Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Steffen Wohlers

Echtzeit-Verarbeitung von Big Data via Complex Event Processing und Apache Flink - Evaluation mittels einer Beispielanwendung

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Steffen Wohlers

Echtzeit-Verarbeitung von Big Data via Complex Event Processing und Apache Flink - Evaluation mittels einer Beispielanwendung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Wirtschaftsinformatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Olaf Zukunft
Zweitgutachter: Prof. Dr. Jens-Eric von Düsterlho

Eingereicht am: 14. August 2018

Steffen Wohlers

Thema der Arbeit
Echtzeit-Verarbeitung von Big Data via Complex Event Processing und Apache Flink - Evaluation mittels einer Beispielanwendung

Stichworte
Complex Event Processing, Event-Driven Architecture, Apache Flink, Ereignisorientierung, Big Data

Kurzzusammenfassung
Complex Event Processing (CEP) ermöglicht die Echtzeitauswertung von Big Data, in dem Daten als Ereignisse interpretiert und weiterverarbeitet werden. Das Ziel dieser Bachelorarbeit war es, eine ereignisorientierte Anwendung mittels Apache Flink zu realisieren und abschließend zu evaluieren. Dafür wurde der fachliche Nutzen für Unternehmen und die technischen Konzepte Event-Driven Architecture (EDA) sowie CEP untersucht. Die Evaluation ergab, dass eine Realisierung in angemessener Zeit möglich ist. Durch die Nähe der CEP Bibliothek zum Programmcode konnten auch benachbarte Konzepte verwendet werden. Allerdings sorgt dies auch für eine höhere Intransparenz. Für Unternehmen bietet sich ein Nutzen durch eine Erhöhung der Agilität und eine verkürzte Reaktionszeit.

Title of the paper
Real-time processing of big data via complex event processing and Apache Flink - evaluation using a sample application

Keywords
complex event processing, event-driven architecture, Apache Flink, event-driven processing, big data

Abstract
Complex Event Processing (CEP) enables real-time evaluation of Big Data by interpreting and processing data as events. The goal of this bachelor thesis was to realize and finally evaluate an event-oriented application using Apache Flink. The functional benefits for companies and the technical concepts of Event-Driven Architecture (EDA) and CEP were investigated. The evaluation showed that implementation is possible within a reasonable time. Due to the proximity of the CEP library to the program code, neighboring concepts could also be used. However, this also leads to greater intransparency. Companies benefit from an increase in agility and a shortened reaction time.

Inhaltsverzeichnis

1. Einleitung

Wie und ob ein Mensch Ereignisse erkennt, diese analysiert und darauf reagiert, bestimmt maßgeblich seinen Lebensweg. Dies lässt sich auch auf Unternehmen als Ganzes oder aber auf einzelne Geschäftsprozesse ableiten [vgl. K10, S. 8]: Wird in der Fabrik erkannt, dass es ein Problem mit einer Maschine in einer durchgetakteten Fließbandproduktion gibt? Erkennt das Unternehmen, dass sein Konkurrent auf einer Onlineplattform zeitweise einen Rabatt auf bestimmte Produktgruppen gegeben hat und wie reagiert es darauf? Der zeitliche Aspekt scheint ein wichtiger Erfolgsfaktor zu sein. Herkömmliche IT-Anwendungen im Big Data Umfeld speichern in der Regel zuerst die Daten ab, um sie zu einem späteren Zeitpunk zu analysieren. Dieser spätere Zeitpunkt kann unter spezifischen Aspekten zu spät sein.
Einen anderen Ansatz verfolgen daher ereignisorientierte Anwendungen, die Complex Event Processing (CEP) nutzen: Sie versuchen Ereignisse und Ereignismuster in Datenströmen zu erkennen und diese in Echtzeit zu verarbeiten, bevor die Daten gespeichert wurden.

1.1. Ziel der Arbeit

Ziel der Arbeit ist es, eine geeignete prototypische Anwendung mit Apache Flink und der CEP Library zu konzipieren und umzusetzen, um anschließend mit einer Evaluation abzuschließen. Außerdem soll auf den fachlichen Nutzen einer ereignisorientierte Anwendung innerhalb eines Unternehmens eingegangen werden.
Im erweiterten Kreis der Betrachtung stehen das Architekturmuster Event-Driven Architecture (EDA), in der ereignisorientierte Anwendungen interagieren sowie die Event Processing Language, die die Ereignismuster für CEP definiert.

1.2. Abgrenzung

Auf Basis der identifizierten Zielsetzung, wird im Rahmen dieser Arbeit aufgrund von limitierten Ressourcen und Zeitrahmen auf eine professionelle Verteilung auf mehreren Servern verzichtet. Die Möglichkeiten der Verteilung der Anwendung werden nur grundlegend getes-

tet. Auch der performance-orientierte Vergleich mit ähnlichen Produkten entfällt daher. Aus betriebswirtschaftlicher Sicht sollen vor allem qualitative Faktoren beschrieben werden.

1.3. Aufbau und Vorgehen

In dieser Arbeit wird das allgemeine Vorgehen nach dem Top-Down-Ansatz verfolgt und fachliche Aspekte werden vor den technischen Aspekten betrachtet. Daraus ergibt sich folgender Aufbau:
In Kapitel 2 werden grundlegende Begriffe beschrieben, um die theoretischen Grundlagen für den weiteren Untersuchungsgang zu definieren und elementare Begriffe abzugrenzen. In Kapitel 3 wird zunächst das ereignisorientierte Handeln in Unternehmen analysiert, um auf diese Weise Anwendungsgebiete zu ergründen sowie Stärken und Schwächen von ereignisorientierten Systemen in Unternehmen darzustellen. Anschließend werden die technischen Konzepte von EDA und CEP beschrieben. Darauf aufbauend erfolgt jeweils eine Extraktion der Eigenschaften und Charakteristika. Abschließend werden Anforderungen definiert und eine Bewertung vorgenommen. Innerhalb des Kapitels 4 erfolgt die fachliche und technische Konzeption der prototypischen Beispielanwendung auf Grundlage der vorangegangenen Analyse. In Kapitel 5 wird das Ergebnis mittels Apache Flink realisiert und auf nennenswerte Aspekte thematisiert. In Kapitel 6 erfolgt die Evaluation der Beispielanwendung und FlinkCEP durch den Anforderungsabgleich sowie einer kritischen Betrachtung. Abschließend fasst Kapitel 7 die wichtigsten Ergebnisse zusammen und gibt im Ausblick Ansatzpunkte für mögliche weiterführende Publikationen.

2. Grundlagen

In diesem Kapitel soll ein grundlegendes Verständnis für die Problemstellung geschaffen und erstes Wissen zur Verfügung gestellt werden.

2.1. Big Data

Die Informationsgenerierung und Informationsverarbeitung durch IT Systeme auf der ganzen Welt ist die Grundlage des alltäglichen Lebens im 21. Jahrhundert [vgl. Luc01, S. 3]. Egal ob in der Industrie durch Sensoren, im globalen Finanzhandel durch High-frequency-trading (HFT) oder auf Social Media Plattformen durch ständig neue Postings - überall nimmt die Datenflut zu. Der Begriff *Big Data* spezifiziert diesen fortschreitenden Prozess und beschreibt ihn mittels der drei Begriffe *Volume*, *Variety* und *Velocity* (VVV). Wenn Big Data als Technologie verstanden werden soll, darf nicht nur von einer einzigen Technologie ausgegangen werden, sondern sollte es als eine Kombination von mehreren alten und neuen Technologien aufgefasst werden. Ziel von Big Data ist es, dass sein Nutzer eine sehr große Menge (volume) von unterschiedlich formatierten und strukturierten Daten (variety) in geeigneter Zeit (velocity) verarbeiten kann [vgl. Hur+13, S. 16].
Da sich diese Arbeit mit der Echtzeitauswertung von Big Data befasst, wird der Fokus ab diesem Zeitpunkt auf relevante Themen für diesen Bereich von Big Data gesetzt. Der traditionelle Ansatz von Big Data ist es, gespeicherte Daten in einer oder mehreren Datenbanken, sogenannten statischen Datenbeständen, zu verarbeiten (*Batch-Processing*). In ereignisgesteuerten Systemen werden eingehende Daten direkt verarbeitet und nicht erst in einer Datenbank gespeichert (*Stream-Processing*). Der Fokus einer solchen Anwendung liegt auf der Bereitstellung der nötigen Velocity.

2.2. Datenströme

Ein Datenstrom ist eine *potenziell unendliche Sequenz* von *atomaren und kontinuierlichen Vorkommnissen* in einer *zeitlichen Sortierung* von einer bestimmten Entität. Diese Vorkommnisse besitzen in der Regel ein wohldefiniertes Schema [vgl. CEA11, S. 38]. Es ist möglich, dass eine

Entität unterschiedliche Arten von Vorkommnissen über den Datenstrom versendet. Datenströme lassen sich in homogene und heterogene Datenströme unterteilen [vgl. EN10, S. 45]. Bezieht man sich auf das Exempel Sensor, kann man auch von ihm annehmen, dass er bis zum Defekt eine unendliche Anzahl von Temperaturänderungen, dem Vorkommnis einer Zustandsänderung, über den Datenstrom distribuiert. Da es mühsam ist von einem Vorkommnis einer Zustandsänderung zu sprechen, hat sich der Begriff *Ereignis* (engl. *event*) im Forschungsfeld durchgesetzt. Dieser Begriff wird im nächsten Abschnitt genauer spezifiziert. Deshalb wird auch von einem *Ereignisstrom* (engl. *event stream*) gesprochen [vgl. BD15, S. 3].

> Ein *Ereignisstrom* (engl. *event stream*) ist eine linear geordnete Sequenz von Ereignissen [vgl. LW08, S. 14].

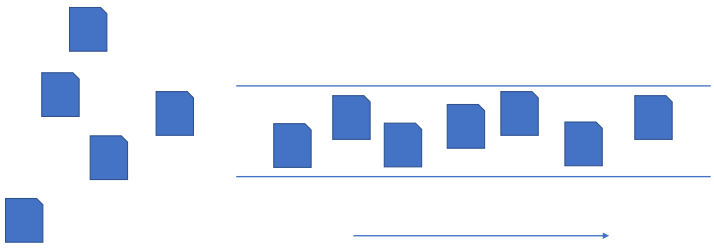

Abbildung 2.1.: Visualisierung eines Datenstroms durch welchen mehrere Datensätze fließen [vgl. BD10].

2.3. Ereignisse

Teilt man das Wort Ereignisstrom in seine Bestandteile auf, erhält man neben dem bereits beschriebenen Wortteil Strom einen weiteren elementaren Begriff für diese Arbeit: das *Ereignis*. Die erhobene Daten (aus beispielsweise einem Warenkorb eines Online-Shops oder eines Sensors) drücken eine Zustandsgröße in einem System aus. Eine Zustandsänderung des Systems wird durch eine Änderung der Zustandsgröße repräsentiert. Hier kann zwischen einer diskreten Zustandsänderung, einem zeitlich punktuellen Geschehnis wie dem Hinzufügen eines Produkts in den Warenkorb oder einer kontinuierlichen Zustandsänderung wie die Änderung der Temperatur eines Objektes differenziert werden [vgl. Hed17, 1f].

Die Ursachen für ein Ereignis können vielfältig sein: eine Aktivität, ein Vorgang, eine Entscheidung, etc. “an event is anything that happens” [K10, S. 1]. Anzumerken ist noch, dass

der Begriff Ereignis mit mehreren Bedeutungen behaftet ist. Er drückt sowohl ein *fachliches Auftreten* in der realen Welt aus, als auch eine *technische Abstraktion* dieses Ereignisses in einem System. Im Verlauf dieser Arbeit werden technische Abstraktionen auch als *Ereignisobjekt* oder *Instanz* bezeichnet.

> Das (erwartete) Vorkommen bzw. die Ursache einer Zustandsänderung in einem System wird als *Ereignis* (engl. *event*) bezeichnet [vgl. Hed17, S. 2].

2.4. Charakterisierung von Ereignissen

Betrachtet man Ereignisse genauer, erkennt man schnell, dass es für die Verarbeitung sinnvoll ist, die Ereignisse zu klassifizieren und ihnen eine geeignete Struktur zuzuweisen. Im Folgenden werden die grundlegenden Charakteristika beschriebenen und visualisiert.

Ereignistypen und Ereignisinstanzen

Wie beschrieben, können Ereignisse aus vielen unterschiedlichen Anwendungen entstehen und versendet werden. Daher ist es hilfreich, nicht nur eine implizite Klassifizierung durch eine Benennung wie beispielsweise das “Änderung des Warenkorbs Ereignis” oder das “Temperatur-Änderungs-Ereignis” vorzunehmen, sondern eine explizite Typisierung zu erzeugen.
Betrachtet man eine Menge gleichartiger Ereignisse, wie die von einem Temperatursensor, kann man erkennen, dass sie die gleiche Struktur bzw. die gleichen Attribute besitzen und die gleiche Art von Informationen in sich tragen. Anstatt die Struktur jedes einzelnen Ereignisses zu definieren, übernimmt man aus der objektorientierten Programmierung die Konzepte des Typs und der Klasse [vgl. EN10, S. 62]. Listing 2.1 veranschaulicht einen solchen *Ereignistyp*.

```
1  Temperaturaenderung
2  timestamp: Integer
3  id: Integer
4  ----------------------------------------
5  alterWert: Integer
6  neuerWert: Integer
7  aenderungsbetrag: Integer
8  sensorName: String
9  breitengrad: Double
10 laengengrad: Double
```

Listing 2.1: Ein Ereignistyp

> "Ein Ereignistyp (engl. event type, event class) spezifiziert die grundlegenden Eigenschaften einer ganzen Klasse von gleichartigen Ereignissen. Ein Ereignistyp definiert insbesondere die Attribute (event properties) einer Klasse von Ereignissen, so dass alle Ereignisse eines bestimmten Typs dieselbe Struktur der in ihnen enthaltenen Daten besitzen." [BD10, S. 88]

Ein konkretes Ereignis, welches zu einem bestimmten Zeitpunkt geschehen ist, ist eine Ausprägung einer Klasse [vgl. Hed17, S. 13]. Analog zu der objektorientierten Programmierung bezeichnet man dieses Ereignis als *Ereignisinstanz* (engl. *event instance*) oder *Ereignisobjekt* (engl. *event object*). Abbildung 2.2 visualisiert exemplarisch zwei Ereignisinstanzen vom Typ *Temperaturaenderung*.

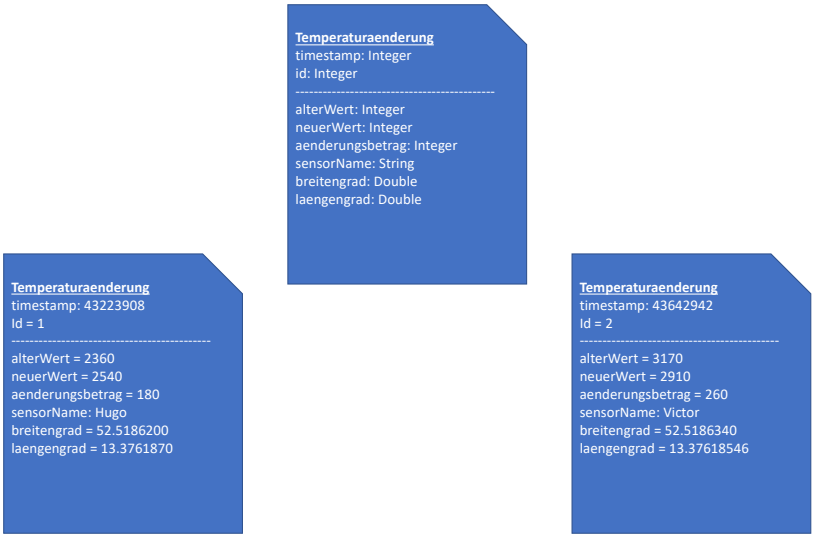

Abbildung 2.2.: Zwei Ereignisinstanzen vom Ereignistyp *Temperaturaenderung* [angelehnt an BD10, S. 88].

Attribute der Ereignistypen

Ereignisinstanzen liefern Informationen über das jeweilige Ereignis. Diese Informationen sind in den Attributen gespeichert und enthalten Informationen darüber, was, wann und wo geschehen ist [vgl. EN10, S. 63]. Des Weiteren kann man Attribute in die zwei Unterkategorien *Metadaten* und *Kontextdaten* unterteilen [vgl. BD10, S. 87].

Metadaten

Mittels der Attribute im Header wird eine Ereignisinstanz auf Meta-Ebene beschrieben. Die drei folgenden Attribute müssen definiert sein, um das Ereignis weiterverarbeiten zu können [vgl. BD15, S. 10]. Zusätzlich gibt es optionale Meta-Attribute, die eine Instanz besitzen kann. Jene werden allerdings nicht weiter in dieser Arbeit thematisiert. Eine Übersicht liefern Etzion, Opher und Niblett [EN10].

Type Identifier Der Type Identifier gibt den Typen der Instanz an. In der Abbildung 2.3 ist "Temperaturaenderung" als der Attribute Identifier angegeben und drückt aus, dass diese Instanz vom Typ Temperaturaenderung ist. In einigen Systemen muss dieses Attribut nicht explizit angegeben werden, da aus dem Kontext auf den Wert geschlossen werden kann oder das System beim Erzeugen den Wert implizit erstellt [vgl. EN10, S. 65]. Als Beispiel kann an dieser Stelle die Programmiersprache Java genannt werden.

Timestamp Der Timestamp gibt Informationen über den genauen Zeitpunkt der Zustandsänderung bekannt. Dieser Attributwert ist identisch mit der bereits weiter oben beschriebenen diskreten Zustandsänderung. Handelt es sich um eine kontinuierliche Zustandsänderung, wird als Timestamp der zeitlich punktuelle Messvorgang angegeben [vgl. Hed17, S. 2]. Der Timestamp ist essenziell, um die Ereignisse sortieren zu können und die zeitlichen Entwicklungen zu erkennen.

Ereignis-ID Als drittes Attribut muss jede Ereignisinstanz eine eindeutige Ereignis-ID besitzen. Sie dient dazu, dass ein Ereignis zweifelsfrei von anderen Ereignissen unterschieden werden kann [vgl. BD15, S. 10].

Kontextdaten

Die Kontextdaten beschreiben den eingetretenen Sachverhalt [vgl. BD15, S. 10]. Diese Attribute sind meist von Ereignistyp zu Ereignistyp unterschiedlich und passen zum jeweiligen Kontext. Für das Ereignis Temperaturänderung könnten die Informationen über die Höhe der Temperaturschwankung sowie der Ort des Sensors sicherlich von Bedeutung sein.

2.5. Ereignismuster und komplexe Ereignisse

Betrachtet man ein einzelnes Ereignis, welches die Zustandsänderung ausdrückt, spricht man von einem *atomaren Ereignis, low level event* oder *primitiven Ereignis* (engl. *primitive event*)

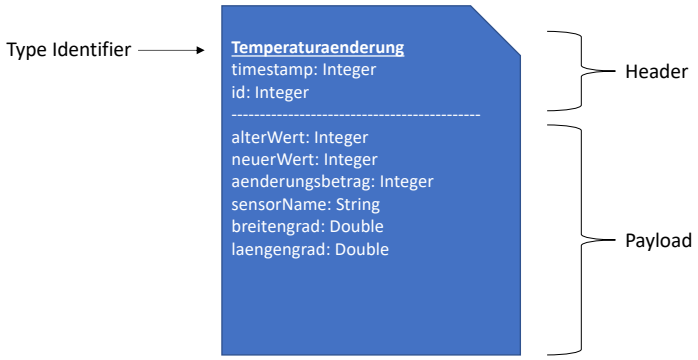

Abbildung 2.3.: Unterteilung von Attributen

[vgl. Hed17, S. 14]. Aus den Bezeichnungen wird deutlich, dass aus ihnen nur ein geringer Informationsgehalt gezogen werden kann. Es ist beispielsweise schwierig, aus einem einzelnen Temperaturänderungs-Ereignis weitere Rückschlüsse zu ziehen. Bei genauerer Inspektion des *Ereignisstroms* und somit der Betrachtung mehrerer Ereignisse (beispielsweise über einen längeren Zeitraum) ist es aber gegebenenfalls möglich *Beziehungen* zu erkennen [vgl. BD15, S. 3]. Durch *temporale*, *räumliche* oder *kausale* Zusammenhänge sowie *fachliche* Korrelationen entstehen Beziehungen zwischen Ereignissen, die ein bestimmtes *Muster* aufzeigen [vgl. BD15, S. 10]. *Ereignismuster* (engl. *event pattern*) spezifizieren Zusammenhänge, Abhängigkeiten und Korrelationen zwischen Ereignissen und ermöglichen es erst so das ereignisgesteuerte System zu verstehen [vgl. BD10, S. 91]. Zum Beispiel hat eine Reihe von Temperaturänderungs-Ereignissen, die eine Erhöhung der Temperaturänderung aufweisen, eine wesentlich höhere fachliche Relevanz als ein einzelnes Ereignis. Daher ist es das zentrale Ziel bei der Verarbeitung von Ereignisströmen Muster zu erkennen und aus ihnen einen Informationsgewinn zu erhalten [vgl. BD15, S. 3].

> Ein *Ereignismuster* (engl.*event pattern*) ist ein Template, welches eine Menge von Ereignissen spezifiziert, die in einer bestimmten Beziehung zueinander stehen [vgl. EN10, 216f].

Diese spezifizierte Menge von Ereignissen wird abstrahiert und durch ein neues, höherwertiges Ereignis repräsentiert. Dieses Ereignis wird als *komplexes Ereignis* bezeichnet. Durch die Korrelation von mehreren Ereignissen befindet es sich auf einer höheren Abstraktionsebene [vgl. BD15, 4f]. Abbildung 2.4 stellt diesen Prozess grafisch dar. Als Beispiel für ein komplexes Ereignis dient das zuvor beschriebene Ereignismuster: Es abstrahiert mehrere Ereignisse aus dem Ereignisstrom zu einem “Temperaturanstiegs-Ereignis” mit einer wiederum erhöhten

fachlichen Relevanz. Abschließend ist anzumerken, dass mittels weiterer Ereignismuster auf höheren Abstraktionsebenen zusätzlich komplexe Ereignisse auf der nächsten Abstraktionsebene generiert werden können. Es entsteht eine *Ereignishierarchie*.

> Ein *komplexes Ereignis* (engl. *complex event*) ist eine Abstraktion von mehreren anderen Ereignissen [vgl. LW08, S. 9].

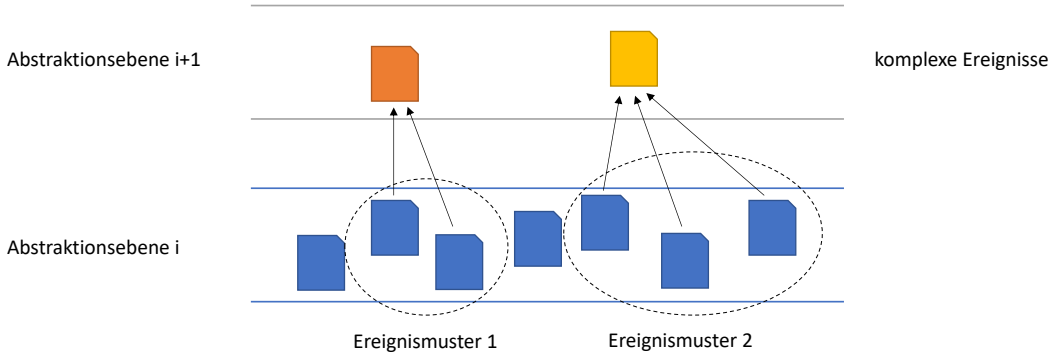

Abbildung 2.4.: Generierung von komplexen Ereignissen auf einer höheren Abstraktionsebene [vgl. BD15, S. 5].

2.6. Ereignisorientiertes Handeln

Analysiert man die Denk- bzw. Verhaltensweise eines Menschen und einer nicht intelligenten Maschine, lässt sich daraus eine geeignete Analogie ableiten: Menschen nehmen Situationen und Ereignisse in ihrer Umwelt wahr, können diese in einen größeren Kontext setzen, abstrahieren Entscheidendes und treffen Entscheidungen. Eine Maschine folgt einem definierten Ablauf und führt diese Schritt für Schritt aus. In Bezug auf die vorherigen Absätze stellt die Umwelt den Ereignisstrom dar. Aus Erfahrungen und Erlerntem bezieht der Mensch seine Ereignismuster. Das Gehirn eines Menschen erkennt die Ereignismuster, erzeugt neue komplexe Ereignisse und kommt zu einer Entscheidungsfindung [vgl. BD15, S. 1]. Ein Mensch kann so auf unvorhersehbare Ereignisse eingehen, wohingegen eine Maschine mit hoher Geschwindigkeit einen vordefinierten Prozess bearbeiten kann. Ein Mensch forciert *ereignisorientiertes Handeln*, während eine Maschine *ablauforientiertes Handeln* ausführt [vgl. BD15, S. 7].

