senden an das Messaging-System Das Senden der generierten AlertEvents findet über einen Kafka-Produzenten statt. Der Kafka-Produzent sorgt dafür, dass die Ereignisse an Kafka (das Messaging-System) übertragen werden. Er ist als Flink-Kafka-Connector (siehe Punkt 2.2 auf Seite 11) vorhanden und muss lediglich eingebunden werden.

Das Senden über einen Kafka-Producer ist – im Gegensatz zur Konvertierung – für alle Producer Jobs gleich und standardisiert. Hierfür muss im Job lediglich die Funktion *createAlertEventProducer* aufgerufen werden (siehe 5.2 auf der vorherigen Seite).

Die Funktion *createAlertEventProducer* ist zentral definiert und kann in jedem Job aufgerufen werden. Sie abstrahiert das Senden von AlertEvents an das Messaging-System. Durch den Aufruf dieser Funktion wird ein Kafka-Produzent mit dem richtigen Thema (topic) erstellt, welcher automatisch die Verbindung zum Messaging-System (zum Kafka-Broker) aufbaut und jedes AlertEvent des Datenstroms an Kafka sendet.

Der Vorteil einer zentralen Funktion zum Senden an Kafka liegt darin, dass der Producer-Job an Komplexität verliert. Der Entwickler muss sich keine Gedanken über das Erstellen des Kafka-Produzenten mit der richtigen Konfiguration machen, sodass er sich auf das Wesentliche konzentrieren kann.

AlertEvent-Consumer

Der AlertEvent-Consumer nimmt die AlertEvents entgegen, die vom AlertEvent-Producer produziert wurden. Seine Aufgabe ist es, alle AlertEvents mit den Filterregeln des Anwenders abzugleichen. Trifft ein Filter auf ein Ereignis zu, wird eine, vom Anwender definierte, Reaktion zu dieser Filterregel ausgeführt. Der Anwender kann momentan nur LED-Reaktionen definieren. Eine LED Reaktion übergibt ein individuelles LEDEvent an das Messaging-System, sodass es später von der LED-Anzeige ausgelesen werden kann.

Nachfolgend werden die Bestandteile und die Funktionsweisen dieser Unterkomponente genauer erläutert.

Empfangen aus dem Messaging-System Das Empfangen der AlertEvents findet über einen Kafka-Consumer statt. Dieser ist als Connector in Flink (siehe 2.2 auf Seite 11) enthalten und muss lediglich eingebunden werden.

Der Kafka-Consumer, der die AlertEvents empfängt, wird über die zentrale Funktion (*createAlertEventConsumer*) erstellt. Diese Funktion vereinheitlicht und abstrahiert das Erstellen eines Konsumenten. Sie benötigt keine Argumente, da hier nur ein lesender Zugriff stattfindet und die AlertEvents immer unter dem selben Thema (topic) im Messaging-System vorhanden sind. Die Funktion gibt eine Ereignisquelle zurück, welche das Flink-Interface SourceFunction implementiert.

Consumer-Job Der AlertEvent-Consumer benötigt nur einen Job, im Gegensatz zum AlertEvent-Producer. Der Job wird hier „Consumer-Job“ genannt, da mit diesem Ereignisse aus dem Messaging-System empfangen werden. Der Consumer-Job beginnt mit dem Erstellen des Kafka-Konsumenten über die oben genannte Funktion. Darauf folgt das Einbinden der Ereignisquelle über die Flink-Methode *addSource*. Nach diesen beiden Schritten folgt das Filtern und Priorisieren der Ereignisse, welches nachfolgend im Absatz *Filterung und Priorisierung* beschrieben wird.

Eine Konvertierung der Ereignisse ist nicht notwendig, da diese schon vom Typ AlertEvent sind. Die Ereignisse werden allerdings von JSON-Zeichenketten wieder in Java-Objekten umgewandelt, sodass elegant auf die Datentyp-Funktionen des AlertEvents zugegriffen werden kann (siehe 4.3 auf Seite 48).

Auch der Consumer-Job hat mehrere Abhängigkeiten, welche durch ein Klassendiagramm in Abbildung 5.3 auf der nächsten Seite dargestellt werden. Dabei ist zu beachten, dass eine Reaktion im Diagramm als „Action“ bezeichnet ist. Die LED-Reaktion (*LEDEventAction*) ist exemplarisch für jegliche Form von Reaktion.

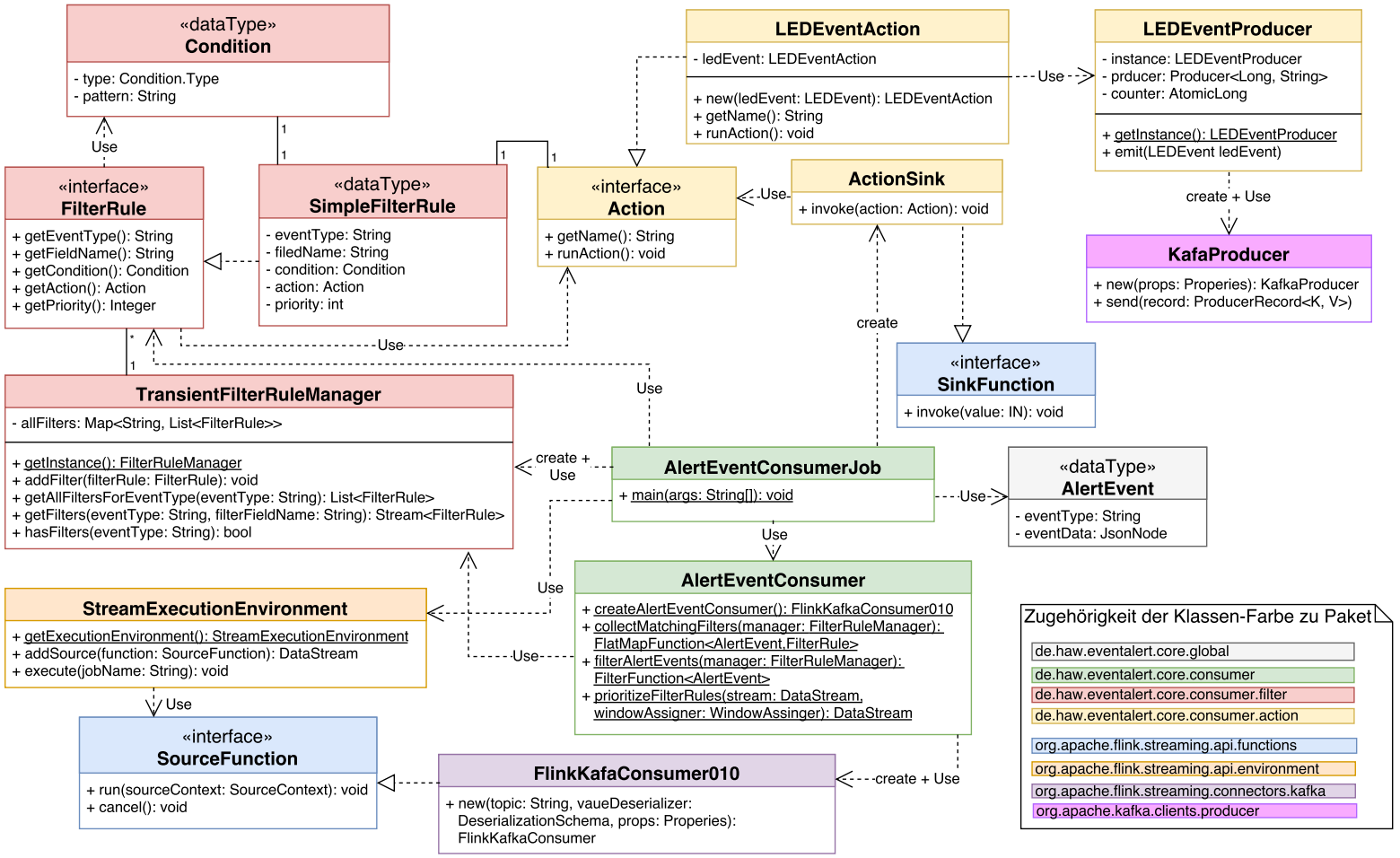


Abbildung 5.3: Klassendiagramm: Abhängigkeiten des Consumer-Jobs

Bedingungstyp	Interner Name	Bedingung
enthält	contains	Feldinhalt enthält Muster
startet mit	startWith	Feldinhalt beginnt mit Muster
endet mit	endWith	Feldinhalt endet mit Muster
gleich	equals	Feldinhalt entspricht Muster
größer als	greaterThan	Zahl aus Feldinhalt ist größer als Muster-Zahl
kleiner als	lessThan	Zahl aus Feldinhalt ist kleiner als Muster-Zahl
Regulärer Ausdruck	regex	Feldinhalt trifft auf reg. Ausdr. im Muster zu

Tabelle 5.1: Umgesetzte Bedingungstypen mit ihren Bedingungen

Filterung und Priorisierung Die Filterung der AlertEvents erfolgt anhand von einfachen Bedingungen. Zur Priorisierung der Filterregeln, kann der Anwender für jede Filterregel einen Prioritätswert festlegen. Der Ablauf der Filterung und Priorisierung wurde bereits über ein Aktivitätsdiagramm (siehe Abbildung 4.3 auf Seite 45) im Entwurf formuliert und wurde dementsprechend umgesetzt.

Wie im Entwurf (siehe Punkt 4.2.1 ab Seite 43) schon erwähnt, besteht eine Filter-Regel unter anderem aus einer Bedingung. Die Bedingung besteht wiederum aus einem Bedingungstyp und einem Muster, welches auf das Feld zutreffen muss. Das Muster kann aus einer Zeichenkette, einer Zahl oder einem regulären Ausdruck bestehen. Während der Realisierung wurden sieben verschiedene Bedingungstypen umgesetzt, welche der Tabelle 5.1 entnommen werden können.

Die Filterung und Priorisierung wurde so umgesetzt, dass der Consumer-Job lediglich auf den *FilterRuleManager* (siehe auch in Abbildung 5.3 auf der vorherigen Seite) zugreifen muss, welcher als Schnittstelle zu den Filterregeln dient.

Der FilterRuleManager verwaltet die Filterregeln des Anwenders. Über ihn kann anhand von Ereignis-Typ und dem zu überprüfenden Feldnamen effizient auf vorhandene Filterregeln zugegriffen werden. Dadurch wird eine kostspieliges Durchsuchen der Gesamtmenge aller Filterregeln vermieden.

Test von EventAlert

Um sicherzustellen, dass EventAlert nach Änderungen korrekt ausgeführt werden kann, wurden Tests für die zentralen Klassen geschrieben. Dafür wurde das Framework *JUnit* verwendet.

Die Tests decken dabei vor allem das Konvertieren von Ereignissen ab (zum Beispiel „Alert-Event zu JSON“), da Fehler in diesem Bereich fatal für die Kommunikation zwischen den Hauptkomponenten wären.

Außerdem wird die Filterung und Priorisierung anhand von Test-Filterregeln getestet. Dafür wird auch die Funktion der *ActionSink* überprüft, indem eine Test-Reaktion ausgeführt und dessen Ergebnis überprüft wird.

Die Ereignisquellen werden jeweils separat getestet. Dort ist vor allem wichtig, dass die Ereignisse richtig strukturiert werden und dabei keine Parameter verloren gehen. Dafür werden je Quelle individuelle Testereignisse in strukturierter und unstrukturierter Form erstellt oder während des Tests generiert, sodass die Strukturierung durch die Quelle getestet werden kann. Dies war allerdings nicht für jede Quelle möglich, da bei manchen Quellen kein direkter Zugriff auf die unstrukturierten Ereignisse besteht (zum Beispiel bei Telegram).

5.1.2 Überprüfung der Anforderungen an EventAlert

In diesem Abschnitt soll überprüft werden, ob alle nicht funktionalen Anforderungen an EventAlert erfüllt wurden. Die nicht funktionale Anforderung an EventAlert zielen auf die Erweiterbarkeit von EventAlert ab.

Die beiden funktionalen Anforderungen an die Filterung und Priorisierung (A1 und A2) wurden bereits im Entwurf durch das Aktivitätsdiagramm (Abbildung 4.3 auf Seite 45) erfüllt (siehe 4.2.1 auf Seite 44).

Außerdem erfüllt der Entwurf auch die Anforderung A3 (siehe 4.1 auf Seite 41), welche gefordert hat, dass die Komponenten von EventAlert keine Abhängigkeiten zueinander haben.

Nachfolgend werden die übrigen Anforderungen an EventAlert (A4 bis A6) überprüft.

Anforderung an die Dokumentation Um das Erweitern und Weiterentwickeln von EventAlert zu erleichtern fordert die Anforderung A4, dass alle Schnittstellen die zum Hinzufügen einer neuen Ereignisquelle benötigt werden, dokumentiert sind (siehe A4, ab Seite 96). Mit „Dokumentation“ ist in diesem Kontext das Javadoc gemeint. Das Javadoc wird anhand von Kommentaren über Methoden und Klassen definiert. Später lässt sich aus den Kommentaren eine HTML Dokumentation generieren.

Während der Entwicklung wurde darauf geachtet, dass alle wichtigen öffentlichen Klassen und Methoden dokumentiert sind. Die Dokumentation enthält dabei eine kurze Beschreibung der Funktionsweise oder Aufgabe der Klasse beziehungsweise Methode. Des Weiteren werden bei Methoden Ein- und Ausgabeparameter, sowie mögliche Fehlerzustände beschrieben.

Die Anforderung A4 konnte also erfüllt werden.

Anforderung an das Hinzufügen von neuen Ereignisquellen Die Anforderung A5 fordert, dass das Hinzufügen einer neuen Ereignisquelle zu EventAlert einen Zeitaufwand von

weniger als zwei Stunden beträgt (siehe A5, ab Seite 96). Dabei gibt es die Einschränkung, dass die Ereignisquelle nicht komplex sein darf. Mit nicht komplexen Ereignisquellen sind Ereignisquellen gemeint, welche POJOs (einfache Java-Objekte ohne Abhängigkeiten) als Ereignis produzieren.

Bei der Realisierung von EventAlert wurde darauf geachtet, dass Anforderung A5 eingehalten werden kann. Zum Hinzufügen einer Ereignisquelle muss lediglich ein neuer Producer-Job erstellt werden, der die Ereignisse aus der neuen Quelle zu AlertEvents konvertiert. Zur Konvertierung stellt EventAlert zentrale, statische Methoden bereit, die jedes POJO zu einem AlertEvent konvertieren. Anschließend muss der Datenstrom von AlertEvents nur noch an EventAlert übergeben werden. Auch dafür ist eine zentrale, statische Methode von EventAlert bereitgestellt.

Um zu demonstrieren, dass der Zeitaufwand weniger als zwei Stunden beträgt, wird nachfolgend der Quellcode (siehe Quellcode 5.1) eines Beispiel-Producer-Jobs gezeigt, der die oben genannten Schritte durchführt.

```
1  //get the execution environment
2  final StreamExecutionEnvironment env = StreamExecutionEnvironment.
       getExecutionEnvironment().setParallelism(1);
3  //create the desired source (must implement the SourceFunction<>
       interface)
4  MyEventSource eventSource = new MyEventSource();
5  //add source to environment (MyEvent has to be a simple POJO)
6  DataStream<MyEvent> myEventStream = env.addSource(eventSource);
7  //convert myEvents to alertEvents
8  DataStream<AlertEvent> alertEventStream = myEventStream.flatMap((
       myEvent, out) -> {
9          try {
10                 out.collect(AlertEvents.createEvent("myEventTypeName
                       ", myEvent));
11         } catch (Exception e) {
12                 //Error logging if needed
13         }
14 });
15 //provide alertEventStream to EventAlert
16 EventAlertProducer.provideDataStream(alertEventStream);
17 //execute the job
18 env.execute("MyAlertEventProducer");
```

Quellcode 5.1: Beispiel-*Producer Job*: Hinzufügen einer Ereignisquelle zu EventAlert

Es wird davon ausgegangen, dass der Aufwand, einen ähnlichen Producer-Job zu erstellen, weniger als zwei Stunden beträgt. Somit kann die nicht funktionale Anforderung A5 an EventAlert als erfüllt angesehen werden.

