CARL VON OSSIETZKY universität OLDENBURG

VERY LARGE BUSINESS APPLICATIONS
Carl von Ossietzky Universität Oldenburg

Datenarchivierung von Energiedaten (DAvE)

Projektdokumentation

Themensteller:	Prof. Dr.-Ing. Jorge Marx Gómez
Betreuer:	Dr.-Ing. Andreas Solsbach M. Eng. & Tech. Viktor Dmitriyev M. Sc. Jad Asswad
Abgabetermin:	31.03.2018

Abstract

Die Digitalisierung der Energiebranche stellt diese vor neuen Herausforderungen. Um Daten von Smart Metern im Kontext von Big Data zu verarbeiten, existieren bereits Ansätze. Die Archivierung von diesen und die Analyse mit passenden Werkzeugen können neue Mehrwerte generieren.

In diesem Projekt wurde ein prototypisches System (DAvE) entwickelt, um riesige Datenmengen zu archivieren und daraus mithilfe von Machine Learning Algorithmen Vorhersagemodelle zu generieren. Die genutzte Machine Learning Bibliothek ist scikit-learn. Um einen entkoppelten bidirektionalen Datenstrom zu garantieren, wird Apache Kafka als Messaging System implementiert. Für den Datentransport zwischen den einzelnen Subsystemen des DAvE-Systems und zu externen Messaging Systemen wird Apache NiFi genutzt. Als zuverlässiges und skalierbares Datenbankmanagementsystem wird Apache Cassandra verwendet.

Das Ergebnis ist ein prototypisches System, welches Smart Meter Daten und Wetterdaten archivieren und bereitstellen kann. Ebenfalls stellt das DAvE-System eine Plattform zur Wissensgenerierung dar. Erzeugte Vorhersagemodelle können per PMML ausgetauscht werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungzverzeichnis

AMQP Advanced Message Queuing Protocol
AXT Agent for XML Transportation
CAP Consistency, Availability und Partition tolerance
CLI Command Line Interface
COSEM Companion Specification for Energy Metering
CQL Continuous Query Language
CSV Comma Separated Values
DASH Data AnalyticS with Hadoop
DAvE Datenarchvierung von Energiedaten
DLMS Device Language Message Specification
DMM Data Mining Modelle
DWD Deutscher Wetterdienst
ELT Extract, Load und Transform
FIFO first in first out
FTP File Transfer Protocol
HDFS Hadoop Distributed File System
ID Identifikation
JSON JavaScript Object Notation
JVM Java Virtual Machine
KDD Knowledge Discovery in Databases
LV Low Voltage
MB Message Broker
MLP Multi Layer Perceptron
MQ Message Queue
NiFi NiagaraFiles
OAS OpenAPI Specifikation
PMML Predictive Model Markup Language
PV Photovoltaikanlage
QL Query Language
QN Qualitätsniveau
REST Representational State Transfer
SMARD Strommarktdaten
SVM Support Vector Machine
UUID Universally Unique Identifier
VM Virtuelle Maschine
XML Extensible Markup Language

1. Einleitung

Durch die Einführung des Gesetztes zur Digitalisierung der Energiewende, wurde das Startsignal für eine digitale Infrastruktur mit Smart Grids inklusive Smart Metern in Deutschland gesetzt. Dies führt zum Austausch aller herkömmlichen Ferraris-Zähler durch intelligente Messsysteme bis zum Jahr 2032.

Der Ausgleich zwischen Stromangebot und -nachfrage stellt eine zentrale Herausforderung dabei dar, wodurch die Digitalisierung der Energiewende anhand der Vielfalt von Daten eine Schlüsselfunktion zur Lösung dieser Aufgabe beitragen kann. Anhand der effizienten Nutzung der Energie können Stromnetze stabilisiert werden und Leistungsengpässe verringert werden, sodass die Steuerung und Regelung mit digitalen Technologien weiter in das Zentrum des zukünftigen Energiesystems treten.[Bun16a]

Diese Einführung konfrontiert die Netzbetreiber mit der Bewältigung riesiger Datenmengen. Durch einen umfassenden Einsatz von Informations- und Kommunikationstechnologien können aus der Herausforderung neue Chancen generiert werden. Es ist möglich aus den Datenmengen umfassende Mehrwerte für die beteiligten Akteure zu gewinnen, damit Angebot und Nachfrage zeitnah erfasst und besser aufeinander abgestimmt werden können. Die anfallenden Daten können nicht nur archiviert werden, sondern bieten weitläufige Potentiale zu Datenanalyse für eine Stabilisierung des Stromnetzes und die Bildung variabler Versorgungstarife. Darüber hinaus können in der neuen Energiewelt zukunftsfähige Geschäftsmodelle entstehen, die neben der Optimierungsprozesse des Energiesystems weitere Vorteile für Erzeuger und Verbraucher bieten.[Ger16]

In dem Kontext der Energiewende mit der Einführung von intelligenten Messsystemen gilt es eine Architektur zu entwickeln, die Daten persistent über Jahrzehnte archivieren kann. Mit geeigneten Analysewerkzeugen sollen neue Mehrwerte durch die Auswertung dieser Daten generiert werden. Die Architektur zur Archivierung und Analyse von großen Datenmengen sollen in unterschiedlichen Branchen angewendet werden. Zunächst wird sie prototypisch im Energiesektor umgesetzt. Zusätzlich sollen geeignete Analysewerkzeuge implementiert werden, um Muster aus den archivierten Daten zu erkennen. Es muss möglich sein, Stromerzeuger und -anbieter in die Lage zu versetzen, Vorhersagen über Energieverbräuche und Stromerzeugung zu bilden und nutzen zu können. Anhand von generierten Vorhersagemodellen können dazu Auskünfte in unterschiedlichen Zeitperioden getroffen werden. Durch den Datenzuwachs ist es möglich, die Vorhersagemodelle stetig zu optimieren und somit immer bessere Prognosen zu erhalten. Dies ermöglicht es den Stromerzeugern, den Energieverbrauch zu prognostizieren und somit die bereitzustellende Menge an Strom besser kalkulieren zu können.

Die Umsetzung dieses Vorhabens wird im Rahmen der Projektgruppe DAvE an der Carl von Ossietzky Universität Oldenburg realisiert. Das zugehörige Konzept und dessen Umsetzung werden im Folgenden erläutert.

1.1. Zielsetzung

Das Projekt beschäftigt sich mit der Leitfrage, wie ein datengetriebener Energiemarkt in der Zukunft aussehen kann. Dabei gilt es eine Kombination aus Netz, Daten und Analyse zu schaffen und anhand dieser Erkenntnisse neue Geschäftsmodelle aufzeigen zu können. Es muss die Möglichkeit geschaffen werden, die anfallenden Daten aus Smart Metern archivieren zu können. Dazu wird eine Infrastruktur geschaffen, mit Hilfe dieser die Daten entgegengenommen und in einem Archiv abgelegt werden. Zusätzlich können die archivierten Daten zur Wissensgenerierung, z. B. zur Vorhersage des Energieverbrauchs, ausgewertet werden. Die generierten Vorhersagemodelle werden zur Verfügung gestellt und dadurch für Dritte nutzbar gemacht.

Neben der primären Datenquellen, einem Niederspannungsnetz, sollen zusätzlich externe Datenquellen eingebunden werden. Dazu gehören Wetterdaten vom Deutschen Wetterdienst. Dazu werden die stündlich bereitgestellten Daten angefragt und archiviert. Diese beinhalten u. a. Informationen zu Windgeschwindigkeiten und Sonnenstunden und ermöglichen Prognosen für die potentielle Energieerzeugung mit Wind- und Solarkraft. Zusätzlich ist es möglich, die Daten auch aufgrund ihrer geographischen Informationen auszuwerten.

Das System muss dabei durch das hohe Datenaufkommen den gestiegenen Anforderungen an sehr große Datenmengen gerecht werden. Durch den Einsatz von Big Data Technologien und Data Science Verfahren werden Anfragen bezüglich Last- und Vermarktungsprognosen auf hochaufgelösten und umfangreichen Langzeitdaten ermöglicht, welche beteiligten Akteuren zukünftig Mehrwerte in den Prozessen ihrer Leistungserstellung ermöglichen können. Damit Anfragen an das Archiv gestellt werden können, ist eine Schnittstelle zu diesem notwendig. Mit der Schnittstelle sollen archivierte Daten angefragt und Prognosemodelle angefordert werden.

1.2. Projekteinbettung

Die Projektgruppe DAvE wird in der Abteilung VLBA unter der Leitung von Prof. Dr.-Ing. Jorge Marx-Gómez betreut. Die Umsetzung erfolgt als Teil des Projekts NetzDatenStrom (NDS) am OFFIS in Oldenburg unter der Leitung von Dr.-Ing.Sven Rosinger. Das Projekt NetzDatenStrom dient der Erforschung einer standardkonformen Integration von quelloffenen Big Data Lösungen in existierende Netzleitsysteme. Das Projekt hat zum Ziel die Definition von offenen und standardisierten Schnittstellen sowie einer Referenzarchitektur für Netzleitsysteme. Darüber hinaus soll ein Datenaustausch über einen zentralen Enterprise Service Bus geregelt werden. Eine Erweiterung von vorhandenen Archiv- und Datenbankstrukturen von Leitsystemlösungen durch Big Data Komponenten ist vorgesehen, sodass die Speicherung und Verarbeitung von großen Datenmengen ermöglicht wird. Darüber hinaus soll eine Auswertung und Verarbeitung von Mess- und Sensordaten in Echtzeit sowie die Anbindung externer und Integration heterogener Daten ermöglicht werden. Die

Struktur des Projekts NetzDatenStrom wird in der folgenden Abbildung dargestellt.

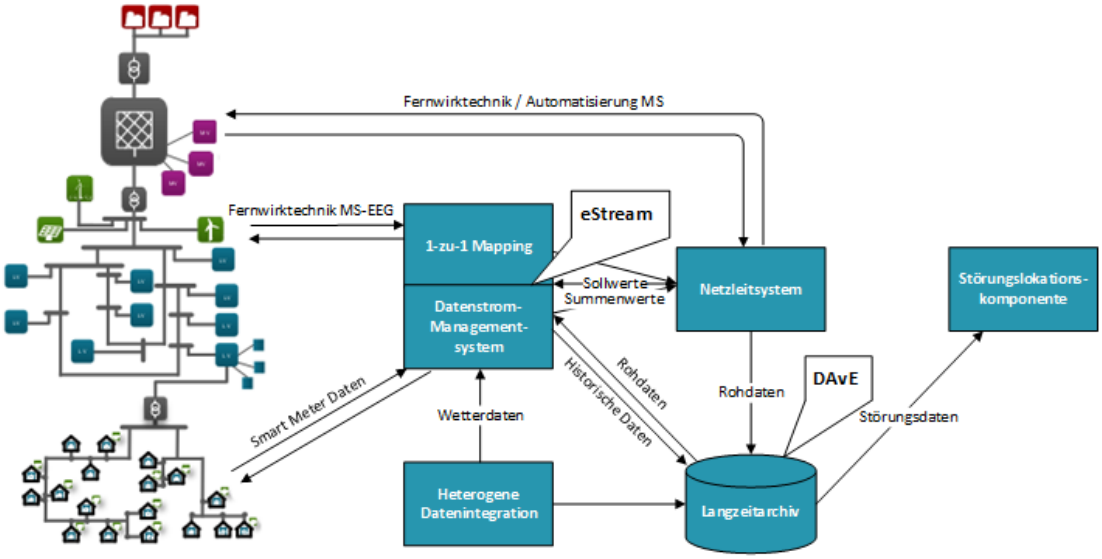

Abbildung 1: NDS Projektübersicht (vgl. [OFF18])

In Rahmen des Projekts übernimmt das DAvE-System die Langzeitarchivierung von Daten. Ebenfalls wird DAvE von der Projektgruppe E-Stream verwendet um archivierte Daten und Vorhersagemodelle abzufragen.

Als Ergebnisse des NDS Projektvorhabens gelten eine stärkere Kundenindividualisierung der Netzbetriebsführung sowie eine Kosteneinsparung durch die Big Data Lösungen. Großvolumige Mess- und Topologiedaten sollen für das Netzleitstellenpersonal nutzbar gemacht werden, sodass bei einer steigenden Häufigkeit von Eingriffen in das System eine Verkürzung der Reaktionszeiten erfolgen kann.

1.3. Aufbau der Projektdokumentation

Diese Projektdokumentation gliedert sich in neun Kapiteln. In der Einleitung wird auf das Themengebiet und der Zielsetzung des Projektes eingegangen. Danach erfolgt eine Einordnung des Projektes in das Projekt Netzdatenstrom.

Das zweite Kapitel beinhaltet die Projektorganisation, dabei werden die Mitglieder der Projektgruppe vorgestellt sowie die Stakeholder. Weiterhin werden das Projektvorgehen und die Projektphasen näher erläutert. Nach dem Kapitel der Projektorganisation wird das Konzept des Projektes genauer beschrieben. So werden zuerst die vier Use Cases vorgestellt. Daraufhin werden die technischen und funktionalen Anforderungen der einzelnen

Komponenten des Projektes erörtert. Abschließend erfolgt eine Beschreibung der Systemarchitektur. Die darauf folgende Technologieauswahl wird in Kapitel 4 erläutert. Es werden Anforderungen für die Technologieauswahl in den jeweiligen Bereichen aufgestellt. Daraufhin erfolgt eine Beschreibung der untersuchten Technologien sowie eine kurze Auswertung, die zeigt, welche Technologien verwendet werden. Das fünfte Kapitel behandelt die Installation der einzelnen Komponenten, welche im vorherigen Kapitel ausgewählt wurden.

Nach der Installation wird im sechsten Kapitel die Realisierung näher beschrieben. So werden zuallererst die ausgewählte Datenquellen vorgestellt. Daraufhin wird die Implementierung der jeweiligen Komponenten im Einzelnen geschildert.

In dem siebten Kapitel wird die Gesamtarchitektur des Systems der Projektgruppe vorgestellt und beschrieben, wie die einzelnen Komponenten miteinander verbunden sind.

Im Anschluss folgt das Kapitel Test und Evaluation. Es wird auf die Testumgebung eingegangen sowie das Evaluationskonzept, die Komponententests, die Integrationstest und die Systemtests näher erläutert. Abschließend wird ein Fazit und ein Ausblick über das gesamte Projekt gegeben. Dabei werden die Erfahrungen der PG beschrieben und in Aussicht gestellt, in wie weit das entwickelte System nach dem Abschluss noch weiterentwickelt werden könnte.

2. Projektorganisation

In diesem Kapitel wird Organisation des Projektes dargestellt und erläutert. Zunächst werden die Mitglieder der Projektgruppe sowie die vergebenen Rollen und Aufgaben innerhalb der Projektgruppe gennant. Darauf folgt die Vorstellung des Vorgehens innerhalb des Projekts in 2.3: Projektvorgehen. Geschlossen wird dieses Kaptitel mit einer Übersicht über die Projektphasen in 2.4: Projektphasen.

2.1. Mitglieder der Projektgruppe DAvE

Die Projektgruppe umfasst drei Betreuer und elf Projektgruppenmitglieder. Im Folgenden sind alle beteiligten Personen in der Tabelle 1: Betreuer und Projektgruppenmitglieder alphabetisch aufgeführt.

Betreuer	PG-Mitglieder
M.Sc Jad Asswad	Maged Ahmad
M. Eng. & Tech. Viktor Dimitryev	Julian Brunner
Dr,-Ing. Andreas Solsbach	Alexander Eguchi
	Jens Feldhaus
	Nils Meyer
	Jakob Möller
	Miriam Müller
	Yasin Oguz
	Henning Schlotmann
	Martin Sonntag
	Helge Wendtland

Tabelle 1: Betreuer und Projektgruppenmitglieder

Zu Beginn der Projektgruppe wurden einige Sonderrollen vergeben, welche durch einzelne Projektgruppenmitglieder wahrgenommen wurden. Die Rollen umfassen folgende:

Projektmanager Aufgabe des Projektmanagers ist es, die Koordination der Gruppe sicherzustellen. Dazu ist es notwendig, dass der Projektmanager einen Teil seiner wöchentlichen Arbeitszeit dafür aufwendet, den aktuellen Stand des Projektes zusammenzutragen. Der Projektmanager koordiniert darüber hinaus Termine mit den Betreuern und der Projektgruppe E-Stream.

Dokumentationsbeauftragter Der Dokumentationsbeauftragte ist dafür zuständig, Vorlagen für alle Dokumente vorzubereiten und die Einhaltung der Vorlage sicherzustellen.

Dokumente in diesem Kontext sind u. a. die Seminararbeiten und die Abschlussdokumentation des Projektes. Das Ziel des Dokumentationsbeauftragten ist es, ein einheitliches Aussehen und einen einheitlichen Aufbau aller Dokumente zu gewährleisten.

Blogbeauftragter Die Aufgabe des Blogbeauftragten ist es, während der Laufzeit der Projektgruppe Ereignisse und Events zu dokumentieren. Diese werden auf einem Blog auf der Webseite der Abteilung VLBA veröffentlicht.

Serveradministrator Der Serveradministrator hat den Auftrag, die Funktionalität der Infrastruktur sicherzustellen. Ein Ausfall der Infrastruktur hätte eine negative Auswirkung auf den Projektfortschritt zur Folge und würde die Weiterarbeit behindern. Um die Verfügbarkeit des Administrators zu gewährleisten wurden zwei Stellvertreter zusätzlich benannt.

Testbeauftragter Der Testbeauftragte ist dafür zuständig, dass die Projektergebnisse getestet werden. Der Träger dieser Rolle stellt sicher, dass ein Testkonzept erarbeitet wird und das Konventionen für das Testen aufgestellt werden. Weiterhin werden Testphasen geplant, dessen Einhaltung gewährleistet. Das Testen übernehmen jedoch alle Mitglieder in der Projektgruppe.