Ereignisorientiertes Handeln beschreibt die Fähigkeit, unvorhersehbare Situationen/Ereignisse aus der Umwelt wahrzunehmen, sie zu abstrahieren und basierend darauf Entscheidungen zu treffen.

2.7. Ereignisgesteuerte Systeme

Betrachtet man das Exempel aus dem vorangegangen Beispiel und verändert es leicht, kann man sich die Mehrheit ereignisgesteuerter Systeme (engl- *event-driven systems*) wie in Abbildung 2.5 vorstellen. Es gibt in der Umwelt des Systems eine Menge von *Ereignisproduzenten*, welche Änderungen *erkennen*, Ereignisse produzieren und via Ereignisstrom auf das ereignisgesteuerte System einwirken. Das System *verarbeitet* die eintreffenden Ereignisse, untersucht sie auf vorhandene Ereignismuster und erzeugt/abstrahiert komplexe Ereignisse auf einem höheren Abstraktionslevel. Dies ist ein iterativer Prozess (siehe Abbildung 2.6). Wird ein bestimmtes Abstraktionslevel erreicht, werden die Ereignisse an die *Ereigniskonsumenten* weitergegeben. Die Ereigniskonsumenten sind die letzte Instanz, die ein Ereignis erreicht und *reagieren* auf dieses beispielsweise indem sie einen Geschäftsprozess anstoßen oder aber im vorherigen Beispiel eine Warnmeldung bezüglich eines zu hohen Temperaturanstiegs visualisieren.

Als *ereignisgesteuertes System* wird ein System bezeichnet, welches auf eintreffende Ereignisse reagiert und diese weiterverarbeitet [vgl. LW08, S. 18].

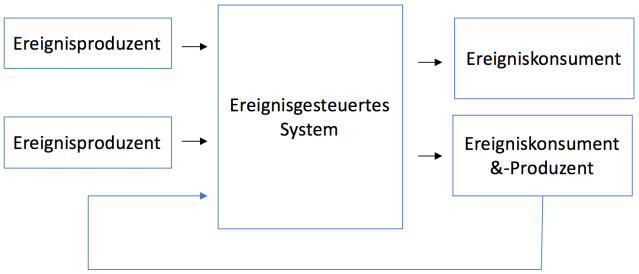

Abbildung 2.5.: Umwelt eines ereignisteuerten Systems (angelehnt an [Fli18c]).

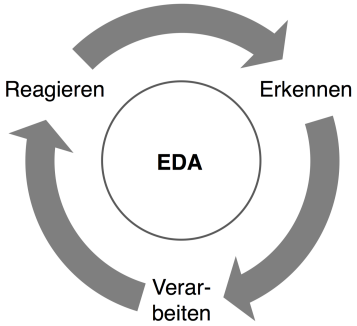

Abbildung 2.6.: Grundzyklus ereignisgesteuerter Systeme [BD10, S. 59].

Im folgenden Kapitel sollen ereignisgesteuerte Systeme genauer untersucht werden. Einführend erfolgt die Analyse des ereignisorientierten Handelns und ereignisgesteuerter Systeme im Unternehmenskontext.

3. Analyse

Die Analyse ist in zwei Abschnitte unterteilt. In Abschnitt 3.1 wird der fachliche Nutzen für ereignisorientierte Anwendungen analysiert. Der Abschnitt 3.2 untersucht bestehende technische Konzepte. Wenn für den jeweiligen Abschnitt geeignet, wird das zugrundeliegende Konzept besprochen und daraus relevante Eigenschaften und Charakteristika abstrahiert. Daraus ergeben sich Anforderungen und eine Bewertung.

3.1. Ereignisorientierte Anwendungen in Unternehmen

Bevor auf technische Konzepte für die Echtzeitauswertung von Big Data eingegangen wird, folgt zunächst die fachliche Untersuchung im Unternehmenskontext.

3.1.1. Ereignisgesteuerte Unternehmen

Betrachtet man vor den Unternehmenscharakteristika die jeweilige Wirtschaftsform in der die Unternehmen agieren, kann daraus ein vereinfachter Vergleich abgeleitet werden: Während in der Planwirtschaft der Staat eine Planung vorgibt, die es abzuarbeiten gilt, reagiert in der sozialen Marktwirtschaft der Staat flexibel auf bestimmte Ereignisse, die in dem Markt stattfinden. Die Welt ist jedoch zu komplex und dynamisch um langjährige statische Pläne durchzuführen. In dieser schnelllebigen Umwelt müssen Unternehmen flexibel auf Ereignisse reagieren können. Die Ungewissheit über Entwicklungen in der Zukunft erschweren dies zunehmend [vgl. K10, S. 2].
Die in den letzten Jahren immer schnellere Digitalisierung von Unternehmen und die damit einhergehende Automatisierung der Märkte, Fabriken und Telekommunikationswege hat dafür gesorgt, dass vorher unzugängliche Ereignisse verarbeitet werden können [vgl. CEA11, S. 8]. Vor allem durch das Internet und die schnellen sowie häufigen Interaktionen sowohl unternehmensintern als auch mit der Umwelt führen dazu, dass ständig und immer zügiger auf Ereignisse reagiert werden muss. Ereignisgesteuerte Unternehmen sind in der heutigen Zeit eher der Standard als die Ausnahme. Nach Bruns und Dunkel in [BD10] sind ereignisgesteuerte Unternehmen durch folgende Eigenschaften gekennzeichnet:

- Das Unternehmen ist in der Lage Ereignisse zu *erkennen*, zu *analysieren* und darauf zu *reagieren*.
- Das Unternehmen *kommuniziert* mittels (Ereignis-)Nachrichten mit anderen Unternehmen.
- Durch Ereignisse werden im Unternehmen *Aktionen* ausgelöst.

Erfolgreiche Unternehmen im 21. Jahrhundert kennzeichnet, dass sie durch eine hohe Agilität innerhalb kürzester Zeit Ereignisse erkennen können, diese verarbeiten und anschließend reagieren, indem sie ihre Geschäftsprozesse anpassen (siehe Abbildung 2.6).

3.1.2. Unternehmensprozesse und Ereignisse

Wird ein typischer Unternehmensprozess analysiert, ist zu erkennen, dass er zeitgesteuerte, anfragegesteuerte und ereignisgesteuerte Aspekte aufweist. Zeitgesteuertes Verhalten tritt in Erscheinung, wenn Aktivitäten zeitlich genau geplant werden können. Anfragegesteuertes Verhalten tritt auf, wenn die Aktivität von allen Teilnehmern bestätigt und definiert ist, jedoch der Zeitpunkt nicht vorhergesagt werden kann. Ereignisgesteuertes Verhalten ist vorhanden, wenn weder die Zeit noch die jeweilige Situation hervorgesagt werden kann, aber kurzfristig reagiert werden muss.[vgl. K10, 1f].
Die Tabelle 3.1 soll mittels des Beispielprozesses “Schrauben nachbestellen” die Verhaltensweisen exemplarisch darstellen.

Verhalten	**Beispiel**
zeitgesteuert	Ein Mitarbeiter bestellt alle x Tage neue Schrauben
anfragegesteuerte	Ein Mitarbeiter prüft den Schraubenbestand. Ist die Kiste leer, löst er einen automatisierten Bestellprozess aus.
ereignisgesteuert	Durch einen Sensor wird das Ereignis “Füllstand niedrig“ erkannt und automatisiert eine Bestellung ausgelöst.

Tabelle 3.1.: Verhaltensweisen in Unternehmensprozessen

Anhand des Beispiels kann die mögliche Stärke ereignisorientierter Geschäftsprozesse erkannt werden. Der ereignisgesteuerte Prozess sorgt für den höchsten Automatisierungsgrad und triviale Aufgaben müssen nicht manuell von Arbeitnehmern vollzogen werden. Ein weiterer Vorteil ist die geringe Kapitalbindung und die Einsparung von Lagerkosten durch Just-in-time-Lieferung.

3.1.3. Ereignisse in Unternehmen

Geschäftsereignisse sind bedeutsame Ereignisse innerhalb eines Unternehmens, welche tagtäglich im gesamten Unternehmen auf unterschiedlichen Abstraktionsebenen auftreten [vgl. K10, S. 1]. Es gilt, dass einzelne Ereignisse für sich betrachtet nur wenig aussagekräftig sind [vgl. BD10, S. 7]. Der Kontext, in dem die Ereignisse auftreten ist entscheidend [vgl. EN10, S. 144]. Ziel des ereignisgesteuerten Unternehmens ist es, Korrelationen zwischen den verfügbaren Ereignissen aus *internen* und *externen* Ereignisströmen zu erkennen, sie auf eine fachliche Ebene zu abstrahieren und abschließend zu reagieren. In Abbildung 3.1 sind exemplarisch Ereignisse entlang der Wertschöpfungskette nach Porter dargestellt.

Abbildung 3.1.: Wertschöpfungskette nach Porter mit Ereignisquellen (Struktur entnommen aus [dea])

3.1.4. Anwendungsgebiete

Geschäftsbereiche, deren funktionale und nicht-funktionale Anforderungen in die folgenden Kategorien fallen, bieten sich als Anwendungsbereiche für ereignisgesteuerte Anwendungen an (u. a. [BD15], [BD10], [Luc01], [ALG17]). Stark strukturierte Geschäftsprozesse die keine Echtzeitauswertung fordern oder in denen nur selten Ausnahmen auftreten, kommen hingegen meistens nicht in Frage [vgl. BD10, S. 39].

- *Komplexe Analyse von Daten*: Aggregation von homogenen und heterogenen Daten, Erkennen von Ereignismustern, Abstraktion von Ereignissen
- *Große Datenvolumina*: Verarbeitung von sehr großen Eingangsdaten, die mittels Datenstrom eintreffen (Skala 1000 - ∞) Daten pro Sekunde

- *Geringe Latenzzeit*: Verarbeitung von Daten/Ereignissen in nahezu Echtzeit (engl. real-time)

Um exemplarisch einige Anwendungsgebiete aufzuzeigen, die diese Anforderungen aufweisen, gliedert die Abbildung 3.2, angelehnt an den Kategorien einer PEST-Analyse, mögliche Anwendungsgebiete in öffentliche und infrastrukturelle, unternehmerisch operative, Social Media sowie forschende und medizinische Anwendungen (u. a. nach [Hed17], [ALG17], [BIT14], [Luc01], [K10]).

Abbildung 3.2.: Beispielhafte Anwendungsgebiete ereignisgesteuerter Anwendungen

3.1.5. Bewertung

Die aufgeführten Eigenschaften sowie Anwendungsgebiete deuten auf die Stärken und Schwächen von ereignisgesteuerten Anwendungen hin. In diesem Abschnitt soll ein strukturierter Überblick über die Vor- und Nachteile mittels einer für diese Bedürfnisse angepasste SWOT Analyse geschaffen werden. Der Überblick beschränkt sich auf die Kern-Eigenschaften. Weitere Charakteristika werden u. a. in [EN10], [CEA11] beschrieben.

Stärken

Informationsverfügbarkeit In vielen Unternehmensbereichen ist Zeit ein entscheidender Faktor (vgl. Abbildung 3.3). Mittels ereignisgesteuerten Anwendungen ist eine Reaktion auf

Zustandsänderungen nahezu in Echtzeit möglich. Betrachtet man beispielsweise eine Produktionsstätte, in der Sensoren den Zustand der Maschinen überprüfen, können ereignisgesteuerte Anwendungen frühzeitig eine Warnung ausgeben und Produktionsstopps verhindert werden.

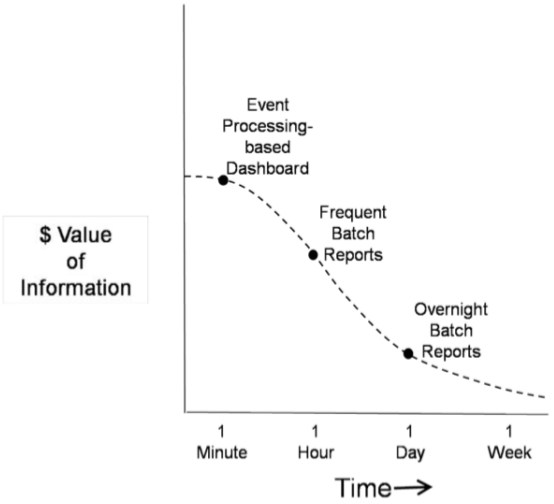

Abbildung 3.3.: Wert einer Information abhängig zur Zeit [CEA11, S. 16]

Einsparung von Kosten Ereignisgesteuerte Anwendungen können schneller sowie kosteneffizienter eintönige Berechnungen und Ereignismuster-Erkennungen durchführen als Menschen [vgl. CEA11, S. 19]. Reaktionen auf Ereignisse können automatisiert werden und wenig Personal eine große Anzahl von Systemen etc. überwachen. Das direkte Verarbeiten von Ereignissen macht die Speicherung von unnötigen Datensätzen obsolet und spart dadurch Hardware und Wartungsarbeiten [vgl. EN10, S. 15].

Hohe Agilität Ein Unternehmen sollte möglichst schnell reagieren können, wenn sich die Umwelt oder interne Prozesse ändern [vgl. K10, S. 5]. Ereignisgesteuerte Systeme sind in der Regel nur lose gekoppelt [vgl. K10, S. 12]. Dies führt zu geringerem Programmieraufwand bei Änderung (vgl. Abschnitt 3.2.1.2). Ereignisorientierte Anwendungen, die Complex Event Processing (CEP) nutzen, besitzen eine einfach zu ändernde Regelsprache (vgl. Anhang C), die auch von fachlichen Experten genutzt werden kann [vgl. EN10, S. 15].

Schwächen

Nötiges Expertenwissen Damit ereignisgesteuerte Systeme/Anwendungen einen Mehrwert liefern, braucht es Expertenwissen für die jeweilige Domäne. Ist falsches Wissen in dem

System implementiert, kann es durch den hohen Automatisierungsgrad schnell zu schwerwiegenden Fehlentscheidungen kommen [vgl. ALG17, S. 3].

Verlust von Informationen Im Grundkonzept ereignisorientierter CEP Anwendungen werden Informationen nicht gespeichert. Daten können verloren gehen und lassen sich für längere zeitliche Analysen nicht verwerten. Generell erfolgt die Speicherung nach der Verarbeitung [vgl. Hed17, S. 105].

Fehlende Standards Sowohl Etzion und Niblett [EN10] als auch Bruns und Dunkel [BD10] beschreiben als derzeitige Schwäche die fehlenden Standards. Derzeit gibt es durch die unterschiedlichen Ansätze sowohl bei kommerziellen als auch bei Open-Source Anwendungen proprietäre Standards (vgl. Anhang C). Dies erschwert einen späteren Wechsel der Plattform und sorgt für eine hohe Abhängigkeit [vgl. BD10, S. 224].

Chancen

Daten(-Ströme) nehmen zu Das Aufkommen von Daten (-Strömen) wird durch weitere Anwendungen von IoT oder Industrie 4.0 zunehmen. Durch eine frühe Implementierung von ereignisgesteuerten Anwendungen können Unternehmen Know-how in diesem Bereich aufbauen und womöglich einen langfristigen Wettbewerbsvorteil erlangen.

Wandel von reaktiven zu proaktiven Anwendungen Derzeitige ereignisgesteuerte Anwendungen werden oft nur reaktiv genutzt, indem sie auf auftretende Ereignisse reagieren und eine Aktion hervorrufen [vgl. EN10, S. 312]. Proaktives Vorgehen könnte realisiert werden, indem man mithilfe von Machine Learning Techniken automatisiert Ereignismuster erstellen lässt [vgl. ALG17, S. 2], [Fel+13].

Risiken

Preisgabe von Informationen Ereignisgesteuerte Anwendungen sind besonders effektiv, wenn sie mit der Umwelt kommunizieren. Allerdings hat das Unternehmen nach der Preisgabe der Informationen keinen Einfluss mehr auf seine Daten.

3.2. Konzepte für ereignisorientierte Anwendungen

Nachdem die fachlich interessanten Aspekte für Unternehmen und u. a. die Verortung von Ereignissen in Unternehmen beschrieben wurden, befasst sich dieser Abschnitt mit den da-

hinter liegenden technischen Konzepten. Zunächst wird das größere Konstrukt *Event-Driven Arichtecture* (EDA) behandelt, in der sich eine ereignisorientierte Anwendung befindet. Anschließend befasst sich diese Arbeit mit dem *Complex Event Processing* (CEP), welches eine Möglichkeit der Verarbeitung von Datenströmen in einer ereignisorientierten Anwendung bietet.

3.2.1. Event-Driven Architecture

Dieser Abschnitt beschreibt die konzeptionelle Architektur eines ereignisgesteuerten Systems, in welcher ereignisorientierte Anwendung interagieren.

3.2.1.1. Konzept

Wie in Abschnitt 3.1 beschrieben, bezeichnet man Unternehmen die auf Ereignisse reagieren als ereignisorientierte Unternehmen. Dieses Konzept der Ereignisorientierung (engl. *event-driven*) kann auch auf Software-Systeme angewendet werden. Die Architektur eines solchen Systems wird als *Event-Driven Architecture* (u. a. [EN10]) bezeichnet [vgl. K10, S. 8]. Die grundlegende Organisation eines Systems und die Beschreibung von Beziehungen und Interaktionen der unterschiedlichen Komponenten bezeichnet man als *Architektur* [vgl. BD10, S. 50]. Ablauforientierte Geschäftsprozesse mit ihren zeit- und anfragegesteuerten Prozessschritten (vgl. Abschnitt 3.1.2) lassen sich gut mit einer *service-orientierten Architektur* (SOA) ergänzen [vgl. BD15, S. 7]. Soll ereignisorientiert gehandelt werden, benötigt man eine Architektur, die Datenströmen sowie dem unvorhersehbaren Eintreten von Ereignismustern adäquat gegenübersteht und diese angemessen verarbeiten kann. “Event-Driven Architecture bietet einen entsprechenden Architekturstil, der die Ereignisbehandlung als zentrales Architekturkonzept in den Mittelpunkt stellt.” [BD15, S. 7].

Die folgende Abbildung 3.4 stellt die Event-Driven Architecture dar. Mithilfe dieses Schaubilds sollen die einzelnen Komponenten genauer beschrieben und anschließend ihre Eigenschaften analysiert werden. Anzumerken ist, dass sie eine detailliertere Repräsentation der Abbildung 2.5 aus Abschnitt 2.7 darstellt. Der zentrale Bestandteil dieser Architektur ist die Ereignisverarbeitung. Im Folgenden wird die Softwaretechnologie hinter der Ereignisverarbeitung als *Complex Event Processing* (CEP) beschreiben. Sie wird im Abschnitt 3.2.2 gesondert behandelt.

1. Schicht Ereignisquelle Eine *Ereignisquelle* ist eine Entität, die Ereignisse *erkennt*, *erzeugt* und anschließend Ereignisobjekte *versendet* [vgl. LW08, S. 11]. Beispiele für Ereignisquellen

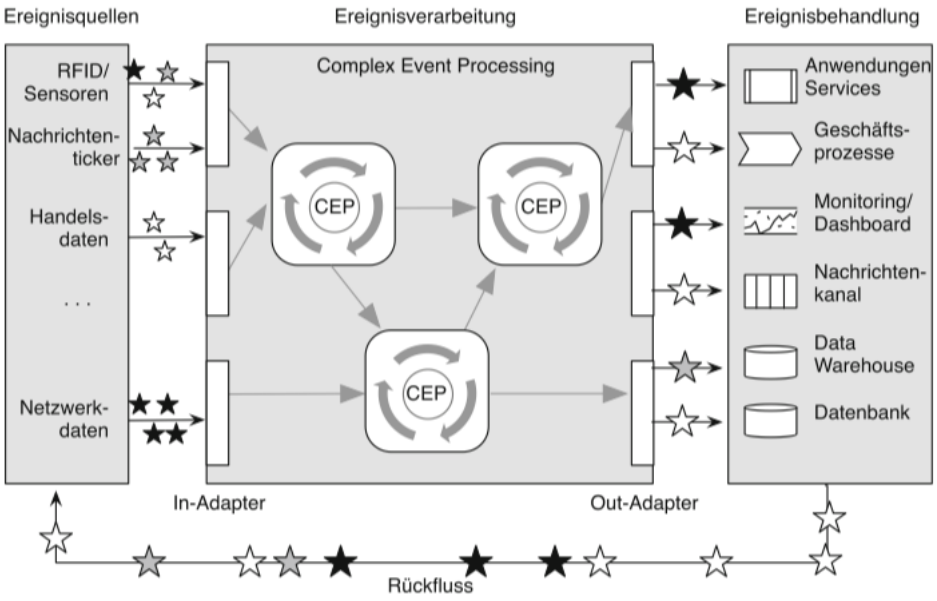

Abbildung 3.4.: Darstellung einer Event-Driven Architecture [BD15, S. 15]

sind in Abbildung 3.4 aufgeführt. Auch Ereigniskonsumenten können als Ereignisquellen fungieren [vgl. BD15, S. 16].

2. Schicht Ereignisverarbeitung Die *Ereignisverarbeitung* (engl. *event processing*) empfängt die Ereignisobjekte mittels der Ereignisströme von den Ereignisquellen. Sie *verarbeitet* die Ereignisobjekte, prüft die Ereignisströme auf Ereignismuster und abstrahiert komplexe Ereignisse [vgl. BD15, S. 16]. Eine tiefer gehende Untersuchung findet in Abschnitt 3.2.2 statt.

3. Schicht Ereignisbehandlung Die *Ereignisbehandlung* (engl. *event handling*) empfängt Ereignisse [vgl. LW08, S. 11]. In der 3. Schicht befinden sich vor allem die produktiven Unternehmensanwendungen, die auf Ereignisse *reagieren* [vgl. BD15, S. 16]. Typische *Ereigniskonsumenten* sind in Abbildung 3.4 dargestellt.

> *Event-Driven Architecture* ist ein Architekturstil, indem eine oder mehrere Komponenten des Systems *ereignisgesteuert* und diese zentraler Bestandteil der Architektur sind [vgl. LW08, S. 16].

3.2.1.2. Eigenschaften und Charakteristika

Sowohl Chandy [K10], [CS07], Etzion und Niblett [EN10] als auch Bruns und Dunkel [BD10] beschreiben ähnliche Eigenschaften einer EDA. Diese Arbeit folgt der Strukturierung von [BD10] und reichert spezifische Punkte mit wertvollen Beiträgen aus anderen Publikationen an. Am Ende dieses Abschnitts liefert Tabelle 3.2 eine komprimierte Übersicht der Eigenschaften und Charakteristika.

Verarbeitungsmodell einer EDA

In Abbildung 3.4 sind die wesentlichen Verarbeitungsschritte zu erkennen: Es gibt Komponenten die Ereignisse erzeugen, welche die konsumieren und jene die sowohl Ereignisse konsumieren als auch Komponente erzeugen. Anders als in Abbildung 3.4 kommunizieren die verschiedenen Komponenten nicht direkt miteinander, sondern über eine weitere Komponente, die gleichzeitig als Trennschicht fungiert und als *Mediator* bezeichnet wird. Wie in Abbildung 3.5 aufgezeigt, sendet die Ereignisquelle eine Nachricht mit den Daten über das aufgetretene Ereignis an den Mediator [vgl. BD10, S. 51]. Der Mediator distribuiert die Nachricht unmittelbar an die Ereignissenken. Durch diese Trennkomponente sind Erzeuger und Konsument *entkoppelt*.

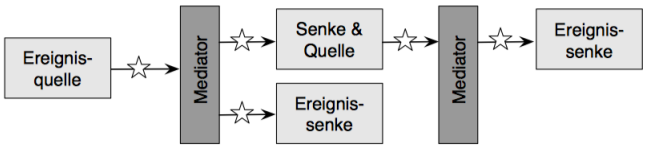

Abbildung 3.5.: Verarbeitungsmodell einer Event-Driven Architecture [BD10, S. 52]

Kommunikationsmuster von EDA

Nach Bruns und Dunkel [BD10] und Etzion und Niblett [EN10] weist das Kommunikationsmuster innerhalb einer EDA die Eigenschaften *Asynchronität*, *Publish/Subcribe Pattern* und *Push-Modus* auf. Diese Eigenschaften werden im Folgenden beschrieben und veranschaulicht:

Asynchronität Bei einer *asynchronen* Kommunikation wartet die abfragende Komponente nicht auf eine Antwort der angefragten Komponente, sondern fährt gleich mit dem nächsten Verarbeitungsschritt fort. Die angefragte Komponente muss weder erreichbar sein noch existieren [vgl. BD10, S. 53]. Setzt man die Asynchronität und das zuvor beschriebene Verarbeitungsmodell in Beziehung, ergibt sich Folgendes: Die *Ereignisquellen* versenden, ohne

auf eine Antwort zu warten, die Ereignisobjekte an die jeweiligen *Mediatoren*. Die Mediatoren buffern die Ereignisobjekte falls die Ereignissenke nicht verfügbar ist. Ansonsten distribuiert der Mediator die Ereignisobjekte an die Ereignissenken.
Daraus resultieren zwei wesentliche Unterschiede bei der Interaktion in EDA im Gegensatz zu SOA [vgl. EN10, S. 35]:

- Eine Ereignisquelle hat keine Erwartungshaltung gegenüber der Ereignissenke wie sie mit den Ereignissen umgeht.
- Kommunikation findet nur in eine Richtung statt. Die Ereignisquelle erwartet keine Antwort und beginnt mit der nächsten Aufgabe.