Anforderung an das Hinzufügen von neuen Reaktionsempfängern Auch neue Reaktionsempfänger sollen mit einem Zeitaufwand von weniger als zwei Stunden hinzugefügt werden können (siehe A6, ab Seite 96). Auch hier gilt wieder die Einschränkung, dass die Anforderung nur für nicht komplexe Reaktionsempfänger gilt. In diesem Kontext heißt nicht komplex, dass die Reaktion nur ein Fremdsystem ansteuert (zum Beispiel Aufruf einer URL oder Produzieren eines neuen Ereignisses). Die Reaktion soll also keine komplexe Logik innerhalb von EventAlert haben.

Dabei ist zu beachten, dass der Reaktionsempfänger durch die Reaktion selbst angesteuert wird. Zum Beispiel wird bei einer LED-Reaktion ein LEDEvent an das Messaging-System gesendet, welches durch die LED-Anzeige empfangen wird. Die LED-Anzeige ist in dem Beispiel der Reaktionsempfänger. Daher muss nur eine neue Reaktion hinzugefügt werden, die einen Reaktionsempfänger beziehungsweise ein Fremdsystem ansteuert.

Das Hinzufügen einer neuen Reaktion wurde dabei möglichst einfach gestaltet, sodass es in wenigen Sätzen erklärt werden kann: Die Reaktion muss lediglich das Interface *Action* implementieren. Das Interface stellt sicher, dass die Methoden *getName* und *runAction* implementiert werden. Die Methode *runAction* wird von EventAlert ausgeführt, sobald eine Filterregel zutrifft und diese die neue Reaktion enthält. In dieser Methode kann zum Beispiel ein Fremdsystem (ein Reaktionsempfänger) angesteuert werden.

Da eine Reaktion nur zwei Methoden benötigt, wird auch hier wird davon Ausgegangen, dass der Entwicklungsaufwand weniger als zwei Stunden beträgt. Somit kann auch Anforderung A6 als erfüllt angesehen werden.

5.1.3 LED-Anzeige

Die Hauptkomponente LED-Anzeige empfängt, verarbeitet und visualisiert die LEDEvents, welche durch LED-Reaktionen produziert werden. Das Ziel der LED-Anzeige ist die Visualisierung von LEDEvents über einen LED-Streifen (LED-Strip).

Die LEDEvents wurden zuvor vom Anwender über die LED-Reaktion definiert und enthalten alle Parameter, die zur Visualisierung notwendig sind. Sie werden versendet, wenn ein AlertEvent auf einen oder mehrere Filterregeln zutrifft.

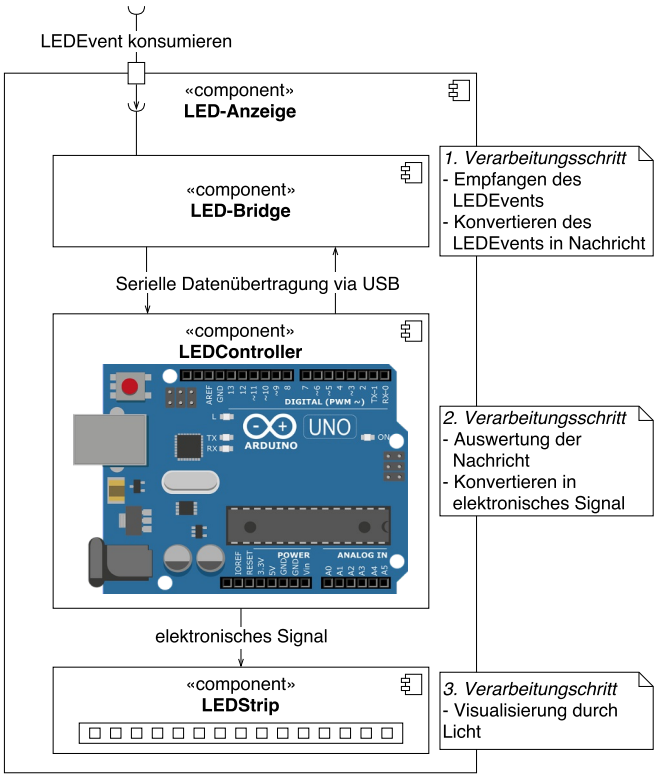


Abbildung 5.4: Verarbeitungskette der LED-Anzeige

Die LED-Anzeige besteht aus drei Unterkomponenten (siehe 4.2.2 auf Seite 46). Jede Unterkomponente stellt einen Schritt in der Verarbeitung vom LEDEvent zur Visualisierung dar. In Abbildung 5.4 ist eine Übersicht der Verarbeitungsschritte jeder Unterkomponenten grafisch dargestellt. Diese Abbildung konkretisiert die Abbildung 4.4 aus dem Entwurf.

Nachfolgend wird auf die Unterkomponenten LED-Bridge und LED-Controller mit ihren speziellen Aufgaben in der Verarbeitungskette eingegangen. Außerdem werden die Funktionsweisen und die Besonderheiten in der Programmierung erläutert. Der Fokus liegt hier bei der Unterkomponente LED-Bridge, da sie die umfangreichste Verarbeitung der LEDEvents vornimmt.

Die Komponente LED-Strip wird hier nicht näher erläutert, da sie keine programmiertechnische Arbeit erfordert hat. Informationen zur Auswahl und Ansteuerung des LED-Streifens sind unter Punkt 5.2 ab Seite 69 aufgeführt.

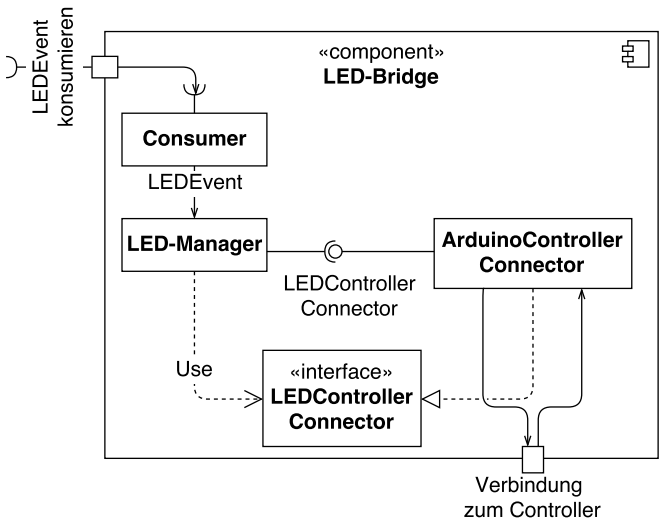


Abbildung 5.5: Innensicht der Unterkomponente LED-Bridge

LED-Bridge

Die LED-Bridge ist eine Softwarekomponente, die einen einen Kafka-Konsumenten beinhaltet und die Kommunikation zu dem LED-Controller herstellt.

Ihre Aufgabe ist das Empfangen der LEDEvents aus dem Messaging-System und die Übergabe von Parametern an den LED-Controller. Sie ist notwendig, da der LED-Controller selbst nicht in der Lage ist eine direkte Verbindung zum Messaging-System herzustellen.

Neben ihrer Notwendigkeit bietet die LED-Bridge auch den Vorteil, dass die Verarbeitung der LEDEvents unabhängig von der konkreten Hardwareimplementation des LED-Controllers ist. Dies sorgt für eine Austauschbarkeit des LED-Controllers und ist für zukünftige Erweiterungen sinnvoll.

Die LED-Bridge bildet also gewissermaßen die Brücke zwischen Messaging-System und LED-Controller. Sie übernimmt damit den ersten Verarbeitungsschritt des LEDEvents (siehe Abbildung 5.4 auf der vorherigen Seite).

Dieser erste Verarbeitungsschritt wird nachfolgend auf Basis der Innensicht von LED-Bridge (Abbildung 5.5) dargelegt. Es wird auf die interne Verarbeitung sowie die Kommunikation zwischen LED-Bridge und LED-Controller eingegangen.

Consumer Der Consumer (Kafka-Consumer) ist für das Empfangen der *LEDEvents* aus dem Messaging-System zuständig. Die Verbindung zum Messaging-System wird allerdings nicht über einen Flink-Connector hergestellt, da sich die *LED-Anzeige* außerhalb des Flink-Kontext

befindet. Stattdessen wird die *Consumer API* verwendet, dies ist eine Bibliothek aus dem Kafka-Projekt (Apache Software Foundation, j).

Die Bibliothek bietet eine ähnliche Funktionalität wie der Flink-Connector für Kafka. Allerdings gibt es den Unterschied, dass die Ereignisse nicht automatisch in einem *DataStream* eintreffen, sondern in regelmäßigen Abständen abgeholt werden müssen.

Nach der Initialisierung des Consumers wird die Methode *consumer.poll* regelmäßig aufgerufen, um die Ereignisse abzuholen. Diese liefert ein *ConsumerRecords*-Objekt zurück, welches alle Ereignisse (records) enthält, die zwischenzeitlich eingetroffen sind. Anschließend wird über das *ConsumerRecords*-Objekt iteriert, sodass jedes Ereignis verarbeitet wird.

Der Consumer übergibt die eingetroffenen LEDEvents anschließend an den LED-Manager.

LED-Manager Der LED-Manager nimmt LEDEvents des Consumers entgegen, liest das jeweilige Ziel der LEDEvents (den Zielcontroller) aus und leitet diese an den entsprechenden LEDConnectorController weiter.

Das Interface LEDControllerConnector gibt vor, dass jeder ControllerConnector eine Funktion implementiert, die LEDEvents erhalten kann. Die eigentliche Übersetzung findet erst in der jeweiligen Implementation des LEDControllerConnector-Interface statt (zum Beispiel im *ArduinoControllerConnector*, siehe unten).

Die Kapselung über ein Interface zielt auf eine spätere Austauschbarkeit ab: Der LED-Manager kann durch das Interface unabhängig von der konkreten Implementation arbeiten. Falls sich der LED-Controller ändern sollte oder ein weiterer LED-Controller angeschlossen werden soll, muss dieser nur das LEDControllerConnector-Interface implementieren und könnte über den LED-Manager eingebunden werden.

In dieser Arbeit wird als LED-Controller ein *Arduino* verwendet (siehe unter 5.2.1 ab Seite 69). Nachfolgend wird die Implementation des Interfaces LEDControllerConnector für einen Arduino-LED-Controller erläutert.

ArduinoControllerConnector Der ArduinoControllerConnector stellt die direkte Kommunikation zum Arduino her und implementiert das Interface LEDControllerConnector.

Die Steuerung des Ardunios erfolgt über serielle Kommunikation (siehe 4.3 auf Seite 50). Für die serielle Kommunikation wird die Bibliothek *Ardulink 2* verwendet. Sie bietet grundlegende Funktionen für die serielle Kommunikation mit einem Arduino, z.B. das Senden und Empfangen von seriellen Nachrichten (Ardulink). Die zu sendenden Nachrichten müssen allerdings selbst definiert werden.

Durch das Implementieren des LEDControllerConnector-Interfaces werden die LEDEvents vom LED-Manager an den ArduinoControllerConnector geleitet. Dieser muss daraufhin den Arduino ansteuern. Dazu werden die Parameter des LEDEvents, wie zum Beispiel die Leuchtfarbe, ausgelesen und mittels einer seriellen Nachricht an den Arduino übertragen.

LED-Controller

Wie schon erwähnt, wurde sich bei dem LED-Controller für einen Arduino entschieden. Damit der Arduino seine Aufgabe als LED-Controller erfüllen kann, muss er programmiert werden. Zur Programmierung des Arduinos wird eine eigene IDE (integrated development environment; zu deutsch: Integrierte Entwicklungsumgebung) vom Hersteller angeboten. Die IDE bietet einige Features an, die das Programmieren erleichtern, wie zum Beispiel die Übertragung des Programms auf den Arduino (Arduino, b). Die IDE ist jedoch nicht zwingend Notwendig für die Programmierung (Arduino, c).

Im Regelfall benötigt der Arduino nur zum Übertragen des Programms eine einmalige Verbindung zum Computer. Zur Ausführung seines Programms wird keine Verbindung zum Computer benötigt. Allerdings braucht der Arduino für den Zweck als LED-Controller eine dauerhafte Verbindung zum Computer, da hierüber die Kommunikation zwischen LED-Bridge und Arduino stattfindet. Arduino und LEDBridge sind über USB verbunden (siehe 5.4 auf Seite 64).

Programmierung des Arudino Die Programmierung des Arduinos erfolgt in einer eigenen Programmiersprache, welche auf den Sprachen *C* und *C++* basiert und dessen Befehle vollständig unterstützt (Arduino, c). Die Sprache bietet außerdem einige Funktionen an, die für das Programmieren eines Mikrocontrollers hilfreich sind, wie zum Beispiel die Steuerung der analogen und digitalen Ein- und Ausgänge des Arduinos (Arduino, d).

Ein Arduino-Programm wird „Sketch“ (zu deutsch Skizze / Aufzeichnung) genannt. Ein Sketch besteht mindestens aus den Funktionen *setup()* und *loop()*. Die *setup()*-Funktion wird bei jedem Start des Sketches (Hochfahren des Arduinos) einmal ausgeführt, in ihr können Variablen definiert oder Bibliotheken initialisiert werden. Die Funktion *loop()* wird wiederholt ausgeführt, bis der Sketch beendet wird. Sie ist eine Endlosschleife und das Kernstück des Sketches (Arduino, d).

Arduino als LED-Controller Zur Verwendung des Arduinos als LED-Controller musste ein Sketch programmiert werden, der seriell übertragene Nachrichten auswertet und daraufhin elektronische Signale an den richtigen PIN sendet, um den LED-Strip zu steuern.

Zum Auswerten der seriellen Nachrichten wird der sogenannte *serial receive buffer* fortlaufend überprüft, dies geschieht über die Methode *Serial.available()*. In dem *serial receive buffer* werden die ankommenden Bits vorgehalten, damit sie später ausgelesen werden können (Arduino, a). Falls der Buffer nicht leer ist, wird die Nachricht in den Sketch eingelesen. Das Ende einer Nachricht markiert ein Zeilenumbruch (linefeed bzw. „\n“).

Sobald die Nachricht vollständig eingelesen wurde, wird eine Antwort-Nachricht verschickt, sodass die LED-Bridge weiß, dass die Nachricht angekommen ist. Dabei liegt es nicht in der Hand des Arduinos, ob die LED-Bridge auf eine Bestätigung des Arduinos wartet, bevor sie weitere Nachrichten sendet.

Anschließend wird der Nachrichteninhalt in der *loop()*-Funktion des Sketches analysiert. Hierzu wird die Nachricht von links nach rechts Stückweise ausgewertet. Am Anfang der Nachricht steht immer der Befehl, der ausgeführt werden soll. Danach folgen die Parameter, die für die Ausführung des Befehls benötigt werden (siehe auch Punkt 4.3 auf Seite 50).

Wurde ein Befehl in der Nachricht erkannt, wird eine selbst entwickelte Sketch-Funktion aufgerufen, die mit diesem Befehl verknüpft ist. Die Sketch-Funktion erhält den verbleibenden Nachrichtenteil (die Parameter) und sorgt dafür, dass die Parameter richtig geparst werden. Aus der Sketch-Funktion werden gegebenenfalls weitere Funktionen mit Teilen der verbleibenden Nachricht aufgerufen. Die Nachricht wird also Stückweise zerteilt und an entsprechende Funktionen übergeben. Am Ende des Zerteilungsprozesses wird eine Funktion aufgerufen, die den LED-Strip zum Leuchten bringt.

Ansteuerung des LED-Strips Zur Ansteuerung des LED-Strip wird vom Hersteller des LED-Streifens eine Bibliothek für den Arduino angeboten. Die Bibliothek heißt *Adafruit NeoPixel* und ermöglicht ein objektorientiertes Arbeiten auf dem LED-Strip. Dazu muss lediglich konfiguriert werden, an welchen PIN der LED-Strip angeschlossen ist, um welche Art LED-Streifen es sich handelt und wie viele LEDs der Streifen beinhaltet (Burgess, 2016a).

Die Steuerung erfolgt dann auf einem Objekt. Auf diesem kann zum Beispiel die Funktion *setPixelColor* aufgerufen werden, mit welcher die Farbe einer LED des Streifens gesetzt wird. Die gesetzten Farbkombinationen werden über die Objekt-Funktion *show* automatisch an den LED-Strip übertragen (Burgess, 2016a). Für Leuchtanimationen (sich bewegende Leuchteffekte) müssen allerdings eigene Funktionen programmiert oder weitere Bibliotheken eingebunden werden. Die NeoPixel-Bibliothek bietet hier nur Beispiele.

Test der LED-Anzeige Die LED-Anzeige besteht sowohl aus Soft- als auch aus Hardwarekomponenten. Das Testen findet nur für die Softwarekomponente LED-Bridge statt. Dafür wurde das Framework *JUnit* verwendet.