Scrum Master Da zwischenzeitlich ein agiles Vorgehen angelehnt an Scrum verwendet wurde, wurde für diesen Zeitraum ein Scrum Master bestimmt, der den Prozess koordinierte und evaluierte.

2.2. Stakeholder

Stakeholder, die ein Interesse an der Realisierung des Projektes haben waren sowohl die Projektbetreuer als auch die Projektgruppe E-Stream.

Projektbetreuer Die Projektbetreuer Dr,-Ing. Andreas Solsbach, M. Eng. & Tech. Viktor Dimitryev und M. Sc. Jad Asswad unterstützen die Projektgruppe als erste Ansprechpartner bei inhaltlichen und organisatorischen Fragen rund um das Projekt. Außerdem wurden alle Projektmeilensteine und -fortschritte ihnen vorgestellt und reflektiert.

Projektgruppe E-Stream Über das Projekt NetzDatenStrom besteht eine Kooperation mit der parallel verlaufenen Projektgruppe E-Stream. Über die Schnittstelle des DAvE-Systems greift die Projektgruppe auf archivierte Daten und generierte Vorhersagemodelle zu.

2.3. Projektvorgehen

Im folgenden Abschnitt wird das Vorgehen der Projektgruppe vorgestellt. Dazu wird die Herangehensweise an Aufgaben und die Kommunikation innerhalb der Gruppe beschrieben. Zuerst wird die Projektgruppenorganisation im Unterabschnitt 2.3.1: Kleingruppen vorgestellt. Im Anschluss daran wird das Vorgehensmodell in 2.3.2: Vorgehensmodell beschrieben. Abschließend werden die verwendeten Kommunikationskanäle genannt.

2.3.1. Kleingruppen

Das Projektteam wurde in Kleingruppen aufgeteilt. Die Aufteilung in Kleingruppen hatte die Motivation die Abarbeitung von Aufgabenpaketen zu parallelisieren. Außerdem hatte es die Auswirkung, dass diese Gruppen sich auf Subsysteme spezialisieren konnten. Die Kleingruppen hatten dabei unterschiedliche Aufgabenschwerpunkte. Insgesamt wurden fünf Kleingruppen gebildet:

- Archivierung
- Schnittstelle E-Stream
- Data Science
- Date Engineering
- Evaluation

Die Bildung der Kleingruppen ist ein Vorgriff auf die Konzeption des DAvE-Systems im Kapitel 3: Konzept und repräsentiert die darin enthaltenen einzelnen Subsysteme. Im Verlauf des Projektes wurde die Gruppe Data Engineering aufgelöst, da die Aufgaben der Gruppe abgeschlossen wurden. Die Gruppe Evaluation wurde zum Ende der Projektgruppe gebildet, um das Testvorgehen für den Prototypen zu planen.

Ebenfalls zu bemerken ist, dass sich die Zusammenstellung der Kleingruppen während des Projektes geändert hat. Gründe waren neben dem Auflösen, bzw. Bilden von Gruppen auch die Umverteilung von Projektgruppenmitgliedern. Damit die Gruppen ihre Aufgabenpakete bewältigen konnten, war das Umverteilen von Ressourcen notwendig.

2.3.2. Vorgehensmodell

Während des Projekts hat sich ein agiles Projektvorgehen etabliert. Aufgrund der Komplexität der Aufgabenstellung und Größe der Projektgruppe war ein iteratives Vorgehen erforderlich. Des Weiteren konnte die Terminierung von Projektabschnitten gemäß der Meilensteinplanung nicht in allen Fällen fristgerecht eingehalten werden.

Wie bereits im vorherigen Unterabschnitt 2.3.1: Kleingruppen erläutert, hatte jede Kleingruppe dabei unterschiedliche Themenschwerpunkte und darauf zugeschnittene Aufgabenpakete. Der Fortschritt der Aufgabenpakete wurde in einem wöchentlichen Gruppentreffen von den Kleingruppen vorgestellt. Dieses Vorgehensmodell erwies sich gerade in den frühen Projektphasen als sinnvoll, da der tatsächliche Umfang der Aufgabenpakete, aufgrund von fehlendem Fachwissen, nur schwer festzustellen war. Die neuen Arbeitspakete wurden jeweils auf den Ergebnissen der vorherigen Woche aufgebaut.

Im späteren Projektverlauf wurde das agile Vorgehen stärker strukturiert, angelehnt am Vorgehensmodell Scrum. Begründet war diese Anpassung dadurch dass der Technologieauswahlprozess arbeitsintensiver war. Zur Unterstützung dieses Vorgehens wurde die Projektmanagement Software Jira genutzt. Alle erfassbaren Aufgabenpakete wurden in das System übernommen. Der Planungsvorgang erfolgte alle zwei Wochen, dadurch waren die Aufgabenpakete besser strukturiert. Darüber hinaus ermöglichte Jira eine wesentlich bessere Dokumentation des Projektfortschritts. Zudem wurden Verantwortliche für die Aufgabenpakete benannt, welche die Qualität des Ergebnisses und des Prozesses sicherstellten.

Nach knapp zwei Monaten wurde jedoch die Verwendung von Jira zu aufwendig. Die eingesetzte Zeit zur Organisation der Aufgaben in Jira hat keinen angemessenen Mehrwert für die Projektgruppe. Darüber hinaus waren viele Aufgabenpakete, welche sich aufgrund neuer verwendeter Technologien ergeben haben, nicht so gut einzuschätzen wie zunächst gedacht, was dazu führte das viele Aufgabenpakete nicht in der dafür vorgesehenen Zeit bearbeitet werden konnten. Durch die gute Zusammenarbeit und die offene Kommunikation innerhalb der Projektgruppe war es möglich, auch ohne die präzisen Strukturen, welche Jira ermöglicht, schnell und nachvollziehbar Ergebnisse zu erzielen. Durch das Wegfallen des Aufwands zur Organisation der Aufgaben in Jira wurde eine Menge Zeit eingespart.

2.3.3. Kommunikation

Die Kommunikation innerhalb der Projektgruppe wurde über zwei Kanäle gesteuert. Als primärer Kommunikationskanal wurde E-Mail verwendet. Für eine unmittelbarer Kommunikation, insbesondere innerhalb der Projektgruppe, wurden die Messaging Dienste WhatsApp und Slack genutzt. Die Kommunikation mit der Projektgruppe E-Stream wurde zunächst via Slack geführt, jedoch führte dies nicht zu den gewünschten kurzen Kommunikationswegen. Im späteren Verlauf der Projektgruppe wurde verstärkt E-Mail genutzt.

2.4. Projektphasen

Der Projektverlauf lässt sich in drei Phasen einteilen. Diese werden im folgenden dargestellt und erläutert. Ebenfalls werden die in den Phasen erstellen Meilensteinpläne gezeigt.

2.4.1. Erste Phase - Seminararbeiten & Konzeption

Um einen Überblick über die Thematik des Projektrahmens und -ziels zu erhalten, wurde zunächst eine Seminararbeitsphase durchgeführt. Darin hat sich jedes Projektgruppenmitglied in einen relevanten Bereich der Projektdomäne eingearbeitet und die Erkenntnisse in einer Seminararbeit festgehalten und der gesamten Gruppe vorgestellt. Die Bearbeitungszeit dafür war auf einen Monat begrenzt. Die Präsentationen schlossen sich daran an. Währenddessen konnten Fragen geklärt werden, die im Zusammenhang mit dem Themenbereich aufgetreten sind. Die erste Phase bot die Möglichkeit Speziallisten für bestimmte Projektbereiche zu finden. Eine weitere Aufgabe in der ersten Phase war das Projektziel eindeutig zu formulieren und die Projektmitglieder in Kleingruppen aufzuteilen.

Während der gesamten ersten Projektphase wurden Workshops und kleinere Ideenfindungsseminare abgehalten. Diese Workshops dienten dazu, das Projektziel genauer zu bestimmen und einordnen zu können. Dieses Vorgehen war notwendig, da zum Projektstart keine eindeutige Zielformulierung, Anforderungen oder genaue Beschreibung der Projektumgebung definiert war. In dieser Phase konnten die unspezifischen Vorgaben an die Projektgruppe konstruktiv erarbeitet und definiert werden.

Als Ergebnis wurde u. a. ein grober Meilensteinplan erstellt. Dieser ist der folgenden Tabelle zu entnehmen.

Meilenstein	Datum
Entwurf - Architektur	19.04.2017
Entwurf - Big Data Archive	31.05.2017
Prototyp testen	17.07.2017
Prototyp präsentieren	31.07.2017
Produkt vorstellen	04.12.2017
Abschluss der Nachbearbeitung	12.02.2018
Abschluss der Dokumentation	23.02.2018
Abschlusspräsentation	09.04.2018

Tabelle 2: Meilensteinplan, April 2017

2.4.2. Zweite Phase - Entwicklung

In der zweiten Projektphase erfolgte die Aufteilung der Projektgruppenmitglieder in die Kleingruppen. Wie im Unterabschnitt 2.3.1: Kleingruppen beschrieben, hatte jede Kleingruppe dabei eine eigene Zielsetzung die erfüllt werden sollte. Die Aufgaben der Gruppen wurden von ihr selbst definiert. In wöchentlichen Treffen wurden die Fortschritte präsentiert und besprochen sowie die Aufgabenpakete der nächsten Woche vorgestellt.

Ebenfalls wurden Workshops zusammen mit der Projektgruppe E-Stream veranstaltet, um

die Schnittstelle des DAvE-Systems zu definieren. Dies geschah auf Grundlage von gemeinsam definierten Anwendungsfällen, welche eine Schnittmenge mit den Anwendungsfällen dieser Projektgruppe ist (siehe hierzu auch 3.1: Use Cases).

Zum Ende dieser Projektphase konnte ein aktualisierter Meilensteinplan erstellt werden. Dieser ist in Tabelle 3: Meilensteinplan, Dezember 2017 dargestellt.

Meilenstein	Datum
Definition Architekturkonzept	20.06.2017
Definition Use Cases PG E-Stram	18.10.2017
Fertigstellung Prototyp	08.12.2017
Präsentation Prototyp	11.12.2017
Umsetzung Use-Cases	31.01.2018
Fertigstellung Dokumentation	31.03.2018

Tabelle 3: Meilensteinplan, Dezember 2017

2.4.3. Dritte Phase - Testen, Korrekturen, Dokumentation

In der letzten Phase des Projektes wurde der Prototyp getestet. Der Test erfolgte anhand strukturierter Komponenten- und Integrationstests. Etwaige Fehler konnten dadurch identifiziert und behoben werden. In einem abschließenden Workshop mit der Projektgruppe E-Stream wurde die Schnittstelle finalisiert und von beiden Projektgruppen abgenommen.

Darüber hinaus stand die Dokumentation der Projektergebnisse im Zentrum der Projektarbeit. Daneben wurden für die ECDA-Konferenz ein Abstract der Projektergebnisse eingereicht und ein Full Paper für die BUIS-Tage 2018, welche von der Abteilung VLBA veranstaltet wird, verfasst.

3. Konzept

Das folgende Kapitel beschreibt das Konzept der Projektgruppe. Zuerst werden in dem Abschnitt 3.1: Use Cases die Anwendungsfälle beschrieben. Danach werden die Anforderungen auf Grundlage der Anwendungsfälle im 3.2: Anforderungen (Technische und Funktionale Anforderungen) definiert. Das Kapitel wird mit der vorläufige Systemarchitektur im Abschnitt 3.3: Systemarchitektur abgeschlossen.

3.1. Use Cases

Bei der Entwicklung der Use Cases lag der Fokus darauf, realistische Anwendungsfälle zu definieren. Aus diesem Grund wurde der Fokus auf die gesamtsystem-notwendigen Funktionen gelegt. Das primäre Ziel der Projektgruppe ist es, die nachfolgenden Use Cases umzusetzen. Die Use Cases konnten zu jederzeit erweitert werden – zusätzliche Funktionalitäten wurden nicht von der Implementierung ausgeschlossen. Es wurden von der Projektgruppe vier Use Cases entworfen.

1. Daten archivieren
2. Anfragenbearbeitung von Daten (spez. COSEM-Format)
3. Modellerzeugung für das Power-Forecasting
4. Modellerzeugung für das Load-Forecasting

Im Folgenden sind die vier Use Cases beschrieben und die dazugehörigen Use Case Diagramme skizziert.

3.1.1. Daten archivieren

Der Use Case soll die Archivierung der Daten realisieren. Es werden heterogene Datenquellen angeschlossen, bspw. Wetterdienste. Alle Datenquellen werden auf die selbe Art und Weise angeschlossen. Der Archivierungsprozess besteht ggfs. aus der Erweiterung der Rohdaten mit zusätzlichen Metadaten für die Datenarchivierung. Die eingehenden Daten werden strukturiert in der Datenbank abgelegt. Des Weiteren werden unterschiedliche Logging-Methoden durchgeführt. Die Daten werden persistent archiviert. Die Rohdaten werden unverändert archiviert, die Metadaten werden lediglich zu den Rohdaten hinzugefügt. Stromerzeuger und Kunden stellen Rohdaten bereit. Diese werden in DAvE archiviert.

Essentielle Schritte:

1. Datenquellen beschreiben

2. Datenströme entgegennehmen
3. Metadaten zu Rohdaten hinzufügen
4. Daten archivieren
5. Logging durchführen

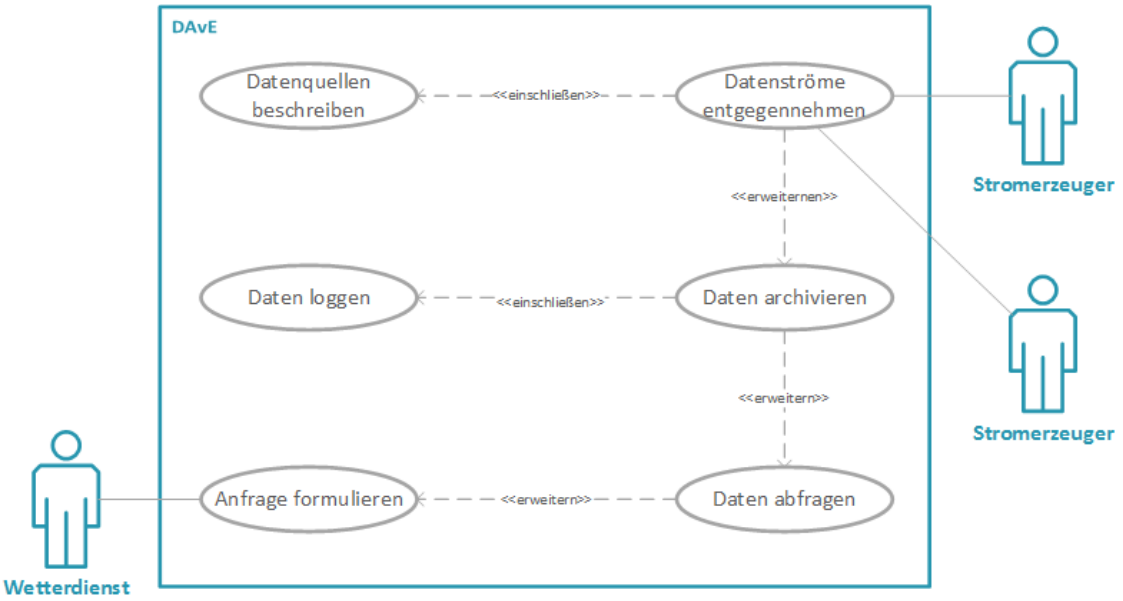

Abbildung 2: Use Case 1: Daten archivieren

3.1.2. Anfragenbearbeitung von Daten

Der Use Case soll die Bearbeitung von Anfragen an das System realisieren. Die Anfragen an das System werden vom System zunächst empfangen. Nachdem eine Anfrage empfangen wurde, wird diese geprüft. Bei einer validen Anfrage wird diese ausgeführt. Es können Datensätze angefragt werden. Die Daten werden je nach Anfrage selektiert und im Anschluss versendet.

Essentielle Schritte:

1. Anfrage überprüfen
2. Anfrage übersetzen
3. Übersetzte Anfrage ausführen
4. Daten versenden

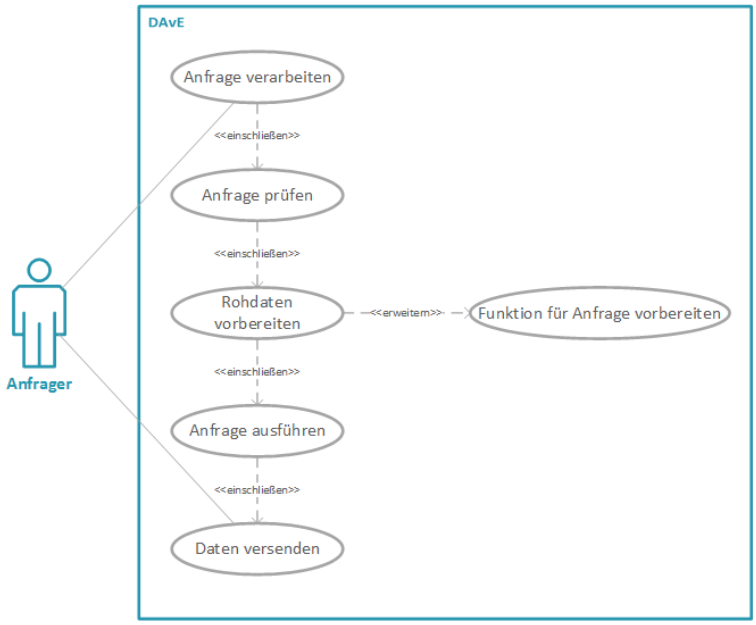

Abbildung 3: Use Case 2: Anfragenbearbeitung von Daten

3.1.3. Modellerzeugung für das Power Forecasting

Der Use Case soll die Erstellung von Power Forecasting Modellen für Windkraft und PV-Anlagen realisieren. Innerhalb eines KDD-Prozesses werden Vorhersagemodelle entwickelt und Data Mining Modelle (DMM) werden innerhalb von DAvE ausgeführt. DMM werden auf selektierte archivierte Datensätze angewendet. Das Modell wird im Anschluss exportiert.