Publish/Subscribe Ereignisproduzenten senden bzw. publizieren (engl. *publish*) die Ereignisse nicht direkt an einen Ereigniskonsumenten, sondern an eine Middleware. Ereigniskonsumenten registrieren sich an der Middleware und abonnieren (engl. *subcribe*) die eintreffenden Ereignisströme [vgl. BD10, S. 54].

Push-Modus Nachrichten werden initiativ von der Ereignisquelle "gepusht". Die Ereignisquelle entscheidet wann die Nachricht an den Empfänger gesendet wird. In einer anfragegesteuerten Anwendung würde die Anfrage den "pull" auslösen. Im Pull-Modus gibt es daher eine Verzögerung zwischen dem Auftreten Ereignis in der realen Welt und der Ankunft bei dem Empfänger [vgl. K10, S. 10]. Abbildungen 3.6 und 3.7 veranschaulichen dies.

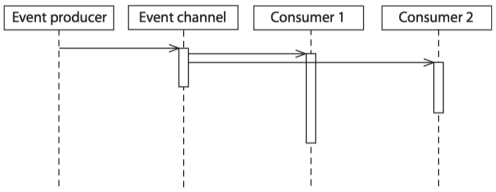

Abbildung 3.6.: Kommunikation im Push-Modus [EN10, S. 35]

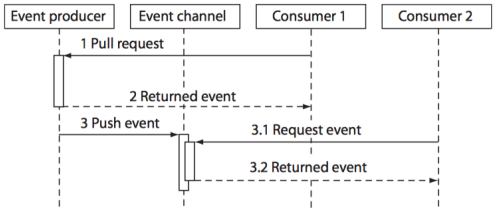

Abbildung 3.7.: Kommunikation im Pull-Modus [EN10, S. 38]

Aus den zuvor beschriebenen Eigenschaften und Auflistungen in [K10] und [BD10] ergeben sich folgenden Eigenschaften in komprimierter Form:

Eigenschaft	Erklärung
Fokussierung auf einzelne Ereignisse	Ereignisse sind zentraler Bestandteil der Architektur. Jedes Ereignis wird individuell übertragen und behandelt.
Lose Kopplung zwischen den Komponenten	kennen die Ereignissenken nicht. Ereignisobjekte enthalten kein Wissen über die folgende Verarbeitung. Die Fachlogik über die jeweilige Reaktion kennen ausschließlich die Ereigniskonsumenten.
Modularität	Durch den Einsatz von Mediatoren sind Quellen und Senken getrennt und es lassen sich leicht neue Komponenten in das System einfügen.
Schneller Durchlauf durch das Verarbeitungs-muster	Realisiert durch Push-Modus und asynchrone Interaktion.
Skalierung	Einzelne Komponenten der Schichten lassen sich durch die lose Kopplung auch physikalisch verteilen.

Tabelle 3.2.: Eigenschaften von EDAs

3.2.1.3. Anforderungen

Aus den Eigenschaften ergeben sich folgende funktionale Anforderungen an eine Event Driven Architecture:

EDA-F1	Ereigniskonsumenten können durch das unilaterale Verarbeitungsmodell einfach in das bestehende System eingefügt werden.
EDA-F2	Die Konsumenten sind durch das Kommunikationsmuster lose gekoppelt.
EDA-F3	Die einzelnen Komponenten lassen sich einfach verteilen.

Tabelle 3.3.: Anforderungen an eine EDA

3.2.1.4. Bewertung

Die beschriebenen Eigenschaften und Charakteristika besitzen spezifische Vor- und Nachteile. Diese werden in Tabelle 3.4 ersichtlich. Anschließend wird für jeden Aspekt die prägende Eigenschaft und der Kontext beschrieben.

Vorteile	**Nachteile**
Robustheit	Redundanz
Verteilung	Debugging
Agilität	Agilität
Großer Datendurchsatz	
Effektivität	

Tabelle 3.4.: Vorteile und Nachteile einer EDA

Vorteile

Robustheit Komponenten sind durch die *lose Kopplung* weitestgehend unabhängig und können voneinander getrennt geändert werden. Durch den *Push-Modus* wird keine Antwort erwartet und Fehlermeldungen müssen nicht behandelt werden [vgl K10, S. 12].

Verteilung Durch die *lose Kopplung* können die einzelnen Komponenten physikalisch voneinander gut getrennt werden [vgl. BD10, S. 74]. Durch die Asynchronität kommt es zu keiner Blockierung, falls Komponenten nicht verfügbar sind.

Agilität Die *lose Kopplung* ermöglicht es, einfach neue Ereignisquellen oder Ereigniskonsumenten in das System zu implementieren.

Großer Datendurchsatz Die Eigenschaften *Push-Modus* und *Asynchronität* verhindern eine hohe Latenz durch eine erwartete Antwort von der angefragten Komponente [vgl. EN10, S. 35]. Durch die erzeugte *Nebenläufigkeit* mittels Mediatoren, können mehrere Ereignisobjekte parallel verarbeitet werden.

Aktualität Die getätigten Auswertungen innerhalb einer EDA sind für den Nutzer in nahezu real-time verfügbar. Durch den *Push-Modus* werden Ereignisobjekte sehr schnell von Komponente zu Komponente weitergereicht.

Nachteile

Redundanz Eine negative Folge von *loser Kopplung* ist eine verstärkte Redundanz. Redundante Daten und Funktionalitäten können in mehreren Komponenten existieren [vgl. BD10, S. 74].

Debugging Durch die *Asynchronität* und die *Nebenläufigkeit* kann die Nachvollziehbarkeit von Fehlern schwer werden [vgl. Mic17, S. 3]. Die Nachvollziehbarkeit eines dynamischen Kontrollflusses ist in diesem Rahmen nicht einfach.

Komplexität Ähnlich wie für den Aspekt Debugging ist es schwer einen Kontrollfluss zu verfolgen. Wenn sich durch eine einfache Veränderung der Fluss stetig verändern lässt, ist es noch schwerer diesem zu folgen. Es gibt ein gewisses Maß von Unsicherheit durch Veränderungen am System [vgl. BD10, S. 75].

3.2.2. Complex Event Processing

Nachdem im vorherigen Abschnitt die Architektur beschrieben wurde, beschäftigt sich dieser Abschnitt mit der zentralen Komponente einer Event-Driven Architecture: der Ereignisverarbeitung via *Complex Event Processing*. Mittels ihr kann Ereignisorientierung voll ausgeschöpft werden [vgl. BD10, S. 58].

3.2.2.1. Konzept

Die Softwaretechnologie *Complex Event Processing* (CEP) ermöglicht eine dynamische Analyse von großen und unterschiedlichen Ereignisströmen in Echtzeit. CEP stellt Techniken bereit, um auf Beziehungen zwischen Ereignissen bzw. auf *Ereignismuster* zu reagieren und weitere Aktionen durchzuführen [vgl. Luc01, S. XVII]. CEP besteht grundsätzlich aus drei Komponenten:

- *Ereignisregeln:* Definieren einen *Bedingungsteil*, der ein Ereignismuster beschreibt und einem *Aktionsteil*, der ausgeführt wird wenn eine Ereignissequenz den Bedingungsteil erfüllt [vgl. BD15, S. 12].
- *Event Processing Agents (EPAs):* Überwachen die Ereignisströme und führen die Ereignisregeln aus [vgl. BD15, S. 13].
- *Event Processing Networks (EPN)* Ein Netzwerk von EPAs die in einem Netzwerk zusammen interagieren [vgl. BD15, S. 14].

Ereignisregeln und Event Processing Language

Menschliche Fachexperten formulieren ihr Wissen oft in Regeln [vgl. BD15, S. 12]. Mittels der Ereignisregeln soll dieses Fachwissen in das System integriert werden. Die Formulierung dieser Regeln erfolgt durch eine *Event Processing Language* [vgl. BD15, S. 12]. Eine Ereignisregel besteht aus zwei Teilen und beschreibt wie auf ein Erkennen eines Musters reagiert werden soll [vgl. BD15, S. 12].

Bedingungsteil Der Bedingungsteil (engl. *condition*) ist ein *Ereignismuster* [vgl. Luc01, S. 118] und prüft, ob eine zeitlich geordnete Sequenz von Ereignissen in das Muster passt. Um ein Ereignismuster genauer spezifizieren zu können, benötigt man eine Menge von definierten Operatoren die in einer *Ereignisalgebra* zusammengefasst sind. Eine detailiertere Betrachtung von Ereignisalgebren lässt sich in [ZU99] und [BD10] finden. Eine beispielhafte Ereignisalgebra mit den wichtigsten Operatoren ist in Anhang B.1 aufgeführt.

Windows Theoretisch können alle auftretenden Ereignisse im Arbeitsspeicher gespeichert werden um sie auf Ereignismuster zu untersuchen. Da CEP allerdings im Big Data Kontext eingesetzt wird, kann dies den RAM schnell füllen. Eine Möglichkeit besteht darin, zeitliche Kontexte zu erstellen in denen Ereignisse betrachtet werden. Diese Zeitfenster werden als Window betitelt [vgl. Hed17, S. 25].

Aktionsteil Eine Mustererkennung löst eine Aktion (engl. *action*) aus. Nach Luckham [Luc01] kann eine Aktion den Zustand des EPAs ändern, ein neues Ereignis generieren oder ein Ereignis passieren lassen. Bruns und Dunkel [BD15] beschreiben, dass Ereignisse erzeugt und Dienste angestoßen werden können. In diesem Schritt werden die in den Grundlagen beschriebenen *komplexen Ereignisse* erzeugt. Möglichen Optionen werden in komprimierter Form in Anhang B.2 aufgelistet. Listing 3.1 stellt eine mögliche Ereignisregel dar.

```
1 //Bedingungsteil
```

```
if(aktieApple.wert > 170) {
    //Aktionsteil
       return new VerkaufenEreignis(aktieApple, anzahl=200)
}
```

Listing 3.1: Eine einfache Ereignisregel

Event Processing Language Eine Event Processing Language (EPL) ist eine High Level Programmiersprache, die es auch Nicht-Programmieren ermöglichen soll Ereignisregeln zu definieren [vgl. LW08, S. 13]. Sie stellt eine Realisierung der Ereignisalgebra (siehe Anhang B.1) für ein bestimmtes System oder Anwendung dar und soll die Ereignisregeln von der logischen Programmierung entkoppeln. Durch die verfügbare EPL wird entschieden, wie einfach es ist komplexere Ereignisregeln zu definieren. Die EPL bestimmt, welche primitiven bzw. grundsätzlichen Datentypen genutzt werden können [vgl. EN10, S. 13]. In Anhang C werden exemplarisch zwei EPLs analysiert, um daraus Anforderungen für die spätere Evaluation zu generieren.

Event Processing Agent

Ein *Event Processing Agent* ist ein Softwaremodul, welches Ereignismuster erkennen kann und komplexe Ereignisse verarbeitet [vgl. LW08, S. 13]. Er führt Ereignisregeln aus [vgl. Luc01, S. 175]. Meistens sind sie modular auf eine Aufgabe ausgerichtet [vgl. Hed17, S. 102]. Im Kontext zum Verarbeitungsmodell einer EDA kann ein EPA Ereignisquelle, Ereignissenke und Hybrid sein. Die wichtigsten Aufgaben eines EPA sind die *Überwachung des Ereignisstroms*, die *Mustererkennung* und die anschließende *Reaktion* [vgl. BD10, 139f]. Im Folgenden werden die einzelnen Komponenten eines EPA beschrieben, anschließend drei wichtige EPAs vorgestellt und zuletzt leichtgewichtige und schwergewichtige EPAs verglichen. Die Abbildung 3.8 visualisiert den Aufbau eines EPA.

Ereignismodell Das Ereignismodell spezifiziert die erlaubten Ereignistypen, deren Attribute sowie die Beziehungen und Abhängigkeiten zwischen diesen Typen [vgl. BD15, S. 13]. Das Ereignismodell enthält keine Verarbeitungslogik. Die Trennung von Struktur und Verarbeitungslogik sorgt für eine Entkopplung [vgl. BD10, S. 95].

Ereignisregeln Ein EPA besitzt eine Menge von Ereignisregeln. Der Event Processing Agent prüft den Ereignisstrom auf diese Ereignisregeln [vgl. BD15, S. 14].

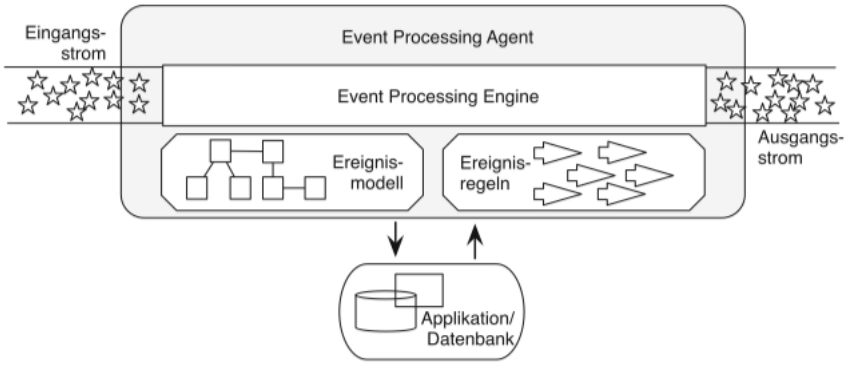

Abbildung 3.8.: Aufbau eines Event Processing Agent [vgl. BD15, S. 13]

Event Processing Engine Die Hauptaufgabe der *Event Processing Engine* (EPE) ist das Erkennen und Verarbeiten von Ereignismustern [vgl. BD15, S. 32]. Die EPE erbringt die *Arbeitsleistung*, wohingegen die anderen beiden Komponenten Struktur und Logik bereitstellen. Sie ist eine spezielle Art von *Regel Interpreter*, die auf die kontinuierliche Verarbeitung von großen Datenmassen im Arbeitsspeicher optimiert ist [vgl. BD10, S. 68].

Leichtgewichtige vs. schwergewichtige EPAs Event Processing Agents können theoretisch eine beliebige Anzahl von Ereignisregeln verwalten. In der Regel sorgt eine Aufteilung der EPAs in klar definierte Teilaufgaben und die Verarbeitung in kleinen Schritten dafür, dass die Komplexität sinkt und damit das Verständnis über das System steigt. Dieses Vorgehen sorgt für eine *lose Kopplung* [vgl. BD10, S. 71]. Die Agenten sind *modular* realisiert und CEP wird den Multiagentensystemen zugeordnet [vgl. Hed17, S. 102]. Dies ermöglicht außerdem eine physikalische Verteilung.
Schwergewichtige EPA sind rückschlüssig *schwieriger wartbar*, *weniger transparent*, aber oft *hoch optimiert* [vgl. BD10, S. 72].

Unterschiedliche EPAs Es ist zielführend den Agenten kleine Aufgaben ausführen zu lassen. In [EN10] und [CEA11] werden drei wichtige Aufgaben beschrieben: *Filter*, *Map*, *Constraints*. Luckham [Luc01] betrachtet einen EPA als Interface und EPA Instanzen als die Realisierung. Bei der Implementation eines EPA wird das nötige Ereignismodell und die Ereignisregeln für die jeweilige Aufgabe eingesetzt.

Filter EPA Filter EPAs sind wichtig, um die Datenlast im System gering zu halten. Es ist günstiger und schneller Ereignismuster auf relevanten Daten zu suchen, anstatt auf al-

len [vgl. Luc01, S. 187]. Mittels entsprechender Ereignisregeln ist es beispielsweise möglich, uninteressante oder doppelte Ereignisse herauszufiltern.

Map EPA Map EPAs aggregieren mittels Ereignisregeln viele Ereignisse zu einem neuen Ereignis. Durch Aggregation werden fachlich relevantere bzw. *komplexere Ereignisse* auf einer höheren Abstraktionsebene erzeugt [vgl. Luc01, S. 192].

Constraint EPA Die Hauptaufgabe von Constraint EPAs ist das detektieren von Vorfällen. Sie haben weniger eine produzierende, sondern vielmehr eine überwachende Aufgabe. Mittels Ereignisregeln wird überprüft, ob fachliche Regeln wie Sicherheitsbestimmungen oder Geschäftsprozesse eingehalten werden [vgl. Luc01, S. 195].

> Ein *Event Processing Agent* ist ein Softwaremodul, welches *Ereignismuster* in *Ereignisströmen* erkennen kann und *komplexe Ereignisse* verarbeitet [vgl. LW08, S. 13].

3.2.2.2. Eigenschaften und Charakteristika

Aus dem beschriebenen Konzept und deren einzelnen Komponenten ergeben sich folgende Eigenschaften und Charakteristika:

Eigenschaft	Erklärung
Modularität	Die *hohe Kohäsion* und *geringere Kopplung* einzelner EPAs sorgt dafür, dass sie als eigenständiges Modul betrachtet werden können.
Echtzeitfähigkeit	*Event Processing Engines* sind dafür konzipiert große Datenmengen zu verarbeiten. Durch *Filter Agents* wird die Datenlast verringert. Durch viele EPAs wird eine Nebenläufigkeit erzeugt.
Dynamik	Durch die *Modularität* und *lose Kopplung* können EPAs im laufenden Betrieb erzeugt oder freigegeben werden.
Skalierbarkeit	Die logische und fachliche Abgrenzung der EPAs untereinander sorgt dafür, dass auch eine physikalische Verteilung auf anderen Servern gelingt.
Explizite Wissensrepräsentation	Wissen ist deklarativ mittels *EPL* in Regeln formuliert und explizit in der CEP Komponente angegeben [vgl BD10, S. 61].

Tabelle 3.5.: Eigenschaften von CEP

3.2.2.3. Anforderungen

Aus den Eigenschaften ergeben sich folgende funktionale Anforderungen an das Complex Event Processing:

CEP-F1	CEP ist in der Lage unterschiedliche Ereignisse in Ereignisströmen in nahezu Echtzeit auszuwerten.
CEP-F2	CEP besitzt eine Ereignisalgebra und Window Funktionen, um Ereignisse in Ereignisströmen ausreichend zu verarbeiten.
CEP-F3	EPAs sind lose miteinander gekoppelt und können physikalisch verteilt werden.
CEP-F4	CEP verfügt über genügend und einfache Adapter, um Ereignisströme aufnehmen und abgeben zu können.
CEP-F5	CEP besitzt eine einfache, explizite Event Processing Language, um Ereignisregeln zu definieren.

Tabelle 3.6.: Anforderungen an CEP

Aus Anhang C Exemplarische Event Processing Languages ergeben sich folgende funktionale Anforderungen an eine Event Processing Language:

EPL-F1	Die ereignisorientierte Anwendung besitzt eine EPL, die einfach genug zu verstehen ist, sodass auch Nicht-Programmierer Regeln in das System einpflegen können.
EPL-F2	Die ereignisorientierte Anwendung besitzt eine EPL, die nicht proprietär ist.

Tabelle 3.7.: Anforderungen an eine EPL

3.2.2.4. Bewertung

Die beschriebenen Eigenschaften und Charakteristika besitzen Vor- und Nachteile. Tabelle 3.8 beschreibt diese. Anschließend wird für jeden Aspekt die prägende Eigenschaft und der Kontext beschrieben.

Vorteile	Nachteile
Agilität	Limitierte Historie
Wartbarkeit	Undurchsichtiger Kontrollfluss
Skalierbarkeit	Nachvollziehbarkeit
Effizienz	
Wiederverwendbarkeit	
Explizite Regeln	

Tabelle 3.8.: Bewertung von CEP

Vorteile

Agilität CEP ist durch die *Modularität* und *Dynamik* flexibel und anpassungsfähig. EPAs können im laufenden Prozess hinzugefügt oder entfernt werden, um Lastspitzen auszugleichen [vgl. Luc01, S. 212].

Wartbarkeit Durch die hohe *Kohäsion* und die *geringe Kopplung* lassen sich Agenten explizit einzeln testen. EPAs können aus dem EPN entfernt werden. Dies hilft zusätzlich beim *Debugging* [vgl. Luc01, S. 212].

Skalierbarkeit Durch die logische Abtrennung mittels *Modularität* lassen sich EPAs gut physikalisch verteilten.

Effizienz CEP wurde designt um massive Datenströme zu verarbeiten. Die Event Processing Engines sind darauf optimiert, Ereignisströme im Arbeitsspeicher zu bearbeiten.

Wiederverwendbarkeit Die *Modularität* sorgt dafür, dass EPAs eines bestimmten Typs mehrfach als eigenständige Instanzen im System oder anderen Projekten genutzt werden können.

Explizite Regeln Dadurch, dass die Ereignisregeln von der Programmierlogik getrennt sind, können Experten eigenständig die Ereignisregeln anpassen.

Nachteile

Limitierte Historie Ereignisse können nicht ewig im Arbeitsspeicher verweilen. Sind Intervalle nicht groß genug gewählt, werden wichtige Ereignismuster eventuell nicht erkannt.

Undurchsichtiger Kontrollfluss Auf Grund der *hohen Nebenläufigkeit* durch die einzelnen Ebenen der EPAs, ist es bei vielen Knoten schwer den Kontrollfluss zu verfolgen.

Nachvollziehbarkeit Durch die manuelle Anpassung von Regeln, ist es schwer Aktionen im System nachzuvollziehen.

4. Konzeption

Auf Grundlage der Analyse aus Kapitel 3 wird in diesem Kapitel ein Konzept für eine ereignisorientierte Anwendung mittels Apache Flink und CEP entwickelt und vorgestellt.
Die Gliederung der Konzeption folgt dem Aufbau der Analyse. Zunächst wird in Abschnitt 4.1 mittels fachlicher Aspekte eine geeignete ereignisorientierte Anwendung mit funktionalen Anforderungen definiert. In der technischen Konzeption in Abschnitt 4.2 wird diese Anwendung dann nach demTop-Down-Ansatz entworfen.

4.1. Fachliche Konzeption

Für die in diesem Kapitel folgende technische Konzeption, der darauf folgenden Realisierung und der abschließenden Evaluation wird eine geeignete Beispielanwendung benötigt. Die fachliche Konzeption einer solchen Beispielanwendung folgt dem Abschnitt 3.1 aus der Analyse. Die dort erarbeiteten Erkenntnisse sollen helfen, eine sinnvolle Anwendung zu konzipieren, die ereignisorientiert ist und theoretisch einen wirtschaftlichen Nutzen aufweist.

4.1.1. Beschreibung der Beispielanwendung

Das als dritte Epoche bezeichnete Informationszeitalter stellt neue Herausforderungen an Unternehmen und deren Interaktion mit der Umwelt dar (vgl. Abschnitt 3.1.1). Das Aufkommen von sozialen Netzwerken in denen viele Millionen Menschen miteinander kommunizieren und Informationen austauschen, führt einerseits dazu, dass Unternehmen direkter mit seinen Stakeholdern kommunizieren können, andererseits kann sich die öffentliche Meinung bezüglich eines Unternehmens (engl. *image* bzw. *public relations*) innerhalb von Sekunden verändern. Vor allem sich schnell verbreitende Nachrichten von (nicht immer) renommierten Zeitungen und Magazinen oder Beiträge von Nutzern mit erheblicher Anzahl von Followern (engl. *influencer*) können ein Unternehmen massiv beeinflussen. Als sich etwa die Influencerin Kylie Jenner mittels Tweet über das Unternehmen Snapchat äußerte, stürzte die Aktie zeitweise um 8% und vernichtete so einen Börsenwert von 1,7 Milliarden Dollar [FAZ18].
Auch der digitale und automatisierte Aktienhandel hat mit der Digitalisierung Einzug ge-

halten. Wie das Beispiel zeigt, werden Aktienkurse viel schneller durch Ereignisse in der Außenwelt beeinflusst und eine schnelle Reaktion auf diese scheint entscheidend zu sein. In der prototypischen Beispielanwendung sollen daher aktuelle Twitter- und Finanzdaten in Echtzeit ausgewertet und miteinander in Verbindung gebracht werden, um Kausalitäten bzw. Korrelationen zu erkennen. Um die Anwendung überschaubar zu halten, sollen Ereignisse, die das Unternehmen Tesla betreffen analysiert und zu komplexen Ereignissen zusammengefügt werden. Mit diesen komplexen Ereignissen sollen Handlungsentscheidungen/Informationen für/über das Unternehmen Tesla generiert werden. Im Folgenden soll diese lose Beschreibung genauer spezifiziert werden.