Die Tests decken dabei vorwiegend das Transformieren der LEDEvents ab, welche sicherstellen, dass die Konvertierung aus und zu JSON richtig funktioniert. Dafür werden einige Test-LEDEvents erstellt, die zu JSON und zurück transformiert werden. Anschließend wird überprüft, ob das Test-Ereignis mit dem transformierten Ereignis übereinstimmt, hierfür wird sowohl der Typ beider Klassen als auch dessen Parameter verglichen. Besonders das Vergleichen des Typs ist wichtig, da bei den LEDEvents mit Vererbung gearbeitet wurde.

Des weiteren findet ein Test des LED-Managers statt, dadurch soll sichergestellt werden, dass eine korrekte Zuordnung von LEDEvents zum LEDControllerConnector erfolgt. Dafür mussten mehrere Test-ControllerConnectors erstellt werden, da das Testen mit einem echten LED-Controller schwierig ist (der Arduino müsste hierfür angeschlossen sein),

Das Testen des echten Arduino-Controllers kann nicht automatisiert werden, daher muss dies manuell – durch einen Menschen – stattfinden. Um den manuellen Test zu erleichtern, wurde eine Testsequenz erstellt, welche verschiedene Leuchteffekte an den Arduino sendet. Der Mensch kann am Leuchten des LED-Strip überprüfen, ob alle Leuchteffekte richtig vom Arduino empfangen und an den LED-Strip übertragen wurden.

5.2 Hardwaretechnische Umsetzung

Die hardwaretechnische Umsetzung bezieht sich auf die Hauptkomponente LED-Anzeige, da diese spezielle Hardware für den LED-Controller und den LED-Strip benötigt. Die Hauptkomponente EventAlert hingegen benötigt keine spezielle Hardware, sie besteht nur aus Software, die auf jedem handelsüblichen Computer ausgeführt werden kann.

Dieser Punkt behandelt vorwiegend die Auswahl eines LED-Controllers und LED-Strips nach den aufgestellten Anforderungen A7 bis A13 (siehe Anhang A ab Seite 96). Außerdem werden die erforderlichen Schritte zur Umsetzung, wie zum Beispiel die Verbindung der beiden Hardware-Komponenten, erläutert.

Die Aufgabe des LED-Controller ist es, den LED-Strip zu steuern. Die Steuerungsanweisungen erhält der LED-Controller von der LED-Bridge (siehe Punkt 5.1.3 ab Seite 63).

5.2.1 Auswahl eines Controllers

Die Auswahl an Hardware, die den LED-Strip theoretisch steuern könnte, ist groß. Es kommt im Prinzip jeder programmierbare Mikrocontroller infrage, der eine Verbindung zu einem Com-

puter aufweist und an den sich die Adern des LED-Strips anschließen lassen. Mikrocontroller sind kleine Computer, an die externe Hardware (wie z.B. Aktoren und Sensoren) angeschlossen werden kann. Sie sind meist in elektronische Geräte zur Steuerung eingebaut, zum Beispiel in Mikrowellen oder Waschmaschinen.

Neben Mikrocontrollern gibt es auch vorgefertigte Controller zur Steuerung von LED-Streifen. Die vorgefertigten Controller können softwareseitig über das DMX-Protokoll gesteuert werden. DMX (*Digital Multiplex*) ist ein digitales Steuerprotokoll, welches in der Licht- und Veranstaltungstechnik zum Einsatz kommt, um diverse Geräte der Veranstaltungstechnik, wie z.B. bewegbare Scheinwerfer oder eben LED-Streifen, zu steuern.

Entscheidung gegen DMX Nach gründlicher Recherche ist die Idee, den LED-Strip über einen DMX-Controller zu steuern, verworfen worden. Der Hauptgrund dafür war, dass DMX-Controller keine direkte Programmierung des Controllers ermöglichen. Damit wäre die Anforderung A11, welche eine Programmierbarkeit der Steuerung fordert, nicht erfüllt.

Ein weiter Grund ist die Komplexität: DMX ist auf große Projekte ausgelegt, bei denen verschiedenste Geräte gesteuert werden. In dieser Arbeit soll hingegen nur ein ganz spezifisches Gerät gesteuert werden, nämlich der LED-Strip.

Auswahl eines Mikrocontrollers Aufgrund, dass sich gegen die Steuerung mittels eines DMX-Controllers entschieden wurde, bleibt nur die Steuerung über einen Mikrocontroller übrig. Unter den programmierbaren Mikrocontrollern stechen zwei Plattformen aufgrund ihrer hohen Verbreitung heraus: Der Raspberry Pi und der Arduino.

Dabei ist zu beachten, dass der Raspberry Pi kein Mikrocontroller im eigentlichen Sinne ist, bei dem Raspberry Pi handelt es sich vielmehr um einen Ein-Platinen-Computer, auf welchem ein vollwertiges Linux-basiertes Betriebssystem installiert werden kann. Des Weiteren lässt sich der Raspberry Pi auch als Personal Computer verwenden: Er bietet unter anderem Hardwareschnittstellen für einen Monitor (HDMI) und für Peripheriegeräte (USB) an. Außerdem wird ein SD-Karten-Slot zur Verfügung gestellt, auf dem sich Daten persistieren lassen.

Dennoch lässt sich der Raspberry Pi als LED-Controller verwenden, entscheidend dafür sind seine programmierbaren Steckverbindungen, an welche sich die Adern eines LED-Strips anschließen lassen. Der LED-Controller wäre dann nur ein Programm, das auf dem Betriebssystem des Raspberry Pis ausgeführt wird.

Bei dem Arduino handelt es sich hingegen um einen „echten“ programmierbaren Mikrocontroller. Sein Betriebssystem ist eingebettet und dient nur zur Ausführung der – unter Punkt 5.1.3 erwähnten – Sketche. Er lässt sich somit nicht als Personal Computer verwenden. Dafür

ist er darauf ausgelegt, dass genau ein Programm (Sketch) wiederholt ausgeführt wird. Der Arduino bietet – genauso wie der Raspberry Pi – programmierbare Steckverbindungen an, an welche sich die Adern eines LED-Strips anschließen lassen.

Sowohl der Raspberry Pi als auch der Arduino erfüllen die meisten Anforderungen an die Steuerung der LED-Anzeige (siehe A7 bis A13 in Anhang A, ab Seite 96):

- Anforderung A7: Beide Geräte können die Leuchtfarbe und -stärke des LED-Strips steuern.
- Anforderung A8: Die Geräte können über eine standardisierte Hardwareschnittstelle mit dem ausführenden Computer verbunden werden: Beim Raspberry Pi kann dies über den LAN-Anschluss (indirekt über ein Netzwerk) geschehen. Der Arduino stellt eine USB-Schnittstelle (direkte Verbindung) bereit.
- Anforderung A9: Durch den flexiblen Anschluss der LED-Strip-Adern an die Steckverbindungen der Geräte, lassen sich verschiedene Arten von LED-Streifen an beide Geräte anschließen.
- Anforderung A11: Raspberry Pi und Arduino unterstützen eine objektorientierte Programmierung. Beim Raspberry Pi lässt sich sogar zwischen unterschiedlichen Programmiersprachen wählen.
- Anforderung A12: Für beide Geräte sind Bibliotheken zur Steuerung des LED-Strips vorhanden, die kein elektrotechnisches Wissen erfordern.
- Anforderung A13: Für den Arduino sind Bibliotheken zur seriellen Kommunikation mit dem ausführenden Computer vorhanden. Beim Raspberry Pi müsste die Kommunikation über ein Netzwerkprotokoll stattfinden, für dieses sind verschiedene Bibliotheken für unterschiedliche Sprachen vorhanden.

Die Anforderungen A10 (gemeinsame Stromversorgung für LED-Streifen und Steuerung) kann nur bedingt erfüllt werden: Die übliche Stromversorgung beider Geräten reicht nicht aus, um den LED-Streifen mit Strom zu versorgen. Eine externe Stromquelle für den LED-Streifen ist also in jedem Fall erforderlich. Allerdings gibt es Möglichkeiten, den Arduino beziehungsweise den Raspberry Pi auch über die externe Stromquelle mit Strom zu versorgen. Üblich ist jedoch die Versorgung über ein USB-Netzteil oder eine USB-Verbindung zu einem Computer.

Die Steuerung eines LED-Strips ist also mit beiden Plattformen nach den aufgestellten Anforderungen möglich. Weiter haben die Plattformen eine große Nutzerbasis, sodass viele Beispielprojekte, auch wie sich ein LED-Strip steuern lässt, vorhanden sind.

Entscheidung für Arduino Nach Durchsicht von diversen Beispielprojekten beider Plattformen, in welchen ein LED-Strip gesteuert wird, ist die Wahl auf den Arduino (*Arduino Uno*) gefallen.

Anhand der Beispielprojekte wurde deutlich, dass die Programmierung einer Steuerung mit dem Arduino weniger Aufwand erfordert als mit dem Raspberry Pi. Dies liegt vor allem daran, dass der Arduino kein variables Betriebssystem besitzt und somit keine Installationen und Konfigurationen erfordert. Dies ermöglicht das Fokussieren auf die eigentliche Programmierung des LED-Controllers. Gewissermaßen ist die Einschränkung der Möglichkeiten durch das eingebettete Betriebssystem beim Arduino der entscheidende Vorteil gegenüber zum Raspberry Pi. Da die Überprüfung der Anforderungen schon für beide Mikrocontroller stattgefunden hat, wird auf eine erneute Überprüfung verzichtet.

5.2.2 Auswahl und Verbindung des LED-Strip

Für diese Anwendung kommt grundsätzlich jeder nicht adressierbare (analoge) RGB-LED-Strip infrage, dessen Adern sich an den LED-Controller anschließen lassen.

Allerdings gibt es bei nicht adressierbaren LED-Streifen einen Nachteil. Sie benötigen für den Anschluss an den LED-Controller weitere Hardware, welche die elektrische Spannung und Stromstärke des Arduinos auf die Anforderungen des LED-Streifens anpasst (Cooper, 2017).

Der Fokus dieser Arbeit liegt auf der softwaretechnischen Umsetzung mit Flink, daher soll die hardwaretechnische Umsetzung möglichst simpel gehalten werden. Aufgrund dessen wurde die Visualisierung mit nicht adressierbaren LED-Streifen für zu aufwendig befunden.

Die Alternative zum analogen, nicht adressierbaren LED-Streifen bietet ein digitaler, adressierbarer LED-Streifen. Bei diesem werden die Steuerungssignale digital über eine Ader an den LED-Streifen übertragen. Er benötigt keine elektrotechnischen Anpassungen. Außerdem haben digitale LED-Streifen den Vorteil, dass jede LED des Streifens individuell gesteuert werden kann, wodurch eine größere Vielfalt an Leuchteffekten ermöglicht wird (siehe 2.6 ab Seite 19). Dies käme auch dem Telegram-Szenario (siehe 3.1 auf Seite 22) zugute, in welchem gefordert wurde, dass der LED-Streifen in zwei Farben gleichzeitig leuchten soll. Die Umsetzung des Szenarios wäre mit einem adressierbaren LED-Streifen möglich.

Unter den digitalen LED-Streifen ist das Angebot ähnlich groß wie bei den analogen LED-Streifen. Bei der Recherche nach digitalen LED-Streifen hat sich der Hersteller *Adafruit Industries* positiv von anderen Herstellern abgehoben. Adafruit bietet diverse Hardware für den Arduino und andere Mikrocontroller an, darunter auch digitale LED-Streifen. Der große Vorteil

bei diesem Hersteller ist, dass Beispielprojekte, Anleitungen und Bibliotheken zur Anwendung der Hardware mit angeboten werden.

Durch die Verknüpfung von Hardware mit dazugehörigen Bibliotheken, Beispielprojekten und Anleitungen kommt beim Hersteller Adafruit alles aus einer Hand. Dies hat den Vorteil, dass mögliche Komplikationen bei der Umsetzung, wie zum Beispiel fehlende Kompatibilität von Hard- und Software oder unspezifische Anleitungen, reduziert werden. Da die hardwaretechnsiche Umsetzung möglichst unkompliziert sein soll, scheint ein LED-Streifen von Adafruit optimal für diese Arbeit.

Nach Begutachtung des Angebots von Adafruit wurde sich für den „Adafruit NeoPixel Digital RGBW LED Strip - White PCB 144 LED/m“ entschieden, dies ist ein digitaler LED-Streifen mit 144 LEDs pro Meter. Jede LED des Streifens besteht aus einer roten, grünen, blauen und weißen LED, dadurch kann dieser LED-Streifen auch echtes weißes Licht (siehe 2.6 auf Seite 19) darstellen (Adafruit Industries).

Verbindung des LED-Strip mit dem LED-Controller

Nachdem die Entscheidung für einen Controller und einen LED-Streifen gefallen ist, soll nachfolgend erläutert werden, wie beide Hardwarekomponenten miteinander verbunden werden. Die Verbindung wird anhand der Abbildungen B.1 und B.2 (Anhang B, Seite 100) deutlich, welche die elektrotechnischen Details der Verbindung zeigen. In diesem Abschnitt wird nur auf die Besonderheiten der Verbindung eingegangen.

Stromversorgung Die Stromversorgung des LED-Strip findet über ein externes Netzteil statt, da die Leistung des Arduino nicht ausreicht. Vor die Stromversorgung des LED-Strips ist ein Kondensator geschaltet, um den LED-Streifen vor plötzlichen Spannungsschwankungen – vor allem beim Einschalten – zu schützen (Burgess, 2016c,b).

Der Arduino bezieht seinen Strom nicht aus dem Netzteil des LED-Strips, er wird über die USB-Verbindung zum Computer mit Strom versorgt.

Dies führt zur Nichterfüllung von A10 (Anforderung an die gemeinsame Stromversorgung; siehe Anhang A, ab Seite 96). Es sind zwar Möglichkeiten vorhanden, den Arduino über das Netzteil des LED-Strip zu versorgen (siehe 5.2.1 auf Seite 70), allerdings wird eine USB-Verbindung ohnehin zur Übertragung der Befehle benötigt.

Steuerung Die Übermittlung des Signals zur Steuerung des LED-Strip findet über eine Ader statt, welche die digitale Steckverbindung des LED-Controller mit dem Dateneingang des

LED-Strip verbindet. Die Nummer, der digitalen Steckverbindung (z.B. PIN 6), muss bei der Programmierung des LED-Controller angegeben werden.

Zwischen die Verbindung von Steuerung und Streifen ist ein Widerstand verbaut. Der Widerstand wird verwendet um Spannungsspitzen abzufangen, da diese die erste LED des Streifens schädigen können (Burgess, 2016b).

6 Erfahrung und Kritik zu Apache Flink

Dieses Kapitel behandelt die eigenen Erfahrungen und eine kritische Auseinandersetzung mit Apache Flink, wodurch ein Eindruck über die Entwicklung diesem Framework vermittelt und eventuelle Verbesserungen aufgezeigt werden sollen.

Als Grundlage für die Auseinandersetzung werden die unter Punkt 3.6 ab Seite 35 aufgestellten Anforderungen an Flink verwendet. Eine Auflistung der Anforderungen findet sich im Anhang A ab Seite 96 (Anforderungen A14 bis A28). Alle Anforderungen werden jeweils im Einzelnen überprüft.

Dafür ist dieses Kapitel in fünf Kategorien (*Installation und Einrichtung*, *Dokumentation und externe Hilfestellungen*, *Verständlichkeit von Methoden und Parametern*, *Belastungstest* und *Entwicklung mit Apache Flink*) unterteilt. Jede Kategorie geht auf einen oder mehrere Aspekte der eigenen Erfahrung mit Flink ein. Am Ende jeder Kategorie wird die Erfahrung in einem Fazit zusammengefasst.

6.1 Installation und Einrichtung

Diese Kategorie geht auf die ersten Schritte mit Apache Flink ein. Dafür wurde zunächst nach Anleitungen, vorzugsweise auf deutsch, für den Einstieg gesucht. Jedoch war die Recherche nach Anleitungen für den Einstieg mit Flink weitgehend erfolglos. Die einzigen vielversprechenden Anleitungen sind in englischer Sprache und werden auf der Internetseite von Apache Flink als „Guides“ angeboten. Auf Basis dieser Guides werden nachfolgend die ersten Eindrücke und Erfahrungen mit Flink beschrieben.