Essentielle Schritte:

1. Auswahl der Datenquelle
2. Datenset erstellen
3. Daten integrieren
4. Daten vorverarbeiten - ETL, Datenset bereinigen
5. Auswahl der DMM
6. Anwendung DMM - Ergebnis ist ein Modell
7. Evaluieren des Modells
8. Versenden des Vorhersagemodells

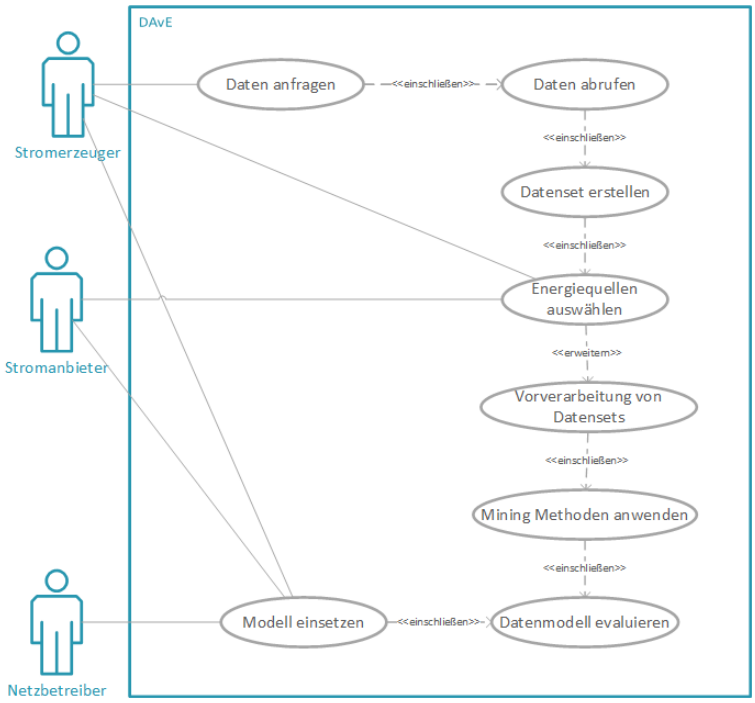

Abbildung 4: Use Case 3: Modellerzeugung für das Power Forecasting

3.1.4. Modellerzeugung für das Load Forecasting

Der Use Case soll die Erstellung von Load Forecasting Modellen realisieren. Innerhalb eines KDD-Prozesses werden Vorhersagemodelle entwickelt und DMM werden innerhalb von DAvE ausgeführt. DMM werden auf selektierte archivierte Datensätze angewendet. Das Modell wird im Anschluss exportiert.

Essentielle Schritte:

1. Datenset erstellen
2. Daten integrieren
3. Daten vorverarbeiten - ETL, Datenset bereinigen
4. Auswahl der DMM
5. Anwendung DMM - Ergebnis ist ein Modell
6. Evaluieren des Modells
7. Versenden des Vorhersagemodells

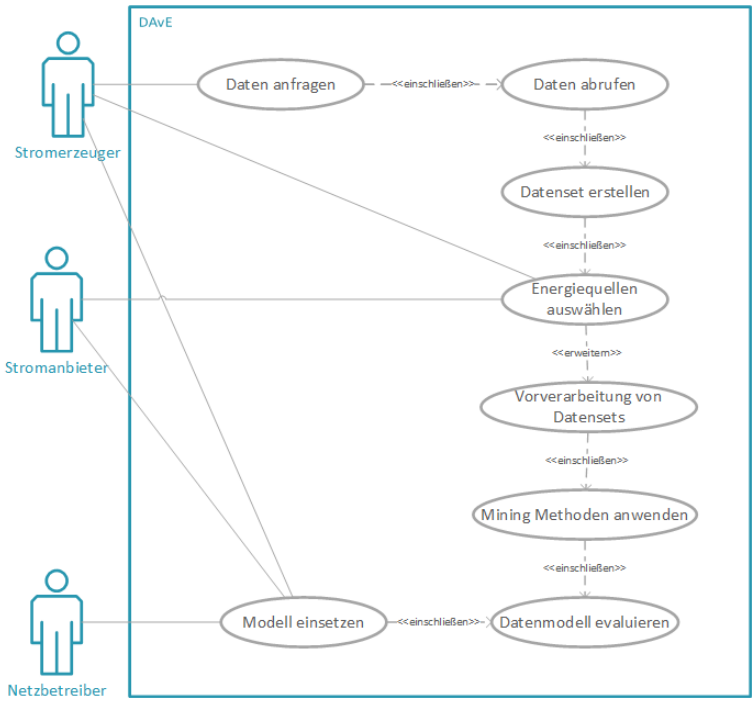

Abbildung 5: Use Case 4: Modellerzeugung für das Load Forecasting

3.2. Anforderungen (Technische und Funktionale Anforderungen)

Zur Umsetzung des Datenarchivs und seiner Komponenten ist es notwendig, dass im Vorfeld genaue Anforderungen an das System definiert werden. Diese beschreiben detailliert, was das System leisten muss. Die Anforderungen zeigen die Kompetenzen des Systems auf.

Es wurden zum Projektbeginn Anforderungen definiert, die während der Entwicklung umzusetzen sind. Diese Anforderungen wurden dabei allerdings im Projektverlauf durch Änderungen der Rahmenbedingungen für das System angepasst. Solche werden in funktionale, nichtfunktionale sowie Anforderungen für das Verhalten im Fehlerfall gegliedert. Ziel ist es keine Lösungsvorgaben zu treffen, sondern lediglich zu definieren, welche Vorgaben durch eine Lösung erfüllt werden müssen. Diese werden in den folgenden Tabellen dargestellt.

3.2.1. Anforderungen Datenanalyse

Das DAvE-System muss die Möglichkeit bieten, anhand der importierten Daten Vorhersagemodelle erstellen zu können. Dabei sollen folgende Modelle unterstützt werden:

- Lineare Regression

- Random Forest
- Support Vector Machine
- Neuronales Netz

Anforderungsnummer	Anforderungen
Funktionale Anforderungen	
DAN-01	Das System muss Algorithmen für Vorhersagemodelle bereitzustellen
DAN-02	Das System muss in der Lage sein, verschiedene Datenquellen anzubinden
DAN-03	Das System muss PMML-Modelle exportieren können
DAN-04	Das System muss PMML-Modelle einlesen können
DAN-05	Das System muss Modelle zum Load-Forecasting erstellen können
DAN-06	Das System muss eine Archivierung der Modelle in Cassandra anbieten
Nichtfunktionale Anforderungen	
DAN-07	Das System soll in der Lage sein, in einer angemessenen Durchlaufzeit Vorhersagemodelle zu erstellen
DAN-08	Der Score der Vorhersagemodelle sollte eine geringe Fehlertoleranz (<20%) aufweisen
DAN-09	Das System sollte nachvollziehbar dokumentiert sein

Tabelle 4: Anforderung Datenanalyse

3.2.2. Anforderungen Datenarchivierung

Das DAvE-System muss die Möglichkeit bieten, Daten verschiedener Datenquellen archivieren zu können. Dabei sollen bspw. Wetter- und Smart-Meter-Daten importiert und archiviert. Die archivierten Daten müssen aus dem Archiv wieder abrufbar sein, sei es als Rohdaten oder als Modelle aus der Analyse dieser Daten.

3.2.3. Anforderungen Datenimport

Das DAvE-System muss die Möglichkeit bieten, folgende Daten in das Archiv importieren zu können:

Das DAvE-System muss die Möglichkeit bieten, folgenden Datenquellen in Apache Cassandra importieren zu können:

Anforderungsnummer	Anforderungen
Funktionale Anforderungen	
DAR-01	Das System muss in der Lage sein, die importierten Daten zu archivieren
DAR-02	Das System muss in der Lage sein, ein Schema zur Datenablage bereitzustellen
DAR-03	Das System muss in der Lage sein, Daten dem Datenbankschema entsprechend abzulegen
DAR-04	Das System muss in der Lage sein, Daten in relevanten Datenformaten speichern zu können
DAR-05	Das System muss Daten in dem Format in dem sie gespeichert wurden, wieder ausgeben können
Nichtfunktionale Anforderungen	
DAR-06	Das System muss in der Lage sein, Daten fehler- und verlustfrei abzulegen
DAR-07	Das System muss jederzeit zur Datenarchivierung verfügbar sein

Tabelle 5: Anforderungen Datenarchivierung

- Wetterdaten (Wind/Sonne)
- Smart-Meter-Daten

Dabei werden die Wetterdaten über Apache NiFi vom FTP-Server des DWD importiert und gemäß dem Datenbakschema in der Cassandra Datenbank archiviert.

Die Smart-Meter-Daten werden vom Mosaik-System bereitgestellt und ebenfalls über Apache NiFi als JSON in der Cassandra Datenbank archiviert.

3.2.4. Anforderungen Schnittstelle

Das DAvE System muss in der Lage sein, Daten über eine Schnittstelle empfangen und bereitstellen zu können. Dazu wird die Möglichkeit geboten, Rohdaten (Smart-Meter-Daten) zu empfangen und diese Daten ebenfalls wieder bereitzustellen. Darüber hinaus können bereits importierte Wetterdaten über die Schnittstelle angefragt werden. Dabei wird ein Kafka Topic von dem Anfragesteller zum Datenabruf gestellt.

3.3. Systemarchitektur

In diesem Abschnitt wird die konzeptionelle Architektur des DAvE Systems dargestellt und erläutert. Der Entwurf der konzeptionellen Architektur ist das Ergebnis der dokumentieren Anforderungen (siehe 3.2: Anforderungen (Technische und Funktionale Anforderungen)) und den entworfenen Use Cases (siehe 3.1: Use Cases).

Um den primären Anwendungsfall, das Archivieren von Smart-Meter-Daten, zu adressieren, ist eine Archivierungsebene innerhalb der Architektur des DAvE-Systems notwendig. Innerhalb dieser sind Datenbanksysteme dafür zuständig, alle zu archivierenden Daten persistent zu speichern. Um den Anforderungen von Big Data gerecht zu werden, ist darauf zu achten dass die Archivierungsebene mit ihren Systemen skalierbar ist um auch riesige Mengen von Daten speichern zu können. Um die archivierten Daten Konsumenten zur Verfügung zu stellen oder auf ihnen Modelle für verschiedene Vorhersagezwecke zu erstellen, ist es ebenfalls essentiell dass die Daten abgefragt werden können.

Als weitere konzeptionelle Schicht in der Architektur ist die Data-Mining-Schicht zu planen. Darin enthaltene Komponenten müssen in der Lage sein Vorhersagemodelle basierend auf den archivierten Smart-Meter-Daten zu generieren. Da das DAvE-System die Generierung dieser Modelle beherrscht, diese jedoch nicht im Kontext von DAvE eingesetzt (deployed) werden sollen, ist bereits in den Anforderungen spezifiziert dass die generierten Modelle mit der standardisierten Datei-Schnittstelle für Data-Mining-Modelle PMML exportiert werden können.

Um externen Konsumenten die Funktionen des DAvE-Systems zur Verfügung zu stellen und um die Daten externer Datenquellen in die Archivierungsebene zu überführen, ist letztlich eine Kommunikationsebene notwendig. Zum einen beinhaltet diese eine Schnittstelle um das Bereitstellen von archivierten Daten und generierten Modellen zu ermöglichen, und zum anderen steuert diese den Datenfluss externer Datenquellen, z. B. Smart-Meter-Daten oder Wetterdaten, in die Archivierungsebene.

Um eine offene Architektur zu schaffen, wird die Auswahl von Technologien welche die konzeptionelle Architektur realisieren, auf standardisierte, freie und quelloffene Technologien, Software und Systeme beschränkt. Die eben beschriebenen Komponenten werden in 6: Übersicht der Komponenten in DAvE abgebildet.

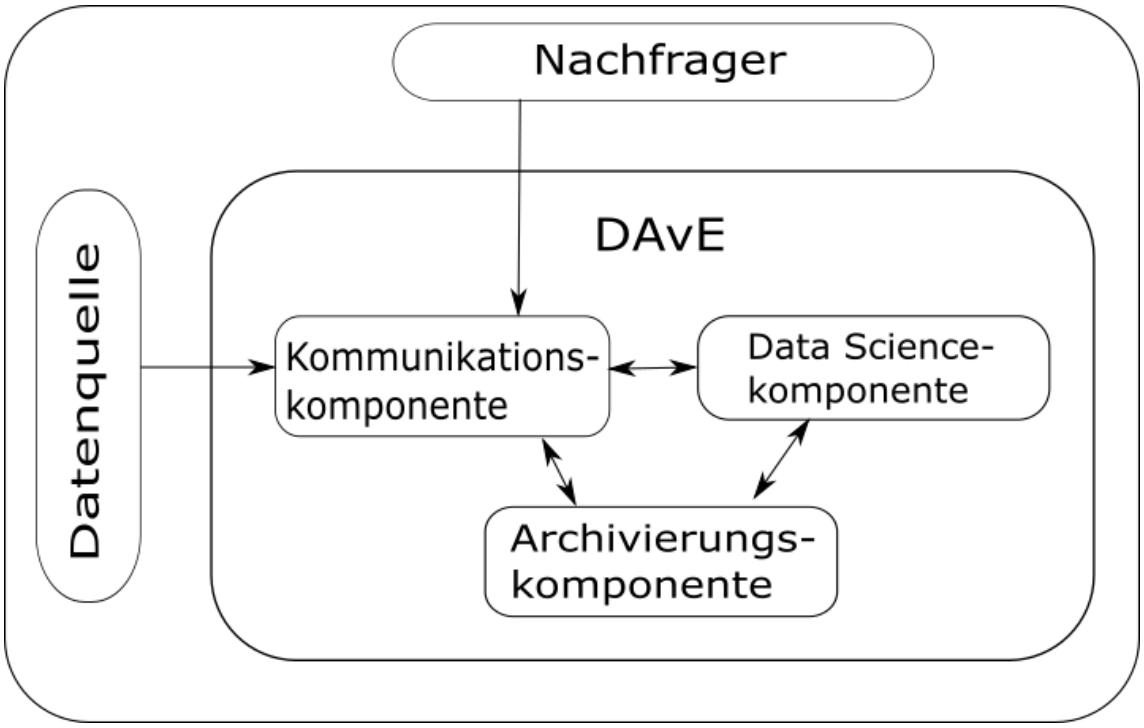

Abbildung 6: Übersicht der Komponenten in DAvE

Anforderungsnummer	Anforderung
	Funktionale Anforderungen
DIM-01	Das System muss tagesaktuelle Wetterdaten (Wind) des DWD automatisch abrufen können
DIM-02	Das System muss tagesaktuelle Wetterdaten (Sonne) des DWD automatisch abrufen können
DIM-03	Das System muss Smart-Meter-Daten importieren
DIM-04	Das System muss die Daten täglich zu einem definierten Zeitpunkt abrufen können
DIM-05	Das System muss die importierten Rohdaten um generierte Metadaten ergänzen
DIM-06	Das System muss die importierten Daten in der Datenbank archivieren
DIM-07	Das System muss die Möglichkeit bieten, die Daten wiederholt und gesteuert zu überprüfen und zu aktualisieren
DIM-08	Das System muss die Möglichkeit beiten, die zu importierenden Daten während des Imports gemäß dem Datenbankschema anzupassen
	Nichtfunktionale Anforderungen
DIM-09	Das System sollte in der Lage sein, den Datenimport zwischen den definierten Importzyklen performant durchführen zu können
DIM-10	Der Datenimport sollte möglichst ressourcenschonend durchgeführt werden
DIM-11	Das System sollte einen vollständigen Import der Daten durchführen können
	Verhalten im Fehlerfall
DIM-12	Das System sollte eine Fehlermeldung ausgeben, falls der Datenimport fehlschlägt
DIM-13	Das System muss bei fehlgeschlagenem Datenimport den Prozess neu anstoßen

Tabelle 6: Anforderungen Datenimport

Anforderungsnummer	Anforderung
	Funktionale Anforderung
DSC-01	Das System muss die Möglichkeit bieten, Smart-Meter-Daten über eine Schnittstelle bereitstellen zu können
DSC-02	Das System muss die Möglichkeit bieten, Smart-Meter-Daten über eine Schnittstelle bereitstellen zu können
DSC-03	Das System muss die Möglichkeit bieten, Wetterdaten über eine Schnittstelle bereitstellen zu können
DSC-04	Das System muss die Möglichkeit bieten, eine zeitliche Eingrenzung für die bereitgestellten Daten vornehmen zu können
DSC-05	Das System muss die Möglichkeit bieten, PMML Modelle über eine Schnittstelle bereitstellen zu können
DSC-06	Das System muss die Möglichkeit bieten, Daten über ein Topic bereitzustellen
DSC-07	Das System muss die Möglichkeit bieten, Daten über ein Topic abzurufen
	Nichtfunktionale Anforderungen
DSC-10	Das System darf nur authentifizierten Nutzern Zugriff auf die Schnittstelle gewährleisten
DSC-11	Die Schnittstelle muss jederzeit von Clients erreichbar sein

Tabelle 7: Anforderungen Schnittstelle

4. Technologieauswahl

In diesem Kapitel wird die durchgeführte Technologieauswahl vorgestellt und begründet. Zunächst erfolgt die Beschreibung des Technologieauswahlprozesses im Bereich der Archivierung für ein Datenbanksystem und ein Dataflow Management-System in den Abschnitten 4.1: Archivierung: Datenbank und 4.2: Archivierung: Dataflow Management. Die Auswahl für Systeme zur Kommunikation wird im Abschnitt 4.3: Messaging System begründet. Im Abschnitt 4.4: Data Science werden ausgewählte Komponenten für den Bereich Data Science im DAvE-System beschrieben. Im letzten Abschnitt 4.5: Data Engineering wird das Kapitel der Technologieauswahl durch die Auswahl im Bereich Data Engineering abgeschlossen.