Bruns und Dunkel [BD10] beschreiben ein mögliches Vorgehen bei der Entwicklung von EDA-Anwendungen. Diese Arbeit folgt der dort beschriebenen Empfehlung mit möglichen Ereignisquellen zu beginnen. Im Kapitel 2 werden bereits implizit Anforderungen an geeignete Ereignisquellen gestellt, wie etwa das individuelle Eintreffen von vielen Ereignissen. Diese Anforderungen werden sowohl von einem Datenstrom von *Tweets* von der Social Media Plattform Twitter als auch von einem Strom von *Aktienkursen* erfüllt.
In Abschnitt 3.2 werden Kategorien genannt, in denen sich geeignete Anwendungsbereiche für ereignisgesteuerte Anwendungen befinden. Mittels der Kategorien soll die Anwendung konkretisiert bzw. die Ereignisquellen auf die Eignung überprüft werden:

Komplexe Analyse von Daten Die heterogenen Daten, bestehend aus Tweets sowie Aktienkursen sollen aggregiert, Ereignismuster zwischen den unterschiedlichen Ereignissen und Ereignistypen erkannt und neue Ereignisse abstrahiert werden.

Große Datenvolumina Aktienkurse ändern ständig in kurzer Zeit ihren Wert und mit einer aktiven Nutzerzahl von bis zu 336 Millionen [Sta] können viele Tweets entstehen.

Geringe Latenzzeit Für die Verarbeitung beider Ereignisquellen bietet sich eine geringe Latenzzeit an. Interaktionen folgen meistens direkt auf auftretende Ereignisse.

Es soll auf Stärken und Schwächen aus Abschnitt 3.1.5 gesondert eingegangen werden, um den theoretisch wirtschaftlichen Nutzen aufzuzeigen:

Informationsverfügbarkeit Vor allem im High-frequency-trading (HFT) ist der zeitliche Wert der Information immens. Auch im Bereich Public Relations ist oft eine schnelle Reaktion von Nöten [vgl. Chl+11], [vgl. WH08].

Einsparung von Kosten/Verlust von Informationen Das Aggregieren bzw. die Synthese von neuen komplexen Ereignissen und die Speicherung dieser in einer Datenbank sorgt für weniger Hardwarekosten und der Verlust von Informationen verringert sich. In der Beispielanwendung soll exemplarisch eine Datenbank angeschlossen werden.

4.1.2. Anforderungen

Eine solche Anwendung kann für eine Vielzahl der Stakeholdergruppen interessant sein. Umso größer können die Anforderungen an solch eine Anwendung sein. Der Fokus dieser Arbeit beruht nicht auf einer möglichst vollständigen Anwendung, sondern auf einer Evaluation von CEP mittels Apache Flink. Daher werden exemplarisch einige funktionale Anforderungen konstruiert.

ANW-F1	Die Anwendung erzeugt eine Mitteilungen, wenn die letzten fünf Kursänderungen der Tesla Aktie positiv verlaufen sind.
ANW-F2	Die Anwendung erzeugt eine Mitteilung, wenn sich der Aktienkurs der Tesla-Aktie innerhalb innerhalb von 5 Minuten um mindestens 3% verändert.
ANW-F3	Die Anwendung erzeugt eine Mitteilung, wenn relevante Nachrichten-Seiten mindestens 3 mal über das Unternehmen Tesla oder Elon Musk in den letzten 20 Minuten getwittert haben.
ANW-F4	Die Anwendung erzeugt eine Mitteilung, die jede Minute eine Statistik zur öffentlichen Meinung über Tesla auf Twitter ausgibt.
ANW-F5	Die Anwendung zeigt, ob ein Tweet von Tesla oder Elon Musk zu einer Kursveränderung (ANW-F4) innerhalb von 15 min führt.
ANW-F6	Die Anwendung zeigt an, wie positiv/negativ ein Tweet von Tesla oder Elon Musk von den Twitter-Nutzern aufgenommen wurde.
ANW-F7	Die Anwendung speichert die aus ANW-F5 und ANW-F6 relevanten Tweets in einer Datenbank.

Tabelle 4.1.: Fachliche Anforderungen an die Beispielanwendung

4.2. Technische Konzeption

Die fachliche Konzeption soll im Folgenden durch die technische Konzeption konkretisiert werden. Um die Erkenntnisse aus der Analyse erfolgeich umsetzen zu können, beruht die technische Konzeption auf den Schilderung von Siedersleben [Sie04], Luckham [Luc01] sowie Bruns und Dunkel [BD10]. Alle drei Werke liefern zum Teil unterschiedliche oder isolierte Beiträge zur Entwicklung des Konzepts, von denen jeweils für diese technische Konzeption der passendste Ansatz gewählt wird. Folgende Punkte werden jedoch von allen aufgeführt:

Entwurfsmuster-Orientierung Entwurfsmuster sind erprobte Problemlösungen für ein wiederkehrendes Entwurfsproblem [vgl. BD10, S. 177]. Im Allgemeinen sind Entwurfsmuster bekannt bzw. leicht zu finden. Sie helfen bestehende Lösungsansätze zu nutzen und ersparen dem Architekten/Programmierer Zeit bei der Erstellung. Durch die Bekanntheit wird es Dritten ermöglicht sich schneller in Ideen hineinzudenken. Dies erhöht die *Wartbarkeit* und verkürzt die Einarbeitungszeit [vgl. Qui05].

Komponenten-Orientierung Eine Komponente ist eine *modulare Einheit* mit klar definierten Schnittstellen. Sie versteckt die Implementierung und kann durch eine andere Komponente, welche die gleichen Schnittstellen exportiert und importiert ersetzt werden. Außerdem eignen sich Komponenten zur *Wiederverwendbarkeit* [vgl. Sie04, S. 42f].

Referenzarchitektur-Orientierung Eine Architektur beschreibt, wie Komponenten interagieren und kooperieren. Sie beschreibt die übergeordnete Struktur eines Softwaresystems [vgl. BD10, S. 202]. Eine Referenzarchitektur kann als eine Blaupause für eine konkrete Architektur gesehen werden.

Auftretende Begriffe, wie beispielsweise *Wartbarkeit* und *Modularität* lassen sich in vorangegangen Analyse häufig als gewünschte Eigenschaften oder in den Bewertungen finden. Daher werden diese drei Designentscheidungen elementarer Bestandteil dieses Entwurfs.

4.2.1. Event-Driven Architecture

In diesem Abschnitt wird eine Event-Driven Architecture entworfen, die auf Grundlage des theoretischen Teils 3.2.1 aus der Analyse die fachliche Konzeption in Abschnitt 4.1 umsetzen soll.

4.2.1.1. Systemarchitektur

Um den Top-Down-Ansatz konsequent umzusetzen, wird den Ausführungen aus [Sie04] gefolgt und zuerst eine Systemarchitektur entworfen, die im späteren Verlauf aus verschiedenen Perspektiven betrachtet und dekomponiert wird. Als Grundlage dient die beschriebene EDA-Architektur (Abschnitt 3.2.1.1) aus [BD10]. Sie stellt eine Basis dar, um die einzelnen Komponenten direkt zu verorten. Die Abbildung 4.1 zeigt die konkrete Systemarchitektur.

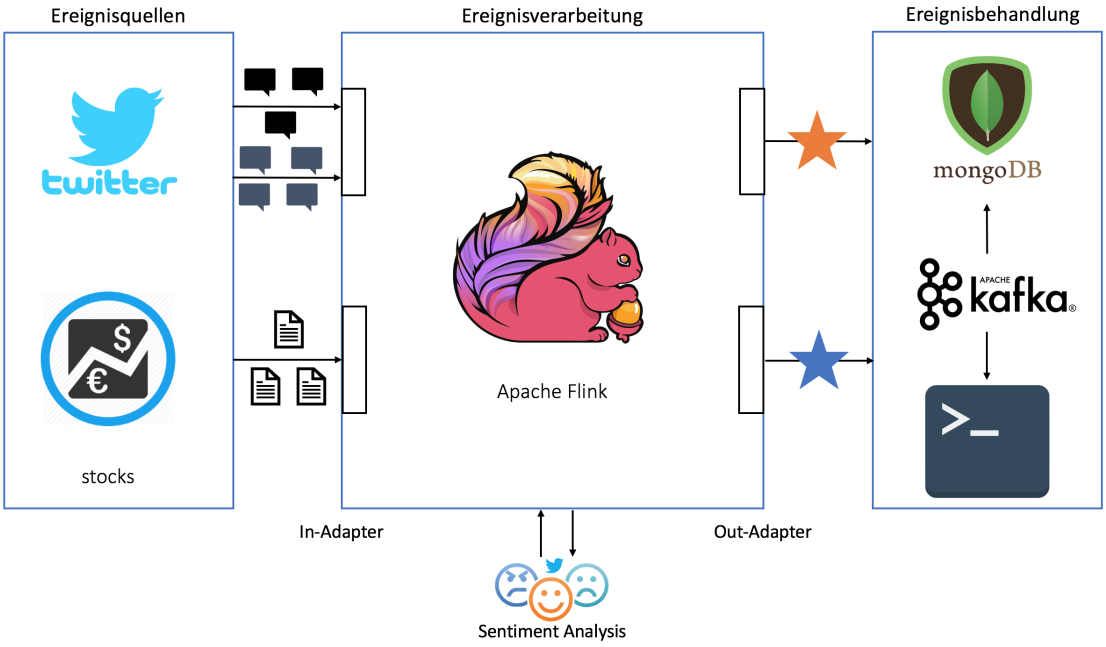

Abbildung 4.1.: Übersicht über die Systemarchitektur (angelehnt an [BD15, S. 15])

Die Event-Driven Architecture stellt die Außensicht der ereignisorientierten Anwendung dar. In ihr werden die Schnittstellen der Ereignisquellen, der Ereignisverarbeitung und der Ereignisbehandler definiert. Dem typischen Kommunikationsmuster einer EDA (vgl. Abschnitt 3.2.1.2) wird gefolgt. Die Ereignisquelle Twitter stellt initiativ über ihre Schnittstelle Tweets zur Verfügung, die in Apache Flink weiterverarbeiteten werden sollen. Die Ereignisverarbeitung hat keinen Einfluss darauf, zu welchem Zeitpunkt die Tweets an der Schnittstelle bereitstehen (Push-Modus). Die Ereignisquelle Twitter besitzt keine konsumierende Schnittstelle und hat keine Erwartungshaltung gegenüber Apache Flink. Die Kommunikation findet nur in eine Richtung statt (Asynchronität). Dies gilt analog zu der anbietenden Schnittstelle der Ereignisquelle Aktienmarkt.
Mittels der Komponente Sentiment Analysis sollen Tweets auf ihre Haltung bzw. ihrer posi-

tiven oder negativen Tendenz untersucht werden und bei der erfolgreichen Umsetzung von ANW-F4, ANW-F6 und ANW-F7 (siehe Tabelle 4.1) beitragen. Anders als Ereignisquellen und Ereignisbehandler ist die Kommunikation synchron und pull-basiert. Die Ereignisverarbeitung sendet den Tweet an die API und erhält im Gegenzug ein Ergebnis.
Die konkreten Ereignisbehandler Terminal und MongoDB konsumieren von Apache Kafka. Apache Kafka folgt dem Kommunikationsmuster in einer EDA und abonniert die von der Ereignisverarbeitung bereitgestellten Ereignisströme (Publish/Subcribe). Apache Flink erhält keine rückwärtigen Informationen über die Verarbeitung der Ereignisse. Diese Komponenten sollen nun einzeln beschrieben werden.

4.2.1.2. Ereignisquellen

Als Ereignisquellen sollen folgende Entitäten dienen, die unterschiedliche Ereignisse bereitstellen.

Twitter Innerhalb der ereignisorientierten Anwendung werden Tweets von bestimmten Nutzern und Tweets mit bestimmten Inhalten benötigt. Um die fachlichen funktionalen Anforderungen zu erfüllen, ist nur eine Teilmenge der Tweets nötig. Folgende Nutzer und Tweets mit bestimmten Inhalten sollen abonniert werden:

- Tweets von den Nutzern: Tesla, Elon Musk, CNN, Bloomberg, The Economist, The Telegraph, Handelsblatt
- Tweets, die folgende Wörter beinhalten: Tesla, TSLA, Elon Musk

Für die Ausführung von Tests soll noch ein weiterer Testnutzer hinzugefügt werden, um bestimmte Ereignisregeln absichtlich auslösen zu können.

Aktienmarkt Für die Erfüllung der fachlichen funktionalen Anforderungen sind Aktienkurse des Unternehmens Tesla (Symbol: TSLA) nötig. Das Abfragen von Echtzeit-Aktienkursen von unterschiedlichen Anbietern ist mit hohen Kosten verbunden und reale Aktienkurse lassen sich nicht für Testzwecke beeinflussen. Daher soll eine eigene Quelle für Aktienkurse geschaffen werden. Um eine gewisse Ähnlichkeit zu einem realen Aktienstrom zu erzeugen, sollen zusätzlich Aktienkurse der Unternehmen BMW, Daimler und VW in die Datenquelle implementiert werden. Ein Ereignis soll die Form *symbol, price* als String repräsentieren.

4.2.1.3. Ereignisverarbeitung und Apache Flink

Die Ereignisverarbeitung und somit die ereignisorientierte Beispielanwendung soll in der Beispielanwendung mittels Apache Flink und seiner CEP Library stattfinden. Apache Flink ist ein Framework und eine verteilte *processing engine* (vgl. Abschnitt 3.2.2.1) für die Verarbeitung von Datenströmen im Arbeitsspeicher [vgl. Fli18f]. Als Kern API steht hierfür die Streaming API zur Verfügung. Sie stellt Klassen und Funktionen bereit, um die Datenströme zu verarbeiten. Eine quantitative Untersuchung liefern Jacobs und Surdy [JS16].

Konnektoren Durch die von der Streaming API bereitgestellten Konnektoren werden die Schnittstellen zu den anderen Komponenten der Event-Driven Architecture realisiert. Mittels dieser Schnittstellen erfolgt der technische Zugriff auf Ereignisquellen und das Publizieren der komplexen Ereignisse als Datenstrom. Für die Beispielanwendung sollen unter anderem der Twitter Konnektor als Ereignisquelle und der Apache Kafka Konnektor als Ereignissenke dienen. Weitere Konnektoren sind unter [Fli18e] aufgeführt.

FlinkCEP FlinkCEP ist die Complex Event Processing Library von Apache Flink. Sie stellt besondere Operationen für die Verarbeitung von Ereignissen bereit. Der Inhalt der Library spiegelt somit die Ereignisalgebra wieder und ihre Funktionen entsprechen der Event Processing Language. Diese soll in der Beispielanwendung genutzt werden.

Verteilung Apache Flink bietet eine Reihe von Möglichkeiten für eine Ausführung der Anwendung auf mehreren PCs. Eine Auflistung der unterschiedlichen Methoden lässt sich unter [Fli18d] finden. Eine detaillierte Erstellung der Verteilung lässt sich in Abschnitt 4.2.3 finden.

4.2.1.4. Sentiment-Analyse

Die Sentiment-Analyse (engl. sentiment analysis) ist eine Kategorie des Natural Language Processing (NLP) und befasst sich mit der Klassifizierung von Texten. Mittels Sentiment-Analyse soll für einen Text erkannt werden, ob der Text bzw. der Verfasser eine positive, neutrale oder negative Meinung vertritt und ob es sich eher um einen objektiven oder subjektiven Text handelt.
Die Sentiment-Analyse soll in der Beispielanwendung genutzt werden, um Tweets die sich auf das Unternehmen Tesla beziehen auf ihre Meinung zu untersuchen. Das Ergebnis soll dem Tweet hinzugefügt werden (*content enrichment*), um in folgenden Analysen Aussagen über die allgemeine Reaktion auf Tweets oder das Stimmungsbild auf Twitter zu untersuchen.

Anders als bei der sonstigen unilateralen Kommunikation in der EDA findet hier eine bilaterale Verständigung statt, indem eine Anfrage mit dem zu untersuchenden Text gesendet und auf eine Antwort gewartet wird (*request-repsonse pattern*). Die Sentiment-Analyse zählt zum Bereich des Text Minings und benötigt daher für eine korrekte Klassifikation viele Beispieldaten. Um schnell korrekte Ergebnisse zu erhalten, soll die Analyse durch einen externen Anbieter mittels API Aufruf erfolgen.

4.2.1.5. Ereignisbehandlung

Als Ereignisbehandler sollen folgende Entitäten dienen, die unterschiedliche Aufgaben erfüllen:

Apache Kafka Apache Kafka ist eine verteilte Streaming-Plattform und wird genutzt, um *real-time data pipelines* zu realisieren [vgl. Kaf18]. Ziel ist es, Datenströme über mehrere Knoten zu verteilen und sie parallel für mehrere Anwendungen konsumierbar zu machen. Apache Kafka beruht auf dem Publish/Subscribe Pattern und ist daher geeignet in dieser konkreten Event-Driven Architecture eingesetzt zu werden [vgl. Sha17]. Die Software kann als eine Art Verbindungskomponente zwischen mehreren Anwendungen, Systemen und Datenbanken gesehen werden. Eine umfassende Einarbeitung und Beschreibung in Apache Kafka ist innerhalb des Zeitrahmens nicht möglich, trotzdem soll der theoretisch mögliche Einsatz der Anwendung in einer größeren Umgebung gezeigt werden. Die Kernabstraktion von Apache Flink ist das Topic [vgl. Kaf18]. Ein Topic lässt sich als Daten-Pipeline betrachten in dem ein oder mehrere Produzenten Daten publizieren (*publish*). Ein oder mehrere Konsumenten abonnieren (*subscribe*) ein Topic und konsumieren den Datenstrom (vgl. Abschnitt 3.2.1.2).
Die später realisierte Anwendung soll in mehrere Topics unterschiedliche Datenströme schreiben. Neben den für die fachlichen Anforderungen benötigten Topics sollen ebenfalls “Debugging Topics“ angelegt werden, die gegebenenfalls für Kontroll- und Debuggingzwecke genutzt werden sollen. Abbildung 4.2 stellt die geplanten Topics und die jeweiligen Interaktionen dar.

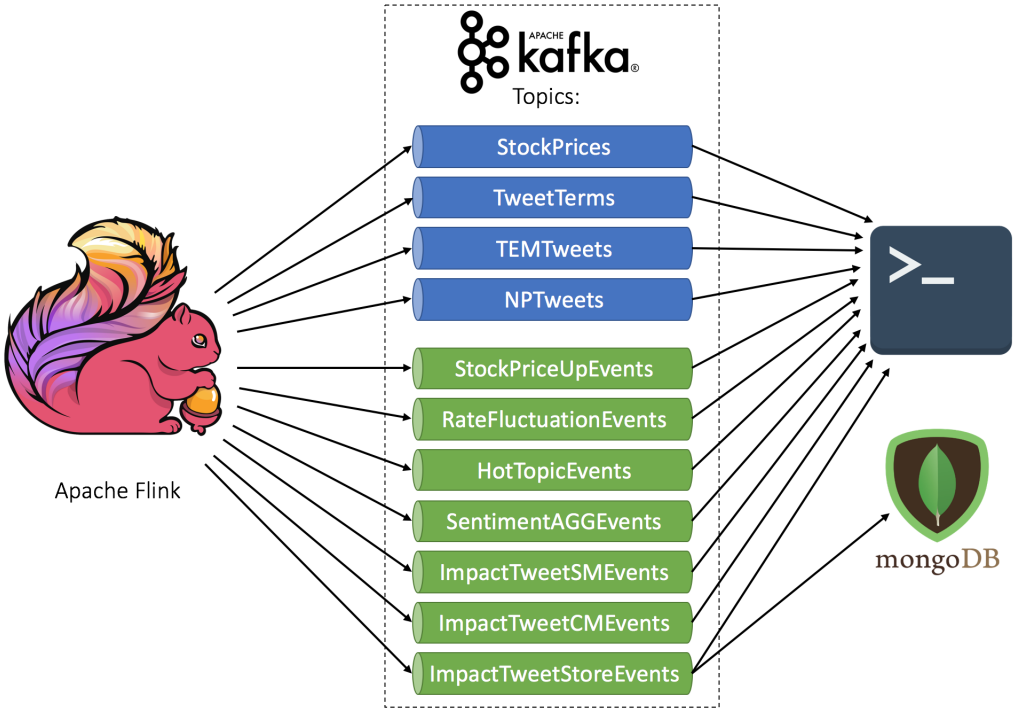

Abbildung 4.2.: Interaktion zwischen Ereignisverarbeitung und Ereignisbehandlern sowie verbindende Kafka Topics.

MongoDB MongoDB ist eine dokumentenorientierte Datenbank, kann Daten schemafrei speichern und ist für eine horizontale Skalierung optimiert [Ded+13]. Eine tiefere Beschreibung und Einarbeitung soll, genau wie für Apache Kafka, für diese Arbeit nicht erfolgen. Um ANW-F7 zu erfüllen, sollen mittels eines Hilfs-Programms Ereignisse aus dem *ImpactTweetStoreEvents* Topic konsumiert und automatisch in die Datenbank *ImpactTweets* abgelegt werden.

Terminal Als einfache Ereignisbehandlung soll die Ausgabe eines Ereignisstroms als String im Terminal erfolgen. Diese Behandlung soll andere Anwendungsprogramme repräsentieren, die ein Kafka Topic abonnieren und komplexe Ereignisse aus Apache Flink verarbeiten können.

4.2.2. Complex Event Processing

Nachdem die Ereignisquellen und Ereignisbehandler spezifiziert wurden, soll eine ereignisorientierte Anwendung konzipiert werden, die die fachlichen Anforderungen mittels Apache Flink und FlinkCEP technisch umsetzt. Die folgenden Planungen können als Konzeption der Innensicht der ereignisorientierten Anwendung verstanden werden. Als Erstes werden die

fachlichen Anwendungsfälle in Ereignisse dekomponiert und in eine Ereignishierarchie (Abschnitt 2.5) eingeordnet. Aus dem Ergebnis wird ein Ereignismodell (Abschnitt 3.2.2.1) und ein Kausalmodell erstellt. Anschließend erfolgt die Modellierung der Anwendungsarchitektur.

4.2.2.1. Ereignishierarchie

Luckham [Luc01] stellt eine Methode vor, um mittels Ereignishierarchie einen ersten Wechsel in die technische Denkweise vorzunehmen. Diese Methode ist später Grundlage bei der Erstellung des Ereignismodells.
Die Erstellung der Ereignishierarchie baut sich etappenweise aus den verfügbaren Ereignisquellen, den funktionalen Anforderungen und den Kafka Topics als Repräsentation der Ereignisbehandler auf, bis man in der Mitte einzelne Schritte verbindet. Aus den Ereignisquellen sollen auf der ersten Hierarchieebene ankommende Ereignisse in jeweils Ereignisobjekte der Klassen `StockPrice` `Tweet` transformiert werden. Auf der höchsten Hierarchieebene steht das Ereignis mit der höchsten Komplexität. Eine Instanz der Klasse `ImpactTweetStore` soll an das Kafka Topic *ImpactTweetStoreEvents* publiziert werden. Weitere Klassen lassen sich inkrementell erstellen und sollen sich der Namensgebung der Topics anpassen.

Ebene	Anforderungen	Erläuterung	Ereignistyp
5	ANW-F7	Aggregation der ImpactTweets zum speichern	ImpactTweetStore
4	ANW-F6, ANW-F7	Zeigt, welche Reaktionen ein Tweet in den sozialen Medien hinterlassen hat	ImpactTweetSM
3	ANW-F5, ANW-F7	Tweet hat eine Reaktion am Finanzmarkt erzeugt	ImpactTweetCM
3	ANW-F4, ANW-F6	Aggregation von TweetSentiments innerhalb eines Zeitraums	TweetSentimentAGG
2	ANW-F4	Tweet + Sentiment	TweetSentiment
2	ANW-F3	Drei Zeitungen berichten drei mal in einem Zeitintervall von 20 min. über Tesla	HotTopic
2	ANW-F2 , ANW-F5	Kurschwankungen von 3% innerhalb eines Zeitintervalls von 5 min.	RateFluctuation
2	ANW-F1	die letzten 5 Kursveränderungen waren positiv	StockPriceUp
1	ANW-F3, ANW-F5, ANW-F6	vom In-Adapter	Tweet
1	ANW-F1, ANW-F2	vom In-Adapter	StockPrice

Tabelle 4.2.: Ereignishierarchie der Beispielanwendung

Folgende Designentscheidungen werden getroffen:

Benennung der Ereignistypen Die Benennung der Ereignisse führt nicht zu einer Veränderung der Funktionalität bzw. der beinhaltenden Attribute, dient aber der Beschreibung und der Verständlichkeit. Der entstehende Trade-off zwischen Generalisierung und Spezialisierung kann mittels der Begriffe aus der CEP Bewertung (siehe Abschnitt 3.2.2.4) genauer beschrie-

ben werden: *Wiederverwendbarkeit* und *Agilität* ist einfacher zu erreichen mit allgemeineren Begriffen, wobei genauere Beschreibungen helfen die *Nachvollziehbarkeit* zu erhöhen. In der Beispielanwendung sollen sich komplexe Ereignisse sowohl an den Kafka Topics als auch an den Vorgänger-Ereignissen orientieren, um über den Namen die Beziehung darzustellen. Als einfaches Beispiel dient `Tweet`, `TweetSentiment` und `SentimentAGG`.