Quickstart-Guide Zum ersten Einstieg wurde der Flink Quickstart-Guide durchgearbeitet. Der Quickstart Guide basiert auf einem lokalem Flink Cluster, welches als gepacktes Archiv heruntergeladen werden kann (Apache Software Foundation, d).

Das Archiv enthält einen Beispieljob, welcher als Java Archiv vorliegt. Der Quickstart-Guide beschreibt, wie man ein lokales Flink Cluster startet und auf diesem den Beispieljob ausführt. Außerdem wird der Quellcode des Beispieljobs anhand von Kommentaren erklärt. Um den Beispieljob auszuführen, wird ein lokaler Server benötigt, welcher Textzeilen zur Verfügung

stellt. Die Funktion des Beispieljobs ist es, die Wörter aus den Textzeilen des lokalen Servers zu zählen und dessen Anzahl auszugeben.

Die Einrichtung des lokalen Clusters hat keine Probleme bereitet, allerdings ist der Guide auf eine Unix-Umgebung ausgerichtet. Für eine Einrichtung unter Windows gibt es eine extra Anleitung, die aber nur auf das Starten des Clusters eingeht. Die Installation unter Windows wird nicht detailliert erläutert. Hierfür braucht es Informatik-Grundkenntnisse, zum Beispiel wie ein Archiv entpackt werden kann. Außerdem setzt der Guide voraus, dass eine aktuelle Java-Version (Java 8) installiert ist (Apache Software Foundation, d).

Die Ausrichtung auf eine Unix-Umgebung bereitet dem Windows-Benutzer weitere Overheads, welche nachfolgend erläutert werden. Um den Beispieljob auszuführen, muss ein lokaler Server gestartet werden, welcher die erwähnten Textzeilen zur Verfügung stellt. Im Guide wird dafür ein *netcat*-Server gestartet (Apache Software Foundation, d), dies ist ein Programm, welches Konsolen-Eingaben über eine Netzwerkverbindung zur Verfügung stellt. Netcat ist unter Windows nicht vorinstalliert, da es eigentlich ein Unix-Programm ist. Es muss also zunächst eine netcat-Nachbildung für Windows gefunden und installiert werden, um den Beispieljob auszuführen.

Weiter muss für die Überprüfung der Ausgabe des Beispieljobs (die gezählten Wörter) die Logdatei des Clusters geöffnet werden. Der Guide verwendet dafür das Unix-Tool *tail* (Apache Software Foundation, d), welches fortlaufend die neuen Zeilen der Logdatei auf der Konsole ausgibt. Auch dieses Programm gibt es in der Windows-Umgebung nicht. Es muss also erneut eine Windows-Nachbildung des Programms gefunden und installiert werden.

Diese Hürden sind gerade zum Einstieg sehr lästig. Es vermittelt das Gefühl, dass Flink nicht auf die Benutzung mit Windows ausgelegt ist und mit weiterem extra Aufwand zu rechnen ist. Informatikern, die vorwiegend mit Windows arbeiten, dürfte dies aber ein bekanntes Problem sein.

Nachdem alle „Windows-Hürden“ überwunden wurden, ließ sich das Beispielprojekt problemlos ausführen. Der Quellcode des Beispieljobs ist verständlich erklärt und das Ziel des Jobs ist klar erkenntlich. Allerdings kann die Syntax, vor allem das Fluent-Interface und die Methodennamen (wie z.B. *flatMap*) von Flink, anfänglich etwas verwirrend sein für jemanden, der noch nicht mit Streams (Datenströmen) gearbeitet hat.

Einrichtung des Projekts Nach dem Durcharbeiten des Quickstart-Guides soll ein eigens Projekt eingerichtet werden. Dazu bietet Flink je eine Anleitung für die Programmiersprachen Java und Scala an. Da EventAlert in Java programmiert werden sollte, wurde die Anleitung zum „Sample Project in Java“ gewählt.

Die Anleitung beschreibt, wie ein eigenes Flink Projekt in Java eingerichtet wird. Dazu wird ein Maven-Projekt zur Verfügung gestellt, welches weitere Beispieljobs enthält (Apache Software Foundation, e).

Die Einrichtung des Maven-Projekts erfolgt weitgehend automatisch über das *Maven Archetype Plugin*, welches das Generieren eines eigenen Projekts auf Basis eines bestehenden Projekts ermöglicht. Für die Einrichtung wird vorausgesetzt, dass Maven und Java installiert und über die Konsole ausführbar sind (Apache Software Foundation, e). Im Einrichtungsdialog müssen die Maven-Parameter *groupId* und *artifactId* gesetzt werden, außerdem wird ein Java-Paketname gefordert. Anhand dieser Parameter wird ein neues – eigenes – Maven-Projekt generiert. Als Alternative zur Einrichtung mit Maven wird ein Shell-Skript zur Verfügung gestellt, welches allerdings nur unter Unix-Systemen funktioniert.

Die Anleitung geht nach der Einrichtung auf die Struktur des Beispiel-Projekts ein (Apache Software Foundation, e). Das Projekt besteht aus vier Klassen, von denen zwei ausführbare Beispieljobs sind. Die anderen beiden Klassen sind Vorlagen für eigene Jobs. Unter den Beispieljobs ist ein Beispiel für die Echtzeit-Datenstrom-Verarbeitung und ein Beispiel für die Batch-Verarbeitung vorhanden.

Außerdem wird darauf hingewiesen, dass sich die Jobs über *main*-Methoden ausführen lassen (Apache Software Foundation, e). Durch das Ausführen der Main-Methoden wird Flink im Entwicklungsmodus gestartet. Dadurch lässt sich Flink innerhalb der Entwicklungsumgebung starten und testen, ein Cluster muss nicht gestartet werden. Dieser Hinweis ist leider nicht prominent platziert, sodass der Vorteil im Vergleich zur komplizierteren Ausführung über das Cluster nicht deutlich wird.

Die Anleitung, wie aus dem Projekt ein Java Archiv gebaut wird, das auf einem Cluster ausgeführt werden kann, ist hingegen recht prominent sichtbar. Dies sorgte bei der Entwicklung von EventAlert anfänglich dafür, dass der komplizierte Weg über das Cluster zum Testen der Jobs genutzt wurde. Eine Art „Best practice“-Hinweis, wie man am besten mit Flink entwickelt, wäre hier hilfreich gewesen.

Weiter geht die Anleitung des Beispielprojekts auf den Job für die Batch-Verarbeitung ein. Der Batch-Job hat – wie der Job aus dem Quickstart-Guide – das Ziel die Worthäufigkeit in einem Text zu zählen. Der Quellcode ist auch hier anhand von Kommentaren erklärt. Der Job kann allerdings ohne den netcat-Server ausgeführt werden, da es sich bei diesem Beispieljob um ein Beispiel für die Batch-Verarbeitung handelt.

Die Anleitung endet nach der Erklärung des Batch-Jobs mit einem Hinweis auf die weiterführende Dokumentation.

Fazit Die Installation und Einrichtung war mit einigen Schwierigkeiten verbunden, welche unter anderem der Entwicklung in einer Windows-Umgebung geschuldet sind.

Der Einstieg über den Quickstart-Guide hat sich rückblickend nicht zielführend herausgestellt, da dieser eher auf die Ausführung von existierenden Flink-Jobs ausgelegt ist und somit nicht für das Ziel geeignet ist, mit Flink zu entwickeln.

Das „Sample Project in Java“ stellt einen besseren Einstieg dar, wenn mit Flink entwickelt werden soll. Leider wird dort nur auf einen Batch-Beispieljob eingegangen, der im Beispielprojekte vorhandene Echtzeit-Beispieljob wird nicht erklärt. Außerdem wird nicht deutlich genug darauf hingewiesen, dass es kein Flink-Cluster für die Ausführung der Beispiele braucht.

Vorteilhaft wäre, wenn der Quickstart-Guide und das „Sample Project in Java“ zu einer Anleitung zusammengefasst werden würden. Dies würde einen umfassenderen Einblick in die Arbeit mit Flink eröffnen. Des Weiteren würde dadurch der Unterschied zwischen der Ausführung auf einem Cluster und der Ausführung über die Main-Methode der Jobs deutlich werden.

Zusammenfassend lässt sich sagen, dass die Installation und Einrichtung in Flink nicht besonders leicht war: Eigene Recherchen und Testversuche waren notwendig, um einen Einstieg zu Flink zu bekommen. Dies ist allerdings bei den meisten komplexeren Frameworks der Fall, deswegen kann die Installation und Einrichtung auch nicht als besonders schwer eingestuft werden. Der Einstieg wäre vermutlich leichter gefallen, wenn es externe Anleitungen, die auf Flink-Neulinge zugeschnitten sind, gegeben hätte.

6.2 Dokumentation und externe Hilfestellungen

Diese Kategorie geht auf die Aspekte der Dokumentation und auf die Verfügbarkeit von externen Hilfestellungen ein. Mit „Dokumentation“ ist die Flink-eigenen Dokumentation gemeint. Unter externen Hilfestellungen werden hingegen Flink-fremde Hilfestellungen und Beispiele verstanden.

Sowohl die Dokumentation als auch externe Hilfestellungen sind essentiell für die Arbeit mit umfangreichen Frameworks wie Flink. Die Dokumentation gibt dem Entwickler häufig wichtige Anhaltspunkte, wie Methoden und Parameter zu verwenden sind, während externe Hilfestellungen zumeist Beispiele aus der Praxis bieten und eine Orientierung geben, wie mögliche Probleme zu lösen sind.

Dokumentation Die Anforderung A21 formuliert den Anspruch, dass alle Klassen, Methoden und Parameter, die für das Benutzen von Flink von Bedeutung sind, dokumentiert sind

(siehe A21, ab Seite 96). Nachfolgend wird überprüft, ob diese Anforderung erfüllt werden konnte.

Flink bietet zwei Formen der Dokumentation an: Zum einen den *Programming-Guide* und zum anderen das *Javadoc*. Der Programming-Guide erklärt die zentralen Funktionen und Vorgehensweisen mit Flink auf eine nicht technische Weise. Außerdem werden dort einfache Codebeispiele aufgeführt, welche die Erklärungen unterstützen. Das Javadoc hingegen dokumentiert jede Funktion des Frameworks mit allen ihren Parametern und ist technischer als der Programming-Guide. Das Javadoc hat einen direkten Bezug zum Quellcode von Flink, da es aus Methoden- und Klassen-Kommentaren des internen Quellcode generiert wurde. Sowohl der Programming-Guide als auch das Javadoc bieten eine gute Unterstützung bei der Programmierung mit Apache Flink.

Der Programming-Guide gibt dem Entwickler einen Überblick über die verschiedenen Möglichkeiten von Flink. Er ist in einzelne Themenbereiche mit weiteren Unterpunkten unterteilt. Die Erklärungen sind in einfacher Sprache mit wenigen Fachbegriffen formuliert und werden gegebenenfalls mit Abbildungen unterstützt (zum Beispiel in Apache Software Foundation, g), um komplexe Funktionsweisen zu präzisieren. Die Programmierbeispiele sind auf die Erklärungen abgestimmt und mit Kommentaren versehen.

Das Javadoc ist vorwiegend eine Unterstützung bei der Entwicklung mit einer IDE. Je nach IDE und Einstellung wird dort das Javadoc automatisch zu der gerade verwendeten Klasse, Methode oder dem verwendeten Parameter angezeigt. Sie bietet dem Entwickler zusätzliche Informationen, wie zum Beispiel über die Parameter einer Methode.

Alternativ lässt sich das Javadoc auch als HTML-Seite aufrufen (siehe Apache Software Foundation, k). Die Darstellung als Seite ist jedoch recht unübersichtlich, da jede Klasse dokumentiert wurde und das Framework verhältnismäßig groß ist: In der Dokumentation sind 5185 dokumentierte Klassen aufgeführt (Stand: 18.10.2017) (Apache Software Foundation, k). Von diesen Klassen wird man in der Praxis jedoch – abhängig vom Projekt – nur einen Bruchteil benutzen.

Externe Hilfestellungen Für Frameworks, die eine hohe Nutzung in der Praxis aufweisen, steht oft eine große Bandbreite von externen Hilfestellungen zur Verfügung. Diese lassen sich zumeist in Portalen oder privaten Seiten frei im Internet abrufen.

Gerade zum Einstieg bieten externe Hilfestellungen erfahrungsgemäß eine bessere Unterstützung an als die Framework-eigene Dokumentation. Dies liegt vor allem daran, dass externe Hilfestellungen häufig von Entwicklern geschrieben wurden, die das Framework selbst praktisch anwenden. Die Framework-eigene Dokumentation hingegen wird oft von den Entwicklern des Frameworks selbst formuliert und geht vorwiegend auf die Möglichkeiten und technischen Details des Frameworks ein. Durch den Praxisbezug sind externe Hilfestellungen häufig verständlicher, als die Framework-eigene Dokumentation.

Nachfolgend soll begutachtet werden, wie reichhaltig das Angebot von externen Hilfestellungen ist und in welcher Form diese angeboten werden.

Leider hat sich die Suche nach externen Hilfestellungen als nicht ertragreich erwiesen. Der Großteil der gefundenen externen Hilfestellung beschreibt keine konkrete Anwendung von Flink sondern geht auf Flink generell ein. In solchen generellen Hilfestellungen wird zumeist erklärt, wie Flink arbeitet und wozu es angewendet werden kann, wobei diese Art von Hilfestellungen kaum einen Mehrwert gegenüber der Flink-eigenen Dokumentation bietet. Sie stellen eher eine vereinfachte Form der Flink-Dokumentation dar, ohne zusätzliche und neue Informationen zu geben.

Dennoch sind vereinzelt Hilfestellungen vorhanden, welche auch auf das Vorgehen in der Praxis eingehen und erklären, wie Flink in der Praxis anwendet werden kann. Nachfolgend wird auf zwei Hilfestellungen beispielhaft eingegangen.

Eine der gefundenen Hilfestellungen wurde von *data Artisans*, dem ursprünglichen Flink-Entwicklungsteam (vor der Übernahme durch die *Apache Software Foundation*), erstellt und ist damit sehr vielversprechend. Diese Hilfestellung ist als „Training“ aufgebaut, welches Erklärungen und Übungen enthält (data Artisans, a). Die Erklärungen sind sehr umfassend und zeigen zum Beispiel, wie Flink eingerichtet wird oder welchen Nutzen einzelne Methoden haben und wie diese angewendet werden können (siehe data Artisans, d,b). Darunter befinden sich immer wieder kleine Quellcode-Beispiele, die praktisch verwendet werden können.

In den Übungen kann das Gelernte umgesetzt werden. Dafür werden Testdaten zur Verfügung gestellt, mit welchen eine Aufgabe umgesetzt werden soll (data Artisans, c). Zur Überprüfung der Richtigkeit sind Musterlösungen vorhanden. Ein Nachteil an dieser Hilfestellung ist, dass die Erklärungen nur als digitale Präsentationsfolien vorhanden sind, die mutmaßlich als Basis für einen Vortrag dienen. Daher bestehen die Erklärungen zum Teil nur aus wenigen Stichwörtern und sind nicht detailliert genug.

Ähnlich gut geeignet ist eine Hilfestellung des Entwicklers *Philip Wagner*. In seiner Hilfestellung wird anhand eines Beispiels erklärt, wie ein Projekt mit Flink umgesetzt werden kann. Das Ziel seines Beispielprojekts ist die Analyse von Wetterdaten mehrerer Wetterstationen mithilfe von Flink (Wagner, 2016). Anhand dieses Projekts werden alle notwendigen Schritte erläutert, des Weiteren wird stets der vollständige Quellcode zu den einzelnen Schritten gezeigt. Das Gesamtprojekt kann außerdem auf der Plattform *GitHub* eingesehen werden (Wagner). Diese Form der Hilfestellung bietet viele Anregungen für eigene Flink-Projekte und zeigt, wie sich ein Projektziel mit Flink umsetzen lässt. Allerdings ist dafür ein Grundlagenwissen von Flink empfehlenswert, da nicht alle im Projekt verwendeten Flink-Methoden und -Klassen ausführlich erklärt werden. Oft wird lediglich auf die Flink-Dokumentation verwiesen.

Neben diesen zwei Beispielen waren keine weiteren praxisorientierten, externen Hilfestellungen zu Flink im Internet aufzufinden.

Flink im Portal Stack Overflow Dies spiegelt sich auch in dem Portal *Stack Overflow* wider. Stack Overflow ist ein hoch frequentiertes Frage-Antwort Portal für Entwickler, in welchem diverse Fragen zu verschiedenen Frameworks und Programmiersprachen gestellt werden können. Die Antworten der Community sind oft sehr hilfreich und enthalten zumeist anwendbare Codebeispiele.