4.1. Archivierung: Datenbank

Eine entscheidende Komponente für ein Archiv stellt die Datenbank dar, in welcher die Daten persistent gespeichert und bei Bedarf abgerufen werden können. Auf dem Markt existiert eine Vielzahl an Datenbanksystemen, so galt es für die DAvE Architektur ein passendes System zu wählen. Im Folgenden wird das Vorgehen bei der Wahl der Datenbank beschrieben.

Aus der zum Projektbeginn einhergehenden Seminarphase erschloss sich, dass relationale Datenbanken für die Herausforderungen von Big Data nicht geeignet sind. Relationale Datenbanken sind der Vielzahl an Daten und verschiedenen Datenquellen nicht gewachsen. So konnte die verfügbare Auswahl in einem ersten Schritt auf NoSQL-Datenbanken beschränkt werden, da diese für die Herausforderungen im Big Data Kontext ausgelegt sind (vgl. [Har15], S. 30).

DB1 Die Datenbank muss einem nicht relationalen Ansatz verfolgen (NoSQL).

In einem nächsten Schritt wurde ein Überblick über den Markt an NoSQL-Datenbanken geschaffen. Die Auswahl an NoSQL-Datenbanken fällt im Vergleich zur relationalen Datenbanken noch relativ gering aus, da diese einhergehend mit Big Data erst in den letzten zehn Jahren an Bedeutung gewonnen haben (vgl. [Har15], S. 30). In der aktuellen Auswahl an Systemen stellten sich die folgenden Datenbanken als eine der etabliertesten NoSQL-Datenbanksysteme heraus (vgl. [Sol18]):

- Apache Cassandra
- Apache HBase
- MongoDB

Diese drei Systeme sind in der Literatur mit am häufigsten vorzufinden und werden vielfach

miteinander verglichen, daher beschränkte sich die weitere Recherche auf diese Systeme (vgl. [Sol18, OBALB15, MH13]).

Die Datenbanken wurden auf ihre Eigenschaften hin untersucht und es wurde eine Tabelle aufgestellt (siehe A.1: Tabelle Auswahl Datenbanken), welche das technische Datenblatt der Systeme vergleicht. Eine der Angaben in dieser Tabelle ist der Typ des Datenbanksystems.

Der Typ des Systemes stellt ein entscheidendes Kriterium, für die Wahl des Datenbanksystems dar. Dieser Typ gibt an, auf welche Weise und in welchem Format die Daten abgespeichert werden, außerdem werden einige Operationen durch diesen Typ limitiert (vgl. [Sul15], S. 332f.). Apache Cassandra und Apache HBase sind den Collumn Family Databases zugehörig und MongoDB dagegen den Document Databases. Collumn Family Databases eignen sich für Anwendungen, welche jederzeit Daten in die Datenbank schreiben müssen, verteilt sind auf mehrere Datenzentren, und hohe Datenmengen im Bereich von mehreren hundert Terabyte aufkommen lassen können.
todozu langer Satz, kürzer Außerdem wird dieser Typ bereits erfolgreich für Analysen im Aktienmarkt genutzt, also einem Markt bei dem schnell reagiert werden muss und es unter anderem gilt Prognosen aufzustellen, wie es auch bei Energiedaten der Fall ist (vgl. [Sul15], S. 334 f.). Somit entspricht der Typ der Collumn Family Databases dem Anwendungszweck unseres Archives am ehesten.

DB2 Die Datenbank muss dem Typ der Collumn Family Databases zugehörig sein.

Mit Bestimmung des für unseren Anwendungszweck geeigneten Types der Datenbank konnte das System MongoDB ausgeschlossen werden. Die Wahl zwischen Apache Hbase und Apache Cassandra verlief äußerst wechselhaft. Die Datenblätter der beiden Systeme unterscheiden sich nur an wenigen Punkten, dies erschwerte die Wahl und erforderte weitere Aspekte in Betracht zu ziehen. Die DAvE Architektur sollte ursprünglich mit dem Hadoop Eco-System umgesetzt werden, somit war Apache Hbase, welches eine Komponente dieses System ist, eine favorisierte Wahl. Dies sollte allerdings kein primäres Argument für die Wahl des Datenbanksystemes sein.

Um eine Wahl festigen zu können, strebten wir einen Praxisversuch der Systeme an. Dazu nutzten wir Virtuelle Maschinen auf einigen der Rechnern der PG-Mitglieder in Verbindung mit der Software Vagrant (vgl. [Has18]). So konnte die Installation der beiden Datenbanksysteme erprobt werden. Der produktive Praxisbetrieb hingegen erwies sich als nicht durchführbar, da die Leistung der genutzten Rechner nicht ausreichte. Als Anforderung für die Installation der Datenbank ergab sich zunächst, dass die Installation erfolgreich erprobt werden konnte. Somit konnten wir sicherstellen, dass die Installation auf dem Server möglich sein dürfte.

DB3 Die Installation der Datenbank muss auf den Testsystemen der PG-Nutzer erfolgreich durchgeführt werden können.

Des Weiteren betrachteten wir für die Wahl das CAP Theorem. Das CAP Theorem besagt, dass eine verteilte Datenbank zur selben Zeit nur zwei der folgenden Eigenschaften abdecken kann:

- **Consistency:** Die Consistency beschreibt die Eigenschaft, dass jeder Nutzer der Datenbank in jeder Instanz (Kopien der Daten auf unterschiedlichen Server-Clustern) eine identische Ansicht der Daten hat.
- **Availability:** Die Availability beschreibt die Eigenschaft, dass bei dem Ereignis eines Fehlers, die Datenbank operational verfügbar bleibt. Alle Anfragen werden beantwortet.
- **Partition tolerance:** Die Partition tolerance beschreibt die Eigenschaft, dass die Datenbank auch bei Netzwerkfehlern, einzelner Netzknoten oder Partition des Netzes weiter arbeitet.

Demnach muss sich bei der Wahl eines verteilten Datenbanksystems für zwei dieser Eigenschaften entschieden bzw. priorisiert werden (vgl. [Har15], S. 43 f.). Wir favorisieren hierbei die Verfügbarkeit gegenüber der Konsistenz der Daten und somit die CAP Theorem Kombination AP.

DB4 Die Datenbank muss nach dem CAP Theorem eine Kombination aus *Availability* und *Partition tolerance* bieten.

4.1.1. Anforderungen

In diesem Abschnitt werden die Anforderungen, welche in Abschnitt 4.1: Archivierung: Datenbank aufgestellt wurden zusammenfassend aufgelistet, damit sie im Abschnitt 4.1.2: Beschreibung für den Vergleich der Datenbank und die Auswahl hinzugezogen werden können.

DB1 Die Datenbank muss einem nicht relationalen Ansatz verfolgen (NoSQL).

DB2 Die Datenbank muss dem Typ der Collumn Family Databases zugehörig sein.

DB3 Die Installation der Datenbank muss auf den Testsystemen der PG-Nutzer erfolgreich durchgeführt werden können.

DB4 Die Datenbank muss nach dem CAP Theorem eine Kombination aus *Availability* und *Partition tolerance* bieten.

4.1.2. Beschreibung

In diesem Abschnitt werden die, aus der in Abschnitt 4.1: Archivierung: Datenbank angesprochenen Marktanalyse resultierenden Datenbanksysteme Apache Hbase und Apache Cassandra in Kurzform beschrieben und auf die Anforderungen in Abschnitt 4.1.1: Anforderungen untersucht. Die Datenbank MongoDB wurde in Abschnitt 4.1: Archivierung: Datenbank bereits aus der Auswahl ausgeschlossen.

Apache Hbase Bei Apache Hbase handelt es sich um die verteilte, skalierbare NoSQL Datenbank des Apache Hadoop Eco-Systems und dient hierbei als Big Data Speicher. Diese Datenbank eignet sich nach Herstellerangaben, wenn ein stichprobenartiger Echtzeitzugriff auf die Massendaten notwendig erscheint. Ziel dieses Open Source Projektes ist es sehr große Tabellen zu hosten, was Millionen von Reihen und Spalten beinhaltet. Apache HBase ist nach dem Vorbild von Google's Bigtable entstanden. Statt wie Google's Bigtable auf das Google File System zu setzen, setzt Apache HBase auf Hadoop und HDFS auf. Einige Funktionen dieser Datenbank sind (vgl. [The18a]):

- Lineare und modulare Skalierbarkeit
- Konsistente Schreib- und Lesezugriffe
- Automatisch und konfigurierbares sharding von Tabellen
- Eine einfach zu nutzende Java API für den Client-Zugriff

Weitere Information zum Hadoop System sind der Seminararbeit Hadoop Ecosystem Components: Ambari, HDFS and MapReduce (siehe Anhang) zu entnehmen.

Apache Hbase ist somit ganz nach Anforderung **DB1**: Anforderungen eine NoSQL Datenbank, außerdem zählt sie zu den Column Family Databases (s. Anforderung **DB2**: Anforderungen)(vgl. [Sul15], S. 334 f.). Die Installation dieser Datenbank ist relativ aufwendig, da sie eine Komponente des Hadoop Ecosystems darstellt und somit ist sie abhängig von dessen Komponenten und setzt diese voraus. Die Installation verlief erfolgreich mithilfe von Apache Ambari (Abschnitt xyz) (s.Anforderung **DB3**: Anforderungen). Hbase deckt nach dem CAP Theorem die Eigenschaften Consistency und Partition tolerance ab, dies widerspricht der Anforderung **DB4**: Anforderungen (vgl. [Sah15]).

Apache Cassandra Die Apache Cassandra Datenbank ist eine skalierbare NoSQL Datenbank mit hoher Avaibility ohne dabei auf Performance verzichten zu müssen. Lineare Skalierbarkeit und scalability sowie bewiesene Fehlertoleranz verhelfen kritische Daten sicher zu speichern. Apache Cassandra ermöglicht Replikationen der Daten zwischen verschiedenen Datenzentren und gehört in diesem Gebiet laut Herstellerangaben zu den Besten. Diese Fähigkeit verschafft den Nutzern geringere Latenzzeiten und löst das Problem regionaler Grenzen. Die Eigenschaften mit denen der Hersteller sich auszeichnet sind (vgl. [The16]):

- Fehlertoleranz, die Daten werden automatisch auf verschiedene Nodes repliziert
- Performanz, so schlägt Apache Cassandra die Konkurrenz mehrfach in Performance Vergleichen
- Dezentralisierung, es gibt keinen Single Point of Failure und keine Netzwerk Flaschenhalse, da jeder Node im Cluster identisch ist

Apache Cassandra ist ganz nach Anforderung **DB1**: Anforderungen eine NoSQL Datenbank, außerdem zählt sie zu den Collumn Family Databases (s. Anforderung **DB2**: Anforderungen) (vgl. [Sul15], S. 334 f.). Die Installation dieser Datenbank ist im Vergleich zur Installation von Apache Hbase unkompliziert, da keine zusätzlichen Komponenten vorausgesetzt werden. Die Installation verlief erfolgreich, somit wird die Anforderung **DB3**: Anforderungen erfüllt. Apache Cassandra deckt nach dem CAP Theorem die Eigenschaften Availability und Partition tolerance ab und erfüllt damit die Anforderung **DB4**: Anforderungen (vgl. [Sah15]).

Ergebnis Nach den aufgestellten Anforderungen setzt sich in diesem Vergleich Apache Cassandra durch. Die Installation erwies sich als simpel und setzte keine weiteren Komponenten voraus, womit sich diese Datenbank im Vergleich zu Apache Hbase um einiges flexibler erweist. Außerdem erfüllt Apache Cassandra die Anforderung **DB4**: Anforderungen, nach dem CAP Theorem die Eigenschaften Availability und Partition tolerance zu vereinen. Weiterhin bietet Apache Cassandra die Konfigurationsmöglichkeit für jede Tabelle ein so genanntes consistency level zu konfigurieren, womit je Tabelle eingestellt werden kann, ob mehr zur Availability oder zur Consistency tendiert wird (vgl. [Dat18]). Dies macht Apache Cassandra dahingehend flexibel, ob die Präferenzen nach dem CAP Theorem eher zur Availability oder zur Consistency tendieren.

4.2. Archivierung: Dataflow Management

Für ein Archiv stellt neben dem Sichern von Daten, das Abrufen der gesicherten Daten eine bedeutende Funktion dar. Nach dem Abruf der Daten innerhalb der DAvE-Architektur steht die Übermittlung dieser Daten an die Nutzer aus. Für diesen Zweck wurde der Message Broker Apache Kafka gewählt, mit jenem Datenströme verwaltet werden können und so das Abrufen und Versenden von Daten ermöglicht wird (Querverweis Kapitel Kafka, Usecases).

Um Daten aus der Datenbank an Apache Kafka zu übermitteln galt es, die Datenbank (in diesem Fall Apache Cassandra) mit Apache Kafka zu verbinden und den Datenfluss zwischen diesen Komponenten zu managen. Es wurde eine Recherche eingeleitet, um diesem Zweck dienliche, mögliche Technologien ausfindig zu machen. Als Ergebnis dieser Recherche ergaben sich folgende Kandidaten:

- Kafka Connect
- Apache NiFi

Unseren Use Cases entsprechend muss es ermöglicht werden, Daten aus der Datenbank in Apache Kafka zu übermitteln.

DFM1 Die Dataflow Management Lösung muss den Datenfluss von der Datenbank, hin zu Apache Kafka ermöglichen.

Weiterhin sollten Daten aus Apache Kafka in die Datenbank übertragen werden können.

DFM2 Die Dataflow Management Lösung muss den Datenfluss von Apache Kafka, zu der Datenbank ermöglichen.

Die Übertragung sollte möglichst automatisiert stattfinden können.

DFM3 Die Dataflow Management Lösung sollte den Datenfluss automatisiert managen können.

Um eine Wahl festigen zu können, wurden die Testkandidaten auf virtuellen Maschinen auf einigen der Rechnern der PG-Mitglieder installiert. Im Anschluss wurden diese Lösungen erprobt. Als Anforderung für die Installation der Datenbank ergab sich zunächst, dass die Installation erfolgreich erprobt werden konnte. Somit konnten wir sicherstellen, dass die Installation auf dem Server möglich sein dürfte.

DFM4 Die Installation der Dataflow Management Lösung muss auf den Testsystemen der PG-Nutzer erfolgreich durchgeführt werden können.

Eine weitere Anforderung ist, dass die Nutzung der Technologie für die angedachten Zwecke (Verbindung von Apache Kafka und Cassandra) erfolgreich auf den Testsystemen durchgeführt werden kann. Auf diese Weise kann ein erfolgreicher Betrieb gewährleistet werden. Außerdem kann so sichergestellt werden, dass die gewählte Lösung den nötigen Funktionsumfang abdeckt.

DFM5 Die Dataflow Management Lösung muss in einem Praxistest erfolgreich den Datenstrom zwischen Apache Kafka und Cassandra managen.

4.2.1. Anforderungen

In diesem Abschnitt werden die Anforderungen, welche in Abschnitt 4.1: Archivierung: Datenbank aufgestellt wurden zusammenfassend aufgelistet, damit diese im Abschnitt 4.1.2:

Beschreibung für den Vergleich der Datenbank und die Auswahl hinzugezogen werden können.

DFM1 Die Dataflow Management Lösung muss den Datenfluss von der Datenbank, hin zu Apache Kafka ermöglichen.

DFM2 Die Dataflow Management Lösung muss den Datenfluss von Apache Kafka, zu der Datenbank ermöglichen.

DFM3 Die Dataflow Management Lösung sollte den Datenfluss automatisiert managen können.

DFM4 Die Installation der Dataflow Management Lösung muss auf den Testsystemen der PG-Nutzer erfolgreich durchgeführt werden können.

DFM5 Die Dataflow Management Lösung muss in einem Praxistest erfolgreich den Datenstrom zwischen Apache Kafka und Cassandra (Datenbank) managen.

4.2.2. Beschreibung

In diesem Abschnitt werden die aus der in Abschnitt 4.2: Archivierung: Dataflow Management angesprochenen Recherche resultierenden Dataflow Management Lösungen Kafka Connect und Apache NiFi, in Kurzform beschrieben und auf die Anforderungen in Abschnitt 4.2.1: Anforderungen untersucht.

Kafka Connect Kafka Connect ist ein Tool, dass Daten zwischen Apache Kafka und anderen Systemen zuverlässig sowie skaliert überträgt. Es ermöglicht auf einfache und schnelle Weise Konnektoren zu definieren, welche Datenmengen in und aus Apache Kafka bewegen. Kafka Connect kann komplette Datenbanken oder Metriken von den Applikationen eines Servers, in einen Kafka Topic aufnehmen und die Daten für das Stream Processing verwenden (vgl. [The18b]).