Anzahl der Ereignistypen Eine höhere Anzahl von Zwischen-Ereignissen sorgt für eine feinere Granularität, aber mehr Aufwand für den Programmierer. Für diese Beispielanwendung sollen möglichst wenig Klassen genutzt werden, um den Aufwand gering zu halten. Beispielsweise soll für die Filterung nach Tweets, die durch den Twitter-Nutzer Elon Musk versendet wurden, keine explizite neue Klasse erstellt, sondern weiterhin die Klasse `Tweet` genutzt werden, da der Inhalt der Klasse der gleiche wäre.

4.2.2.2. Ereignismodell

Die definierten Ereignisse sollen durch ein Ereignismodell genauer spezifiziert werden. Anders als in Abschnitt 3.2.2.1 beschrieben, wird das Ereignismodell aufgeteilt, um die Komplexität zu verringern. Das Ereignismodell (Abbildung 4.3) soll in diesem Fall nur die Vererbungsbeziehungen und die Attribute der Klassen in Form eines UML-Klassendiagramms (entspricht der Definition *Ereignishierarchie* aus [BD10]) darstellen. Die kausalen Beziehungen werden in das kausale Modell 4.2.2.3 ausgelagert.

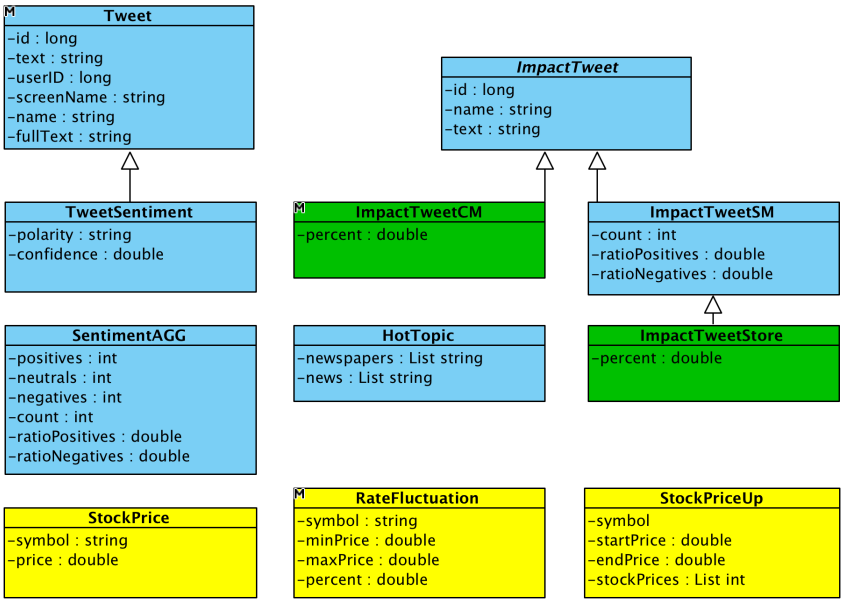

Abbildung 4.3.: Ereignismodell der Beispielanwendung

Folgende Designentscheidungen werden getroffen:

Modellierung der Ereignisse Wie in der Grundlagen beschrieben, können Ereignisse unterschiedlich modelliert werden. In dieser Konzeption sollen sie jeweils als Plain Old Java Object (POJO) konzipiert und später realisiert werden. Folgende Gesichtspunkte sprechen für eine Nutzung von POJOs:

- Apache Flink erkennt POJOs und lässt exklusive Methodenaufrufe zu [vgl. Fli18a].
- POJOs sind auch von Nicht-Programmieren einfach zu verstehen.
- Es lassen sich Java-spezifische Eigenschaften wie Vererbung anwenden, die es ermöglichen Quellcode nicht doppelt zu schreiben.
- Einzelne Sub-Ereignistypen können leichter in einen Ereignisstrom vom Obertyp zusammengefügt werden [vgl. Roh16].

4.2.2.3. Kausales Modell

Mittels des kausalen Modells aus [Luc01] werden die kausalen Zusammenhänge zwischen den Ereignissen sowie das erste Mal die konkreten Ereignisregeln beschrieben. Anhand dieses Modells kann anschließend die Strukturschicht (oder wie in [Sie04] genannt Anwendungsar-

chitektur) modelliert werden. Folgend sollen exemplarisch die kausalen Zusammenhänge der Ereignistypen für die funktionale Anforderung ANW-F-4 in Abbildung 4.4 gezeigt werden.

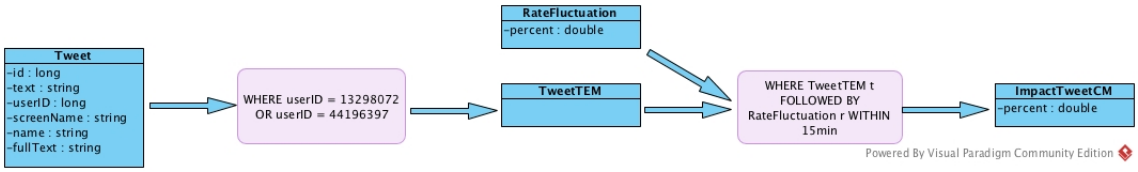

Abbildung 4.4.: Auszug aus dem kausalen Modell für ANW-F-4

Folgende Designentscheidungen werden getroffen:

Modellierungssprache Die gewählte Modellierungssprache ist (anders als in [Luc01]) nicht maßgeblich an der realisierten EPL angelehnt und grafisch gehalten. Mittels dieser Entscheidung soll es den potenziell Ereignisregeln erstellenden Experten möglich sein, die Ablaufschritte nachzuvollziehen. Die genutzte EPL in den Ereignisregeln stellt eine Mischung aus SQL, natürlicher Sprache und ersten Elementen der später genutzten EPL dar. Jenes soll erneut den Wechsel von fachlicher zu technischer Beschreibung einleiten.

4.2.2.4. Anwendungsarchitektur

Aus den vorhergegangen Überlegungen soll die Anwendungsarchitektur (oder nach Bruns und Dunkel Struktursicht) modelliert werden. Die Anwendungsarchitektur ordnet die Bestandteile des Systems aus der Sicht der Anwendung und beschreibt die Schnittstellen und Beziehungen zueinander. Sie ist die zentrale Komponente bei der Realisierung und soll dafür sorgen, dass die Zuständigkeiten klar voneinander getrennt sind. Ziel ist es, die fachlichen Anforderungen in einzelne Komponenten zu zerlegen [vgl. Sie04, S. 151].
In der Wissenschaft (siehe [PV09], [BD10], [Alv+10]) und in der Unternehmenswelt (siehe [Kri11], [SR13]) gibt es ein allgemeines Verständnis darüber, wie eine Referenzarchitektur einer ereignisorientierten Anwendung aussieht. Auf Basis dieser Architekturen und den Entwurfsmustern aus [BD10] soll die Anwendungsarchitektur für die Beispielanwendung (Abbildung E.1) erstellt werden.

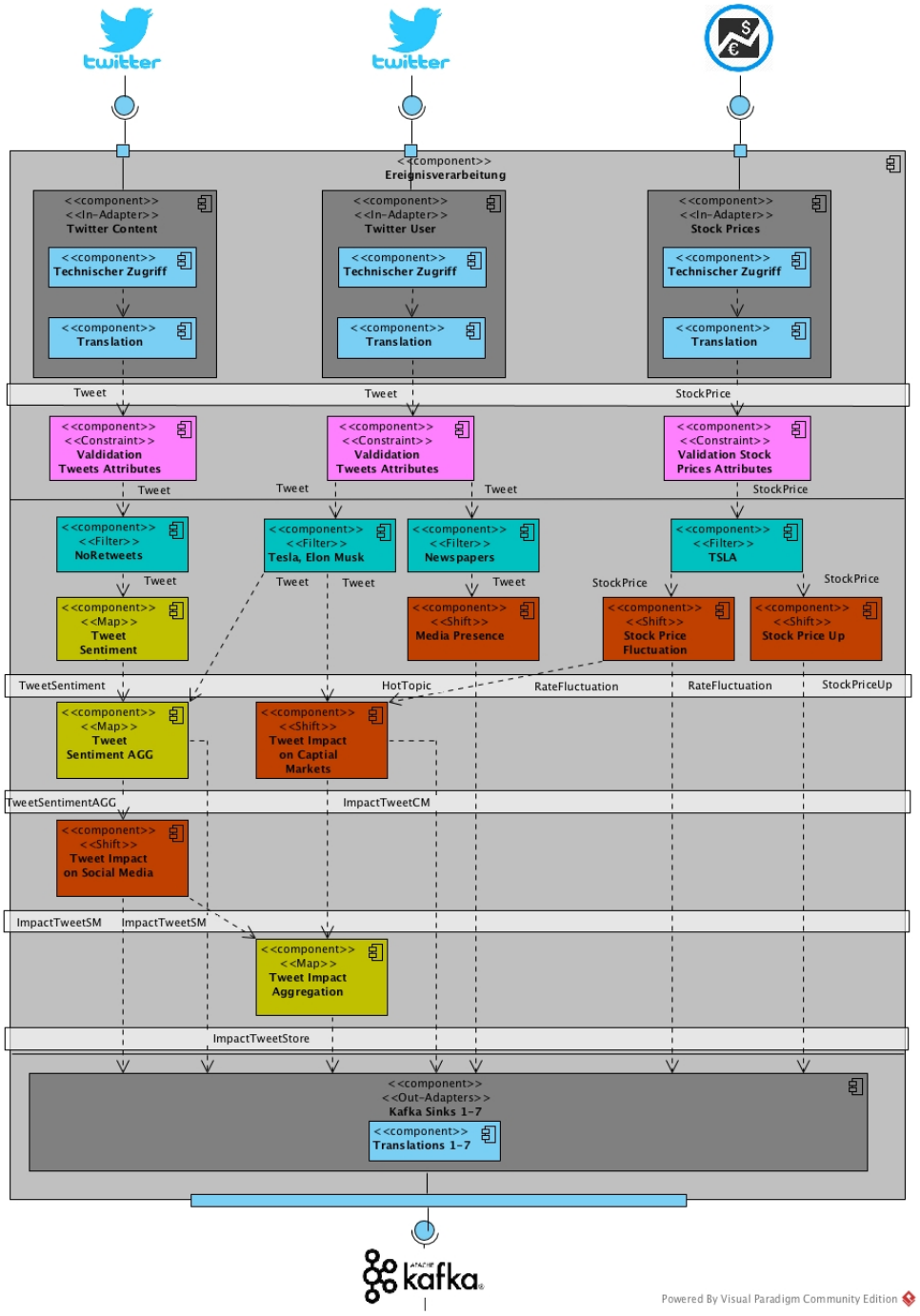

Abbildung 4.5.: Anwendungsarchitektur der Beispielanwendung

Neben der Darstellung der einzelnen Hierarchieebenen in der Anwendungsarchitektur, kann der implizite Kontrollfluss innerhalb der Anwendung beobachtet und damit der Wechsel von atomaren zu komplexen Ereignissen eingesehen werden. Bei der Erstellung der Anwendungsarchitektur wurden folgende Designentscheidungen getroffen:

Schichtenarchitektur Alle referenzierten Referenzarchitekturen beruhen auf dem Prinzip der Schichten. In der Beispielanwendung gibt es folgende implizite und explizite Schichten:

1. *Monitoring:* Die Monitoringschicht hat die Aufgabe korrekte homogene Ereignisse bereitzustellen. Mittels der Komponente *technischer Zugriff* wird der Empfang der Daten realisiert. Anschließend werden die Ereignisse mittels Translationsschritt in Klassen der Hierarchieebene 1 umgewandelt. Im letzten Schritt werden die Attribute der Ereignisse auf Korrektheit überprüft.

2. *Ereignisverarbeitung:* In der Ereignisverarbeitung erfolgt die logische Verarbeitung. In ihr gibt es folgende Schritte, die aus einfachen atomaren Ereignissen inkrementell komplexere Ereignisse entstehen lassen:

   a) *Filter:* Im ersten Schritt werden für die weitere Verarbeitung nur relevante Ereignisse herausgefiltert. Beispielsweise soll die Komponente `Newspapers` dafür sorgen, dass im nachfolgenden Datenstrom nur Tweets vorhanden sind, die von den spezifizierten Zeitungen abgesetzt wurden.

   b) *Map:* Map Komponenten sollen Ereignisobjekte entweder in einen anderen Typ überführen oder ähnliche Ereignisse aggregieren. Die Komponente *Tweet Impact Aggregation* soll Instanzen der Klassen `ImpactTweetSM` und `ImpactTweetCM` in ein Objekt der Klasse `ImpactTweetStore` aggregieren.

   c) *Shift:* Die Shift Komponente ist dafür zuständig, höherwertige komplexe Ereignisse zu erzeugen. Dabei werden unterschiedliche Objekte wie `Tweet` und `RateFluctuation` in Beziehung gesetzt und ein komplexes Ereignis vom Typ `ImpactTweetCM` erzeugt.

   Ereignisvorbehandlung: Die komplexen Ereignisse werden mittels Translation für die Weitergabe an die Ereignisbehandler vorbereitet.

Durch die Wahl einer Schichtenarchitektur soll ein zu *undurchsichtiger Kontrollfluss* verhindert werden. Außerdem wird versucht, die zentralen Zuständigkeit voneinander zu trennen und für eine *lose Kopplung* zu sorgen.

Modularität durch Nutzung leichtgewichtiger EPAs Die Anwendungsarchitektur soll mittels Nutzung der in Abschnitt 3.2.2.1 beschriebenen unterschiedlichen Constraint, Filter und Map EPAs für eine hohe Kohäsion und lose Kopplung sorgen. Die Shift Komponente sorgt nochmals für eine klarere Trennung der Aufgaben. Die *Wartbarkeit* und *Wiederverwendbarkeit* soll durch kleine Komponenten erhöht werden.

Transformation von Ereignissen Die Transformation der eintreffenden Ereignisse in ein durch das Ereignismodell bekanntes Format in der Monitoringschicht soll dafür sorgen, dass in weiteren Schichten keine zusätzlichen Überprüfungen mehr stattfinden müssen bzw. Typfehler auftreten können. In der späteren Realisierung sollen daher die Ereignisse in Ereignisobjekte der Klassen `Tweet` und `StockPrice` transformiert werden

Cleaning Der folgende Cleaning-Schritt mittels Constraint EPA soll die Ereignisse auf Korrektheit überprüfen. Dadurch sollen falsch übertragende Ereignisse gar nicht erst in die Verarbeitungsschicht gelangen und dort gegebenenfalls Werte negativ beeinflussen. Die Komponente *Validation Stock Prices Attributes* soll Ereignisse des Ereignistyps `StockPrice` mit einem `price`< 0 herausfiltern. Die Komponente *Validation Tweet Attributes* soll alle `Tweet` Ereignisse herausfiltern, die nicht die spezifizierten Attribute besitzen.

Reduktion der Ereignismenge Das frühe Filtern der Ereignisströme mittels Filterkomponenten soll dazuführen, dass nachgelagerte Komponenten nur relevante Ereignisse erhalten. Ziel ist es, nachgelagerte EPAs zu entlasten. Daher soll wie in der Anwendungsarchitektur dargestellt, eine sofortige Filterung nach den Constraints Komponenten erfolgen.

4.2.3. Architektur der technische Infrastruktur

Die TI-Architektur beschreibt unter anderem die physischen Geräte, die Systemsoftware und die verwendeten Programmiersprachen [vgl. Sie04, S. 145]. Da es sich bei Apache Flink im Kern um eine *distributed process engine* handelt, soll ein Szenario entworfen werden, in dem eine mögliche Verteilung getestet werden kann. Die Modellierung der TI-Architektur bezieht sich auf die Ausführungen von Siedersleben in [Sie04]. Die spezielle Sicht der Verteilung von ereignisorientierten Anwendungen aus [BD10] wird nicht verwendet, da Apache Flink selbständig die Aufgaben (engl. *tasks*) nach Rechenintensität distribuiert [vgl. Fli18b].
Die Basis der TI-Architektur sollen zwei Computer bilden, die in einem Local Area Network (LAN) miteinander kommunizieren. Jeder Knoten (engl. *node*) soll bestimmte Aufgaben erfüllen.

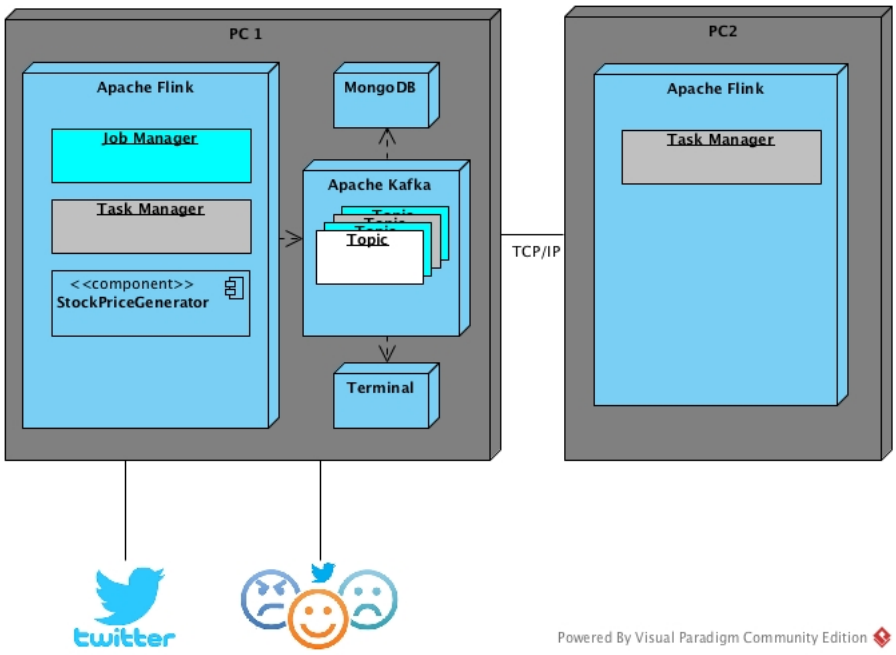

Abbildung 4.6.: TI-Architektur der Beispielanwendung

Apache Flink Das Apache Flink Framework soll auf zwei Knoten verteilt und als Standalone Cluster betrieben werden. Damit eine Anwendung erfolgreich in einem Cluster ausgeführt werden kann, muss im Netzwerk mindestens ein *Job Manager* und mindestens ein *Task Manager* verfügbar sein. Der Job Manager nimmt den Job (die auszuführende Anwendung) entgegen und sorgt für die Organisation. Dabei analysiert er das Programm, sorgt für die entsprechende Verteilung der Tasks und entscheidet darüber, welche Tasks parallel verarbeitet werden sollen. Der Task Manager nimmt die Aufgaben entgegen und bearbeitet sie. In der Beispielanwendung soll es eine Organisationseinheit und eine Verarbeitungseinheit auf PC1 geben. Zusätzlich soll ein weiterer Task Manager auf PC2 bereitgestellt werden, um die Verteilung von Tasks innerhalb Apache Flinks zu untersuchen.

Sonstige Komponenten Die sonstigen Komponenten der EDA wie MongoDB und Apache Kafka werden als Single-Node auf PC1 realisiert. Der Zugriff auf die APIs im Internet kann erst zum Ausführungszeitpunkt definiert werden, da die Verteilung der Tasks erst zu diesem Zeitpunkt feststeht.

5. Realisierung

In diesem Kapitel wird das entstandene Konzept für eine ereignisorientierte Anwendung aus dem Kapitel 4 mittels Apache Flink prototypisch realisiert. Einerseits wird die Tragfähigkeit des Entwurfs verifiziert und andererseits werden Erfahrungen mit Apache Flink und der CEP Library gesammelt, um diese in der anschließenden Evaluation einfließen lassen zu können. Dafür werden ausgewählte und interessante Aspekte der Realisierung präsentiert. Nach der Betrachtung des Realisierungsumfangs gliedert sich das Kapitel nach dem logischen Datenfluss in einem ereignisorientierten System und endet mit der Beschreibung der realisierten Verteilung der Anwendung.

5.1. Realisierungsumfang

Die Grundlage der Realisierung bildet die in Abbildung 4.1 dargestellte Systemarchitektur. Die Implementierung der Ereignisverarbeitungs-Komponente basiert auf der in Abschnitt 4.2.2.4 aufgezeigten Anwendungsarchitektur und ihren vorangegangenen Prozessschritten. Die nur abstrakt beschriebene Architektur der technischen Infrastruktur (Abschnitt 4.2.3) wird durch zwei Personal Computer (PC) mit UNIX-basierten Betriebssystemen konkretisiert.

Auf Seiten der Implementierung der Ereignisverarbeitungs-Komponente wurde aufgrund des begrenzten Zeitrahmens auf die konkrete Implementierung von Fehlerbehandlungen verzichtet. Die Behandlung von Fehlern stellt keinen direkten funktionalen Nutzen für die prototypische Anwendung dar.

Die realisierte Übertragung der Ereignisse aus der Ereignisverarbeitung an die eigentlichen Komponenten der Ereignisbehandlung, wie Terminal-Ausgabe und Speicherung der Daten in der Datenbank durch Apache Kafka ist nur in einfacher Ausführung realisiert. Die Realisierung fokussiert sich auf die Kernkomponente der ereignisorientierten Anwendung, der Ereignisverarbeitung.

5.2. Realisierungsdetails

In diesem Abschnitt werden relevante Realisierungsdetails in den einzelnen Komponenten erörtert, die entweder ein konkretes Problem lösen und oder sonstige nennenswerte Aspekte darlegen.

5.2.1. Ereignisquelle Aktienstrom

Die Implementierung eines eigenen Aktienstroms wird mittels einer selbst erstellten Klasse `SingleStockSource<String>` realisiert, die das von der Flink *DataStream API* bereitgestellte Interface `SourceFunction<T>` implementiert. Über den Konstruktor wird die Aktienkennung (engl. *symbol*) und der Startpreis an die Instanz übergeben. Außerdem wird mittels des im Konstruktor enthalten formalen Parameters `sigma` die Volatilität der Aktie bestimmt (siehe Listing 5.1). Eine Instanz dieser Klasse liefert jede Sekunde einen String, der das Symbol und den Preis beinhaltet. Die Preisfindung erfolgt auf Basis des letzten Preises plus eines Auf-/Abschlags einer zufälligen Zahl, ausgewählt mittels Gauß-Verteilung multipliziert mit der Volatilität (sigma): $price = price + random.nextGaussian() * sigma$.

Um den Aktienstrom realistischer wirken zu lassen, werden mehrere dieser Ereignisquellen für unterschiedliche Unternehmen (Tesla, BMW, Daimler, VW) mit ihren derzeitigen Aktienkursen erzeugt und anschließend zusammengefügt.

5.2.2. Komponenten der Anwendungsarchitektur

Im Folgenden wird auf konkrete Realisierungsdetails eingegangen, die logische Aspekte betrachten und sich in der Anwendungsarchitektur verorten lassen.

5.2.2.1. Monitoring

Die Monitoringschicht sorgt für die Bereitstellung der Datenströme und ist dafür zuständig, die empfangenen Ereignisse in eine homogene Struktur zu transformieren. Die Komponente *technischer Zugriff* wird via einfachem Methodenaufruf von Flink zur Verfügung gestellt, die ein Objekt einer implementierenden Klasse des Interface `SourceFunction` benötigt. Im folgenden Quellcode wird der `StreamExecutionEnvironment env`, der Zugriff auf den Aktienstrom des Unternehmens Tesla ermöglicht und steht der Anwendung anschließend für die weitere Verarbeitung zur Verfügung.

```
1 DataStream<String> TSLA_source = env.addSource(
```

```
2 new SingleStockSource("TSLA", 318, 1));
```

Listing 5.1: Technischer Zugriff auf den Aktienstrom des Unternehmens Tesla

Translation der Twitter-Daten Anders als die Ereignisse im selbst erzeugte Aktienstrom sind die ankommenden Twitter-Ereignisse wesentlich komplexer. Twitter stellt viele Informationen zu einem einzigen Tweet bereit, welcher als String-Interpretation eines JSON Dokuments über den technischen Zugriff in die Ereignisverarbeitung gelangt. Durch diese vorgegebene Struktur muss nicht, wie im Fall der eingehenden Aktienkurse der String aufgetrennt werden und seine einzelne Bestandteile manuell untersucht werden, sondern das Ereignis kann mittels `ObjectMapper` kurzzeitig mit wenig Programmieraufwand in ein JSON Dokument gewandelt, auf seine Eigenschaften mittels deklarativer Methodenaufrufe untersucht und anschließend in ein POJO transformiert werden. Die beschriebenen Funktionen sind in der Klasse `TweetTranslation` zusammengefasst und es wird eine jeweils eigene Instanz für die unterschiedlichen Twitter-Datenströme genutzt.