Zum Thema Flink wurden bei Stack Overflow allerdings verhältnismäßig wenige Fragen gestellt und beantwortetet: Zurzeit sind lediglich 1304 Fragen zu dem Stichwort Flink oder Apache Flink im Portal vorhanden (Stand 24.10.2017). Zum Vergleich dient das Framework Apache Spark, welches eine ähnliche Funktionalität wie Flink bietet: Dort wurden bereits 32 638 Fragen gestellt (Stand 24.10.2017).

Dies liegt vermutlich daran, dass Flink im Vergleich zu Spark ein noch sehr junges Framework ist. Diese These bestätigt sich, wenn das ähnlich junge Framework Apache Storm betrachtet wird: Zu Storm wurden bisher 2093 Fragen auf Stack Overflow gestellt (Stand 24.10.2017).

Fazit Als Leitfaden zur Arbeit mit Flink wird der Programming-Guide angeboten. Dieser gibt einen Überblick über die Möglichkeiten von Flink und enthält praktische Programmierbeispiele, an denen sich der Entwickler orientieren kann. Des Weiteren sind die Erklärungen in einfacher Sprache verständlich verfasst.

Für ein erweitertes Verständnis von einzelnen Klassen, Methoden und Parametern bietet sich das Javadoc als Nachschlagewerk an. Das Javadoc beschreibt vollständig und ausführlich jede Klasse und Methode. Durch die Größe des Frameworks empfiehlt es sich eine IDE zu

benutzen, welche Javadoc unterstützt, dadurch werden die wichtigen Informationen aus dem Javadoc automatisch angezeigt, wenn sie benötigt werden.

Ein Teil der Anforderung A21 ist damit schon erfüllt: Für alle Klassen, Methoden und Parameter ist eine Dokumentation vorhanden.

Das Angebot an weiteren externen Hilfestellung war unbefriedigend. Die meisten externen Hilfestellungen bieten keine konkreten Hilfen zur Umsetzung von Projekten oder zur Lösung von Problemen an, sondern gehen nur auf Flink generell ein. Die zwei gefundenen, praxisorientierten Hilfestellungen sind dafür qualitativ recht hochwertig und bieten gute Beispiele aus der Praxis.

Des Weiteren sind in dem bekannten Frage-Antwort-Portal Stack Overflow vergleichsweise wenige Fragen und Antworten zum Thema Flink vorhanden. Die Vermutung liegt nahe, dass das karge Angebot von externen Hilfestellungen mit dem Alter von Flink zusammenhängt. Flink ist ein noch sehr junges Framework, das noch wenig Verbreitung findet. Dementsprechend existieren bislang nur wenige externe Hilfestellungen.

Allerdings wurde bei der Suche nach Hilfestellungen gelegentlich darauf hingewiesen, dass die Flink-Community aktiv und hilfsbereit sein soll. Diese kann über verschiedene E-Mail-Verteiler oder einen IRC-Kanal erreicht werden (Apache Software Foundation, i).

Zusammenfassend lässt sich sagen, dass die Flink-Dokumentation (inklusive Programming-Guide) die erste Anlaufstelle bei der Entwicklung mit Flink ist. Darüber hinaus ergänzen die zwei gefundenen externen Hilfestellungen das Flink-eigene Angebot gut, da diese praxisorientierter sind. Ein reichhaltigeres Angebot an externen Hilfestellungen wäre von Vorteil ist aber nicht zwingend notwendig für die Entwicklung mit Flink.

6.3 Verständlichkeit von Methoden und Parametern

In dieser Kategorie wird der Aspekt der Bezeichnung von Methoden und Parametern in Flink evaluiert. Neben der Dokumentation und den Beispielen geben die Bezeichnungen von Methoden und Parametern einen Hinweis darauf, wie diese zu verwenden sind. Eine durchdachte Beizeichnung kann dabei helfen, die Funktionsweise und das Ziel einer Methode zu verstehen, ohne die Dokumentation verwenden zu müssen.

Für diese Kategorie ist die Anforderung A22 von zentraler Bedeutung. Diese Anforderung fordert, dass die Methoden von Flink und deren Parameter selbsterklärend bezeichnet und leicht verständlich sein müssen (siehe A22, ab Seite 96).

Bei der Verwendung von Apache Flink werden hauptsächlich Methoden zur Verarbeitung von Datenströmen genutzt. Der Entwickler greift über die Schnittstelle *DataStream API* auf diese Methoden zu (siehe 2.2 ab Seite 9 und Abbildung 2.4 auf Seite 11). Methoden zur Verarbeitung von Datenströmen haben die Aufgabe, den Datenstrom zu transformieren oder selbstdefinierte Methoden auf jeden Datensatz des Stroms anzuwenden (Apache Software Foundation, b).

Dabei folgt die Bezeichnung der Methoden einem einheitlichen Schema, welches teilweise auch in anderen Programmiersprachen für die Verarbeitung von Datenströmen eingesetzt wird. Beispielsweise wurde die Methode, welche jeden Datensatz eines Stroms verarbeitet und entweder keinen, einen oder mehrere Datensätze zurückgibt, mit *flatMap* bezeichnet (Apache Software Foundation, b). Damit trägt diese Methode dieselbe Bezeichnung, welche zum Beispiel auch in der Programmiersprache Java für die gleiche Funktionalität verwendet wird (Oracle, 2016). Es ist hierbei zu beachten, dass mit *flatMap* unter Java keine Flink-Datenströme, sondern Java-Interne Datenströme verarbeitet werden. Eine gleiche Bezeichnung der Datenstrom-Methoden hat für den Entwickler den Vorteil, dass er die – aus anderen Programmiersprachen oder Frameworks bekannten – Methoden-Bezeichnungen für Datenstrom-Operationen auch in Flink nutzen kann und sich so nicht umstellen muss.

Des weiter sind die Flink-spezifischen Methoden zur komplexeren Verarbeitung von Datenströmen kurz und präzise benannt. Die meisten Bezeichnungen bestehen nur aus einem oder zwei Wörtern. Zum Beispiel wurde die Methode zur Anwendung von Zeit- und Längenfenstern (siehe *Sliding Windows* unter 2.1) mit dem Wort *window* (englisch für Fenster) bezeichnet (Apache Software Foundation, g). Diese Bezeichnung ist möglichst einfach gehalten und ist dennoch aussagekräftig im Kontext von Flink.

Außerdem stellt de Window-Methode ein gutes Beispiel für die kurze und präzise Parameterbezeichnung in Flink dar: Die Methode hat einen Parameter, der mit *assigner* (zu deutsch „Bestimmer“ oder „Markierer”) bezeichnet ist und vom Typ *WindowAssigner* sein muss (Apache Software Foundation, g). Die Bezeichnung des Parameters fasst damit alles zusammen, was für den Entwickler wichtig ist: Die Methode erwartet als Parameter einen Wert, der das Zeit- oder Längenfenster bestimmt beziehungsweise markiert.

Die Methode lässt dabei offen, ob der Parameter *assigner* ein Zeitfenster oder Längenfenster enthalten soll. Tatsächlich lassen sich auf die Methode beide Arten von Fenstern anwenden. Der Parameter muss lediglich eine Implementation des Interfaces *WindowAssigner* enthalten (Apache Software Foundation, g), die Methode ist also polymorph. Damit ist die Window-Methode kein Einzelfall, denn die meisten Methoden von Flink sind polymorph.

Polymorphie hat den großen Vorteil, dass dieselbe Methode für verschiedene Implementationen genutzt werden kann. Dadurch wird die Arbeit des Entwicklers vereinfacht, da nur eine Methode für unterschiedliche Zwecke genutzt werden kann. Außerdem wird durch Polymorphie ermöglicht, selbst entwickelte Implementationen mit eigenen Funktionalitäten einzubinden.

Bei der Window-Methode kann zwischen mehreren vorgefertigten Implementationen von Flink gewählt werden oder eben eine selbst entwickelte *WindowAssigner*-Implementation verwendet werden. Die vorgefertigten *WindowAssigner* lassen sich über statische Methoden einbinden. Ein gleitendes Zeitfenster kann zum Beispiel durch den Aufruf der statische Methode *SlidingProcessingTimeWindows.of(Time size, Time slide)* definiert werden. Der Parameter *size* definiert die Länge beziehungsweise die Dauer des Zeitfensters und der Parameter *slide* definiert, wie oft ein neues Zeitfenster gestartet wird. Ein vollständiger Aufruf kann dann folgendermaßen aussehen *SlidingProcessingTimeWindows.of(Time.seconds(10), Time.seconds(5))*. Anhand dieses Aufrufs ist erneut zu sehen, dass die Bezeichnung von Klassen, Methoden und Parametern auf das Notwendigste beschränkt wurde.

Weiter ist zu bemerken, dass unter den Bezeichnungen von Klassen, Methoden und Parametern keine Abkürzungen verwendet wurden. Die Wörter in den Bezeichnungen werden immer vollständig ausgeschrieben, dadurch ist Quellcode, der auf Basis von Flink entwickelt wurde, gut lesbar und verständlich.

Fazit Die Bezeichnungen von Klassen, Methoden und Parametern ist positiv aufgefallen. Sie verläuft nach einem klaren Schema und es werden meist kurze, prägnante und präzise Klassen-, Methoden- und Parameterbezeichnungen verwendet. Hierbei ist besonders aufgefallen, dass keine Abkürzungen verwendet werden.

Die Methoden zur Verarbeitung von Datenströmen sind genau wie in anderen Programmiersprachen bezeichnet worden. Dadurch muss sich der Entwickler nicht umstellen.

Außerdem sind viele Methoden in Flink polymorph, dadurch wird die Anzahl der Methoden reduziert, da eine Methode für verschiedene Zwecke genutzt werden kann. Des Weiteren bietet sie dem Entwickler die Möglichkeit, eigene Implementationen zu programmieren und damit Flink zu erweitern.

Flink erfüllt damit die gestellten Anforderung an die Bezeichnung von Klassen, Methoden und Parametern (Anforderung A22).

6.4 Belastungstest

Diese Kategorie widmet sich den Anforderungen A17 und A18, beide beziehen sich darauf, dass Flink auch hohen Belastungen standhält. A17 fordert, dass die Verarbeitung eines Ereignisses nicht länger als 10 Millisekunden dauert und A18 fordert, dass Flink bis zu 10000 Ereignisse pro Sekunde verarbeiten kann (siehe Anhang Anhang A ab Seite 96).

Aufbau Um diese Anforderungen zu Überprüfen, wurde ein Belastungstest durchgeführt. Dazu wurde eine Test-Ereignisquelle erstellt, die eine vorgegebene Anzahl von Test-Ereignissen generiert und diese mit einem Sendezeitstempel versieht. Die generierten Test-Ereignisse werden von einer Senke empfangen, die das Ereignis mit einem weiteren Zeitstempel zur Empfangszeit versieht. Aus der Differenz von Sende- und Empfangszeitstempel berechnet sich die Dauer der Verarbeitung eines Test-Ereignisses. Das Test-Ereignis hat einen in der Größe einstellbaren Platzhalter, um eine Nutzlast zu simulieren.

Der Flink-Testjob ist so aufgebaut, dass lediglich die Testquelle mit der Testsenke verbunden wird. Es findet also keine Transformation der Ereignisse statt. Dadurch wird sichergestellt, dass nur die interne Verarbeitung der Ereignisse durch Flink gemessen wird. Die Start- und Endzeit des Testjobs wurde jeweils festgehalten, um die Gesamtdauer der Jobausführung zu ermitteln. Es ist zu beachten, dass die Gesamtdauer der Job-Ausführung nicht mit der Verarbeitungsdauer der Ereignisse gleichzusetzen ist.

Der Belastungstest wurde mit verschiedenen Testparametern (Anzahl von Ereignissen und Größe der Nutzlast) durchgeführt. Jeder Belastungstest wurde mehrmals mit denselben Parametern wiederholt. Die Test-Ergebnisse sind in der Tabelle 6.1 einzusehen. Dabei ist zu beachten, dass ein Test in der Tabelle jeweils den Durchschnitt mehrerer Testwiederholungen repräsentiert.

Auswertung In der Tabelle 6.1 auf der nächsten Seite ist zu sehen, dass sich die Anzahl der Ereignisse antiproportional zu der Verarbeitungsdauer pro Ereignis verhält: Je niedriger die Anzahl von Ereignissen desto höher die Verarbeitungsdauer pro Ereignis. Flink wird also mit einer höheren Anzahl von Ereignissen effizienter, was vermutlich darauf zurückzuführen ist, dass der Overhead im Verhältnis zur Anzahl der Ereignisse kleiner wird, je mehr Ereignisse verarbeitet werden.

Des Weiteren ist zu erkennen, dass sich die Größe der Nutzlast sowohl auf die Verarbeitungsdauer pro Ereignis als auch auf die Gesamtdauer des Jobs auswirkt. Eine größere Nutzlast sorgt dabei für eine Erhöhung beider Messparameter.

Nr.	Anzahl Ereignisse	Größe Nutzlast	Gesamtdauer (Job-Ausführung)	Ereignisse pro Sekunde (Job-Ausführung)	Gesamtdauer (Verarbeitung)	Verarbeitungsdauer pro Ereignis
1	100	1 kB	1.33 s	75	132.59 ms	1325.91 µs
2	1000	1 kB	1.36 s	733	142.20 ms	142.20 µs
3	10 000	1 kB	1.39 s	7186	152.80 ms	15.28 µs
4	100 000	1 kB	1.75 s	57 258	172.34 ms	1.72 µs
5	1 000 000	1 kB	2.37 s	428 768	395.85 ms	0.40 µs
6	10 000 000	1 kB	8.91 s	1 123 943	1299.54 ms	0.13 µs
7	100	10 kB	1.35 s	74	138.17 ms	1381.74 µs
8	1000	10 kB	1.37 s	728	146.75 ms	146.75 µs
9	10 000	10 kB	1.71 s	5865	448.03 ms	44.80 µs
10	100 000	10 kB	2.07 s	48 330	378.57 ms	3.79 µs
11	1 000 000	10 kB	4.83 s	206 994	1669.19 ms	1.67 µs
12	10 000 000	10 kB	31.82 s	314 299	13 020.33 ms	1.30 µs
13	10 000	100 kB	1.95 s	5117	341.38 ms	34.14 µs
14	100 000	100 kB	4.36 s	22 920	1542.37 ms	15.42 µs
15	1 000 000	100 kB	23.90 s	41 848	11 870.35 ms	11.87 µs

Tabelle 6.1: Ergebnisse des Belastungstests

Fazit Anhand des Belastungstest ist deutlich geworden, dass Flink die Anforderung A17 (Verarbeitung eines Ereignis in unter 10 Millisekunden) erfüllt. Anhand Tabelle 6.1 ist zu sehen, dass Flink selbst bei einer niedrigen Zahl von Ereignissen nur knapp über eine Millisekunde braucht um ein Ereignis zu verarbeiten (siehe Zeile 1 und 7). Bei hoher Ereignisanzahl und einer geringer Nutzlast sind sogar Verarbeitungszeiten von unter einer Mikrosekunde möglich (siehe Zeile 5 und 6).

Anforderung A18 hingegen wurde nicht vollständig erfüllt. Gefordert war, dass Flink bis zu 10 000 Ereignisse pro Sekunde verarbeiten kann. Der Belastungstest hat gezeigt, dass dies nur für eine höhere Anzahl von Ereignissen gilt (siehe Zeile 3+4, 9+10 und 13+14). Bei einer sehr geringen Anzahl von Ereignissen ist der Overhead im Verhältnis zur Ereignisanzahl so groß, dass Flink zum Teil nur 75 Ereignisse pro Sekunde verarbeitet (siehe Zeile 1).

6.5 Entwicklung mit Apache Flink

In dieser Kategorie werden Aspekte evaluiert, die bei der Entwicklung mit Flink aufgefallen sind. Die Evaluierung orientiert sich an den übrigen Flink-Anforderungen aus 3.6. Es sollen dabei alle übrigen Anforderungen an Flink überprüft werden.

Zunächst ist hier zu sagen, dass Flink die Anforderung A14 erfüllt hat. In dieser wurde gefordert, dass Flink das Empfangen, Filtern, Analysieren, Konvertieren und Weiterleiten von Ereignissen ermöglicht (siehe A14, ab Seite 96). Diese Anforderung war essenziell für

die Entwicklung von EventAlert, da eine Umsetzung nur mit Erfüllung dieser Anforderung möglich war. Nachfolgend ist bereits bei der Entwicklung des Prototyps zu sehen, dass Flink diese elementare Anforderung erfüllt.