Kafka Connect enthält folgende Funktionen (vgl. [The18b]):

- Ein Framework für Kafka Konnektoren - Kafka Connect ermöglicht eine standardisierte Integration von anderen Dateisystemen mit Apache Kafka
- REST-Schnittstelle - Management von Konnektoren über eine einfach zu bedienende REST-Schnittstelle
- Streaming/batch Integration - Kafka Connect nutzt die vorhandenen Fähigkeiten von Apache Kafka und ist damit eine ideale Lösung für die Überbrückung von Streaming- und Batch-Datensystemen

Kafka Connect erfüllt die zuvor aufgestellten funktionalen Anforderungen. Die Installation

und Bedienung erwies sich allerdings als problematisch. Die Dokumentation und einschlägigen Tutorials sind mangelhaft und teils veraltet. Während der Installation und Bedienung traten vermehrt Fehlermeldungen und Probleme auf, sodass die praktische Nutzung sich als nicht durchführbar erwies.

Apache NiFi Apache NiFi unterstützt das Nutzen skalierbarer gerichteter Graphen für den Transport und das Transformieren von Daten. Dies wurde entwickelt, um Datenflüsse zwischen verschiedenen Systemen zu automatisieren. Datenfluss ist in diesem Kontext der automatisierte und gesteuerte Fluss von Informationen zwischen Systemen. Solch eine Lösung erweist sich als notwendig, seit dem Unternehmen mehr als ein System betreiben. Die Systeme erstellen eigens Daten und konsumieren Daten anderer Systeme. Apache NiFi ermöglicht diesen Datentransport zu konfigurieren und zu steuern (vgl. [The18c]).

Einige der Features sind (vgl. [The18c]:

- Ein Web-based User Interface
- Hochgradig Konfigurierbar
- Datenflüsse können zur Laufzeit modifiziert werden
- Das Erstellen eigener Prozessoren ist möglich

Apache NiFi erfüllt die funktionalen Anforderungen, welche an eine Dataflow Management Lösung gesetzt wurden. Die Installation erwies sich als unkompliziert, von der Herstellerseite wird ein Paket heruntergeladen, entpackt und der NiFi Prozess muss dann nur noch gestartet werden. Die Bedienung wird durch das Web-Interface erleichtert. Apache Kafka mit einer Datenbank zu verbinden erfolgt problemlos. Apache NiFi ermöglicht es einem Dataflows zu erstellen, in welchen Prozessoren mit gerichteten Graphen miteinander verbunden werden. Die Prozessoren ermöglichen Funktionalitäten, wie das Empfangen, Manipulieren und Senden von *FlowFiles*. FlowFiles beinhalten einen Datensatz sowie zugehörige Attribute. So gibt es Eingangsprozessoren, wie z. B. *QueryCassandra* mit dem Daten aus Cassandra per CQL-Abfrage entnommen werden können, mit weiteren Prozessoren werden diese Daten manipuliert und letzten Endes z. B. mit einem Ausgangsprozessor in einer einem Kafka Topic hinterlegt.

Ein Beispiel für solch einen Dataflow ist in Abbildung 7: NiFi Dataflow Beispiel zu sehen.

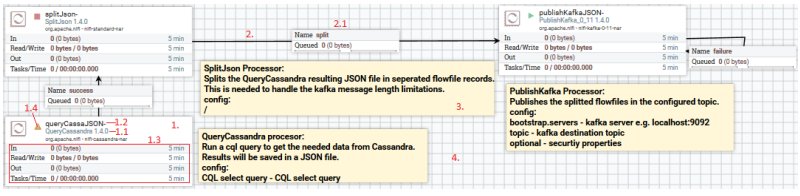

Abbildung 7: NiFi Dataflow Beispiel

Die einzelnen Komponenten des dargestellten Dataflows sollen im Folgenden beschrieben werden, hierzu wird auf die in roter Schrift markierten Punkte *1.* bis *4.* eingegangen.

1. Prozessoren: Dies sind die NiFi Komponenten welche genutzt werden, um eintreffende Daten abzufangen, Daten aus externen Quellen zu beziehen und Daten an externe Quellen zu übermitteln. Weiterhin werden Prozessoren genutzt um Informationen aus einzelnen Daten weiterzuleiten, zu transformieren oder zu extrahieren (vgl. [The18c]). Prozessoren bzw. auch ganze Dataflows können zur weiteren Strukturierung in einer Prozessorgruppe gruppiert werden.
   a) **Prozessortyp:** Bezeichnet den verwendeten Prozessor (die Prozessorklasse).
   b) **Prozessorname:** Ein individuell anpassbarer Prozessorname.
   c) **Prozessormonitoring:** Hier werden unter anderem eintreffende Daten (***In***) und ausgehende Daten (*Out*) erfasst.
   d) **Prozessorstatus:** Beschreibt den Status des Prozessors, es wird unterschieden zwischen *Probleme liegen vor* dargestellt mit einem Warndreieck, *Gestartet* symbolisiert durch eine grüne Pfeilspitze und dem Status *Gestoppt*, welcher mittels eines roten Quadrates repräsentiert.
2. **Konnektor:** Sobald Prozessoren und andere Komponenten dem NiFi Canvas (Arbeitsfläche) hinzugefügt und konfiguriert wurden, ist der nächste Schritt die Komponenten miteinander zu verbinden, um den Datenfluss zu definieren ([The18c]). Beim Verknüpfen der Komponenten wird eine *Beziehung* gewählt.
   a) **Konnektor-Beziehung:** Die Konnektor-Beziehung gibt ähnlich wie in UML Klassendiagrammen an, in welcher Beziehung die einzelnen Komponenten stehen. So kann z. B. der weitere Fluss der Daten, welche erfolgreich (succes) durch einen Prozessor bearbeitet wurden bestimmt werden und wie die Daten behandelt werden sollen, bei denen Fehler vorkamen.
3. **Label:** Diese Komponente wird genutzt um einzelne Teile eines Dataflows zu dokumentieren (vgl. [The18c]).
4. **Canvas:** Hierbei handelt es sich um die Arbeitsfläche, auf der Komponenten in Apache NiFi platziert werden.

Solche Dataflows können in einem *Template* gesichert und als Vorlage weitergenutzt werden. Diese Templates können auch im XML-Format heruntergeladen, weitergegeben und in Apache NiFi hochgeladen werden.

Der Dialog für die Konfiguration der Prozessoren wird in der folgenden Abbildung 8: NiFi Prozessor Konfiguration Beispiel veranschaulicht. Hier können die einzelnen *Eigenschaften* (*properties*) eingestellt werden.

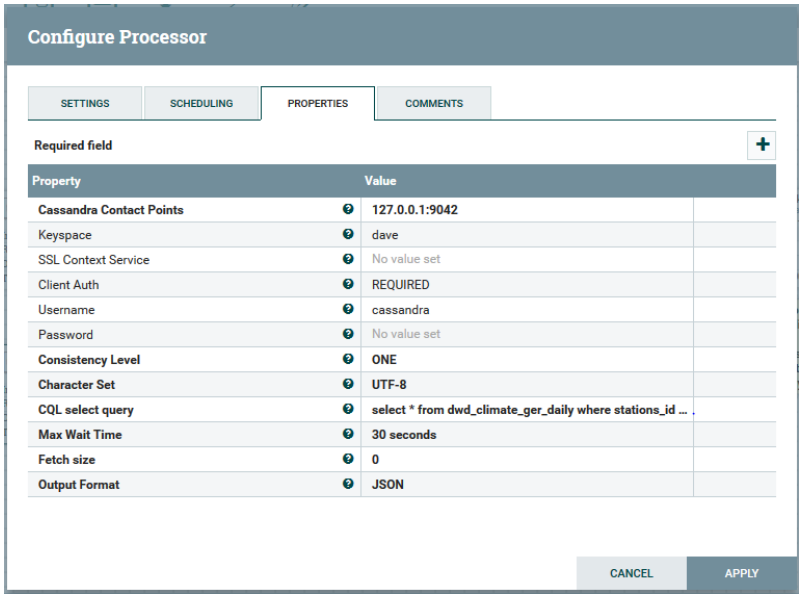

Abbildung 8: NiFi Prozessor Konfiguration Beispiel

Ergebnis Kafka Connect ist durch die schlechte Handhabung mit häufigen Fehlermeldungen ausgeschieden, das verbinden von Apache Kafka mit anderen Systemen erwies sich in der Praxis, nicht als problemlos durchführbar. Apache NiFi erfüllt den angestrebten Zweck und durch das Web-Interface ist die Bedienung leichter. Apache NiFi bietet eine Vielzahl von Prozessoren, welche den Umgang mit den verschiedensten Systemen ermöglichen. Weiterhin können bei Bedarf auch eigene Prozessoren erstellt werden. Durch die Vielzahl an Prozessoren kann Apache NiFi nicht nur die angestrebte Verbindung von Apache Kafka mit einer Datenbank ermöglichen, es kann auch für andere Zwecke genutzt werden. Es können z. B. Daten von FTP-Servern automatisiert heruntergeladen und direkt in einer Datenbank abgelegt werden. So kann Apache NiFi in DAvE dazu genutzt werden, um das Archiv mit Wetterdaten zu erweitern. Dieses Werkzeug erweist sich als vielseitig einsetzbar und mächtig im Umgang mit Daten.

4.3. Messaging System

In diesem Abschnitt werden die Anforderungen für die Datenbereitstellung hergeleitet und benannt. Im Anschluss werden die zwei ausgewählten Messaging Systeme vorgestellt und erläutert. Der Abschnitt wird durch eine kurze Zusammenfassung mit einem Abgleich der Anforderung abgeschlossen.

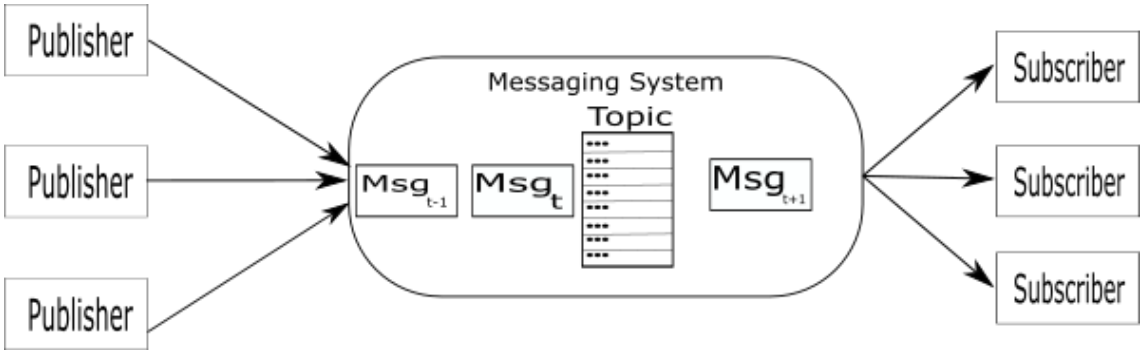

Abbildung 9: Allgemeine Funktionsweise von Messaging Systemen

4.3.1. Anforderung

Die Teilanforderungen für die Kommunikation und Transport der Daten, konnten auf Grund momentanen Stillstand nur schwer definiert werden. Anhand der Use Case *Archivierung von Rohdaten aus dem Niederspannungsnetz* und der voraussichtlichen gesetzlichen sowie normierten Rahmenbedingungen wurden die anfallenden Datenmengen abgeschätzt. Eine Herausforderung ist die Bewältigung der kontinuierlich-ankommenden Daten aus dem Niederspannungsnetz [Bun16b]. Um den Datenstrom von dem System zu Entkoppeln wurde entschieden eine Messaging System zu nutzen. Die grundlegende Funktionsweise eines Messaging System wird in der Abbildung 9: Allgemeine Funktionsweise von Messaging Systemen dargestellt.

Nach einer Recherche über das Format, den Inhalt und der Lieferung der Smart-Meter-Daten aus dem Niederspannungsnetz mussten noch die Randbedingungen definiert werden. Im Gesetz zur Digitalisierung von Energiedaten wurde festgelegt, dass in periodischen Zeitabständen die Informationen versandt werden sollen und dies mehrmals am Tag, der momentane Zyklus liegt bei 15 Minuten. Somit ergab sich, dass es entweder einen kontinuierlichen Datenstrom oder eine Bulk Lieferung von Daten erwartet werden konnte.

Der genaue Umfang und die Größe der erwarteten Smart-Meter-Daten steht momentan noch aus. Mit dieser Ungewissheit und der vermuteten Lieferung der Smart-Meter-Daten, wurde entschieden, dass ein Werkzeug genutzt werden muss, um die Annahme der Smart-Meter-Daten vom Rest des System zu entkoppeln. Wie bereits erwähnt ist eine Leitlinie in diesem Porjekt, ein Open-Source System zu entwickeln. Daher beschränkte sich die Suche zuallererst auf Werkzeuge mit einer Open-Source Lizenz.

Damit ergab sich zum einen die Anforderung eines hohen Durchsatzes und die Allgemeine Anforderung nach einer Open-Source Software.

Anhand der vorliegenden Informationen, wurden erste Anforderungen formuliert. Zur Auswahl standen unterschiedliche Messaging System. Im erste Schritt wurden die Messaging

Systeme genauer auf die Merkmale, Systemanforderungen, Vor- und Nachteile untersucht. Dazu wurde eine Literatur-Recherche durchgeführt. Innerhalb der Literatur-Recherche wurde sowohl die Dokumentation der Werkzeuge also auch nach wissenschaftlichen Bewertungen gesucht.

Unter anderem wurden die Arbeiten von [Ion15] und [JL17] angesehen. In der Arbeit von [Ion15] wurde RabbitMQ mit ActiveMQ verglichen. Dabei wurde festgestellt, dass RabbitMQ schneller Daten an Clients sendet als ActiveMQ, obwohl im selben Test ActiveMQ die Daten schneller empfangen kann.[Ion15] Die Arbeit von [JL17] verglich unterschiedliche Message Broker und Message Queues miteinander. Untersuchungsgegenstand war Apache Kafka und AMQP (advanced message queue protocol). Das Ziel war den Durchsatz und die Zuverlässigkeit zu überprüfen. Daraus resultierte, das Apache Kafka im Vergleich mit AMQP einen besseren Durchsatz hat. Liegt allerdings der Fokus auf einer hohen Zuverlässigkeit so ist Apache Kafka ungeeignet. Mit diesen Informationen haben wir die Auswahl auf zwei Message Broker reduziert. Zur engeren Auswahl standen Apache Kafka und RabbitMQ.

Im zweiten Schritt wurden die Messaging Systeme auf ihre Tauglichkeit untersucht. Dazu wurde ein Testsystem vorbereitet und die Messaging System prototypisch eingerichtet. Die Untersuchung umfasste zwei Etappen. Zuerst wurde ein Single-Node Cluster konfiguriert. In der nächsten Konfiguration wurde das Cluster um zwei Nodes erweitert. Das Testsystem hatte die selben Konfiguration, wie die zu diesem Zeitpunkt geplante Gesamtarchitektur. Das primäre Ziel war hierbei, die Einrichtung und der Betrieb der beiden Messaging Systeme. Die Installation und Konfiguration des Cluster verlief ohne große Schwierigkeiten. Auch bei dem Betrieb wurden zwischen der Single-Node Konfiguration und der Multicluster-Konfiguration keine Unterschiede festgestellt. Es wurden erste kleine Programme geschrieben, um das Erzeugen und konsumieren von Daten zu testen.

Während der ersten Sondierungen konnten noch foglende Anforderungen festgestellt werden:

- Das Messaging Systeme muss zuverlässig arbeiten. D. h. es dürfen keine Nachrichten verloren gehen. Dazu zählt auch bei einem unerwartet Ausfall der Server.
- Das Messaging Systeme muss zuverlässig arbeiten. D. h. es dürfen keine Nachrichten verloren gehen. Dazu zählt auch bei einem unerwartet Ausfall der Server.
- Das Messaging System muss skalierbar sein. Es müssen weiter Knoten einem Cluster während des Betriebs hinzufügbar sein.
- Die Nachrichten sollten nicht während der Übertagung zerlegt werden.

4.3.2. Beschreibung

In diesem Abschnitt werden die beiden Streaming Plattformen Apache Kafka und RabbitMQ vorgestellt. Dabei wird zuerst auf die allgemeine Architektur eingegangen und im Anschluss auf die Arbeitsweise.

Kafka Architektur Apache Kafka ist eine verteilte Streaming Plattform. Dabei werden Nachrichten (Messages) durch sog. Abonnenten (Subscriber) konsumiert und durch sog. Produzenten (Producer) erzeugt. Die erzeugten Daten werden auf einem Server gespeichert. Die Messages können in Apache Kafka sog. Topics kategorisiert werden.

Eine Kafka Topic besteht aus mehreren Bestandteilen. Ein Producer kann auf diversen Partitionen Messages senden. Jede Partition wird protokolliert. Eine Message innerhalb eines Topics besteht aus der Nachricht und einem offset. Ein offset ist ein einzigartige Identifikation innerhalb einer Partition. Der offset kann dazu genutzt werden einen Startpunkt beim abonnieren zu setzen.

Des Weiteren bietet Apache Kafka diverse andere Vorteile: eine hohe Skalierbarkeit, Persistenz und Zuverlässigkeit. Weiterhin können Empfangsbestätigungen von Apache Kafka bereitgestellt werden, welche durch einen Parameter aktiviert werden. Apache Kafka schreibt die Messages auf die Festplate und repliziert diese, daraus erfolgt eine gute Fehlertoleranz. Außerdem wird eine Empfangsbestätigung an den Producer gesendet, solange hält der Producer die Message vor. Es können weitere Knoten während des Betriebs hinzugefügt werden. Zusätzlich können Multi-Broker auf einem Single-Node-Cluster betrieben werden. Ein Broker ist der Kafka Service der über einen Port angesprochen werden kann. Die Replikation bzw. das Loggen auf der Festplatte führt zu einer Dauerhaftigkeit der Messages. Die Abbildung 10: Ein Schema für ein Kafka Architektur zeigt schematisch die Architektur eines Kafka Cluster. Es zeigt mehrere Producer und Consumer sowie die Kafka Umgebung. Ein Kafka Cluster besteht aus mehreren Brokern, welche durch ein Zookeeper Service verwaltet werden vgl. [Tut18] und [The18b].