5.2.2.2. Ereignisverarbeitung

Nach der Homogenisierung der Ereignisse in POJOs und anschließender Überprüfung der Attribute werden die Ereignisobjekte mittels der bereitgestellten Werkzeuge aus der Streaming API und der CEP Library weiterverarbeitet.

Verarbeitung von Datenströmen via FlinkCEP Die Verarbeitung eines Datenstroms via FlinkCEP erfolgt auf drei Ebenen und ist nah am Abschnitt Ereignisregeln 3.2.2.1 aus der Analyse gehalten. Eine Instanz der Klasse `Pattern` beschreibt den Bedingungsteil/das Ereignismuster. Die Anwendung des Patterns auf einem einem Datenstrom erzeugt einen `PatternStream`, der nur Ereignissequenzen beinhaltet, die dem Ereignismuster entsprechen. Der Aktionsteil wird durch die Methode `.select(PatternSelectFunction<IN, OUT>)` realisiert. Allerdings können auch in der implementierenden Klasse weitere Bedingungen gestellt werden. Listing 5.2 zeigt die konkrete Implementierung für das Ereignis HotTopic.

```
1 Pattern<Tweet, ?> mediaPresencePattern =  // Bezeichner des Pattern
2     Pattern.<Tweet>begin("start") //  Anfang des Pattern
3     .times(3) // Folgende Bedinung muss dreimal auftreten
4     .where(new MediaPresenceCondition()) // Bedingung
5     .within(Time.minutes(5)); // innerhalb von 5 min.
6
7 // Anwenden des Pattern auf den Ereignisstrom
```

```
// Erfuellende Sequenzen sind im Patternstream enthalten
PatternStream<Tweet> mediaPresencePatternStream =
    CEP.pattern(NPDataStream, mediaPresencePattern);

// Definierte Aktionen werden ausgefuehrt
DataStream<HotTopic> hotTopicDataStream =
    mediaPresencePatternStream.select(new MediaPresenceAction());
```

Listing 5.2: Implementierung von CEP für das HotTopic Ereignis

Als korrekte Ereignissequenzen aus dem Datenstrom `NPDatastream` (enthält nur Tweets von Nachrichten-Accounts) werden diejenigen definiert, in denen innerhalb von fünf Minuten dreimal die Begriffe Tesla, Musk oder TSLA in dem Tweet vorkommen.

Der genutzte Datenstrom `NPDatastream` wird nicht via CEP Library erzeugt, sondern die Generierung wird durch die Streaming API realisiert. Einfache Filter- und Map-Aufgaben ohne weitere Bedingungen lassen sich mit Hilfe dieser API mit weniger Programmieraufwand vollziehen.

```
DataStream<Tweet> NPDataStream =
    tweetUsersDataStream.filter(new OnlyNPTweets());
```

Listing 5.3: Erzeugen des NPDataStream

Content Enrichement Die Anreicherung eines Tweets mit der jeweiligen Klassifikation in positiver Tweet oder negativer Tweet wird mit dem externen Anbieter Aylien und einer Map-Funktion ohne CEP realisiert. Die Instanz der Klasse `SentimentEnrichment` sendet den Inhalt des Tweets via Methodenaufruf an den API Endpoint des Unternehmens. Als Rückgabewert erhält die Anwendung ein Objekt `sentiment`, welches unter anderem die benötigten Angaben über die Klassifikation und die Sicherheit der Aussage enthält. Der Input-Tweet wird anschließend mit den gewonnen Daten aus der Abfrage angereichert und ein neues Ereignis vom Typ `TweetSentiment` erzeugt sowie anschließend als Output ausgegeben.
Anders als bei anderen Map-Funktionen, wird in diesem Fall eine `RichMapFunction` genutzt, um die Verbindung zur API durch die Erzeugung des Objekts und die anschließende Methode `open` zu initialisieren, sobald die Map-Funktion aufgerufen wird. Ein Methodenaufruf in einer Map-Funktion ist ansonsten nur möglich, wenn die benutzte Klasse das Interface `Serializable` implementiert. Die Listings 5.4 und 5.5 stellen den Ablauf des Content Enrichment dar. Um die Kommunikation mit der API möglichst gering zu halten, wird der Datenstrom `noRTDataStream` genutzt, indem keine Retweets mehr vorhanden sind.

```
DataStream<TweetSentiment> tweetSentimentDataStream =
    noRTDataStream.map(new SentimentEnrichment());
```

Listing 5.4: Außensicht auf den Methodenaufruf der Map-Funktion

```
@Override
public void open(Configuration parameters) throws Exception {
    _textAPIClient = new TextAPIClient("efe1eeb5",
        "6af506e26a54fb3fd2e6cbcd5e0a8895");
}

@Override
public TweetSentiment map(Tweet tweet) throws Exception {
    SentimentParams sentimentParams =
    new SentimentParams(tweet.getText(), null, null);
    Sentiment sentiment = _textAPIClient.sentiment(sentimentParams);

    return new TweetSentiment(tweet, sentiment.getPolarity(),
                    sentiment.getPolarityConfidence());
}
```

Listing 5.5: Innensicht der Klasse SentimentEnrichment

Zusammenführen von unterschiedlichen Datenströmen Für einige funktionale Anforderungen ist das Betrachten von unterschiedlichen Ereignissen der Oberkategorien Tweet und Aktienkurs (bzw. unterschiedlicher Klassen innerhalb dieser Kategorien) gemeinsam erforderlich. In Apache Flink ist es zwar möglich auch Datenströme unterschiedlichen Typs in einen Datenstrom zusammenzufassen, allerdings werden sie bei der Implementierung von Funktionen trotzdem voneinander unterschieden. FlinkCEP hingegen benötigt immer einen Datenstrom mit konsistenten Typ. Um diese Bedingung zu realisieren, wurden zwei unterschiedliche Verfahren genutzt, die jeweils exemplarisch beschrieben werden.

Zusammenfügen mittels Typ-Vererbung Für die Speicherung der Ereignisse `ImpactTweetStore` (ANW-F6) in einer Datenbank müssen die Ereignisse `ImpactTweetCM` und `ImpactTweetSM` aggregiert und dadurch in einem Datenstrom zusammengefasst werden. Durch die gemeinsame Vererbungsbeziehung (siehe Abbildung 4.3 im Kapitel Konzeption) zu der Klasse `ImpactTweet` werden die jeweiligen Datenströme nicht mit dem konkreten Typ, sondern mit dem Obertyp erzeugt. Dadurch können die beiden Datenströme mittels der Methode `.union(DataStream<T>)`

zusammengefügt werden. Eine Unterscheidung der konkreten Typen ist dann mittels `instanceof` möglich. Listing 5.6 veranschaulicht dieses Vorgehen.

```
1  //Erzeugen des impactTweetCMDataStream
2  DataStream<ImpactTweet> impactTweetCMDataStream =
3      impactTweetCMPatternStream.select(new ImpactTweetCMAction());
4
5  //Erzeugen des impactTweetSMDataStream
6  DataStream<ImpactTweet> impactTweetSMDataStream=
7      tweetImpactSMPatternStream.select(new ImpactTweetSMAction());
8
9  //Zusammenfgen der Datenstroeme zu einem Datenstrom
10 DataStream<ImpactTweet> impactTweetDataStream =
11      impactTweetCMDataStream.union(impactTweetSMDataStream);
```

Listing 5.6: Aggregation von Datenströmen mittels Typ-Vererbung

Zusammenfügen durch einen neuen Typ und Type Identifier Um die Kausalität zwischen einem Tweet und der Veränderung des Aktienkurses zu erfassen (ANW-F4), müssen auch diese in einem Datenstrom zusammengefasst werden. Da die Typen gänzlich andere Attribute aufweisen, wird anstatt einen Super-Typ zu definieren die Ereignisobjekte in ein neuen Typ umgewandelt. Zwei zusätzliche Map-Funktionen transformieren Ereignisobjekte der Typen `Tweet` und `RateFluctuation` zu dem Typ `TweetRateFluctuation`, um jeweils einen neuen Datenstrom zu erzeugen. Diese Datenströme des Typs `TweetRateFluctuation` werden anschließend in einem gemeinsamen Datenstrom zusammengefasst und darauf die CEP Pattern angewendet. Um Die Typen trotzdem voneinander unterscheiden zu können, gibt es zwei Konstruktoren, die jeweils das Attribut `originalType` unterschiedlich besetzen. Listing 5.7 veranschaulicht dieses Vorgehen (vgl. Abschnitt 2.4).

```
1  public class TweetRateFluctation {
2          public TweetRateFluctation(Tweet tweet) {
3              originalType = "Tweet"; // Type Identifier
4              ...
5          }
6          public TweetRateFluctation(RateFluctuation rateFluctuation)
             {
7              originalType = "RateFluctuation"; // Type Identifier
8              ...
9          }
10         ...
```

```
}
\\ Wandeln des Typs
DataStream<TweetRateFluctuation> modifiedTEMCMDataStream =
    TEMDataStream
    .map(new TweetToTweetRateFluctuation());

\\ Wandeln des Typs
DataStream<TweetRateFluctuation> modifiedRateFluctuationDataStream =
    rateFluctuationDataStream
    .map(new RateFluctuationToTweetRateFluctuation());

\\ Zusammenfuegen der Datenstroeme
DataStream<TweetRateFluctuation> tweetRateFluctuationDataStream =
    modifiedTEMCMDataStream
    .union(modifiedRateFluctuationDataStream);
```

Listing 5.7: Aggregation von Datenströmen durch Erstellung eines neuen Typs

Das nötige Definieren von zusätzlichen Typen ist in die Konzeption nicht mit eingeflossen. Anhang D zeigt das endgültige Ereignismodell.

5.2.2.3. Ereignisvorbehandlung

In der Ereignisvorbehandlung werden die POJOs mittels einfacher Map-Funktion zu Strings transformiert. Listening 5.8 zeigt die ähnlich einfache Ausgabe eines Ereignisstroms an Kafka, wie Ereignisse von Quellen empfangen werden können.

```
impactTweetStoreOut.addSink(
new FlinkKafkaProducer08<>("192.168.178.23:9092",
    "ImpactTweetStoreEvents", new SimpleStringSchema()));
```

Listing 5.8: Abgabe der Ereignisse an das Kafka Topic ImpactTweetStoreEvents

5.2.3. Ereignisbehandlung

Die zu speichernden komplexen Ereignisse des Typs `ImpactTweetStore` werden in der Datenbank *ImpactTweets* innerhalb der Collection *ImpactTweets* gespeichert. Für die Übertragung der Ereignisse aus dem Kafka Topic *ImpactTweetStoreEvents* in die MongoDB, wird zusätzlich ein Java Programm verwendet, welches auf Grundlage von [Mor16] entstanden ist. Dieses Programm sorgt für die Übersetzung der Ereignisse zwischen Kafka und MongoDB indem es die Bytes aus dem Topic zu einem String umwandelt, den String wieder in ein Objekt des

Typs `ImpactTweetStore` wandelt, um es anschließend als `Document` an die Datenbank zu senden.

5.2.4. Komponenten der TI Architektur und Deployment

Im Folgenden wird konkret auf Realisierungsdetails eingegangen, die sich in der TI-Architektur verorten lassen und zu einem erfolgreichen Deployment des logischen Programm führen. Als *Node 1* wird ein Notebook mit dem Betriebssystem macOs 10.13.5 verwendet. Dieser Node kann als Master betrachtet werden. Als *Node 2* dient eine Virtual Machine (VM) mit dem Betriebssystem Ubuntu 18.04 LTS auf einem weiteren Notebook.

5.2.4.1. Konfiguration Apache Flink

Um Apache Flink 1.5.0 erfolgreich als verteiltes Cluster zu realisieren, müssen neben dem Vorhandensein von Java 1.8.x oder höher und SSH auf jedem Node zusätzliche Konfigurationen gemacht werden.

Interaktion der Knoten Der Job Manager (Master) verteilt die einzelnen Aufgaben (Tasks) an die Task Manager (Slaves). Um eine Interaktion zwischen Job Manager und Task Manager zu realisieren, wird in die Datei *conf/flink-conf.yaml* mittels des Keys `jobmanager.rpc.address` der Hostname bzw.die IP-Adresse angegeben. Dadurch kann eine Kommunikation von Task Manager zu Job Manager via Remote Procedure Call (RPC) erstellt werden. Des Weiteren müssen im Dokument *conf/slaves* die IP-Adressen/Hostnames aller als Nodes angegeben werden, die als Task Manager genutzt werden sollen. Listing 5.9 zeigt die zusätzlich angelegten Konfigurationen. Mit diesem Setup wird beim Starten des Clusters nicht nur der Job Manager auf dem Computer angelegt, sondern ebenfalls ein Task Manager der Tasks ausführen kann.

```
1 // In der flink-conf.yaml Datei
2 jobmanager.rpc.address: 192.168.178.23
3 //In der slaves Datei
4 192.168.178.23
5 192.168.178.38
```

Listing 5.9: Zusätzliche Konfiguration von Apache Flink für die Verteilung

Konfiguration von Zeiten Bei der Ausführung der Anwendung auf beiden Task Managern gleichzeitig kam es zu Abbrüchen durch einen Fehler. Nach einigem Suchen stellte sich heraus, dass bei der Parallelisierung von Subtasks das Deployment auf dem zweiten Node noch nicht

vollzogen war. Dadurch wartete der erste Node vergebens auf eine Antwort und brach nach einiger Zeit ab. Um dieses Problem zu lösen, wird in der Datei *conf/flink-conf.yaml* der Parameter `taskmanager.network.request-backoff.max = 30000` (30 Sekunden) gesetzt, um das Zeitintervall zu verlängern und dem zweiten Node genug Zeit für das Deployment einzuräumen.

5.2.4.2. Apache Kafka

Die erzeugten komplexen Ereignisse in Apache Flink werden zur weiteren Verteilung an Apache Kafka übergeben. Um den Zugriff auf diese Ereignisse auf Node 1 zu realisieren, wird ein Zookeeper Server sowie ein Apache Kafka Server auf Node 1 im einfachen Single Node Cluster gestartet. Anschließend werden die geforderten Topics erstellt. Die Ereignisse werden anschließend mittels einfachen `console-consumer` im Terminal ausgegeben oder durch den bereits beschriebenen Kafka Konsumenten für MongoDB in die Datenbank gespeichert.

6. Evaluation

In diesem Kapitel werden zuerst das in Kapitel 4 konzipierte und anschließend in Kapitel 5 realisierte ereignisgesteuerte System sowie die beinhaltende Beispielanwendung kritisch untersucht. Ziel ist es, festzustellen, ob sich die Beispielanwendung für eine Evaluierung von Apache FlinkCEP eignet. Bei einem positiven Ausgang wird anschließend das Complex Event Processing via Apache Flink evaluiert. Abschließend wird die verwendete Methodik untersucht.

6.1. Event-Driven Architecture und Beispielanwendung

Im Folgenden Abschnitt wird das anhand der konzipierten EDA realisierte ereignisgesteuerte System und die Beispielanwendung evaluiert, um ihre Eignung als Untersuchungsgegenstand zu validieren. Dazu werden technische Aspekte betrachtet, auf mögliche fachliche Szenarien eingegangen und die Gegenstände auf die in Kapitel Analyse definierten Anforderungen überprüft. Abschließend findet eine kritische Betrachtung statt.

6.1.1. Tests

Im ersten Schritt der Evaluation wird das Verfahren zum Testen der Anforderungen beschrieben. Außerdem wird auf die Latenzzeiten zwischen den einzelnen Komponenten eingegangen.

Testen der Anforderungen und der Latenzzeit Tweets in Form von Ereignissen sind ein wesentlicher Bestandteil der Beispielanwendung. Da Ereignisse wie beschrieben jedoch nicht vorgesagt werden können, werden in der Anwendung bestimmte Ereignisse durch einen Twitter-Testnutzer generiert. Dieser Nutzer ist sowohl als Zeitung registriert als auch in dem `TEMDataStream` vertreten und kann bestimmte Ereignisse auslösen. Um Ereignisse auszulösen, die durch den Aktienkurs beeinflusst werden, wurde die Volatilität erhöht. Anhang E.1.1 zeigt eine exemplarische Prüfung der Anforderung ANW-F3 sowie die Beobachtung der Latenzzeiten mit Hilfe des Twitter-Testnutzers, der manuell erfüllende Tweets erzeugen muss. Twitter stellt ankommenden Tweets nur eine Erzeugungsangabe in Sekunden bereit, daher wurden folgende Kennzahlen lose gemessen:

	Zeit in ms
Erzeugung von Tweet auf Twitter bis Ausgabe	< 200
Erzeugung von dritten Tweet auf Twitter bis Ausgabe von `HotTopic`	< 200

Tabelle 6.1.: Latenz von Tweets bis zur Ausgabe

Die Tests zeigten, dass das Erstellen eines Ereignisses in der Ereignisquelle, dem Verarbeiten in der Ereignisverarbeitung und der Ausgabe in in der Ereignisbehandlung in unter einer halben Sekunde innerhalb des ereignisgesteuerten Systems realisiert werden kann.

Sentiment-Analyse Die Sentiment-Analyse und die Ereignisverarbeitung kommunizieren im Request-Response Pattern. Um die Auswirkungen auf die geringe Latenzzeit aus dem vorherigen Test zu überprüfen, wurden mehrere Anfragen an die API gesendet und die Zeit gemessen (siehe Anhang E.1.2). Zehn Messungen ergaben:

	Zeit in ms
Dauer der Kommunikation	$900 - 1300$

Tabelle 6.2.: Dauer der Kommunikation zur Sentiment-Analyse API

Der Test zeigte, dass die Einbindung einer Komponente die nicht im typischen Kommunikationsmuster einer EDA mit anderen Komponenten kommuniziert, das System um ein vielfaches verlangsamen kann (in diesem Fall bis zu über dem sechsfachen). In diesem Fall besteht die Gefahr für ein *bottleneck*.

6.1.2. Szenarien

Im Abschnitt Szenarien werden mögliche fachliche Szenarien konstruiert, um zu überprüfen wie das System auf Änderungen reagiert. Ziel ist es, Eigenschaften zu erkennen und sie in der kritischen Betrachtung wieder aufzugreifen.

Szenario 1 Ein neuer Ereigniskonsument soll in das ereignisgesteuerte System eingepflegt werden, um den Aktienkurs der Konkurrenten zu verfolgen. Anhang E.1.3 zeigt exemplarisch, wie ein weiterer Konsument einen Ereignisstrom bzw. Kafka Topic abonnieren kann. Es zeigte sich, dass eine leichte Einbindung von neuen Ereignisbehandlern möglich ist.

6.1.3. Anforderungsabgleich

Im zweiten Evaluierungsschritt sollen die Untersuchungsgegenstände EDA und Anwendung dahingehend überprüft werden, ob sie den jeweils geforderten Anforderungen standhalten.

6.1.3.1. Event Driven Architecture

In Tabelle 4.1.2 sind die funktionalen Anforderungen aus Abschnitt 3.2.1.3 aufgeführt. In der Spalte *erfüllt* wird betrachtet, ob die EDA die Anforderungen erfüllt.

Anforderung	**erfüllt**
EDA-F1: Ereigniskonsumenten können durch das unilaterale Verarbeitungsmodell einfach in das bestehende System eingefügt werden.	√
EDA-F2: Die Konsumenten sind durch das Kommunikationsmuster lose gekoppelt.	(√)
EDA-F3: Die einzelnen Komponenten lassen sich einfach verteilen.	√

Tabelle 6.3.: Abgleich der Anforderung einer EDA mit dem System

Tabelle 6.3 zeigt, dass alle funktionalen Anforderungen an eine EDA zumindest mit Einschränkungen erfüllt wurden. Die Anforderung EDA-F2 wird nur zum Teil erfüllt, da die Ereignisbehandlung mit der Komponente *Sentiment-Analyse* im Request-Response Pattern und im Pull-Modus kommuniziert (siehe Abschnitt 5.2.2.2 und Test 2 in Abschnitt 6.1.1). Alle anderen Komponenten kommunizieren im in Abschnitt 3.2.1.2 beschrieben Kommunikationsmuster. Ereigniskonsumenten können im laufenden Betrieb des Systems ein Kafka Topic abonnieren und so in das System integriert werden (Abschnitt 6.1.1 Szenario 1). Die Verteilung von Apache Flink, Apache Kafka und MongoDB ist in professioneller Nutzung eher der Regelfall als die Ausnahme.

6.1.3.2. Anforderungen aus der fachlichen Konzeption

Die funktionalen Anforderungen an die Beispielanwendung aus der fachlichen Konzeption aus Abschnitt 4.1.2 sind in Tabelle 6.4 dargestellt.

Anforderung	erfüllt
ANW-F1: Die Anwendung erzeugt eine Mitteilung, wenn die letzten fünf Kursänderungen der Tesla Aktie positiv verlaufen sind.	√
ANW-F2: Die Anwendung erzeugt eine Mitteilung, wenn sich der Aktienkurs der Tesla-Aktie innerhalb von 5 Minuten um mindestens 3% verändert.	√
ANW-F3: Die Anwendung erzeugt eine Mitteilung, wenn relevante Nachrichten-Seiten mindestens 3 mal über das Unternehmen Tesla oder Elon Musk in den letzten 20 Minuten getwittert haben.	√
ANW-F4: Die Anwendung erzeugt eine Mitteilung, die jede Minute eine Statistik zur öffentlichen Meinung über Tesla auf Twitter ausgibt.	√
ANW-F5: Die Anwendung zeigt, ob ein Tweet von Tesla oder Elon Musk zu einer Kursveränderung (ANW-F4) innerhalb von 15 min führt.	√
ANW-F6: Die Anwendung zeigt an, wie positiv/negativ ein Tweet von Tesla oder Elon Musk von den Twitter-Nutzern aufgenommen wurde.	√
ANW-F7: Die Anwendung speichert die aus ANW-F5 und ANW-F6 relevanten Tweets in einer Datenbank.	√

Tabelle 6.4.: Abgleich der fachlichen Anforderung mit der Beispielanwendung

Tabelle 6.4 zeigt, dass alle funktionalen Anforderungen erfüllt wurden. Die Validierung der einzelnen Anforderungen erfolgte mithilfe des implementierten Twitter-Testnutzers und einer Anpassung der Volatilität der Aktienkurse (siehe Abschnitt 6.1.1 Test 1). Die konkrete Realisierung der Anforderungen innerhalb der Beispielanwendung trägt nicht zu einer voll umfänglichen Beurteilung von Apache Flink und der CEP Library bei, zeigt aber, dass ein Prototyp einer verteilten Anwendung, die schnell und automatisiert auf auftretende Ereignisse reagiert innerhalb eines angemessenen Aufwands (ca. 2300 Lines Of Code (LOC) in einer Dauer von 16 Tagen) konzeptioniert und realisiert werden kann.

6.1.4. Kritische Betrachtung

Im Rahmen der prototypischen Realisierung der EDA wurde aus Gründen der Fokussierung auf die Implementierung der Beispielanwendung mittels Apache Flink und CEP auf die physikalische Verteilung der Komponenten Apache Kafka und MongoDB auf weitere Knoten verzichtet. Eine Untersuchung mit positiven Ergebnis findet durch Kleppmann und Kreps [KK15] sowie Dede, Govindaraju und Gunter [Ded+13] statt. Bei einer Weiterentwicklung des Prototyps sollten allerdings auch diese Komponenten auf den zweiten Knoten verteilt werden.
Da es nicht das Ziel der Arbeit ist, selbstständig eine vollständige Text Mining Komponente für die Sentiment-Analyse zu implementieren, sondern eher die Möglichkeit des Content

Enrichment getestet werden sollte, wurde auf einen externen Anbieter zurückgegriffen. Dies hat zur Folge, dass eine kostenfreie Nutzung nur bis zu einer Analyse von bis zu 30.000 Tweets im Monat gegeben ist, was für eine potenzielle Big Data Anwendung gering ist. Die Abfrage nach dem Sentiment dauert ca. eine Sekunde (siehe Abschnitt 6.1.1 Test 2), welches für die Anforderungen im Social-Media-Kontext in einem angemessen Rahmen liegt.