Entwicklung eines Prototyps Nach der Einrichtung des Projekts war die Entwicklung mit Flink im Allgemeinen ohne Probleme möglich. Es hat nur wenige Tage Entwicklungszeit gefordert, bis ein funktionierender Prototyp von EventAlert mit Flink realisiert wurde.

Der Prototyp konnte Änderungen in einem IMAP-Konto über eine selbst entwickelte SourceFunction erkennen und an Flink übergeben. Des Weiteren hat der Prototyp über einen Flink-Job eine rudimentäre Filterung der Ereignisse vorgenommen und diese an eine ebenfalls selbst entwickelte SinkFunction übergeben. Die Senke hatte allerdings noch keine Verbindung zu einem LED-Streifen, sondern hat lediglich das eingehende Ereignis in der Konsole ausgegeben.

Die Hauptarbeit während der Entwicklung lag nicht in der Verwendung von Flink sondern vielmehr in Programmierung eines IMAP-Clients. Die Anbindung des Clients an Flink war einfach, da lediglich ein Interface implementiert werden musste. Die Senke zur Ausgabe der eingehenden Ereignisse war ähnlich einfach umzusetzen.

Flink hat damit die Anforderung A19 erfüllt, welche gefordert hat, dass ein *EventAlert*-Prototyp in unter fünf Tagen auf Basis von Flink entwickelt werden kann (siehe A19, ab Seite 96).

Einfache Verbindung von Quelle und Senke Der im Prototypen entwickelte Flink-Job, welcher die IMAP-Quelle mit einer Senke zur Ausgabe verbindet, wird als Exempel genutzt um die Anforderung A20 zu überprüfen. Diese hat gefordert, dass eine einfache Verbindung zwischen Quelle und Senke weniger als 20 LOC (Code-Zeilen) benötigt (siehe A20, ab Seite 96).

Tatsächlich hat das Exempel aus dem Prototypen diesen Anspruch deutlich übertroffen, denn obwohl dieser Job etwas komplexer ist, als durch die Anforderung gefordert wurde, benötigt er nur 18 LOC.

Um die Ausdrucksstärke und Kompaktheit von Flink zu demonstrieren ist nachfolgend der vollständige Job aus dem Prototypen aufgeführt (siehe Quellcode 6.1).

```
1 // set up the execution environment
2 final StreamExecutionEnvironment env = StreamExecutionEnvironment.
    getExecutionEnvironment()
3         .setParallelism(1);
4 //Create source function
5 SourceFunction<MailMessageEvent> sourceFunction = new
    ImapDataSourceFunction("host","user","password");
```

```
6  //Generate LEDEffect out of MailMessage
7  DataStream<LEDEffect> ledEffectStream = env.addSource(sourceFunction
       , "ImapDataStream") //Add Source
8          .flatMap((mailMessageEvent, out) -> { //Filter sender
9          LEDEffect ledEffect;
10         if(mailMessageEvent.getSenders().contains("email@host.com"))
11                 ledEffect = new LEDEffect(0,0,255,0, 200); //200ms
                       blue LEDEffect
12         else
13                 ledEffect = new LEDEffect(0,255,0,0,200); //200ms
                       green LEDEffect
14         out.collect(ledEffect);
15 });
16 SinkFunction<LEDEffect> ledSinkFunction = new LEDSinkFunction(); //
       Create sink function
17 ledEffectStream.addSink(ledSinkFunction).name("LED_Sink"); //Add
       Sink
18 env.execute("ImapLEDJob"); //Execute the job
```

Quellcode 6.1: Verbindung von Quelle und Senke

Es wäre sogar möglich, diesen Job weiter zu komprimieren, zum Beispiel indem Kommentare entfernt werden oder die Filterung in eine Methode ausgelagert wird.

Der Anspruch an Flink aus Anforderung A20, dass eine Verbindung von Quelle und Senke weniger als 20 LOC benötigt, wurde damit übertroffen.

Unterstützung von Lambda-Ausdrücken Außerdem ist durch das Code-Beispiel deutlich geworden, dass Flink die Verwendung von Lambda-Ausdrücken unterstützt (siehe Zeile 8 bis 15 in 6.1 auf der vorherigen Seite). Unter anderem wird in der Dokumentation von Flink auch explizit auf die Unterstützung von Lambda-Ausdrücken in Java hingewiesen (siehe Apache Software Foundation, c). Die Möglichkeit der Verwendung von Lambda-Ausdrücken wurde in Anforderung A23 gefordert und ist damit erfüllt.

Keine Unterscheidung zwischen eigenen und eingebauten Komponenten Es wurde erwartet, dass sich selbst entwickelte Programmteile nicht von den in Flink eingebauten Programmteilen unterscheiden. Diese Erwartung wurde anhand von Anforderung A24 präzisiert: Der Entwickler verwendet dieselben Interfaces von Flink, die auch von in Flink eingebauten Komponenten genutzt werden (siehe A24, ab Seite 96).

Dass diese Erwartung erfüllt wurde, zeigt sich bei Flink an mehreren Stellen. Wie schon erwähnt wurde, sind die meisten Flink-Methoden polymorph. Dies gibt dem Entwickler die Möglichkeit, eigene Implementationen nahtlos in Flink zu integrieren. Dabei können dieselben Interfaces benutzt werden, die auch eingebaute Komponenten benutzen.

So implementiert zum Beispiel die in Flink eingebaute *TwitterSource* das Interface *RichSourceFunction* (Apache Software Foundation, l), welches eine Erweiterung des schon erwähnten SourceFunction-Interfaces ist. Dasselbe Interface (RichSourceFunction oder SourceFunction) wird auch implementiert, wenn eigene Quellen entwickelt werden.

Die Polymorphie setzt sich im ganzen Flink-Framework fort, und der Entwickler kann stets die Flink-Interfaces implementieren, die auch von in Flink eingebauten Komponenten implementiert werden.

Auftreten von Fehlern Weiter wurde von Flink gefordert, dass es stets fehlerfrei und korrekt arbeitet. Falls Fehler auftreten, sollen diese nur auf den eigenen Quellcode zurückzuführen sein (siehe A15, ab Seite 96). Außerdem soll ein Fehler im eigenem Quellcode nicht zum Absturz des Gesamtsystems führen (siehe A16, ab Seite 96).

Während der Arbeit mit Flink ist kein Fehler im Framework aufgetreten: Der Ursprung von Fehlern lag stets im eigenem Quellcode. Dies war zum Beispiel in der anfänglichen Version der IMAP-Quelle der Fall, dort konnte ein Fehler darauf zurückgeführt werden, dass nicht alle Parameter der Quelle serialisierbar waren. Hier hätte die Fehlermeldung von Flink etwas detailreicher sein können, denn der Ursprung des Fehlers konnte nur durch das Debuggen der Ausführung gefunden werden.

An diesem Fehler hat sich auch gezeigt, dass die Gesamtausführung nicht beeinträchtigt wurde. Flink ist weiter gelaufen obwohl die Quelle nicht gestartet werden konnte. Lediglich der Job, in dem die IMAP-Quelle genutzt wurde, ist fehlgeschlagen.

Flink konnte also beide Anforderungen (A15 und A16) erfüllen.

Debuggen auf Basis von Flink Eine weitere Anforderung war, dass der Quellcode auf Basis von Flink debuggt werden kann (siehe A26, ab Seite 96).

Flink erfüllt diese Anforderungen dadurch, dass sich Programme auf Basis von Flink auch lokal – in der Entwicklungsumgebung – ausführen lassen. In dem Programming-Guide wird sogar explizit darauf hingewiesen, dass man sein Programm debuggen und testen sollte, bevor man es auf einem Cluster ausführt (siehe Apache Software Foundation, b).

Testen von Flink Es wurde gefordert, dass Flink eine Testumgebung zur Verfügung stellt, welche die vollständige Verarbeitung von Ereignissen mit Testdaten simulieren kann (siehe A28,

ab Seite 96). Außerdem wurde gefordert, dass die Testumgebung mit anderen Test-Frameworks kompatibel ist (siehe A27, ab Seite 96).

Leider wurden diese Anforderungen nicht zur vollständigen Zufriedenheit erfüllt. Flink bietet in der hier genutzten Version (1.3) lediglich Werkzeuge zum Testen an. Darunter ist zum Beispiel ein Werkzeug, welches einen Datenstrom (DataStream) so umwandelt, dass man über die Ergebnisse iterieren kann. Weitere Werkzeuge ermöglichen das Erstellen von Datenströmen aus einer Liste von Elementen (Apache Software Foundation, b). Mit diesen Werkzeugen lässt sich zwar eine Verarbeitung von Ereignissen simulieren und testen, jedoch sind dies eben nur Werkzeuge, sie stellen keine richtige Testumgebung dar.

Allerdings hat sich während der Entwicklung von EventAlert herausgestellt, das elegante Tests auch ohne Flink-Testumgebung geschrieben werden können. Dazu ist lediglich eigener Entwicklungsaufwand erforderlich. Dies wird dadurch ermöglicht, dass ein Flink-Job auch lokal innerhalb der JVM (Java Virtual Machine) ausgeführt werden kann, sodass keine zwingende Notwendigkeit einer Flink-Testumgebung besteht.

Zum Beispiel wurde für den Test von EventAlert ein Flink-Job entwickelt, der während der Test-Ausführung dynamisch erstellt und nebenläufig in einem Thread gestartet wird. Jeder Test kann Ereignisse an die Quelle des Jobs übergeben und anschließend überprüfen, welche Ereignisse an der Senke des Jobs angekommen sind. Dies ermöglicht den Test einer Flink-Ausführung innerhalb einer JUnit-Testklasse.

Außerdem werden die zukünftigen Versionen von Flink eine Testumgebung enthalten, welche mutmaßlich die Anforderungen A27 und A28 erfüllen würde. Bereits in der nächsten Version von Flink (1.4) soll eine Testumgebung enthalten sein, dies geht jedenfalls aus der Dokumentation von Flink 1.4 hervor (siehe Apache Software Foundation, h). Die Version 1.4 wurde am 12.12.2017 – noch während der Entwicklung dieser Arbeit – veröffentlicht (Krettek und Winters, 2017).

Connectors zu Fremdsystemen Flink bietet eingebaute Quellen und Senken zu externen Fremdsystemen (z.B. zu Apache Kafka) in Form von sogenannten Connectors an (siehe 2.2 auf Seite 11). Im Rahmen von EventAlert werden die Connectoren für Twitter und für Kafka genutzt. Der Twitter-Connector bietet nur eine Quelle an, für den Kafka-Connector gibt es sowohl Quelle als auch Senke.

Die Anforderung (siehe A25, ab Seite 96) hat gefordert, dass keine Fachkenntnisse des Fremdsystems erforderlich sind, um eine Ereignisquelle über einen Connector einzubinden. Erwartet wurde, dass lediglich eine der zur Verfügung gestellten SourceFunction eingebunden werden muss.

Das Einbinden der Kafka-Quelle hat problemlos funktioniert. Die Flink-Dokumentation zum Kafka-Connector ist ausführlich und hilfreich (siehe Apache Software Foundation, h), sodass die Verbindung zum externen Kafka-System schnell hergestellt war, ohne Details von Kafka zu kennen.

Der Twitter-Connector war hingegen deutlich komplizierter einzubinden. In der Flink-Dokumentation ist ein kurzes Beispiel aufgeführt, wie eine Twitter-Quelle eingebunden werden kann und welche Authentifizierungsparameter dafür benötigt werden (siehe Apache Software Foundation, a). Des Weiteren wird auf ein ausführlicheres Beispiel verwiesen, welches zeigt, wie man die ankommenden JSON-Objekte von Twitter nach Sprache filtert und die Wörter eines Tweets als Datenstrom erhält.

Jedoch wird nicht erklärt, wie die Twitter-Quelle gesteuert werden kann. Diese bietet nämlich die Möglichkeit, die Tweets schon auf der Seiten von Twitter einzuschränken, sodass man nur eine Teilmenge aller Tweets erhält. Die Bandbreite an Einschränkungen ist dabei sehr weitreichend, unter anderem kann eine Einschränkung nach Schlüsselwörtern oder Standorten stattfinden.

Die Dokumentation von Flink weist zwar darauf hin, dass Einschränkungen auf Seiten von Twitter über einen *EndpointInitializer* möglich sind. Allerdings wird nicht erklärt, wie dies Einschränkungen konfiguriert werden können. Diese Informationen mussten selbst recherchiert werden. In der Dokumentation der Twitter-API werden vorteilhafterweise alle Einschränkungsmöglichkeiten aufgelistet (siehe Twitter Inc.). Die Schwierigkeit besteht dann nur noch darin, die Einschränkungen mit dem Twitter-Connector umzusetzen.

Zusammenfassend lässt sich sagen, dass die jetzige Dokumentation zum Twitter-Connector nicht hilfreich ist. Es wird durchaus Fachwissen des Fremdsystems benötigt um den Twitter-Connector mit vollem Funktionsumfang einzubinden. Hier würde ein Beispiel, welches auf die Einschränkungen auf Seiten von Twitter zeigt, das notwendige Fachwissen zum Fremdsystem eingrenzen. Anders ist dies bei dem Kafka-Connector, dort waren keine Fachkenntnisse notwendig, und die Dokumentation war hilfreich.

Anforderung A25 konnte somit nur zum Teil erfüllt werden.

Fazit Die Entwicklung mit Flink war weitgehend problemlos möglich. Probleme oder Schwierigkeiten, die Aufgetreten sind, wurden nur selten durch das Flink-Framework verursacht.

Während der Entwicklung des Prototyps konnte Flink schon einige der gestellten Anforderungen erfüllen. Beispielweise wurde schnell deutlich, dass selbst eine komplexe Verbindung von Quelle und Senke in nur 18 LOC Platz findet. Flink stellte hierdurch seine Ausdrucksstärke

unter Beweis, zu welcher unter anderem auch die Unterstützung von Lambda-Ausdrücken beigetragen hat. Dadurch konnte Flink die Anforderung A20 und A23 erfüllen.

Des weiteren war der besagte Prototyp nach einer Entwicklungszeit von weniger als fünf Tagen funktionsfähig, sodass Flink auch die Anforderung A19 erfüllen konnte.

Außerdem sorgt die Polymorphie vieler Flink-Methoden dafür, dass der Entwickler nahtlos eigene Funktionalität in Flink integrieren kann. Selbst entwickelte Programmteile unterscheiden sich dadurch nicht von den in Flink eingebauten Komponenten. Die Anforderung A24 konnte dadurch erfüllt werden.

Zudem lief Flink stets fehlerfrei und robust. Fehler wurden immer auf den eigenen Quellcode zurückgeführt und haben nicht zum Absturz des Gesamtsystems geführt (Anforderungen A15 und A16) .

Flink konnte auch beim Debuggen überzeugen: Durch die lokale Ausführung von Flink ist das Debuggen genauso möglich wie in jedem anderen lokalen Java-Programm. Flink stellt also kein Hindernis beim Debuggen dar und erfüllt dadurch Anforderung A26.

Die Anforderungen an das Testen von Flink konnten nicht vollständig erfüllt werden: Flink bietet keine eigene Testumgebung an. Flink stellt zwar Testwerkzeuge zur Verfügung, allerdings reicht dies nicht aus, um die Anforderungen A27 und A28 zu erfüllen. Jedoch wurde während der Entwicklung deutlich, dass für diese Anwendung keine Testumgebung erforderlich ist, um elegante Tests zu schreiben.

Auch bei den Connectoren konnte Flink die Anforderung nicht vollständig erfüllen. Es wurde gefordert, dass das Einbinden von Flink-Connectoren keine Fachkenntnisse des Fremdsystems benötigt. Dies traf auf den Kafka-Connector auch zu, jedoch war die Dokumentation des Twitter-Connectors unzureichend, sodass selbst recherchiert werden musste, wie ein Twitter-Connector richtig konfiguriert wird. Dies führt nur zur Teilerfüllung von Anforderung A25.

Flink hat fast alle Anforderungen erfüllt und zum Teil sogar übertroffen. Die Ausdrucksstärke von Flink ist hervorragend, sodass komplexe Operationen hinter einem eleganten, kompakten und lesbaren Quellcode verschwinden. Eigene Funktionalitäten können leicht über Polymorphie eingebunden werden, sodass eigene und eingebaute Komponenten fließend miteinander interagieren. Dabei bietet Flink beim Debuggen den gleichen Komfort wie ein kleines Java-Programm.