RabbitMQ Architektur RabbitMQ zählt ebenfalls zu den Messaging Systemen. Die Arbeitsweise entspricht dabei mehr der Message Queue (MQ). Eine Queue ist eine Sammlung von Messages. Die Queue hat einen eindeutigen systemweiten Namen und die Messages werden in RabbitMQ nach First in First out (FIFO) sortiert. Für eine Queue können diverse Einstellungen gesetzt werden, bspw. aktiviert *Auto-delete* das Löschen einer Queue nachdem der letzte Consumer sich abgemeldet hat. Die Messages werden persistent auf der Festplatte gesichert. Daher existieren nicht nur die Queues, sondern auch die Messages nach einem Neustart der Broker.

Ein Broker ist hier ebenfalls ein Node der die Prozesse verwaltet und verarbeitet. In einem Cluster können zwei Typen von Broker eingestellt werden. So kann ein Broker als RAM

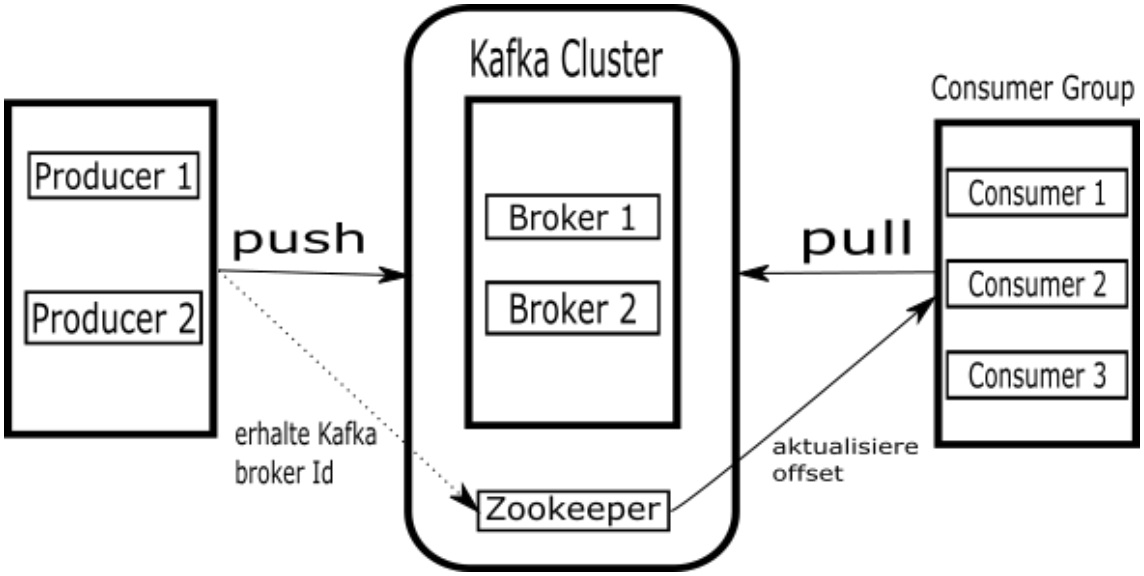

Abbildung 10: Ein Schema für ein Kafka Architektur (vgl.[Tut18])

Node fungieren. Die RAM Nodes speichern die Nachrichten ausschließlich im RAM. Wohingegen eine Disk Node auch die Messages auf der Festplatte speichert. Die Verwaltung eines Clusters wird über das Advanced Message Queuing Protocol gesteuert.

Das AMQP ist in Erlang entwickelt worden. Erlang ist eine Programmiersprache. AMQP ist ein open standard für die Übermittlung von Informationen von und zu Organisationen und Anwendungen. RabbitMQ nutzt das AMQP, um die einzelnen Nodes innerhablb eines Clusters zu verbinden.

RabbitMQ besteht aus mehreren miteinander interagierender Komponenten. Ein Producer sendet eine Message an den Broker. Die Message wird an die Queue weitergeleitet und kann dann von einem Consumer abonniert werden. Die Weiterleitung einer Message zu einer Queue kann unterschiedlich eingestellt werden. Die Message kann direkt auf eine Queue weitergeleitet werden. Messages können auf eine Anzahl unterschiedliche Queues verteilt werden oder Anhand eines Muster auf unterschiedliche Queues sortiert werden.

Ergebnis Die Einrichtung sowie die Wartung von Apache Kafka ist relativ unkompliziert. Weiterhin kann Kafka durch diverse Erneuerungen sehr zuverlässig arbeiten und bietet eine hohe Fehlertoleranz. Das Hinzufügen von weiteren Knoten ist ebenfalls sehr einfach, da die Verwaltung nicht von Kafka sondern von Zookeeper übernommen wird.

Bei RabbitMQ konnten dieselben Eigenschaften festgestellt werden. Es ist ein verteiltes Messaging System und höchst zuverlässig aufgrund einer Empfangsbestätigung. Dieses Vorgehen verursacht aber ein geringen Durchsatz. Die Einrichtung ist ein wenig aufwen-

diger als bei Kafka. Die Skalierbarkeit wird durch das AMQP sichergestellt. Die Wartung kann über eine *Command Line Interface* (CLI) durchgeführt werden. Es zeigte sich, dass keine Entscheidung getroffen werden kann, ohne die Messaging Systeme im Prototypen zu testen. Daher empfiehlt es sich beide Messaging Systeme in der Gesamtarchitektur zu implementieren. Für die Kooperation mit der Projektgruppe E-Stream wurde Apache Kafka eine höhere Priorität zugewiesen als RabbitMQ.

4.4. Data Science

Im folgenden werden die Anforderungen an die Data Science Komponente hergeleitet und betrachtete Frameworks beschrieben und evaluiert.

4.4.1. Anforderungen

Für die Auswahl des Data Science Frameworks wurden sieben Anforderungen formuliert, welche im Folgenden näher erläutert werden und in Tabelle 8: Anforderungen Data Science Framework nochmals zusammengefasst:

Hadoop/Cassandra Grundlage der Vorhersagemodelle sind die, in der Datenbank gespeicherten, historischen Daten. Daher ist es notwendig, dass eine Schnittstelle zur verwendeten Datenbank bereit gestellt wird. Zuerst sollte das Framework eine Schnittstelle zu `Apache Hadoop` unterstützen. Allerdings hat sich die Projektgruppe im späteren Verlauf auf `Apache Cassandra` geeinigt, sodass eine Unterstützung von `Hadoop` nicht mehr notwendig war.
Wichtigkeit: hoch

Programmiersprache Bei dieser Anforderung soll das Framework wenn möglich die Programmiersprache Python unterstützen. Einige Gründe, warum Python zur Datenanalyse verwendet werden sollte, sind im gleichnamigen Abschnitt erläutert. Darüber hinaus wird Python von vielen Machine Learning Frameworks unterstützt, was die Möglichkeiten zur Datenanalyse erheblich erweitert.
Wichtigkeit: mittel

PMML Für das Versenden bzw. den Austausch von Modellen bietet der Standard `PMML` die optimale Lösung. Zudem wurde sich mit der Projektgruppe *E-Stream* auf diesen Standard geeinigt. Daraus ergibt sich die Anforderung, dass das Framework in der Lage sein sollte `PMML` zu unterstützen.
Wichtigkeit: hoch

Open/Close Source Das Framework sollte Open Source sein, da die Projektgruppe beschlossen hat, nur mit Open Source Software zu arbeiten.
Wichtigkeit: hoch

Aktualität-Updates Das Framework sollte in einem regelmäßigen Abstand vom Entwickler aktualisiert werden, in dem mögliche Fehler beseitigt und neue Funktionalitäten hinzugefügt werden. Zudem sollte das Framework möglichst aktuell sein, um den heutigen Standards zu entsprechen.
Wichtigkeit: mittel

Erscheinungsdatum Das Erscheinungsdatum ist ein leichtes Indiz für die Entwicklungsphase und die Reife einer Software. So kann außerdem davon ausgegangen werden, dass ältere Software eine geringere Fehleranfälligkeit aufweist und es eine umfangreichere (Fehler-) Dokumentation vorliegt.
Wichtigkeit: gering

Algorithmen Die Kernaufgabe des Bereichs Data Science ist es Modelle bzw. Prognosen zu erstellen. Für die Erstellung dieser Prognosen ist die Verfügbarkeit eines umfangreichen Pools an Algorithmen essentiell. Daher sollte das Framework eine große Vielfalt an Methoden bereit stellen, damit verschiedene Modelle hinsichtlich ihrer Prognosen betrachtet werden können.
Wichtigkeit: hoch

Anforderungsnummer	Anforderung
DSF-01	Das Framework muss eine Schnittstelle zu Apache Hadoop und Apache Cassandra bieten
DSF-02	Das Framework muss die Sprache Python unterstützen
DSF-03	Das Framework muss den PMML Standard unterstützen
DSF-04	Das Framework muss ein Open-Source Projekt sein
DSF-05	Das Framework muss stetig aktualisiert werden
DSF-06	Das Framework muss sich in einer ausreichenden Reifephase befinden
DSF-07	Das Framework muss eine ausreichende Anzahl an Machine Learning Algorithmen bieten

Tabelle 8: Anforderungen Data Science Framework

4.4.2. Beschreibung

Im Folgenden werden mehrere Frameworks, die für die Datenanalysen in DAvE in Frage kamen, gegenüber gestellt. Da viele Frameworks auf Eignung überprüft wurden, wird nicht auf alle im Detail eingegangen. Tabelle 9: Kriterienkatalog für Machine Learning Frameworks Teil I und Tabelle 10: Kriterienkatalog für Machine Learning Frameworks Teil II stellen die einzelnen Frameworks in Zusammenhang mit den Anforderungen, die im letzten Kapitel erläutert wurden, übersichtlich dar. Installiert und umfangreich getestet wurden nur die Frameworks, die alle wichtigen Anforderungen erfüllten.

`Deeplearning4j` kann zwei wichtige Anforderungen nicht erfüllen. Zum einen werden nicht genügend Algorithmen zur Analyse bereit gestellt, zum anderen wird der PMML-Standard nicht unterstützt. Letzteres wird ebenfalls von den beiden Tools `WEKA` und `mlpy` nicht unterstützt, weshalb diese als Analysewerkzeuge nicht in Frage kamen. Zum Zeitpunkt der Evaluation war noch nicht entschieden, ob Hadoop oder Cassandra als Datenbank verwendet wird. Deshalb wurde `KNIME` auch nicht weiter betrachtet, da hier ein kostenpflichtiger Big Data Connector notwendig gewesen wäre. Bezüglich `TensorFlow` war im Vorfeld nicht ersichtlich, ob eine Schnittstelle zu Cassandra bereit gestellt wird. Somit fielen `scitkit-learn`, `Spark MLlib` und `R Studio`, die alle wichtigen Anforderungen erfüllen, in die engere Auswahl. Installiert und getestet wurden zunächst `Spark MLlib` und `scitkit-learn`, da beide Python als Programmiersprache unterstützen.

scikit-learn `scikit-learn` ist ein Open-Source Projekt für maschinelles Lernen in Python. Es wird stetig weiterentwickelt und verbessert und besitzt eine große Nutzergemeinde, wodurch viele Tutorials und Lösungsvorschläge für diverse Problemstellungen existieren. Daneben wird eine umfangreiche Dokumentation[1] bereitgestellt, die zu vielen Algorithmen zusätzlich ausführliche Code-Beispiele liefert (vgl. [MGR17], S. 6).

`scikit-learn` basiert auf den Python-Bibliotheken `NumPy` und `SciPy`. `NumPy` ist ein grundlegendes Paket für wissenschaftliche Berechnungen in Python, welches die die Funktionalität für mehrdimensionale Arrays und mathematische Funktionen, wie z. B. lineare Algebra enthält. Das `NumPy-Array` ist in `scikit-learn` die fundamentale Datenstruktur. Jegliche Daten werden in Form von `NumPy-Arrays` verarbeitet, bzw. müssen in solche umgewandelt werden. `SciPy` ist eine Sammlung von Funktionen zum wissenschaftlichen Rechnen in Python, die u.a. Routinen für fortgeschrittene lineare Algebra, Optimierung mathematischer Funktionen und statistische Verteilung enthält. `scikit-learn` verwendet Funktionen aus dieser Sammlung, um seine Algorithmen zu implementieren (vgl. [MGR17], S. 8).

Spark MLlib Apache Spark ist ein Framework für Cluster Computing, das unter einer Open-Source-Lizenz öffentlich verfügbar ist. Spark kann sowohl Standalone als auch mit

[1] siehe http://scikit-learn.org/stable/documentation.html

Kriterium	KNIME [KNI18]	scikit-learn [Sci17]	Spark MLlib [The18d]	Deeplearning4j [Sky17]
Betriebssystem	Linux Windows Mac OS	Linux Windows Mac OS	Linux Windows Mac OS	Linux Windows Mac OS
Hadoop	kostenpflichtig	wird unterstützt	wird unterstützt	wird unterstützt
Cassandra	mittels JDBC-Driver	cassandra-driver von datastax	wird unterstützt	über Spark
Programmiersprache	Java	Python	Java, Scala, Python, R	Java, Scala Clojure
PMML	Import und Export	nur Export	Import und Export	nicht unterstützt
Open/Closed Source	Open/kostenpflichtige Erweiterung	Open	Open	Open
Aktualität	keine Angabe	20.06.2017	11.07.2017	10.03.2017
Methoden	Regression Assoziationsanalyse Clusteranalyse Klassifikation	Regression Assoziationsanalyse Clusteranalyse Klassifikation	Regression Assoziationsanalyse Clusteranalyse Klassifikation	Neuronale Netze
Repository	KNIME SDK	Github	Github	Github
Erscheinungsdatum	2006	2007	2009	keine Angabe

Tabelle 9: Kriterienkatalog für Machine Learning Frameworks Teil I. Quelle: eigene Darstellung

Kriterium	WEKA [Uni17]	TensorFlow [Ten18]	R Studio [R17]	mlpy [AVM+12]
Betriebssystem	Linux Windows Mac OS	Linux Windows Mac OS Android	Linux Windows Mac OS	Linux Windows Mac OS
Hadoop	wird unterstützt	wird unterstützt	mittels Hive	nicht unterstützt
Cassandra	wird unterstützt	keine Angabe	JDBC-Driver	nicht unterstützt
Programmiersprache	Java	Java, C Go, Python	R	Python
PMML	Import eingeschränkt kein Export	Import und Export	Import und Export	nicht unterstützt
Open/Closed Source	Open	Open	Open/kostenpflich- tige Erweiterung	Open
Aktualität	16.12.2016	15.06.2017	19.04.2017	05.04.2012
Methoden	Klassifikation Assoziationsanalyse Clusteranalyse	insb. Neuronale Netze Klassifikation Regression	Regression Assoziationsanalyse Klassifikation	Regression Klassifikation Clusteranalyse
Repository	SVN	Github	Github	Github
Erscheinungsdatum	1993	2015	2011	keine Angabe

Tabelle 10: Kriterienkatalog für Machine Learning Frameworks Teil II. Quelle: eigene Darstellung

```
1          # The RDD materialization time is unpredicable, if
               we set a timeout for socket reading operation,
               it will very possibly fail. See SPARK-18281.
2          Exception: could not open socket
```

Listing 1: Fehlermeldung Spark MLlib

YARN oder Mesos betrieben werden und besteht aus den Komponenten Spark Core, Spark SQL, Spark Streaming , Machine Learning Library (MLlib) und GraphX. Für den Bereich Data Science in DAvE ist insbesondere die Komponente MLlib interessant, in der alle gängigen Machine Learning Methoden implementiert sind. Spark-Core bildet die Grundlage des gesamten Systems und stellt grundlegende Infrastruktur-Funktionalitäten bereit. Die Datenstruktur für alle Operationen wird als Resilent Distributet Dataset (RDD) bezeichnet, das einen logischen (Teil-)Bestand von Daten darstellt. Mithilfe von Spark SQL können diese RDD's in DataFrames umgewandelt werden. Zudem bietet Spark eine Schnittstelle um auf HDFS oder HBase zugreifen zu können. Des weiteren werden die Programmiersprachen Java, Scala und Python unterstützt (siehe [The18d])

Beim Testen der `Spark MLlib` wurden einige Fehler identifiziert, die beim Darstellen von DataFrames im Jupyter Notebook auftraten. Folgende Fehlermeldung trat sowohl beim Anzeigen der DataFrames, als auch beim Umwandeln in ein pandas-DataFrame auf:

In der Fehlerdokumentation von Spark wurde zu diesem Fehler auch keine passende Lösung bereit gestellt.

Beim Testen von `scikit-learn` wurden hingegen keine Fehlerfälle identifiziert. Da eine fehlerfreie Anzeige von Daten, insbesondere während der Entwicklung von Modellen, essentiell ist, wurde `scikit-learn` als engültiges Machine Learning Tool ausgewählt.

In den folgenden Abschnitten werden weitere Komponenten, die zur Entwicklung der Machine Learning Modelle benötigt werden, eingehend beschrieben.