In bestimmten Fällen können komplexe Ereignisse, die aufgrund eines Zeitintervalls erzeugt werden, mehrfach auftreten. Bei einer theoretischen Nutzung in einer Produktiv-Umgebung, sollte mit Stakeholdern besprochen werden, ob dies eine gewisse Dringlichkeit repräsentiert oder aber als störend empfunden wird. Des Weiteren sollte in einer Produktiv-Umgebung eine explizite Fehlerbehandlung stattfinden.

Die beschriebenen Optimierungsmöglichkeiten können dabei helfen, später eine korrekte Anwendung in einer Event-Driven Architecture zu realisieren. Die Verbesserungen schränken den Prototypen jedoch nicht insoweit ein, als dass die erworbenen Erfahrungen während Konzeption und Realisierung nicht genutzt werden können. Diese werden folgend für die Anforderungen aus dem Abschnitt Complex Event Processing verwendet.

6.2. FlinkCEP

Im Folgenden Abschnitt sollen die Erfahrungen aus Konzeption und Realisierung, gepaart mit Ausführungen aus der Analyse zu einer Evaluation von Complex Event Processing via Apache Flink führen.

6.2.1. Tests

In diesem Abschnitt werden Tests in einem technischen Rahmen vorgestellt, die die Implementierung von Ereignisregeln innerhalb von Apache Flink untersuchen.

CEP vs. Streaming API In Abschnitt 5.2.2.2 wurde bereits kurz beschrieben, dass für manche Ereignisregeln sowohl eine Nutzung der Streaming API als auch der CEP Library möglich ist. Mittels eines Vergleichs der Implementierungen der Komponente *Newspapers* wurden die unterschiedlichen Implementierungsmethoden untersucht, die aus dem Ereignisstrom `tweetUsersDataStream` die Tweets der Zeitungen herausfiltern (vgl. Anhang E.2.1).

	CEP	Streaming API
Lines Of Code (LOC)	12(+12)	2(+12)
Streams	2	1
Erzeugte Objekte	5	2
Implementierung in Minuten	4(+6))	1(+6)

Tabelle 6.5.: Exemplarischer Vergleich der Implementierung einer Ereignisregel. In Klammern die zusätzliche Betrachtung der ergänzenden Klasse

Der Test zeigte, dass bei wenig komplexen Regeln ein großer Overhead in allen untersuchten Bereichen bei der Nutzung der CEP Library entsteht. Durch die zusätzlichen bereitgestellten Funktionen der Streaming API, bietet es sich an, einfache Filter-Regeln mittels Streaming API zu realisieren.

Aufwand Um den Implementationsaufwand von Ereignisregeln innerhalb von Apache Flink zu untersuchen, wird im Folgenden exemplarisch die Komponente *Tweet Impact on Social Media* betrachtet. Dafür wurde während der Implementation die beanspruchte Zeit gemessen und anschließend die Anzahl der nötigen Zeilen Code inspiziert (Quellcode zu finden unter Anhang E.2.2).

	LOC	Zeit in min.	CEP	API & Java
Vorbereitung: Klasse, Map-Funktionen erstellen, Datenströme zusammenfassen	89	25	-	√
Ereignisregel: Bedingungsteil, Aktionsteil definieren, auf Datenstrom anwenden	37	25	√	-

Tabelle 6.6.

Der Test zeigte, dass die Vorbereitung der einzelnen Datenströme mindestens genauso viel Aufwand erzeugt, wie die eigentliche Implementierung der Ereignisregel. Vor allem die Erstellung der aggregierenden Klasse `TweetSentimentAGG` benötigte sehr viele Lines Of Code (LOC). Außerdem ist die enge Verbindung der Streaming API und der CEP Library zu erkennen. Um CEP erfolgreich anwenden zu können, müssen mit Hilfe der Werkzeuge der Streaming API die Datenströme vorbereitet werden.

6.2.2. Szenarien

Im diesem Abschnitt sollen zwei vereinfachte fachliche Szenarien konstruiert werden, die eine Anpassung der fachlichen Anforderungen erfordern. So soll untersucht werden, wie gut die Anwendung auf Änderungen reagieren kann.

Szenario 1 Elon Musk gibt sein gesamtes Aktienpaket an Daimler ab und scheidet aus dem Unternehmen aus. Daraus ergibt sich, dass die Anforderungen ANW-F5 und ANW-F6 obsolet werden und eine Anpassung der Anwendung erforderlich ist. In Anhang E.2.3 sind die nötigen Änderungen im Quellcode dargestellt, die von einem Programmierer erledigt werden müssen. Es zeigte sich, dass nur minimale Änderungen am Quellcode / an den Ereignisregeln vorgenommen werden müssen, um sich an ändernde Anforderungen in der Anwendung anzupassen. Dies zahlt auf eine hohe Agilität / Flexibilität der Anwendung und des Unternehmens ein.

Szenario 2 Ein Mitarbeiter aus dem Bereich Finanzen möchte, dass sich der Treshold aus ANW-F2 von 3% auf ein 1% ändert und er oder ein Programmierer passen die Regel an. Die Auswirkungen in der Anwendung sind in Anhang E.2.4 dargestellt. Es zeigte sich, dass diese Regeländerung sich auch auf andere Datenströme, Ereignisregeln und letztendlich Anforderungen auswirkt. So wirkt sich beispielsweise die Änderung auch auf die (vielleicht vorhandene) Social Media Abteilung aus, ohne dass sie von der Änderung erfahren hat (ANW-F5). Die Intransparenz und Komplexität größer werdender Systeme steigt schnell.

6.2.3. Anforderungsabgleich

Im ersten Evaluierungsschritt sollen der Untersuchungsgegenstand dahingehend überprüft werden, ob sie den jeweils gegebenen Anforderungen standhalten.

6.2.3.1. Complex Event Processing

In Tabelle 6.7 sind die funktionalen Anforderungen an CEP aus Abschnitt 3.2.2.3 aufgeführt. In der Spalte *erfüllt* wird betrachtet, ob FlinkCEP die Anforderungen erfüllt.

Anforderung	**erfüllt**
CEP-F1: CEP ist in der Lage unterschiedliche Ereignisse in Ereignisströmen in nahezu Echtzeit auszuwerten.	$(\surd)$
CEP-F2: CEP besitzt eine ausreichende Ereignisalgebra und Window Funktionen, um Ereignisse in Ereignisströmen ausreichend zu verarbeiten.	$\surd$
CEP-F3: EPAs sind lose miteinander gekoppelt und können physikalisch verteilt werden.	$\surd$
CEP-F4: CEP verfügt über genügend und einfache Adapter, um Ereignisströme aufnehmen und abgeben zu können.	$\surd$
CEP-F5: CEP besitzt eine einfache, explizite Event Processing Language, um Ereignisregeln zu definieren.	-

Tabelle 6.7.: Abgleich der Anforderungen an CEP mit FlinkCEP

Die Möglichkeit des Zusammenfassens von unterschiedlichen Ereignistypen in einen Ereignisstrom wurde in Abschnitt 5.2.2.2 behandelt und obwohl durch die Nutzung von Java theoretisch kein expliziter Type Identifier als Attribut im POJO auftreten sollte, wird er in FlinkCEP für manche Situationen benötigt (vgl. Abschnitt 2.4 und Abschnitt 6.2.1 Test 2). Unterschiedliche Ereignistypen können nur mittels Typ-Vererbung oder aber durch das Schaffen von speziellen Typen, die explizit einen Type Identifier angeben, für den Verarbeitungsprozess in einem Ereignisstrom genutzt werden. Der zweite Teil der Anforderung, Ereignisse in Echtzeit auszuwerten wurde durch Tests mit dem Twitter-Testnutzer validiert. Die Ereignisströme können aus den beschriebenen Quellen und Senken abonniert bzw. publiziert werden (siehe Abschnitt 6.1.1).
Das Konzept der EPAs wird nur implizit genutzt. Bei der Verteilung des Systems auf unterschiedliche Knoten dekomponiert der Job Manager Aufgaben, welche die Struktur eines schmal gewichtigen Event Processing Agent haben, aus dem Job und verteilt sie automatisiert auf die Knoten.
Da die Implementierung der Ereignisregeln in Java oder Scala realisiert wird, steht eine Vielzahl der in Tabelle B.1 geforderten Operatoren implizit bereit. Des Weiteren baut die FlinkCEP Library auf der Streaming API auf. Daher können die bereitgestellten (deklarativen) Methoden bereits vor dem eigentlichen CEP-Schritt genutzt werden (siehe Abschnitt 6.2.1 Test 1 und Test 2). Dies wertet die ansonsten schmale Library auf, sodass sie Anforderung CEP-F2 erfüllt. Jenes führt aber auch oft zu einer Vermischung von Streaming API und CEP Library, welches ein Lesen der Ereignisregeln erschwert (siehe Abschnitt 6.2.1 Test 2). Außerdem sorgt das CEP Muster in Flink für einen gewissen Overhead, der vor allem bei einfachen Regeln störend sein kann (vgl. Abschnitt 5.2.2.2 und Abschnitt 6.2.1 Test 1).

6.2.3.2. Event Processing Language

In Tabelle 6.8 sind die funktionalen Anforderungen an eine EPL aus Abschnitt 3.2.2.3 aufgeführt. In der Spalte *erfüllt* wird betrachtet, ob die EPL die Anforderungen erfüllt.

Anforderung	**erfüllt**
EPL-F1: Die ereignisorientierte Anwendung besitzt eine EPL, die einfach genug zu verstehen ist, sodass auch Nicht-Programmierer Regeln in das System einpflegen lassen können.	-
EPL-F2: Die ereignisorientierte Anwendung besitzt eine EPL, die nicht proprietär ist.	-

Tabelle 6.8.: Abgleich der Anforderungen an eine EPL mit FlinkCEP

In FlinkCEP gibt es keine explizit beschreibende EPL, wie in Abschnitt 4.2.1.3 erläutert. Die Beschreibung der Ereignisregeln erfolgt durch Klassen und ihre Funktionen, die von der CEP Library bereitgestellt werden. Die Beschreibung des Bedingungsteils und der darauffolgenden Aktion findet daher direkt im Quellcode in der Anwendung statt (siehe u. a. Anhang E.2.4). Dies hat zur Folge, dass Ereignisse sehr programmier-nah verfasst werden müssen und es für Nicht-Programmierer schwer ist, Ereignismuster nachzuvollziehen bzw. sie die Regeln selber in der Anwendung schreiben müssten. Verglichenen mit den beschriebenen EPLs aus Anhang C (vgl. Abschnitt 6.2.1 Test 1), ist die Definition von Regeln wesentlich schwieriger für Nicht-Programmierer.

6.2.4. Kritische Betrachtung

Die Implementierung von Ereignisregeln nah am bzw. im Quellcode sorgt für einige Vor- und Nachteile. Die Nutzung von erweiterten Konzepten aus der Programmierung macht es möglich, die Ereignisregeln mächtiger zu machen. Auch der Aufbau der CEP Library auf der Streaming API sorgt dafür, dass beide APIs gleichzeitig genutzt werden können und so mehr Funktionen bereitstehen (siehe Abschnitt 6.2.1 Test 1 und Test 2). Geübten Programmierern sollte die Einarbeitung leicht fallen, da auf bestehende Konzepte zurückgegriffen wird. Ein weiterer Vorteil ist, dass schon durch kleine Anpassungen des Programms innerhalb der Regeln die Funktionalität schnell geändert werden kann. Eine Änderung der Anforderungen kann leicht in die Anwendung übertragen werden, was im Kontext zu einer höheren Agilität führt (siehe Abschnitt 6.2.1 Test 1).

Die Komplexität sorgt auch für Nachteile. Die Erstellung der Ereignisregeln nimmt durch den zuvor beschriebenen Overhead mittels Generierung der Pattern, PatternStream, DataStream (sie Abschnitt 6.2.1 Test 1) und die zusätzliche Implementierung der Klassen, auf die innerhalb der Verarbeitung zugegriffen wird, Zeit in Anspruch (siehe Abschnitt 6.2.1 Test 2) . Eine vereinfachte Kommunikation zwischen fachlichen Experten und Programmierern durch Ereignisregeln ist möglich, jedoch eine explizite Implementierung von Nicht-Programmierern durch das Nichtvorhandensein einer expliziten EPL eher unwahrscheinlich. Des Weiteren kann die Nähe der Regeldefinition zum Quellcode zu Verständigungsproblemen zwischen Programmieren führen. Gibt es keine festgelegten Regeln, wie Ereignisregeln implementiert und Ereignisströme benannt werden sollen, kann es zu großen Intransparenzen kommen, da einzelnen Programmierern die Inhalte unterschiedlicher Ereignisströme gänzlich unbekannt sein können (siehe Abschnitt 6.2.1 Szenario 2).

Die Verteilung von EPAs auf das Netz und das Hinzufügen neuer Task Manager gelingt. Da es sich um ein Open-Source Produkt handelt, ist der Support eingeschränkt und Erkenntnisse müssen größtenteils von der offiziellen Website gewonnen werden. Allerdings gibt es aktiv genutzte Mailinglisten, die bei Problemen helfen können (u. a. Abschnitt 5.2.4.1 Absatz Konfiguration von Zeiten).

6.3. Methodische Abstraktion

Die sich durch die Arbeit ziehende Trennung zwischen betriebswirtschaftlichen Aspekten, Event-Driven Architecture und Complex Event Processing sorgte für einige Vor- und Nachteile. Die fachliche Untersuchung von ereignisorientierten Anwendungen im Unternehmenskontext führte dazu, dass früh der Mehrwert einer solchen Anwendung definiert werden konnte und dass sie keinem Selbstzweck dient. Die Analyse bzw. Umsetzung des Konzepts Event-Driven Architecture spannte einen größeren Rahmen um den Untersuchungsgegenstand CEP und zeigte den größeren technischen Zusammenhang. Dies hat jedoch zur Folge, dass sowohl die theoretischen als auch praktischen Untersuchungen einen größeren Kontext abbilden, anstatt einzelne Aspekte feingranular herauszuarbeiten.

7. Zusammenfassung und Ausblick

In diesem Kapitel wird der Inhalt dieser Arbeit zusammengefasst und die wesentlichen Ergebnisse aufgeführt. Abschließend wird ein Ausblick darauf gegeben, welche weiterführenden Forschungsansätze aus den gewonnen Erkenntnissen abgeleitet werden können.

7.1. Zusammenfassung

In dieser Arbeit wurde eine geeignete prototypische ereignisorientierte Anwendung mittels Apache Flink und Complex Event Processing realisiert. Dabei wurden in Kapitel 2 zuerst grundlegende Begriffe definiert. In Kapitel 3 wurden mittels der Analyse geeignete Anwendungsgebiete innerhalb eines Unternehmens definiert und der Nutzen einer solchen Anwendung festgestellt. Anschließend wurden die für die Konzeption benötigten Konzepte Event-Driven Architecture (EDA)und Complex Event Processing (CEP) untersucht, um aus den Eigenschaften und Charakteristika erforderliche Anforderungen abzuleiten. Für alle drei Themenpunkte wurde eine Bewertung erstellt. Innerhalb des Kapitels 4 wurde auf Grundlage der fachlichen Analyse ein geeignetes Anwendungsbeispiel geschaffen und durch die identifizierten Anforderungen an eine EDA und CEP ein Konzept für einen Prototypen erstellt. Kapitel 5 befasste sich mit nennenswerten Aspekten der Realisierung sowie Besonderheiten bei der Nutzung von Apache Flink und der CEP Library. In Kapitel 6 wurde die prototypische Anwendung und FlinkCEP durch Anforderungsabgleich und kritischer Betrachtung evaluiert.

Es zeigte sich, dass Unternehmen durch eine Nutzung von EDA und CEP durch einen höheren Automatisierungsgrad schneller auf eintretendes Ereignisse innerhalb und außerhalb des Unternehmens reagieren können. Vor allem die schnelle Verarbeitung von Informationen und die leichte Anpassung der Ereignisregeln sorgt für einen Wettbewerbsvorteil in puncto Agilität und Informationsverfügbarkeit. Andererseits kann die Intransparenz durch viele Datenströme und die lose Kopplung einzelner Komponenten in einer EDA und CEP Anwendungen zu Problemen führen.

Durch den direkten Aufbau der CEP Library auf der Streaming API besitzt Apache Flink keine explizit eigene Sprache für die Definition von Ereignisregeln. Durch direkte Implementierung der Ereignisregeln im Programmcode bietet Apache Flink vielfältige Möglichkeiten, Datenströme vor und zwischen den einzelnen Ereignisregeln mittels der Streaming API zu verarbeiten. Allerdings sorgt diese Nähe auch dafür, dass die ohnehin schon programmiernahe Beschreibungssprache der Ereignisregeln durch Kombination mit anderen Konzepten noch schwerer für Nicht-Programmierer zu verstehen ist. Die Umsetzung der Beispielanwendung gelang in einem angemessen Zeitrahmen und konnte einfach auf zwei Knoten verteilt werden.

7.2. Ausblick

Bei der Untersuchung von Apache Flink und der CEP Library lag der Fokus vor allem auf der Entwicklung eines Prototyps um die Eigenschaften von FlinkCEP zu evaluieren. Das Deployment der Anwendung auf einem Cluster wurde nur grundlegend realisiert. Das professionelle Deployment auf mehreren Servern sowie Performance-Tests mit ähnlichen Anwendungen stellen daher weitere spannende Forschungsfelder dar. Weitere quantitative Untersuchungen werden empfohlen, um zu evaluieren, inwiefern ereignisorientierte Anwendungen wichtige Kennzahlen aus der Betriebswirtschaft und Public Relations beeinflussen.

A. Inhalt der CD-ROM

Dieser Arbeit liegt eine CD-ROM mit folgenden Dateien bei:

- **bachelor_thesis.pdf**: Diese Arbeit im PDF-Format.
- **bachelor_thesis.zip**: Der Quellcode der prototypischen Beispielanwendung.
- **Informationen_zur_Ausfuehrung.pdf**: Informationen bezüglich der Ausführung der Beispielanwendung.
- **lib.tar.gz**: Gegebenfalls benötigte JAR-Files, wenn eine Anwendung außerhalb der IDE erfolgen soll.

B. Ereignisregeln

In diesem Kapitel des Anhangs werden beispielhaft grundlegende Operatoren für die Definition von Ereignismustern und mögliche Aktionen aufgezeigt.

B.1. Ereignisalgebra

Grundlegende Operatoren für eine Ereignisalgebra.

Kategorie	Operatoren(en)	Beschreibung
Operatoren zwischen Ereignistypen	Sequenzoperator($\rightarrow$)	Legt die Reihenfolge, in der Ereignistypen auftreten müssen, fest
	Simultanoperator($\leftrightarrow$)	Zwei Ereignistypen müssen zum gleichen Zeitpunkt auftreten
	Boolesche Operatoren(||,&&)	Stellen UND und ODER Beziehungen dar
	Negationsoperator(!)	Ein Ereignistyp darf in der Ereignisfolge nicht auftreten
Kontext-Bedingungen	Aliasnamen(=)	Geben dem Ereignisobjekt einen Namen
	Attributszugriff(.)	Zugriff auf ein Attribut des Ereignisobjekts
	Methodenaufrufe aus der Anwendungsdomäne	Wissen aus der Anwendungsdomäne kann angewendet werden
	Operationen und Vergleichsoperatoren	Beispielsweise: $+, -, *, <, >$, usw.
Sliding Windows	Längenfenster (W.length(,f, l))	Beschränkt die Anzahl der Elemente in der betrachteten Ereignisfolge
	Zeitfenster(W.time(f,t)	Betrachtet nur Ereignisobjekte, die in der jeweiligen Zeit aufgetreten sind
Aggregations-funktionen	Summe, Durchschnitt, etc. (sum(), ...)	Aggregiert die Ereignisobjekte

Tabelle B.1.: Mögliche Ereignisalgebra mit wichtigen Operatoren

B.2. Aktionen

Grundlegende Aktionen für eine Ereignisregel.

Art	Transformation	Beschreibung
Ereignis erzeugen	Translation	Überführt Daten eines Ereignisobjektes in ein anderes Ereignisobjekt [vgl. BD15, S. 28]
	Filterung	Nur Daten, die ein Kriterium erfüllen, werden in ein neues Ereignisobjekt transformiert
	Aggregation	Fasst mehrere Ereignisobjekte in ein neues Ereignisobjekt zusammen
	Anreicherung	Ereignisse werden mit Kontextwissen angereichert
	Synthese	*Komplexe Ereignisse* werden durch die in einer fachlichen *Beziehung* stehenden einfachen Ereignisse abgeleitet.
Ereignis passieren		Ein bestehendes Ereignis erfüllt eine Bedingung, um weiter versendet zu werden
EPA ändern		Der interne Zustand des EPAs wird geändert
Dienst anstoßen		Dienste von Anwendungen der Ereignisbehandlung werden angestoßen (operative Unternehmensanwendungen)

Tabelle B.2.: Mögliche Aktionen einer Ereignisregeln

C. Exemplarische EPL

Event Processing Languages werden von der Processing Engine definiert und sind eine Konkretisierung der Ereignisalgebra. Um die EPL von Apache Flink besser untersuchen zu können, werden folgend zwei EPLs betrachtet. Die Inhalte stammen wesentlich aus [Hed17] und [BD10] und wurden aggregiert sowie komprimiert.

C.1. Continuous Query Language

Für die Belange des Complex Event Processing wurde die Datenbankabfragesprache SQL an der Stanford University erweitert. Diese Ausformung, die *dynamische Abfragen* auf kontinuierlichen Ereignisströmen erlaubt, wird als *Continuous Query Language* (CQL) bezeichnet und Grundlage ist ebenfalls die Abbildung von Relationen auf Tabellen [vgl. Hed17, S. 64]. CQL beinhaltet eine Vielzahl von Operationen, die von der Algebra definiert werden (inkl. Sliding Windows) [vgl. BD15, S. 23]. Die strukturelle Nähe zu SQL ermöglicht eine einfache Integration in die IT-Landschaft, wenn Ereignisse in Datenbanken gespeichert werden oder aber Wissen in die Ereignisse einfließen soll [vgl. Hed17, S. 65]. Außerdem ist CQL in der Regel relativ einfach zu erlernen, da SQL von vielen Softwareentwicklern verstanden wird. Eine Vermischung von Konzepten und Paradigmen kann die Abfragesprache aber etwas schwerer zu lesen machen [vgl. BD10, 146f].

```
1 SELECT  Symbol, AVG(Price)
2 FROM            StockTickEvent [ROWS 10 SLIDE 5]
3 WHERE           Symbol =    CSCO
```

Listing C.1: Beispielhafte CQL Query

C.1.1. Eigenschaften und Charakteristika

Aus der Beschreibung ergeben sich folgende Eigenschaften und Charakteristika:

- Einfaches Erlernen durch Nähe zu SQL
- Hohe Komplexität durch viele Operationen

- Einfache Integration in bestehende IT-Landschaft

C.1.2. Bewertung

Aus den Eigenschaften und Charakterstark lassen sich folgende Vor- und Nachteile ableiten:

Vorteile

- Einfaches Erlernen durch die Bekanntheit von SQL
- Durch die hohe Komplexität können viele Regeln realisiert werden
- Durch die Nähe zu SQL ist eine Eingliederung in die IT-Landschaft einfach

Nachteile

- Durch die hohe Komplexität wurden Konzepte und Paradigmen vertauscht.

C.2. Esper EPL

Esper EPL ist eine weit verbreitete Sprache, da Esper ein weit verbreitetes Open Source Framework darstellt [vgl. Hed17, S. 66] und wurde von der EsperTech Inc. entwickelt. Esper ist auch an SQL angelehnt, aber anders als bei CQL werden Ereignisobjekte in Schlüssel-Wert-Paaren, in XML oder als POJO (Plain Old Java Object) dargestellt [vgl. BD10, S. 127]. Esper ist ebenfalls komplex und kann nahezu alle Sprachkonzepte einer Ereignisverarbeitungssprache abbilden [vgl. BD10, S. 138].

```
1 select  symbol, avg(price)
2 from            StockTickEvent.win:time(60 sec)
3 where   symbol =    CSCO
```

Listing C.2: Beispielhafte Esper EPL Query

C.2.1. Eigenschaften und Charakteristika

Aus der Beschreibung ergeben sich folgende Eigenschaften und Charakteristika:

- Einfaches Erlernen durch Nähe zu SQL
- Hohe Komplexität durch viele Operationen
- Weite Verbreitung
- proprietär

C.2.2. Bewertung

Aus den Eigenschaften und Charakterstark lassen sich folgende Vor- und Nachteile ableiten:

Vorteile

- Einfaches Erlernen der ersten Ereignismuster [vgl. BD10, S. 138]
- Durch die hohe Komplexität können viele Regeln realisiert werden

Nachteile

- Durch die hohe Komplexität fällt es am Anfang schwer komplexe Ereignismuster zu beschreiben [vgl BD10, S. 138]
- Sprache ist proprietär. Sprache kann in keinem anderen System verwendet werden

C.3. Bewertung

Event Processing Languages ermöglichen eine Trennung von Programmierlogik und Ereignisregeln. Dadurch wird das Bilden von Ereignisregeln vereinfacht und auch Nicht-Programmierer können diese bilden. Allerdings gibt es viele unterschiedliche Sprachen für die unterschiedlichen CEP Produkte, sodass eine Übertragung der Regeln selten möglich ist.