Eine vollwertige Flink-Testumgebung war in der Version 1.3 noch nicht vorhanden, wird aber bereits in der nächsten Version integriert.

Die Dokumentation des Twitter-Connectors wurde für unzureichend befunden. Sie wirkt momentan unvollständig und geht nur auf einen Bruchteil des Möglichen ein.

7 Zusammenfassung und Ausblick

Dieses Kapitel fasst die Ergebnisse dieser Bachelorarbeit zusammen und gibt einen Ausblick auf eine mögliche Weiterentwicklung.

7.1 Zusammenfassung der Ergebnisse

Das Ziel dieser Arbeit war die kritische Auseinandersetzung mit dem Framework Apache Flink. Hierzu sollte eine Beispielanwendung programmiert werden, die auf Basis von Flink IoT-Ereignisse analysiert und diese über eine smarte LED-Anzeige visualisiert. Außerdem sollten Anforderungen an Flink gestellt werden. Anhand der Beispielanwendung und den Anforderungen sollten die eigenen Erfahrungen mit Flink dokumentiert und mögliche Verbesserungen aufgezeigt werden. Des Weiteren sollte dabei auf die Besonderheiten in der soft- und hardwaretechnischen Umsetzung eingegangen werden.

Zu Beginn der Arbeit wurden – nach der Einleitung – die umgebenden Themen in den Grundlagen erläutert (Kapitel 2 ab Seite 4). Anhand dieser Grundlagen wurden die Anforderungen analysiert (Kapitel 3 ab Seite 20). Zunächst wurde dafür ein Arbeitstitel für die zu programmierende Software vergeben: EventAlert. Zur Einführung der Analyse wurden drei mögliche Szenarien aufgeführt (3.1 ab Seite 20). Anhand dieser Szenarien wurden die Anforderungen an EventAlert (3.2 ab Seite 23) und dessen benötigte Hardware entwickelt (3.5 ab Seite 32). Die Anforderungen an EventAlert und die Hardware zielen auf die Möglichkeit zur Weiterentwicklung ab. Außerdem wurden die Anforderungen an Apache Flink entwickelt (3.6 ab Seite 35), auf welchen die kritische Auseinandersetzung basiert.

Durch die Ermittlung der Anforderungen wurde deutlich, was EventAlert können soll und welche Anforderungen an Flink bestehen. Auf dieser Basis wurde ein Entwurf erstellt (Kapitel 4 ab Seite 40). Zunächst wurde sich im Entwurf für eine ereignisgesteuerte Architektur als Programmiermodell entschieden (4.1 ab Seite 40). Anschließend wurde die Software in zwei Komponenten aufgeteilt: EventAlert (4.2.1 ab Seite 42) und LED-Anzeige (5.1.3 ab Seite 63),

welche jeweils weitere Unterkomponenten haben. Die Aufgabe oder Funktion der Komponenten mit ihren Unterkomponenten wurde mithilfe von drei Komponenten-Diagrammen (Abbildungen 4.1 auf Seite 41, 4.2 auf Seite 43 und 4.4 auf Seite 46) veranschaulicht. Zum Schluss wurde im Entwurf auf die Besonderheiten der Kommunikation zwischen den Komponenten eingegangen (4.3 ab Seite 48), dabei gibt es vor allem bei der Kommunikation innerhalb der LED-Anzeige einige Besonderheiten.

Auf Basis des Entwurfs wurde im Kapitel Realisierung und Test (Kapitel 5 ab Seite 53) die Entwicklung von Soft- und Hardware umgesetzt. Anhand der Aufteilung in Komponenten wurde in der softwaretechnischen Umsetzung (5.1 ab Seite 53) auf die Besonderheiten und Funktionsweisen jedes Komponenten eingegangen. Zudem findet für die Komponente Event-Alert (5.1.1 ab Seite 53) eine Überprüfung der Anforderungen statt (5.1.2 ab Seite 61). Zu der Komponente LED-Anzeige (5.1.3 ab Seite 63) wurden Abbildungen erstellt, welche die Kommunikation innerhalb der LED-Anzeige grafisch veranschaulicht (Abbildungen 5.4 auf Seite 64 und 5.5 auf Seite 65). Außerdem wurde zu beiden Hauptkomponenten beschrieben, wie diese getestet werden.

Anschließend wurde – nach der softwaretechnischen Umsetzung – die hardwaretechnische Umsetzung näher erläutert (5.2 ab Seite 69). Es wurde sich für einen Arduino als LED-Controller entschieden (5.2.1 ab Seite 69) und ein passender LED-Streifen ausgewählt (5.2.2 ab Seite 72). Anschließend wurde auf die Besonderheiten der Verbindung zwischen den Hardware-Komponenten eingegangen.

Im letzten Kapitel dieser Arbeit wurden sich kritisch mit Flink auseinandergesetzt und die Erfahrungen während der Entwicklung evaluiert (Kapitel 6 ab Seite 75). Dazu wurde das Kapitel fünf Kategorien unterteilt. Die Grundlage für die Auseinandersetzung waren die Anforderungen an Flink, welche im Einzelnen überprüft wurden. Flink konnte die meisten Anforderungen erfüllen. Im Fazit zu jeder Kategorie wurden die Erfahrungen noch einmal zusammengefasst.

7.2 Ausblick

In dieser Arbeit wurde auf Basis des Frameworks Apache Flink eine Anwendung entwickelt, welche verschiedene IoT-Ereignisse visualisiert. Zur Visualisierung wurde ein LED-Streifen verwendet. Die Software und Hardware wurde dabei darauf ausgelegt, dass sie weiter entwickelt werden können. Zukünftig könnte die Software und Hardware in verschiedene Richtungen weiterentwickelt oder verbessert werden.

Zum einen könnte die Software für weitere IoT-Quellen geöffnet werden. Dazu müssten lediglich weitere Quellen angeschlossen werden. So könnten neben Telegram und E-Mail z.B. weitere Messaging-Dienste angeschlossen werden. Neben den Messaging-Diensten wäre es von Nutzen, wenn Geräte im Haushalt, wie z.B. die Haustürklingel, IoT-Ereignisse generieren könnten. Denkbar wäre auch die Integration von externen Wetter-Daten oder Verkehrsinformationen, die über die LED-Anzeige visualisiert werden.

Zum anderen benötigt die Software eine GUI (Grafische Benutzeroberfläche) zur Konfiguration von Quellen, Filterregeln und Reaktionen. Mithilfe der GUI könnten auch Funktionen gesteuert werden, die nicht von IoT-Quellen abhängig sind. So könnte zum Beispiel ein Wecker entwickelt werden, der den LED-Streifen zur Weckzeit zum Leuchten bringt oder einen Sonnenaufgang simuliert.

Des Weiteren könnte auch die Hardware inklusive ihrer softwaretechnischen Steuerung weiterentwickelt werden. Es könnten weitere Reaktionen hinzugefügt werden, wie zum Beispiel das Abspielen eines akustischen Signals. Außerdem ließen sich weitere Geräte zur Visualisierung an den Arduino anschließen. Denkbar wäre es zum Beispiel, einen kleinen Monitor an den Arduino anzuschließen, um Ereignisse zu visualisieren.

Weiter wäre es auch möglich den Arduino nicht nur zur Visualisierung zu nutzen, sondern ihn auch zur Quelle von IoT-Ereignissen umzufunktionieren. Zum Beispiel könnte ein Temperatursensor an den Arduino angeschlossen werden, um mit EventAlert auf Änderungen der Umgebungstemperatur zu reagieren.

Außerdem könnte der Arudino als LED-Steuerung noch verbessert werden. Momentan wird eine Verbindung zu einem Computer benötigt um den LED-Streifen zu steuern. Es wäre jedoch möglich den Arduino mit WLAN auszustatten, sodass die Steuerung keine mechanische Verbindung zu einem Computer brauchen würde. Dadurch könnte der LED-Streifen zur Visualisierung gewissermaßen mobil werden.

Bei den diversen Möglichkeiten zur Weiterentwicklung muss zukünftig auch der Sicherheitsaspekt beachtet werden. Momentan sollte die Software und Hardware nur in einem privaten, gesichertem Netzwerk verwendet werden, da die Software nicht als sicher eingestuft werden kann. Hierzu könnten die Sicherheitsmechanismen der Frameworks Apache Kafka und Apache Flink zum Einsatz kommen, beide lassen sich so Konfigurieren, dass der Authentifizierungsdienst Kerberos genutzt werden kann. Über diesen könnte eine Authentifizierung der einzelnen Komponenten stattfinden und diese könnten verschlüsselt kommunizieren. Des weiteren müssen die Konfigurationsdateien der entwickelten Ereignis-Quellen gesichert werden. Diese könnte man zum Beispiel verschlüsseln, sodass ein Passwort vom Nutzer benötigt wird, um eine Quelle zu starten.

A Übersicht aller Anforderungen

Diese Übersicht beinhaltet alle in Kapitel 3 gestellten Anforderungen. Es wird zwischen funktionalen und nicht funktionalen Anforderungen unterschieden, außerdem werden die Anforderungen an Apache Flink in einer eigenen Kategorie angeführt. Die Anforderungen sind fortlaufend nummeriert.

Innerhalb dieser Übersicht wurde für die nicht funktionalen Anforderungen noch eine Zuordnung zu den jeweiligen Komponenten (EventAlert und LED-Anzeigen-Steuerung) vorgenommen.

Funktionale Anforderungen an EventAlert

[A1] Der Anwender kann individuelle Reaktionen (z.B. einen Leuchteffekt) für Ereignisse oder Ereignisfolgen festlegen. Die Festlegung erfolgt anhand von Filterregeln, welche auf die Daten und Metadaten des jeweiligen Ereignisses angewendet werden. (Definiert unter Punkt 3.2 auf Seite 24 / Überprüft unter Punkt 4.2.1 auf Seite 44)

[A2] Der Anwender kann seine Filterregeln priorisieren. Die Reaktionen von Filterregeln mit hoher Priorität werden bevorzugt behandelt. (Definiert unter Punkt 3.2 auf Seite 24 / Überprüft unter Punkt 4.2.1 auf Seite 44)

Nicht funktionale Anforderungen an EventAlert

[A3] Die Komponenten von EventAlert sind voneinander vollständig unabhängig. (Definiert unter Punkt 3.2 auf Seite 26 / Überprüft unter Punkt 4.1 auf Seite 41)

[A4] Alle Schnittstellen, die zum Hinzufügen einer neuen Ereignisquelle benötigt werden, sind dokumentiert (JavaDoc). (Definiert unter Punkt 3.2 auf Seite 26 / Überprüft unter Punkt 5.1.2 auf Seite 61)

[A5] Der Aufwand, eine neue und nicht komplexe Ereignisquelle hinzuzufügen, beträgt weniger als zwei Stunden. (Definiert unter Punkt 3.2 auf Seite 26 / Überprüft unter Punkt 5.1.2 auf Seite 62)

[A6] Der Aufwand, einen neuen und nicht komplexen Reaktionsempfänger anzubinden, beträgt weniger als zwei Stunden. (Definiert unter Punkt 3.2 auf Seite 26 / Überprüft unter Punkt 5.1.2 auf Seite 63)

Nicht funktionale Anforderungen an die Steuerung der LED-Anzeige

[A7] Die Steuerung kann Leuchtfarbe und -stärke des LED-Streifens steuern. (Definiert unter Punkt 3.5 auf Seite 33 / Überprüft unter Punkt 5.2.1 auf Seite 71)

[A8] Die Steuerung wird über eine standardisierte Hardwareschnittstelle mit dem ausführenden Computer verbunden. (Definiert unter Punkt 3.5 auf Seite 33 / Überprüft unter Punkt 5.2.1 auf Seite 71)

[A9] Die Steuerung hat eine flexible Verbindung zum LED-Streifen, sodass sich verschiedene Arten von LED-Streifen anschließen lassen. (Definiert unter Punkt 3.5 auf Seite 34 / Überprüft unter Punkt 5.2.1 auf Seite 71)

[A10] LED-Streifen und Steuerung haben eine gemeinsame Stromversorgung. (Definiert unter Punkt 3.5 auf Seite 34 / Überprüft unter Punkt 5.2.2 auf Seite 73)

[A11] Die Steuerung des LED-Streifens ist programmierbar (enthält eigene Logik) und unterstützt eine objektorientierte Programmierung. (Definiert unter Punkt 3.5 auf Seite 35 / Überprüft unter Punkt 5.2.1 auf Seite 71)

[A12] Es sind Bibliotheken für die Steuerung zum Steuern eines LED-Streifens vorhanden. Für die Programmierung der Steuerung ist kein elektrotechnisches Wissen notwendig. (Definiert unter Punkt 3.5 auf Seite 35 / Überprüft unter Punkt 5.2.1 auf Seite 71)

[A13] Es sind Bibliotheken vorhanden, welche die Kommunikation zwischen dem ausführenden Computer und der Steuerung über eine softwaretechnische Kommunikationsschnittstelle ermöglichen. (Definiert unter Punkt 3.5 auf Seite 35 / Überprüft unter Punkt 5.2.1 auf Seite 71)

Anforderungen an Flink

[A14] Flink kann Ereignisse empfangen, filtern, analysieren, konvertieren und an Fremdsysteme weiterleiten. (Definiert unter Punkt 3.6 auf Seite 36 / Überprüft unter Punkt 6.5 auf Seite 86)

[A15] Flink arbeitet zu jeder Zeit korrekt und fehlerfrei. Das Auftreten von Fehlern darf ausschließlich auf den eigenen Quellcode zurückzuführen sein. (Definiert unter Punkt 3.6 auf Seite 36 / Überprüft unter Punkt 6.5 auf Seite 89)

[A16] Die Ausführung von Flink ist robust. Ein Fehler im eigenen Quellcode darf nicht zum Absturz des Gesamtsystems führen. (Definiert unter Punkt 3.6 auf Seite 36 / Überprüft unter Punkt 6.5 auf Seite 89)

[A17] Die Verarbeitung eines Ereignisses von Flink dauert nicht länger als 10 Millisekunden. (Definiert unter Punkt 3.6 auf Seite 36 / Überprüft unter Punkt 6.4 auf Seite 86)

[A18] Flink kann bis zu 10 000 Ereignisse pro Sekunde verarbeiten, ohne dass andere Anforderungen beeinträchtigt werden. (Definiert unter Punkt 3.6 auf Seite 36 / Überprüft unter Punkt 6.4 auf Seite 86)

[A19] Ein funktionierender Prototyp von EventAlert lässt sich in weniger als fünf Tagen mithilfe von Flink entwickeln. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.5 auf Seite 87)

[A20] Es sind weniger als 20 LOC notwendig, um eine schlichte Verbindung von einfacher Ereignisquelle und -senke herzustellen. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.5 auf Seite 87)

[A21] Methoden, Parameter und Klassen von Flink sind dokumentiert. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.2 auf Seite 79)

[A22] Flinks Methoden und deren Parameter sind selbsterklärend bezeichnet und leicht verständlich. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.3 auf Seite 82)

[A23] Flink unterstützt die Verwendung von Lambda-Ausdrücken. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.5 auf Seite 88)

[A24] Der Entwickler benutzt dieselben Interfaces von Flink, die auch von in Flink eingebauten Komponenten verwendet werden. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.5 auf Seite 88)

[A25] Das Einbinden einer Ereignisquelle über einen Flink-Connector erfordert keine Fachkenntnisse über das Fremdsystem hinter dem Connector. (Definiert unter Punkt 3.6 auf Seite 37 / Überprüft unter Punkt 6.5 auf Seite 90)

[A26] Das Debuggen von selbst entwickelten Programmteilen, die auf Flink basieren, ist möglich. (Definiert unter Punkt 3.6 auf Seite 38 / Überprüft unter Punkt 6.5 auf Seite 89)

[A27] Flink stellt ein Test-Framework zur Verfügung, welches mit anderen Test-Frameworks kompatibel ist. (Definiert unter Punkt 3.6 auf Seite 38 / Überprüft unter Punkt 6.5 auf Seite 90)

[A28] Das Test-Framework von Flink ermöglicht die Simulation einer vollständigen Ereignis-Verarbeitung anhand von Testdaten. (Definiert unter Punkt 3.6 auf Seite 38 / Überprüft unter Punkt 6.5 auf Seite 90)