Python Pyhton ist für viele Anwendungen aus dem Bereich Data Science zu einer der beliebtesten Programmiersprachen geworden. Python kombiniert die Ausdruckskraft allgemein nutzbarer Programmiersprachen mit der einfachen Benutzbarkeit einer domänenspezifischen Skriptsprache. Insbesondere im Rahmen von Datenanalysen existieren für Python eine Vielzahl von Bibliotheken, wie z. B. zum Laden von Daten, Visualisieren, Berechnen von Statistiken oder Methoden für das maschinelle Lernen. Dies gibt Data Scientists einen sehr umfangreichen Werkzeugkasten für jegliche Einsatzgebiete. Ein Hauptvorteil von Python ist die Möglichkeit direkt mit dem Code zu interagieren, sei es in einer Konsole oder in einer anderen Umgebung wie dem Jupyter Notebook (vgl. [MGR17], S. 5f.).

pandas `pandas` ist eine Python-Bibliothek zur Datenaufbereitung und Analyse, die um die Datenstruktur DataFrame herum aufgebaut ist. Ein DataFrame kann als eine Art Tabelle verstanden werden. Vorteil gegenüber `NumPy-Arrays` ist, dass `pandas` in jeder Spalte unterschiedliche Datentypen zulässt. `pandas` enthält eine großes Spektrum an Methoden zum Modifizieren und Verarbeiten dieser DataFrames. Insbesondere sind SQL-artige Suchanfragen möglich (vgl. [MGR17], S. 10). Im Rahmen dieses Projektes wird `pandas` verwendet, um Daten von Cassandra in Python auslesen zu können. Zudem wird durch die bereitgestellten Methoden eine effiziente Vorverarbeitung möglich, um beispielsweise Rohdaten in die benötigten Datentypen umzuwandeln.

matplotlib `matplotlib` ist eine Python-Bibliothek zum wissenschaftlichen Plotten, die umfangreiche Funktionen zum Erstellen von Diagrammen, wie z. B. Liniendiagramme, Histogramme oder Streudiagramme, enthält. Die Daten und unterschiedliche Aspekte dieser Daten zu visualisieren, kann wichtige Erkenntnisse liefern und wurde im Rahmen des Teilbereichs Data Science für sämtliche Visualisierungsaufgaben eingesetzt. Im Jupyter Notebook können Diagramme über den Befehl `%matplotlib inline` direkt im Browser dargestellt werden (siehe [MGR17], S. 10).

4.5. Data Engineering

Aufgabe der Gruppe Data Engineering ist die Identifizierung von möglichen Datenquellen und die Analyse der Daten für einen Einsatz im Projekt. Zusätzlich müssen Möglichkeiten zur Einbindung dieser Datenquellen in das Gesamtsystem aufgezeigt werden. Aus diesem Grund wurden an dieser Stelle alle Daten untersucht, die mit dem Projektszenario in Verbindung stehen. Im Anschluss werden die geeigneten Datenquellen selektiert und Möglichkeiten eines Imports in das System sowie die Eignung zur Einbindung in die Gesamtarchitektur untersucht.

4.5.1. Anforderungen

Die Anforderung an die Technologien und die Datenquellen ist die einwandfreie, automatisierte Importmöglichkeit in das Gesamtsystem, da ein manueller Datenimport für den späteren Betrieb nicht in Frage kommt. Zur Umsetzung der Use-Cases sind Wetterdaten und Smart-Meter-Daten nötig. Die Wetterdaten werden vom Deutschen Wetterdienst geliefert und können vom eigenen FTP-Server abgerufen werden. Diese Wetterdaten gilt es direkt vom FTP Abzufragen und in das System einzubinden. Zusätzlich werden Daten für Smart-Meter-Daten mittels des LV-Sim erstellt und bereitgestellt. Mithilfe der ausgewählten Technologie musste es möglich sein, die Daten aus den verschiedenen Quellen, zu jeder beliebigen Zeit automatisch abzurufen und in der Datenbank speichern zu können.

4.5.2. Beschreibung

Die erste Hürde bei dem Import der Datenquellen bestand in der Importmöglichkeit für Smart-Meter-Daten. Da kaum Datensätze für Smart-Meter-Daten zur Verfügung standen, musste eine Alternative Möglichkeit des Datenimports geschaffen werden.

An dieser Stelle kam das Python Tool mosaik mit der Netzdatenstrom-Simulation des Office Energie zum Einsatz. Mit diesem Toolset ist es möglich, in gewisser Weise Echtzeitdaten zu erzeugen. Das Toll wird im folgenden genauer beschrieben.

Für den Import von Wetterdaten wurden verschiedene Möglichkeiten getestet, welche im folgenden beschrieben sind. Der Teilbereich des Data-Engineerings fokussiert sich auf den Import der beschriebenen Datenquellen in das DAvE-System. Dabei soll konkret eine Möglichkeit gefunden werden, Wetterdaten, Smart-Meter-Daten und Daten über Energieerzeugung und -verbrauch im System zu archivieren. Dazu wird eine Technologie benötigt, die einen automatisierten Import dieser Daten in die Cassandra Datenbank, in regelmäßigen Abständen, oder im Bedarfsfall gewährleistet. Zur Implementierung einer solchen Technologie, haben wir folgende Lösungen in Betracht gezogen:

Eigenentwicklung in Java Zunächst wurde sich für eine Eigenentwicklung auf Java Basis entschieden. Während der Entwicklung kam es zu Problemen mit der für den FTP Import notwendigen Java Bibliothek *Apache Commons Net*. Da das Problem nicht vollständig identifizierbar war, wurde sich dazu entschlossen, andere Möglichkeiten des Datenimportes in Betracht zu ziehen, weshalb die Eigenentwicklung auf der Basis von Java an dieser Stelle abgebrochen wurde.

PyFTPToHDFS Ein weiterer Ansatz basierte auf der Nutzung eines Python Skriptes zum automatischen Import der Wetterdaten. Der Projektgruppe wurde ein fertiges Skript zur Verfügung gestellt, welches lediglich hätte angepasst werden müssen. Das Skript erstellt einen Pfad zum FTP-Server der Wetterdaten und importiert diese Daten in HDFS. Aufgrund der Einstellung einer Nutzung von HDFS und mangelnder Erfahrung in der Nutzung von Python, wurde diese Option jedoch zunächst hinten angestellt.

AXT Zum Import von Daten wurde zusätzlich eine Eigenentwicklung der vorherigen Projektgruppe DASH herangezogen. Dabei handelt es sich um das Java-basierte Programm *AXT*. Dies ermöglicht den Datenimport von einer Quelle und überträgt diese Daten in HDFS. Zusätzlich bietet es die Möglichkeit, Dateien mit Meta-Informationen wie Dateinamen oder Dateigröße anreichern zu können. Das Programm bietet somit den Einstiegspunkt für einen ELT-Prozess, sodass nachgelagerte Programme auf die von AXT vorbereiteten und übertragenen Daten zugreifen können. Dabei werden die Informationen aus den Dateien extrahiert oder es werden bereits im System vorhandene Dateien angepasst. AXT kommt

für die Anwendungsfälle von DAvE deswegen in Frage, weil es sich auf einen Datenimport von einem FTP-Server und deren Extraktion in HDFS bezieht, sodass der Workflow zur Archivierung von Wetterdaten abgebildet werden kann. Dabei bietet sich vor allem der Vorteil, dass größere Datenmengen effizient und in kurzer Zeit übertragen werden können. Die Besonderheit dabei liegt allerdings darin, dass in dem Anwendungsfall von AXT ein gesondertes Datenformat vorliegt. Dabei handelt es um ZML-Daten, die in XML-Daten konvertiert werden und als solche archiviert werden.
Für die Anwendungsfälle von DAvE wäre deshalb eine umfangreichere Anpassung des Quellcodes notwendig gewesen. Darüber hinaus wird für die Archivierung von DAvE eine Cassandra Datenbank anstatt von HDFS herangezogen. Aufgrund dieser Anpassungen und der Architektur wurde der Ansatz von AXT verworfen, um eine den Anforderungen entsprechende Lösung zu finden.

Apache NiFi Letztendlich wurde sich für das Tool Apache NiFi entschieden, welches bereits in anderen Bereichen des Projektes zur Anwendung gekommen ist. Apache NiFi ermöglicht einen zeitgesteuerten vollautomatischen Import der Wetterdaten von dem FTP Server des deutschen Wetterdienstes.

Low Voltage Simulator Als Teil des Projekts NetzDatenStrom des OFFIS existiert die Simulationssoftware *Low Voltage Simulator* (LV-Sim) für Niederspannungsnetze, welche Smartmeter(gateway)-Daten zu den simulierten Netzen erzeugt. Die Simulationssoftware (:LV-Simulator) basiert auf dem Simulationsframework `mosaik`[2] des OFFIS und ist in der Programmiersprache Python geschrieben. Während der gesamten Laufzeit dieser Projektgruppe befand sich der LV-Simulator in der Entwicklung. Diese und alle weiteren Informationen sind der nur für Projektmitglieder zugänglichen Onlinedokumentation des LV-Sim sowie E-Mail-Kommunikationen und persönlichen Gesprächen mit den Programmierern entnommen (siehe [OFF18]).

Um den Anwendungsfall des Archivierens von Smart-Meter-Daten des Prototypen des DAvE-Systems testen und durchführen zu können, ist es notwendig Smart-Meter-Daten eines externen Produzenten erstellen zu lassen und über die definierten Schnittstellen zu archivieren. Da sich diese Projektgruppe und das DAvE-System in das Projekt NetzDatenStrom eingliedert, wird der LV-Simu zu diesem Zweck verwendet. Dieses Programm simuliert diesen externen Produzenten eines Datenstroms von Smart-Meter-Daten. Dadurch entfällt der Auswahlprozess für ein geeignetes Softwaresystem, welches diese Aufgabe übernehmen soll.

Der LV-Sim erzeugt Smart-Meter-Daten im JSON-Format, diese können jedoch auch optional im COSEM-XML-Format ausgegeben werden. Jeweils ein Beispieldatensatz für das JSON-Format und das COSEM-XML-Format ist den Abbildungen 2: Beispieldatensatz LV-

[2] siehe `https://mosaik.offis.de/`

```
{
  "ldevs": [
    {
      "ldev_id": "sm_38e3e715-0e70-4585-8bc8-bfa9a82e32f7",
      "objects": [
        {
          "value": 40064.60324999999,
          "capture_time": 314664,
          "status": "000",
          "scaler": -3,
          "logical_name": "TODO: OBIS-ID",
          "unit": 30
        }
      ]
    },
    {
      "ldev_id": "sm_5fdf2caf-c2dd-4981-8e71-a6f87393f108",
      "objects": [
        {
          "value": 57028.70040000003,
          "capture_time": 314664,
          "status": "000",
          "scaler": -3,
          "logical_name": "TODO: OBIS-ID",
          "unit": 30
        }
      ]
    }
  ],
  "logical_name": "smgw_eb868bb1-b5c0-4400-abd2-097d86b21239
      "
}
```

Listing 2: Beispieldatensatz LV-Simulator, JSON-Format

Simulator, JSON-Format und 3: Beispieldatensatz LV-Simulator, COSEM-XML-Format zu entnehmen.

COSEM steht für *Companion Specification for Energy Metering* und ist ein standardisiertes Format zum Austausch von Smart-Meter-Daten. Dieser Standard wird von der *DLMS User Association* verwaltet. DLMS ist wiederum eine Abkürzung für *Device Language Message specification*. Der COSEM-Standard steht als *XML Schema Definiton* zur Verfügung, dadurch können Daten mit der COSEM-Struktur in Form einer XML-Datei als COSEM-XML ausgetauscht werden (vgl. [DLM18]).

Die Generierung der Smart-Meter-Daten kann vielfältig konfiguriert werden, z. B. ist es möglich unterschiedliche Output-Klassen dynamisch in den Programmablauf zu integrieren, um die erzeugten Daten auf verschiedene Arten auszugeben und weiterzuverarbeiten. Dazu gehören u. a. die Möglichkeiten der Ausgabe auf der Console oder das Schreiben in

```
<cosem>
  <ldevs count="1">
    <ldev id="sm_ebba2e0d-8530-4933-a504-76b162ed0fb3">
      <objects count="1">
        <object class-id="4" id="TODO: OBIS-ID" version="0">
          <attributes count="6">
            <attribute id="1">
              <value>
                <string>TODO: OBIS-ID</string>
              </value>
            </attribute>
            <attribute id="2">
              <value>
                <float64>8735.074424999999</float64>
              </value>
            </attribute>
            <attribute id="3">
              <value>
                <integer>-3</integer>
                <enum>30</enum>
              </value>
            </attribute>
            <attribute id="4">
              <value>
                <enum>000</enum>
              </value>
            </attribute>
            <attribute id="5">
              <value>
                <double-long-unsigned>55342</double-long-
                    unsigned>
              </value>
            </attribute>
          </attributes>
        </object>
      </objects>
    </ldev>
  </ldevs>
</cosem>
```

Listing 3: Beispieldatensatz LV-Simulator, COSEM-XML-Format

eine Output-Datei.

Des Weiteren ist der LV-Sim mit einer Topologie zu konfigurieren. Dies ist eine Struktur der zu simulierenden Smart Meter. Die Topologie kann mit einigen Parametern zufallsgeneriert oder per JSON-Datei dem Simulator vorgegeben werden.

5. Installation

In diesem Kapitel werden die ausgewählten Technologien, die im Kapitel 4: Technologieauswahl vorgestellt wurden, installiert. Dazu wird zuerst die Infrastruktur der Systemumgebung im Abschnitt 5.1: Infrastruktur eingegangen. Die Abschnitte 5.2: Archivierung: Datenbank und 5.3: Archivierung: Dataflow Management umfassen die gesamte Installation des Archivs. Es wird seperate auf die Datenbank und danach auf das Dataflow Management eingegangen. Der Abschnitt 5.4: Kommunikationskomponente umfasst die Installation der Messaging Systeme und des Webservers. Abgeschlossen wird dieses Kapitel mit dem Abschnitt 5.5: Data Science. Dabei wird die Installation der Subsysteme der Data Science Komponente erläutert.

5.1. Infrastruktur

In der Systemumgebung befindet sich ein Cluster von Virtuelle Maschinen (folgend :DAvE-Cluster). Die Virtuellen Maschinen (:VMs) des DAvE-Clusters mit ihren Aufgaben und Diensten sind in 11: Systeminfrastruktur des DAvE-Cluster aufgeführt. Innerhalb dieser Tabelle sind bereits Entscheidungen aus dem Kapitel 4: Technologieauswahl sowie teilweise Abschnitte aus den Kapiteln 5: Installation und 6: Realisierung vorgegriffen und sie stellt die finale Infrastruktur dar. Auf allen VMs ist das Betriebssystem Ubuntu in der Version 14.04 LTS installiert.

Die VM *master* beinhaltet die meisten Dienste. Das endgültige Big Data Archivsystem DAvE läuft innerhalb eines Webservers auf dieser VM und greift auf die anderen Dienste dieser und der anderen VMs zu. Ebenfalls ist diese VM der Host für einen Kafka Broker, welcher verwendet wird um Daten aus dem Archiv den externen Anfragestellern zur Verfügung zu stellen. Die Data Mining Komponente des DAvE-Systems befindet sich ebenfalls auf der VM *master*. Dafür ist eine Python3-Umgebung eingerichtet. Damit diese VM in der Entwicklungsphase von allen Projektgruppenmitgliedern genutzt werden kann hat jedes Mitglied einen eigenen Benutzer auf dem System. Administrative Zugänge sind den Systemadministratoren vorbehalten.

Die VMs *slave1* - *slave5* dienen als reine Cassandra Nodes. Auf ihnen werden die zu archivierenden Daten aufgeteilt und gespeichert. Da dies der einzige Zweck der VMs ist, ist der Zugang zu diesen nur von den Systemadministratoren möglich.

Um die Interaktionen mit anderen System zu simulieren wird die VM *mosaik-vm* verwendet. Darauf befindet sich der LV-Simulator sowie eine zum Betrieb dessen notwendige Python3-Umgebung. Um das Importieren von Daten von externen Kafka Brokern zu testen, befindet sich ebenfalls eine weitere Kafka Broker auf dieser VM.