D. Ereignismodell nach der Realisierung

Folgendes Ereignismodell ergab sich nach der Realisierung.

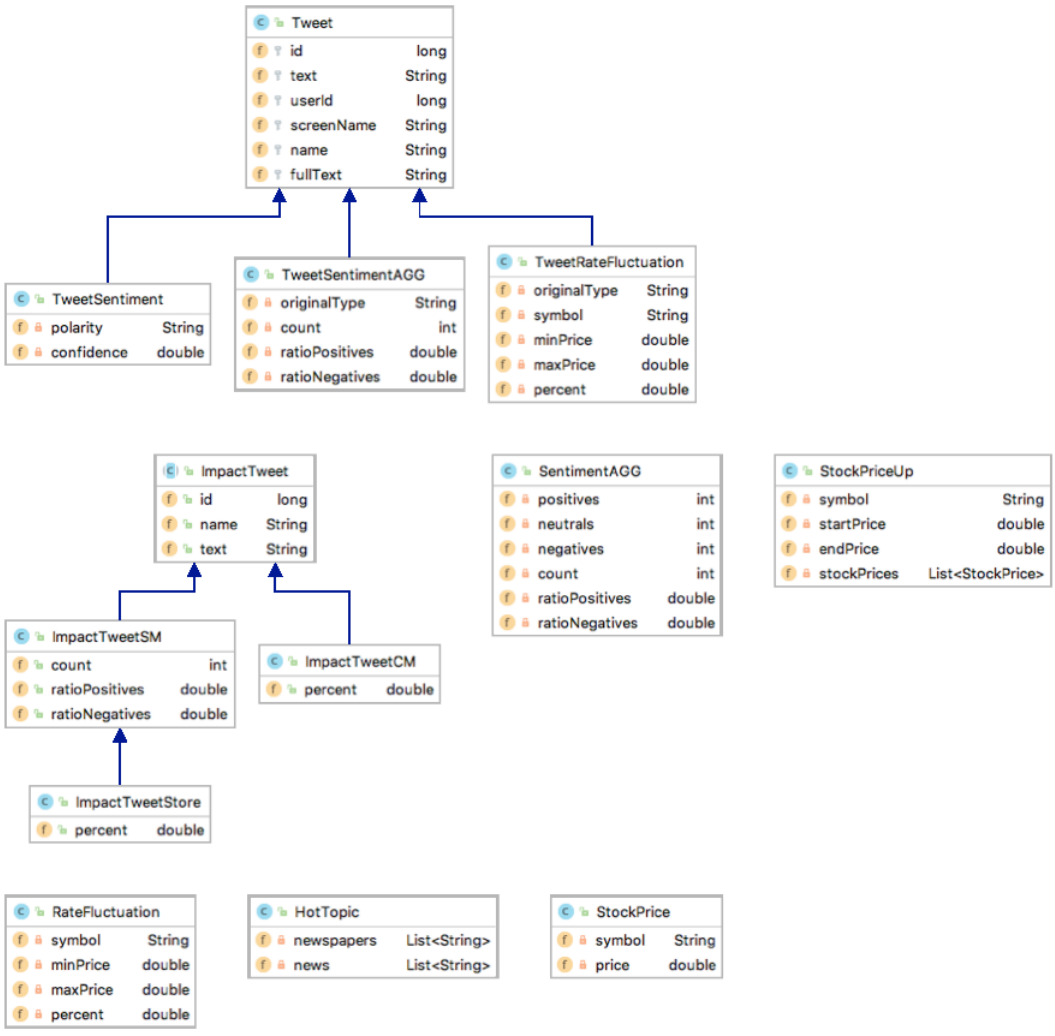

Abbildung D.1.: Das realisierte Ereignismodell

E. Evaluation

E.1. EDA und Beispielanwendung

E.1.1. Test 1

Exemplarisches Testen von Anforderungen durch den Test-Twitternutzer und der Konsole.

```
twitterUsersSource.print();
TEMOutStream.print();
hotTopicOut.print();

TEMOutStream.addSink(new FlinkKafkaProducer08<>(
    "192.168.178.23:9092",
    "TEMTweets", new SimpleStringSchema()));
 hotTopicOut.addSink(new FlinkKafkaProducer08<>(
    "192.168.178.23:9092",
    "HotTopicEvents", new SimpleStringSchema()));
```

Listing E.1: Exemplarisches Prüfen von Latenz und Anforderung

E.1.2. Test 2

Testen der Kommunikationsdauer zwischen einem Klienten und der Sentiment-Analyse API.

```
TextAPIClient client = new TextAPIClient(
    "efe1eeb5", "6af506e26a54fb3fd2e6cbcd5e0a8895");
String text = "John is a very good football player!";
long t = System.currentTimeMillis();
SentimentParams sentimentParams =
    new SentimentParams(text,null,null);
Sentiment sentiment = client.sentiment(sentimentParams);
System.out.println(System.currentTimeMillis() - t);
```

Listing E.2: Prüfen der Kommunikationsdauer zwischen Klient und API

E.1.3. Szenario 1

```
bin/kafka-console-consumer.sh --bootstrap-server 192.168.178.23:9092
--topic StockPrices
```

Listing E.3: Hinzufügen eines Konsumten an ein Kafka Topic

E.2. FlinkCEP

E.2.1. Test 1

Quellcode für die Untersuchung der einzelnen Möglichkeiten in Apache Flink, um eine einfache Ereignisregel zu implementieren.

```
DataStream<Tweet> NPDataStream = tweetUsersDataStream.filter(
    new OnlyNPTweets());
```

Listing E.4: Implementierung mittels Streaming API

```
Pattern<Tweet, ?> NPPattern = Pattern.<Tweet>begin("start")
    .where(new OnlyNPTweets());

PatternStream<Tweet> NPPatternStream = CEP.pattern(
    tweetUsersDataStream, NPPattern);

DataStream<Tweet> NPDataStream2 = NPPatternStream.select(
    new PatternSelectFunction<Tweet, Tweet>() {
        @Override
         public Tweet select(Map<String, List<Tweet>> pattern)
             throws Exception {
             return pattern.get("start").get(0);
         }
});
```

Listing E.5: Implementierung mittels CEP Library

```
private static final long CNN = 759251L;
private static final long BLOOMBERG = 34713362L;
private static final long THE_ECONOMIST = 5988062L;
private static final long THE_TELEGRAPH = 16343974L;
private static final long HANDELSBLATT = 2979574468L;
private static final long TEST_ID = 10065010861009428 48L;

```

```
@Override
public boolean filter(Tweet tweet) throws Exception {
    return tweet.getUserId() == CNN ||
        tweet.getUserId() == BLOOMBERG ||
        tweet.getUserId() == THE_ECONOMIST ||
        tweet.getUserId() == THE_TELEGRAPH ||
        tweet.getUserId() == HANDELSBLATT ||
        tweet.getUserId() == TEST_ID;
}
```

Listing E.6: Inhalt der jeweiligen OnlyNPTweets Klassen

E.2.2. Test 2

```
DataStream<TweetSentimentAGG>  modifiedTEMSMDataStream=
    TEMDataStream.map(new TweetToTweetSentimentAGG());

DataStream<TweetSentimentAGG> modifiedSentimentAGGDataStream =
    sentimentAGGDataStream.map(new
      SentimentAGGToTweetSentimentAGG())

DataStream<TweetSentimentAGG> tweetSentimentAGGDataStream =
    modifiedTEMSMDataStream.union(modifiedSentimentAGGDataStream);
```

Listing E.7: Transformieren und Zusammenfügen der Datenströme

```
Pattern<TweetSentimentAGG, ?> tweetImpactSMPattern=
    Pattern.<TweetSentimentAGG>begin("Tweet")
    .where(new ImpactTweetSMCondition1())
    .followedBy("SentimentAGG")
    .where(new ImpactTweetSMCondition2())
    .within(Time.minutes(15));

PatternStream<TweetSentimentAGG> tweetImpactSMPatternStream =
    CEP.pattern(tweetSentimentAGGDataStream, tweetImpactSMPattern);

DataStream<ImpactTweet> impactTweetSMDataStream =
tweetImpactSMPatternStream.select(new ImpactTweetSMAction());
```

Listing E.8: Definieren und Anwenden der Ereignisregel

E.2.3. Szenario 1

Änderungen, die am Quellcode gemacht werden müssen, um auf das Szenario 1 eingehen zu können.

```
public class OnlyTEMTweets implements FilterFunction<Tweet> {
    private static final long TESLA = 13298072L;
    private static final long ELON_MUSK = 44196397L;

    @Override
    public boolean filter(Tweet tweet) {
        return tweet.getUserId() == TESLA ||
                tweet.getUserId() == ELON_MUSK ||
    }
```

Listing E.9: Vorheriger Code

```
public class OnlyTEMTweets implements FilterFunction<Tweet> {
    private static final long TESLA = 13298072L;
    private static final long DAIMLER = 12637732L;

    @Override
    public boolean filter(Tweet tweet) {
        return tweet.getUserId() == TESLA ||
                tweet.getUserId() == DAIMLER || // Aenderung
    }
```

Listing E.10: Code nach der Änderung

E.2.4. Szenario 2

Auswirkungen der Regeländerung in Szenario 2.

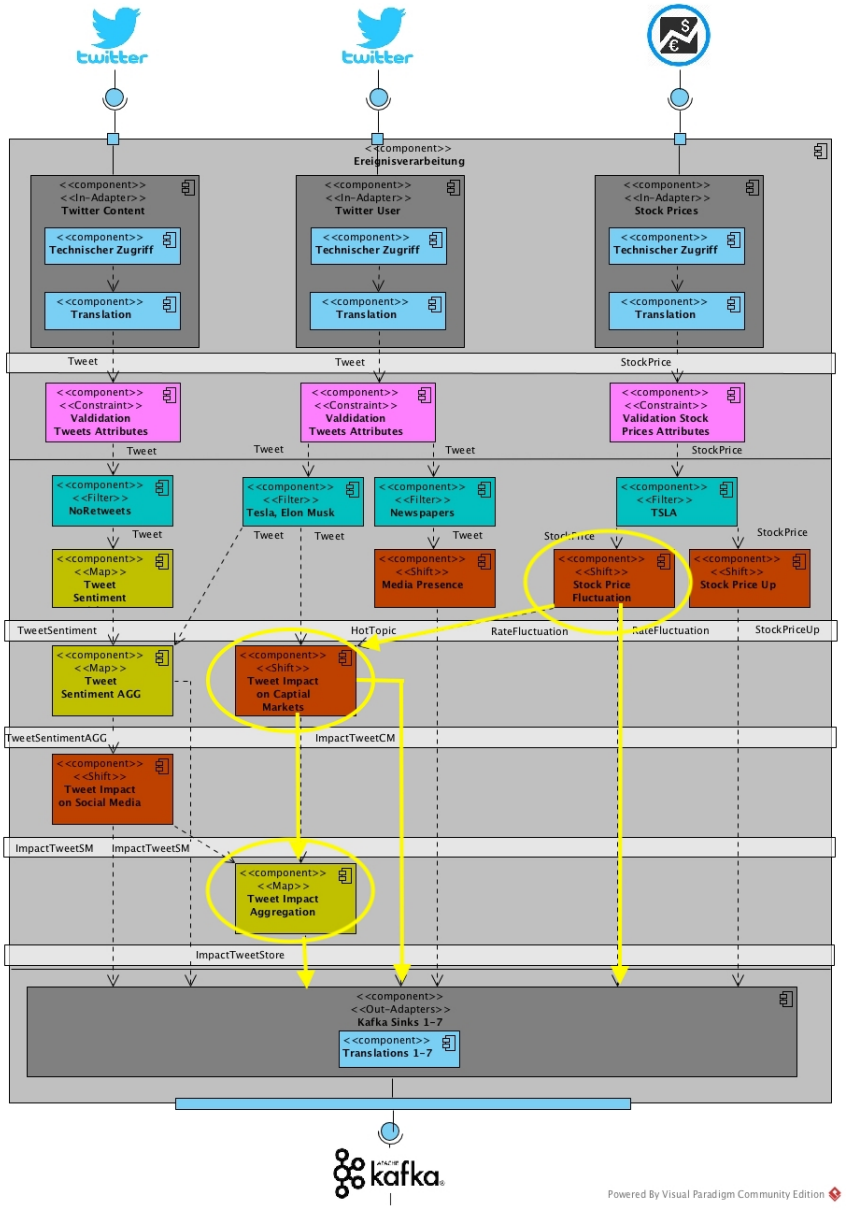

Abbildung E.1.: Auswirkungen durch Regeländerung

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Glossar

Abstraktion Verallgemeinerung oder Methode zur Reduzierung der Komplexität.

Aktionsteil Beschreibt innerhalb der Ereignisregel, welche Aktion bei positiven Bedingungsteil ausgeführt werden soll.

Anforderung Bedingung oder erwünschte Fähigkeit eines Objekts.

Apache Flink Softwaremodul, welches den Rahmen für die Ereignisverarbeitung bereitstellt.

Architektur Bauplan für ein konkretes System.

Attribut Eine Information innerhalb oder eine zusätzliche Beschreibung eines Ereignisobjekts.

Bedingungsteil Besteht aus einem Ereignismuster und stellt innerhalb einer Ereignisregel die Bedingung.

Complex Event Processing Verarbeitung von (komplexen) Ereignissen mittels lesen, erzeugen, transformieren, abstrahieren.

Datenstrom Eine lineare Sequenz von Daten.

Ereignis Das Auftreten einer Veränderung.

Ereignisbehandlung Operative Anwendungen, die die komplexen Ereignisse behandeln .

Ereignismuster Definiert eine bestimmte Sequenz von Ereignissen durch Operatoren.

Ereignisobjekt Ein Objekt, welches ein Ereignis repräsentiert .

Ereignisquelle Eine Entität, die Ereignisse erkennt und versendet.

Ereignisregel Beschreibt, wie Ereignisse verarbeitet werden sollen. Besteht aus Bedingungsteil und Aktionsteil.

Ereignissenke Eine Entität, die Ereignisse konsumiert.

Ereignisstrom Eine lineare Sequenz von Ereignisobjekten.

Ereignistyp Beschreibt, zu welcher Klasse das Ereignisobjekt gehört.

Ereignisverarbeitung Eine Komponente, die Ereignisse mittels Transformation und Filterung verarbeitet.

Event Processing Engine Gleicht den Ereignisstrom mit dem Ereignismuster ab.

Event Processing Agent Ein Softwaremodul, welches Ereignisobjekte verarbeitet.

Event-Driven Architecture Ein Architekturstil, der die Verarbeitung von Ereignissen in den Fokus rückt.

Event Processing Language Eine High Level Programmiersprache, mit der Ereignisregeln definiert werden.

High Level Programmiersprache Programmiersprache mit einem hohen Abstraktionsgrad.

Instanz Beschreibt ein konkretes Objekt und die Zugehehörigkeit zu einer Klasse.

Klasse Beschreibt eine Art Blaupause für Ereignisobjekte. In ihr sind unter anderem die Attribute für ein Ereignisobjekt festgelegt.

Library siehe Bibliothek.

Virtual Machine Bildet Hardware softwaretechnisch nach und kapselt sie.

Akronyme

API Application Programming Interface.

CEP Complex Event Processing.

EDA Event-Driven Architecture.

EPA Event Processing Agent.

EPE Event Processing Engine.

VM Virtual Machine.

Literaturverzeichnis

[ZU99] D. Zimmer und R. Unland. "On the semantics of complex events in active database management systems". In: *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*. IEEE, 1999. DOI: 10.1109/icde.1999.754955. URL: https://doi.org/10.1109/icde.1999.754955.

[Luc01] David C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 0201727897.

[Sie04] Johannes Siedersleben. *Moderne Software-Architektur*. Dpunkt.Verlag GmbH, 2004. ISBN: 3898642925.

[Qui05] Klaus Quibeldey-Cirkel. *Entwurfsmuster*. Online abgerufen am 28.07.2018. Juli 2005. URL: https://gi.de/informatiklexikon/entwurfsmuster/.

[CS07] K.M. Chandy und W.R Schulte. *What is Event Driven Architecture (EDA) and Why Does it Matter*. 2007.

[LW08] David Luckham und Roy W Schulte. "Event Processing Glossary - Version 1.1". In: (Juli 2008). Abgerufen am 13.05.2018. URL: http://www.ep-ts.com.

[WH08] Donald K Wright und Michelle D Hinson. "How blogs and social media are changing public relations and the way it is practiced". In: *Public relations journal* 2.2 (2008), S. 1–21.

[PV09] Adrian Paschke und Paul Vincent. "A Reference Architecture for Event Processing". In: *Proceedings of the Third ACM International Conference on Distributed Event-Based Systems*. DEBS '09. Nashville, Tennessee: ACM, 2009, 25:1–25:4. ISBN: 978-1-60558-665-6. DOI: 10.1145/1619258.1619291. URL: http://doi.acm.org/10.1145/1619258.1619291.

[Alv+10] Alex Alves u. a. *Event Processing Architectures leading to an EPTS Reference Architecture*. Online abgerufen am 31.05.2018. 2010. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.710.9759.

[BD10] Ralf Bruns und Jürgen Dunkel. *Event-Driven Architecture - Softwarearchitektur für ereignisgesteuerte Geschäftsprozesse*. Berlin Heidelberg New York: Springer-Verlag, 2010. ISBN: 978-3-642-02439-9.

[EN10] Opher Etzion und Peter Niblett. *Event Processing in Action*. Birmingham: Manning, 2010. ISBN: 978-1-935-18221-4.

[K10] Chandy K. *Event Processing: Designing IT Systems for Agile Companies* -. Madison: McGraw Hill Professional, 2010. ISBN: 978-0-071-63711-4.

[CEA11] Mani K. Chandy, Opher Etzion und Rainer von Ammon. "10201 Executive Summary and Manifesto – Event Processing". In: *Event Processing*. Hrsg. von K. Mani Chandy, Opher Etzion und Rainer von Ammon. Dagstuhl Seminar Proceedings 10201. Dagstuhl, Germany: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, Germany, 2011. URL: http://drops.dagstuhl.de/opus/volltexte/2011/2985.

[Chl+11] Michael Chlistalla u. a. "High-frequency trading". In: *Deutsche Bank Research* 7 (2011).

[Kri11] Anbu Krishnaswamy. *Oracle Approach to Event Driven Architecture*. Online abgerufen am 31.05.2018. Aug. 2011. URL: https://www.slideshare.net/OTNArchbeat/eda-rakrishnaswamyrws.

[Ded+13] Elif Dede u. a. "Performance evaluation of a mongodb and hadoop platform for scientific data analysis". In: *Proceedings of the 4th ACM workshop on Scientific cloud computing*. Online abgerufen am 28.07.2018. ACM. 2013, S. 13–20.

[Fel+13] Zohar Feldman u. a. "Proactive Event Processing in Action: A Case Study on the Proactive Management of Transport Processes (Industry Article)". In: *Proceedings of the 7th ACM International Conference on Distributed Event-based Systems*. DEBS '13. Online abgerufen am 28.07.2018. Arlington, Texas, USA: ACM, 2013, S. 97–106. ISBN: 978-1-4503-1758-0. DOI: 10.1145/2488222.2488274. URL: http://doi.acm.org/10.1145/2488222.2488274.

[Hur+13] Judith Hurwitz u. a. *Big Data For Dummies*. For Dummies, 2013. ISBN: 1118504224.

[SR13] Mohd. Saboor und Rajesh Rengasamy. *DESIGNING AND DEVELOPING COMPLEX EVENT PROCESSING APPLICATIONS*. Aug. 2013. URL: https://www.sapientglobalmarkets.com/thought_leadership/designing-and-developing-complex-event-processing-applications.

[BIT14] BITKOM. “Big-Data-Technologien - Wissen fÃ¼r Entscheider”. In: (Feb. 2014). Online abgerufen am 13.05.2018. URL: https://www.bitkom.org/Bitkom/Publikationen/Big-Data-Technologien-Wissen-fuer-Entscheider.html.

[BD15] Ralf Bruns und Jürgen Dunkel. *Complex Event Processing - Komplexe Analyse von massiven Datenströmen mit CEP*. Berlin Heidelberg New York: Springer-Verlag, 2015. ISBN: 978-3-658-09899-5.

[KK15] Martin Kleppmann und Jay Kreps. “Kafka, Samza and the Unix Philosophy of Distributed Data.” In: *IEEE Data Eng. Bull.* 38.4 (2015). Online abgerufen am 28.07.2018, S. 4–14.

[JS16] Kevin Jacobs und Kacper Surdy. *Apache Flink: Distributed Stream Data Processing*. Techn. Ber. Online abgerufen am 28.07.2018. Sep. 2016. URL: http://cds.cern.ch/record/2208322.

[Mor16] Andrew Morgan. *MongoDB Data Streaming Implementing a MongoDB Kafka Consumer*. Online abgerufen am 28.07.2018. Juni 2016. URL: https://www.mongodb.com/blog/post/mongodb-and-data-streaming-implementing-a-mongodb-kafka-consumer.

[Roh16] Till Rohrmann. *Introducing Complex Event Processing (CEP) with Apache Flink*. Online abgerufen am 03.07.2018. Apr. 2016. URL: https://flink.apache.org/news/2016/04/06/cep-monitoring.html.

[ALG17] Alaa Alakari, Kin Fun Li und Fayez Gebali. “Complex event processing enrichment: Motivations and challenges”. In: *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, Aug. 2017. DOI: 10.1109/pacrim.2017.8121891. URL: https://doi.org/10.1109/pacrim.2017.8121891.

[Hed17] Ulrich Hedtstück. *Complex Event Processing - Verarbeitung von Ereignismustern in Datenströmen*. Berlin Heidelberg New York: Springer-Verlag, 2017. ISBN: 978-3-662-53451-9.

[Mic17] Brenda M. Michelson. *Event-Driven Architecture Overview*. Online abgerufen am 28.05.2018. Juni 2017. URL: http://www.elementallinks.com/2006/02/06/event-driven-architecture-overview/.

[Sha17] Gwen Shapira. *Apache Kafka: Getting Started Guide*. Online abgerufen am 28.07.2018. Juli 2017. URL: https://www.confluent.io/blog/apache-kafka-getting-started/.

[FAZ18] FAZ. *Internet-Star Ã¼ber Snapchat: Ein Tweet, der 1,7 Milliarden Dollar kostet*. Online abgerufen am 16.06.2018. Feb. 2018. URL: http://www.faz.net/aktuell/finanzen/finanzmarkt/snapchat-tweet-von-kylie-jenner-bringt-aktie-zum-absturz-15463825.html.

[Fli18a] Apache Flink. *Data Types Serialization*. Online abgerufen am 03.07.2018. 2018. URL: https://ci.apache.org/projects/flink/flink-docs-stable/dev/types_serialization.html.

[Fli18b] Apache Flink. *Distributed Runtime Environment*. Online abgerufen am 03.07.2018. 2018. URL: https://ci.apache.org/projects/flink/flink-docs-release-1.5/concepts/runtime.html.

[Fli18c] Apache Flink. *Scalable Stream and Batch Data Processing*. Online abgerufen am 10.06.2018. 2018. URL: https://flink.apache.org/.

[Fli18d] Apache Flink. *Standalone Cluster*. Online abgerufen am 28.07.2018. 2018. URL: https://ci.apache.org/projects/flink/flink-docs-release-1.5/ops/deployment/cluster_setup.html.

[Fli18e] Apache Flink. *Streaming Connectors*. Online abgerufen am 28.07.2018. 2018. URL: https://ci.apache.org/projects/flink/flink-docs-release-1.5/dev/connectors/.

[Fli18f] Apache Flink. *What is Apache Flink?* 2018. URL: https://flink.apache.org/flink-architecture.html.

[Kaf18] Apache Kafka. *Introduction*. Online abgerufen am 16.06.2018. 2018. URL: https://kafka.apache.org/intro.

[dea] deacademic. *Wertkette*. Online abgerufen am 28.07.2018. URL: http://deacademic.com/dic.nsf/dewiki/1502199.

[Sta] Statista. *Twitter - Monatlich aktive Nutzer weltweit 2018 Statistik*. Online abgerufen am 29.06.2018. URL: https://de.statista.com/statistik/daten/studie/232401/umfrage/monatlich-aktive-nutzer-von-twitter-weltweit-zeitreihe/.

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

Hamburg, 14. August 2018 Steffen Wohlers