B Verbindung zwischen LED-Steuerung und LED-Streifen

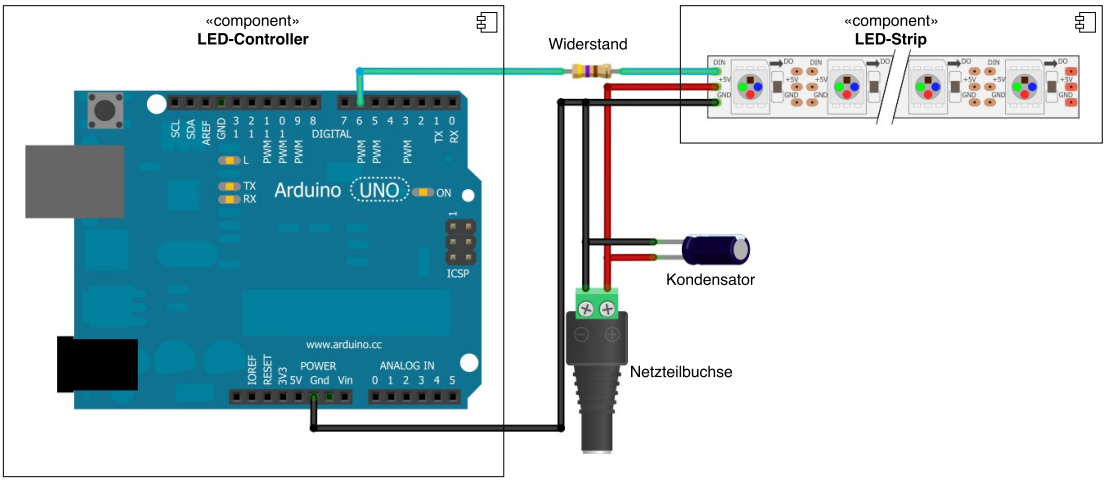


Abbildung B.1: Verbindung zwischen LED-Steuerung und LED-Streifen: Grafische Darstellung

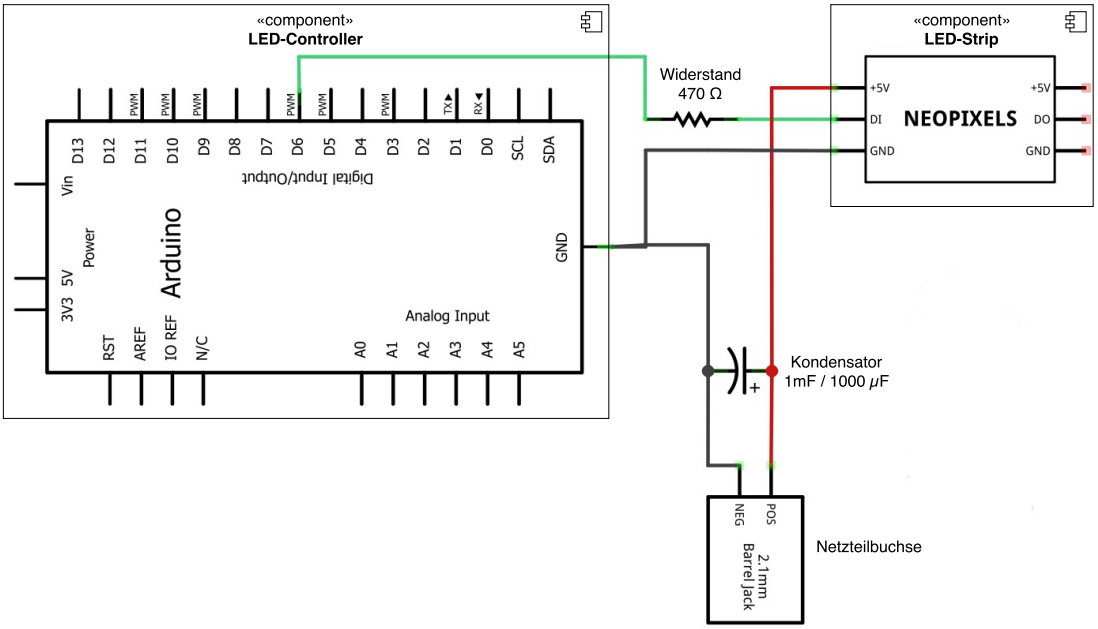


Abbildung B.2: Verbindung zwischen LED-Steuerung und LED-Streifen: Technische Darstellung

Literaturverzeichnis

[Adafruit Industries] Adafruit Industries: *Adafruit NeoPixel Digital RGBW LED Strip - White PCB 144 LED/m.* – URL https://www.adafruit.com/product/2847. – Zugriffsdatum: 2017-10-01

[Andelfinger und Hänisch 2015] Andelfinger, Volker P. ; Hänisch, Till: *Internet der Dinge.* Springer Fachmedien Wiesbaden, 2015. – ISBN 978-3-658-06728-1

[Apache Software Foundation a] Apache Software Foundation: *Apache Flink 1.3 Documentation: Apache Kafka Connector.* – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/connectors/kafka.html. – Zugriffsdatum: 2017-10-19

[Apache Software Foundation b] Apache Software Foundation: *Apache Flink 1.3 Documentation: Flink DataStream API Programming Guide.* – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/datastream_api.html. – Zugriffsdatum: 2017-07-14

[Apache Software Foundation c] Apache Software Foundation: *Apache Flink 1.3 Documentation: Java 8.* – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/java8.html. – Zugriffsdatum: 2017-10-19

[Apache Software Foundation d] Apache Software Foundation: *Apache Flink 1.3 Documentation: Quickstart.* – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/quickstart/setup_quickstart.html. – Zugriffsdatum: 2017-11-27

[Apache Software Foundation e] Apache Software Foundation: *Apache Flink 1.3 Documentation: Sample Project using the Java API.* – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/quickstart/java_api_quickstart.html. – Zugriffsdatum: 2017-11-27

[Apache Software Foundation f] Apache Software Foundation: *Apache Flink 1.3 Documentation: Streaming Connectors*. – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/connectors/index.html. – Zugriffsdatum: 2017-07-14

[Apache Software Foundation g] Apache Software Foundation: *Apache Flink 1.3 Documentation: Windows*. – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/dev/windows.html. – Zugriffsdatum: 2017-10-17

[Apache Software Foundation h] Apache Software Foundation: *Apache Flink 1.4-SNAPSHOT Documentation: Testing*. – URL https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/stream/testing.html. – Zugriffsdatum: 2017-10-19

[Apache Software Foundation i] Apache Software Foundation: *Apache Flink: Community and Project Info*. – URL https://flink.apache.org/community.html. – Zugriffsdatum: 2017-10-24

[Apache Software Foundation j] Apache Software Foundation: *Apache Kafka 1.0 Documentation*. – URL https://kafka.apache.org/documentation/. – Zugriffsdatum: 2017-11-26

[Apache Software Foundation k] Apache Software Foundation: *Overview (flink 1.3-SNAPSHOT API)*. – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/api/java/. – Zugriffsdatum: 2017-10-18

[Apache Software Foundation l] Apache Software Foundation: *TwitterSource (flink 1.3-SNAPSHOT API)*. – URL https://ci.apache.org/projects/flink/flink-docs-release-1.3/api/java/org/apache/flink/streaming/connectors/twitter/TwitterSource.html. – Zugriffsdatum: 2017-10-19

[Arduino a] Arduino: *Arduino - Available*. – URL https://www.arduino.cc/en/Serial/Available. – Zugriffsdatum: 2017-09-25

[Arduino b] Arduino: *Arduino - Environment*. – URL https://www.arduino.cc/en/Guide/Environment. – Zugriffsdatum: 2017-09-22

[Arduino c] Arduino: *Arduino - FAQ*. – URL https://www.arduino.cc/en/Main/FAQ. – Zugriffsdatum: 2017-09-22

[Arduino d] Arduino: *Arduino - Sketch.* – URL https://www.arduino.cc/en/tutorial/sketch. – Zugriffsdatum: 2017-09-22

[Ardulink] Ardulink: *Ardulink.* – URL http://www.ardulink.org/what-is/

[data Artisans a] Artisans data: *Apache Flink Training.* – URL http://training.data-artisans.com/. – Zugriffsdatum: 2017-10-21

[data Artisans b] Artisans data: *DataStream API Basics.* – URL http://training.data-artisans.com/dataStream/basics.html. – Zugriffsdatum: 2017-10-21

[data Artisans c] Artisans data: *How to do the Exercises.* – URL http://training.data-artisans.com/howto-exercises.html. – Zugriffsdatum: 2017-10-21

[data Artisans d] Artisans data: *Setup Development Environment.* – URL http://training.data-artisans.com/devEnvSetup.html. – Zugriffsdatum: 2017-10-21

[Bassett 2015] Bassett, Lindsay: *Introduction to JavaScript Object Notation: A To-the-Point Guide to JSON.* O'Reilly Media, 2015. – ISBN 9781491929483

[Braun 2010] Braun, Torsten: Das Internet der Zukunft. In: *Informatik-Spektrum* 33 (2010), apr, Nr. 2, S. 103–105. – ISSN 0170-6012

[Breur 2015] Breur, Tom: Big data and the internet of things. In: *Journal of Marketing Analytics* 3 (2015), mar, Nr. 1, S. 1–4. – ISSN 2050-3318

[Bruns und Dunkel 2010] Bruns, Ralf ; Dunkel, Jürgen: *Event-Driven Architecture.* Springer Berlin Heidelberg, 2010 (Xpert.press). – ISBN 978-3-642-02438-2

[Bruns und Dunkel 2015] Bruns, Ralf ; Dunkel, Jürgen: *Complex Event Processing: Komplexe Analyse von massiven Datenströmen mit CEP.* Springer Fachmedien Wiesbaden, 2015 (essentials). – ISBN 978-3-658-09898-8

[Burgess 2016a] Burgess, Phillip: *Arduino Library Use.* 2016. – URL https://learn.adafruit.com/adafruit-neopixel-uberguide/arduino-library. – Zugriffsdatum: 2017-09-25

[Burgess 2016b] Burgess, Phillip: *Basic Connections | Adafruit NeoPixel Überguide.* 2016. – URL https://learn.adafruit.com/adafruit-neopixel-uberguide/basic-connections

[Burgess 2016c] Burgess, Phillip: *Best Practices | Adafruit NeoPixel Überguide*. 2016. – URL https://learn.adafruit.com/adafruit-neopixel-uberguide/best-practices. – Zugriffsdatum: 2017-10-01

[Cooper 2015] Cooper, Tyler: *Overview | RGB LED Strips*. 2015. – URL https://learn.adafruit.com/rgb-led-strips/overview. – Zugriffsdatum: 2017-09-27

[Cooper 2017] Cooper, Tyler: *Usage | RGB LED Strips*. 2017. – URL https://learn.adafruit.com/rgb-led-strips/usage. – Zugriffsdatum: 2017-09-27

[Deshpande 2017] Deshpande, Tanmay: *Learning Apache Flink*. Packt Publishing, 2017. – ISBN 9781786466228

[Dunning und Friedman 2016] Dunning, Ted ; Friedman, Ellen: *Streaming Architecture*. O'Reilly Media, 2016. – ISBN 9781491953907

[Estrada und Ruiz 2016] Estrada, Raul ; Ruiz, Isaac: *Big Data SMACK*. Apress, 2016. – ISBN 978-1-4842-2174-7

[Fördergemeinschaft Gutes Licht 2010] Fördergemeinschaft Gutes Licht: Das Licht der Zukunft. In: *licht.wissen* 17 (2010). – URL https://www.licht.de/fileadmin/Publikationen_Downloads/lichtwissen17_LED.pdf. ISBN 978-3-926193-56-8

[Friedman und Tzoumas 2016] Friedman, Ellen ; Tzoumas, Kostas: *Introduction to Apache Flink: Stream Processing for Real Time and Beyond*. O'Reilly Media, 2016. – ISBN 9781491976586

[von Gagern 2017] Gagern, Stefan von: *Was ist was im Internet der Dinge?* 2017. – URL https://www.cio.de/a/was-ist-was-im-internet-der-dinge,3213802,2. – Zugriffsdatum: 2017-12-15

[Garg 2015] Garg, Nishant: *Learning Apache Kafka, Second Edition*. Packt Publishing, 2015. – ISBN 978-1784393090

[Günther und Lehmann 2013] Günther, Markus ; Lehmann, Martin: Lambda-Ausdrücke in Java 8. In: *JavaSPEKTRUM* (2013), Nr. 03/2013. – URL https://www.sigs-datacom.de/uploads/tx_dmjournals/guenther_lehmann_JS_03_13_va54.pdf

[Hager Vertriebsgesellschaft mbH 2012] Hager Vertriebsgesellschaft mbH: *Grundlagen der Beleuchtungssteuerung*. 2012. – URL http://www.hager.de/flash/e-learning-beleuchtung-dimmer/doc/de/hager_beleuchtungssteuerung_2012_05.pdf

[Hedtstück 2017] Hedtstück, Ulrich: *Complex Event Processing: Verarbeitung von Ereignismustern in Datenströmen*. Springer Berlin Heidelberg, 2017 (eXamen.press). – URL http://link.springer.com/10.1007/978-3-662-53451-9. – ISBN 978-3-662-53450-2

[Krettek und Winters 2017] Krettek, Aljoscha ; Winters, Mike: *Apache Flink 1.4.0 Release Announcement*. 2017. – URL http://flink.apache.org/news/2017/12/12/release-1.4.0.html. – Zugriffsdatum: 12-12-2017

[Mattern und Flörkemeier 2010] Mattern, Friedemann ; Flörkemeier, Christian: Vom Internet der Computer zum Internet der Dinge. In: *Informatik-Spektrum* 33 (2010), apr, Nr. 2. – ISSN 0170-6012

[Oracle 2016] Oracle: *Stream (Java Platform SE 8)*. 2016. – URL https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html. – Zugriffsdatum: 2017-10-17

[collaborative research project] project, Stratosphere collaborative research: *Stratosphere - Next Generation Big Data Analytics Platform*. – URL http://stratosphere.eu/. – Zugriffsdatum: 2017-06-23

[Schubert 2006] Schubert, E. F.: *Light-Emitting Diodes*. Cambridge University Press, 2006. – ISBN 9780521865388

[Strese 2016] Strese, Hartmut: Kommt das Smart Home durch die Küche? In: *iit perspektive* Nr. 28 (2016), sep. ISBN 978-3-89750-173-7

[Strese u. a. 2010] Strese, Hartmut ; Seidel, Uwe ; Knape, Thorsten ; Botthof, Alfons: *Smart Home in Deutschland*. Institut für Innovation und Technik (iit) in der VDI/VDE Innovation + Technik GmbH, 2010. – ISBN 978-3-89750-165-2

[Telegram Messenger LLP] Telegram Messenger LLP: *Telegram F.A.Q.*. – URL https://telegram.org/faq/de. – Zugriffsdatum: 2017-07-18

[Twitter Inc.] Twitter Inc.: *Basic stream parameters - Twitter Developers*. – URL https://developer.twitter.com/en/docs/tweets/filter-realtime/guides/basic-stream-parameters. – Zugriffsdatum: 2017-10-19

[Tzoumas u. a. 2017] Tzoumas, Kostas ; Ewen, Stephan ; Hueske, Fabianm ; Marthi, Suneel ; Metzger, Robert ; Sax, J. M.: *Powered by Flink - Apache Flink - Apache Software Foundation*. 2017. – URL https://cwiki.apache.org/confluence/display/FLINK/Powered+by+Flink. – Zugriffsdatum: 2017-06-27

[Wagner] Wagner, Philipp: *bytefish - FlinkExperiments*. – URL https://github.com/bytefish/FlinkExperiments. – Zugriffsdatum: 2017-10-21

[Wagner 2016] Wagner, Philipp: *Building Applications with Apache Flink (Part 1): Dataset, Data Preparation and Building a Model*. 2016. – URL https://bytefish.de/blog/apache_flink_series_1/. – Zugriffsdatum: 2017-10-21

[Wingerath u. a. 2016a] Wingerath, Wolfram ; Gessert, Felix ; Friedrich, Steffen ; Ritter, Norbert: Real-time stream processing for Big Data. In: *it - Information Technology* 58 (2016), jan, Nr. 4, S. 186–194. – ISSN 1611-2776

[Wingerath u. a. 2016b] Wingerath, Wolfram ; Gessert, Felix ; Friedrich, Steffen ; Ritter, Norbert: *Scalable Stream Processing: A Survey of Storm, Samza, Spark and Flink*. 2016. – URL https://medium.baqend.com/6d248f692056. – Zugriffsdatum: 2017-06-27

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 29. Dezember 2017 Tim Wittler