VMs		
Interne IP-Adresse	***Name***	***Dienste/Aufgaben***
192.168.112.10	master	Apache Tomcat Webserver mit DAvE Apache Zookeeper Apache Kafka Broker Jupyter Notebook Python3-Umgebung für Data Mining Apache Cassandra Node Apache NiFi Benutzer für jedes Projektgruppenmitglied
192.168.112.11	slave1	Apache Cassandra Node nur Admin-Nutzer
192.168.112.12	slave2	
192.168.112.13	slave3	
192.168.112.14	slave4	
192.168.112.15	slave5	
192.168.112.16	mosaik-vm	LV-Simulator Python3-Umgebung für den LV-Simulator Apache Zookeeper Apache Kafka Broker nur Admin-Nutzer

Tabelle 11: Systeminfrastruktur des DAvE-Cluster

5.2. Archivierung: Datenbank

Die nachfolgenden Schritte zeigen die Installation und Konfiguration von Apache Cassandra 3.11.1. Dabei wurde nach der offiziellen Anleitung von Apache vorgegangen.[3] Bevor Apache Cassandra installiert werden kann, wird die Oracle Java Version 8 vorausgesetzt. Zuerst muss das *Apache Repository* von Apache Cassandra auf dem Master und alle Slaves angelegt werden. Danach werden die Schlüssel der Repository hinzugefügt und ein Update durchgeführt. Anschließend kann Apache Cassandra heruntergeladen und installiert werden. Nach der Installation wird der Service automatisch gestartet. Zur Konfiguration von Apache Cassandra muss der Dienst gestoppt werden.

```
1 echo "deb http://www.apache.org/dist/cassandra/debian 36x
      main" | sudo tee -a /etc/apt/sources.list.d/cassandra.
      sources.list
2 curl https://www.apache.org/dist/cassandra/KEYS | sudo apt-
      key add -
3 sudo apt-get update
4 sudo apt-get install cassandra
```

Listing 4: Download und Installation von Apache Cassandra

[3] www.cassandra.apache.org/doc/latest/getting_started/installing.html

Damit alle Knotenpunkte auf dem selben Rack sind und miteinander kommunizieren können, müssen verschiedene Eigenschaften in den Dateien *cassandra.yaml*, *cassandra-topology.properties*, *cassandra-rackdc.properties* angepasst werden. Die Dateien sind unter dem Pfad „:/etc/cassandra/“ zu finden. Auf allen Knoten müssen in der Datei cassandra.yaml die Seeds und listen_adress individuell verändert werden. Für den Master sehen die Anpassungen folgendermaßen aus:

```
1 Seeds: "master.local" #auf allen Nodes
2 listen_adress:master.local #z. B. für Slave1: listen_adress:
      slave1.local
```

Listing 5: Anpassungen master cassandra.yaml

Des Weiteren muss auf allen Nodes in der Datei *cassandra-topology.properties* folgende Zeilen auskommentiert und die darunterliegenden Zeilen hinzugefügt werden.

```
1  #192.168.1.100=DC1:RAC1
2  #192.168.2.200=DC2:RAC2
3
4  #10.0.0.10=DC1:RAC1
5  #10.0.0.11=DC1:RAC1
6  #10.0.0.12=DC1:RAC2
7
8  #10.20.114.10=DC2:RAC1
9  #10.20.114.11=DC2:RAC1
10
11 #10.21.119.13=DC3:RAC1
12 #10.21.119.10=DC3:RAC1
13
14 #10.0.0.13=DC1:RAC2
15 #10.21.119.14=DC3:RAC2
16 #10.20.114.15=DC2:RAC2
17
18 192.168.112.10=DC1:RAC1
19 192.168.112.11=DC1:RAC1
20 192.168.112.12=DC1:RAC1
21 192.168.112.13=DC1:RAC1
22 192.168.112.14=DC1:RAC1
23 192.168.112.15=DC1:RAC1
```

Listing 6: Anpassungen cassandra-topology.properties

In der *cassandra-rackdc.properties* sollte noch überprüft werden, ob die folgenden Zeilen vorhanden sind. Ansonsten müssten diese noch dazugeschrieben werden.

```
1 dc=DC1
2 rack=RAC1
```

Listing 7: Überprüfung cassandra-rackdc.properties

Zudem musste noch in der Datei *cqlshrc* die Zeitzone auf die Mitteleuropäische Zeit (CET) geändert werden. Nach den Anpassungen wird Apache Cassandra gestartet und es wird überprüft, ob sich alle Knoten im selben Rack befinden.

```
1 sudo service cassandra start
2 nodetool status
```

Listing 8: Start von Apache Cassandra

```
hduser@master:~$ nodetool status
Datacenter: datacenter1
=======================
Status=Up/Down
|/ State=Normal/Leaving/Joining/Moving
--  Address         Load       Tokens       Owns (effective)  Host ID                               Rack
UN  192.168.112.10  1.78 GiB   256          49.9%             12bf3d18-f303-4f1d-8e6a-1959a8fdc14d  rack1
UN  192.168.112.11  1.71 GiB   256          52.1%             b3b2c158-15b6-4ddd-8ca7-2d586d485921  rack1
UN  192.168.112.12  1.71 GiB   256          49.6%             86c9d50a-55f2-4cd2-8cbb-5e00220dc7fc  rack1
UN  192.168.112.13  1.67 GiB   256          49.8%             47d9a12e-1e97-4986-a9a8-be00138dbfbc  rack1
UN  192.168.112.14  1.65 GiB   256          49.3%             480ef6b0-b917-41ff-a69f-ad725510e397  rack1
UN  192.168.112.15  1.57 GiB   256          49.3%             7519e832-d55c-474a-844a-0ae43d078b32  rack1
```

Abbildung 11: Cassandra Cluster

5.3. Archivierung: Dataflow Management

Nachfolgend wird die Installation und Konfiguration von Apache NiFi gezeigt. Als Vorlage dient hier die offizielle Anleitung von Apache. [4] Voraussetzung der Installation von Apache NiFi ist die Oracle Java Version 8. Bevor Apache NiFi installiert werden kann, laden wir unsere gewünschte Version von der offiziellen Downloadseite herunter. [5] Da wir Ubuntu benutzen und die aktuellste Version 1.4.0 ist, benötigen wir die Binaries mit der Endung „.tar.gz“. Diese werden, wenn möglich, direkt in das gewünschte Verzeichnis heruntergeladen und dort entpackt. Um Apache NiFi nun starten zu können, muss im Terminal in das Installationsverzeichnis von NiFi navigiert werden. Es gibt nun zwei Startmöglichkeiten, aber auch die Möglichkeit den Dienst wieder zu stoppen oder den Status einzusehen:

[4] https://nifi.apache.org/docs/nifi-docs/html/getting-started.html
[5] http://nifi.apache.org/download.html

```
# im Vordergrund starten
bin/nifi.sh run
# im Hintergrund starten
bin/nifi.sh start
# NiFi stoppen
bin/nifi.sh stop
# Status einsehen
bin/nifi.sh status
```

Listing 9: Download und Installation von Apache NiFi

In unserem Fall müssen vorerst aber noch einige Konfigurationen angepasst werden, um auftretenden Problemen vorzubeugen. Voraussetzung dafür ist, dass Apache NiFi gestoppt ist. Erst nach der Änderung kann Apache NiFi wieder gestartet werden. Zum Einen haben wir den Standardport ändern müssen, da dieser bei der Erstinstallation bereits vergeben war. Die Einstellungen können in dem Unterordner für Konfigurationsdateien im ursprünglich entpackten NiFi-Ordner geändert werden. Für das Verändern des Ports wird nun folgendes eingegeben:

```
cd /opt/nifi-1.4.0/conf
sudo nano nifi.properties
# Abändern des Ports
# web properties #
nifi.weg.http.port=1904
```

Listing 10: Konfiguration des Ports von Apache NiFi

Des Weiteren reichen die JVM memory settings nicht aus, um die Datenmengen fehlerlos übermitteln zu können. Deswegen wird die Größe des JVM memories angepasst:

```
cd /opt/nifi-1.4.0/conf
sudo nano bootstrap.conf
# Ändern der JVM memory settings
java.arg.2=Xms16g
java.arg.3=Xms16g
```

Listing 11: Konfiguration der JVM memory settings

Nach der erfolgreichen Installation und Konfiguration von NiFi kann die Benutzeroberfläche in einem beliebigen Browser über die Adresse „localhost:1904/nifi“ angesteuert werden.

5.4. Kommunikationskomponente

In diesem Abschnitt wird die Installation von Apache Kafka und RabbitMQ eingegangen. Dabei wird Schritt für Schritt der gesamte Installation erläutert. Im Abschnitt 5.4.3:

Apache Tomcat und Docker wird auf die Installation von Tomcat und Docker eingegangen.

5.4.1. Apache Kafka

In diesem Abschnitt wird auf die Einrichtung, sowie auf die Konfiguration des Kafka Cluster für DAvE eingegangen. Dazu werden zuerst die Installationsschritte beschrieben. Der zweite Schritt ist die Konfiguration die durchgeführt wurden.

Es wurde Apache Kafka 1.0.0 für den Prototyp verwendet. Bei der Einrichtung wurde die Dokumentation der Apache Kafka herangezogen (vgl. [The18b]). Für die Verwaltung der Topics muss vorher noch ein Apache Zookeeper Service gestartet werden. Der Zookeeper Service wird mit der Installation von Apache Kafka mitgeliefert. Es ist auch möglich das Apache Projekt direkt von der eigenen Website zu erhalten. Für den Prototyp wurde die Projektversion 3.4 genutzt.

Die Wurzelverzeichnis von Zookeeper und Kafka befinden sich innerhalb des */opt*-Verzeichnisses in Ubuntu. In dem bin-Ordner innerhalb der Kafka Verzeichnis sind alle notwendigen Befehl-Skripte, unter anderem die Start und Stopp-Skripte, hinterlegt. Im Config-Ordner befinden sich die Konfigurationsdateien. Der Aufbau bei Zookeeper ähnelt dem von Kafka. Die Start- und Stopp-Skripte befinden sich ebenfalls im */bin*-Verzeichnis. Die Kofniguration befinden sich im */conf*-Verzeichnis. Die Kafkakonfiguration durchlief eine Entwicklung.

In der ersten Version wurde die *server.properties* in */Kafka/conf/* angepasst. Aufgrund der Netzwerkarchitektur musste die Einstellung *advertised.host.name* musste aktiviert werden. Der Wert für diese Einstellung war die Hostname des Master-Server auf dem der Apache Kafka Service läuft. Des Weiteren musste auch die *advertised.port* Einstellung aktiviert und vergeben werden. In der *zookeeper.properties* mussten weitere Einstellung überprüft werden. Dazu zählen die *tickTime*, *dataDir* und *clientPort* Einstellung. Die *tickTime* gibt die minimale Verzögerung an bis eine Verbindung zum Zookeeper Service geschlossen wird. Die Einstellung im Prototyp wurde auf 2000 millisekunden festgelegt. Die Einstellung *dataDir* gibt den Pfad zu den Log-Dateien an. Die *clientPort*-Einstellung wurde auf dem Standardwert belassen. Der Zookeeper Service ist unter dem Port 2181 auf dem *master.local*-Server erreichbar.

In der finalen Version der Konfiguration konnten die Grundeinstellung aller Services genutzt werden. Dafür wurde die *hosts*-Datei auf Fehler überprüft und korrigiert. Die Befehl-Skripte umfassen Start- und Stopp-Befehle, sowie Monitoring Skripte. Für den Prototyp wurden die Start- und Stopp-Befehle, das *Topic*-Skript und Erzeuger/Verbraucher Skripte genutzt. Die Befehle zum Starten der Services werden im Listing 12: Start-Skript gezeigt. Diese Befehle lassen die Services im Hintergrund starten.

Zum Überprüfen der Einstellung wurden die Befehle im Listing 13: Befehle für Kafka genutzt. Der erste Befehl erzeugt ein Topic *Hello Kafka*. Dazu wurden die Parameter

```
1 bin/zkServer.sh config/zoo.cfg
2 bin/kafka-server-start.sh -daemon config/server.properties
```

Listing 12: Start-Skript

```
1 bin/kafka-topics.sh --create --zookeeper localhost:2181 --
      replication-factor 1 --partitions 1 --topic HelloKafka
2 bin/kafka-topics.sh --list --zookeeper localhost:2181
3 bin/kafka-console-producer.sh --broker-list localhost:9092
      --topic HelloKafka
4 bin/kafka-console-consumer.sh --bootstrap-server localhost
      :9092 --topic HelloKafka --from-beginning
```

Listing 13: Befehle für Kafka

übergeben, wo der Zookeeper Services angesprochen werden kann. Des Weiteren werden die Replikationsfaktoren und die Partition angegeben. Der Parameter *–create* erzeugt das Topic. Der zweite Befehl gibt listet die alle vorhanden Topics auf. Dabei muss das Topic *HelloKafka* auftauchen. Die letzten beiden Befehle dienen zur Nachrichtenerzeugung und das Abrufen des Topics *HelloKafka*.

5.4.2. RabbitMQ

Die Installation eines RabbitMQ Server wurde für den Prototyp folgendermaßen durchgeführt. Die Version der RabbitMQ Software ist die Release 3.6 und die Erlang Bibliothek 20.1 für RabbitMQ muss ebenfalls installiert sein. RabbitMQ wurde auf den Server *master.local*, *slave1* und *slave2* installiert. Das Listing 14: Installation von RabbitMQ auf Ubuntu zeigt die verwendeten Befehle um Erlang zu installieren. Dazu muss das Erlang Repository auf Ubuntu hinzugefügt werden und dann kann Erlang heruntergeladen und installiert werden (vlg. [Piv18]).

Der nächste Schritt ist RabbitMQ zu installieren. Dazu muss zuerst der das RabbitMQ Repository hinzugefügt werden. Im Anschluss muss auch der Public Key hinzugefügt werden. Dann kann RabbitMQ installiert werden. Das Listing 15: Installation RabbitMQ zeigt die verwendeten Befehle (vlg. [Piv18]).

```
1 wget https://packages.erlang-solutions.com/erlang-
      solutions_1.0_all.deb
2 sudo dpkg -i erlang-solutions_1.0_all.deb
3 sudo apt-get update
4 sudo apt-get install erlang erlang-nox
```

Listing 14: Installation von RabbitMQ auf Ubuntu

```
1 echo 'deb http://www.rabbitmq.com/debian/ testing main' |
      sudo tee /etc/apt/sources.list.d/rabbitmq.list
2 wget -O- https://www.rabbitmq.com/rabbitmq-release-signing-
      key.asc | sudo apt-key add -
3 sudo apt-get update
4 sudo apt-get install rabbitmq-server
```

Listing 15: Installation RabbitMQ

```
1 sudo update-rc.d rabbitmq-server defaults
2 sudo systemctl enable rabbitmq-server
3 sudo service rabbitmq-server start
4 #Add Admin
5 sudo rabbitmqctl add_user admin password
6 sudo rabbitmqctl set_user_tags admin administrator
7 sudo rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
8 #Einrichtung der Weboberfläche
9 sudo rabbitmq-plugins enable rabbitmq_management
```

Listing 16: Konfiguration RabbitMQ

Das Starten und die Konfiguration der RabbitMQ Services werden im Listing 16: Konfiguration RabbitMQ gezeigt. Zuerst wird der Dienst eingerichtet. Mit dem Befehl in Zeile 2 wird die RabbitMQ Systemctl aktiviert. Damit lässt sich RabbitMQ über eine CLI steuern. Der Befehl in Zeile 3 startet dann den Services. Als nächstes wird ein Admin eingerichtet. Der letzte Befehl aktiviert die Weboberfläche für RabbitMQ. Damit kann unter anderem der Zustand anhand eines Monitors beobachtet werden (vlg. [Piv18]).

Zum Abschluss muss noch das Cluster erzeugt werden. Aufgrund der Hochverfügbarkeit und Zuverlässigkeit wurde ein Cluster mit einer Abbild des *master.local* erstellt. Dazu befindet sich im Verzeichnis */var/lib/rabbitmq/* eine Datei *.erlang.cookie*. Diese Datei enthält eine eindeutige Idenfikationsnummer. Diese Datei oder der Inhalt muss kopiert und auf dem zweiten Server hinterlegt werden. Damit werden wird ein Abbild erzeugt (vlg. [Piv18]).

5.4.3. Apache Tomcat und Docker

Im folgenden wird auf die Erreichbarkeit des System via REST eingegangen. Der Tomcat Service wird innerhalb eines Dockercontainers betrieben. Dazu wurde eine Docker Installation genutzt und ein Tomcat Image. Für die Docker Installation wurde die Version 17.0.4 ausgewählt. Der Apache Tomcat Container wurde vom official Repository mit der Version 8.0.50 genutzt. Die Installation wurde auch wieder nach der offiziellen Dokumentation durchgeführt. Da der einzige erreichbare Server der *master.local* ist, wurde auch nur auf diesem der Tomcat Container erzeugt. Der Port 9090 wurde für die Erreichbarkeit der Schnittstelle ausgewählt. Der Port 9090 muss in der *server.xml* in der *Tomcat/conf* geän-

```
1
2 FROM tomcat:8.0.20-jre8
3
4 COPY /conf/server.xml /usr/local/tomcat/conf/server.xml
5
6 COPY /dave_server/dave_server.war /usr/local/tomcat/webapps/
      dave_server.war
```

Listing 17: Start-Skript

dert werden. Dazu muss das Attribute *Port* im *Connector*-Tag von 8080 auf 9090 geändert werden. Eine alternative Einstellung des Ports in der *server.xml* ist die Einstellung in der *web.xml* im *war*-Verzeichnis von DAvE.

Die DAvE Dockerfile beinhaltet eine Referenz auf die offizielle Tomcat Dockerfile und den Docker-Befehl zum kopieren einer Datei. Das Listing 17: Start-Skript zeigt den Inhalt dieser Dockerfile.

5.5. Data Science

Um die Data Science Funktionalitäten anzubieten, wurden mehrere Komponenten auf der VM installiert. Python wurde bereits mit dem auf der VM installierten Betriebssystem mitgeliefert. Um weitere Python Pakete zu installieren, wurde zunächst pip installiert (Listing 18: Installieren der Data Science Komponenten, Zeile 1) Im Anschluss daran wurde die Entwicklungsumgebung Jupyter installiert (Listing 18: Installieren der Data Science Komponenten, Zeile 3). Diese wurde zum Untersuchen der Daten und zum Entwickeln der Modelle genutzt. Im folgenden wurden die für die Anwendung der Data Science Methoden und das Erstelllen von Modellen relevanten Pakete installiert (Listing 18: Installieren der Data Science Komponenten, Zeile 5-8). Zum Export der erstellten Modelle in das PMML Format wurde weiterhin das Paket sklearn2pmml installiert (Listing 18: Installieren der Data Science Komponenten, Zeile 10). Zum anschließenden Parsen der PMML Dateien war das Paket lxml erforderlich (Listing 18: Installieren der Data Science Komponenten, Zeile 12). Für die Anbindung an die Cassandra Datenbank wurde der von Datastax entwickelte Treiber verwendet (Listing 18: Installieren der Data Science Komponenten, Zeile 13).

```
1  sudp apt-get install python3-pip
2
3  sudo pip3 install jupyter
4
5  sudo pip3 install numpy
6  sudo pip3 install scipy
7  sudo pip3 install scikit-learn
8  sudo pip3 install pandas
9
10 sudo pip3 install git+https://github.com/jpmml/sklearn2pmml.
       git
11
12 sudo pip3 install lxml
13
14 sudo pip3 install cassandra-driver
```

Listing 18: Installieren der Data Science Komponenten

6. Realisierung

Das Ziel dieses Kapitels ist die Erläuterung der Realisierung des Prototyps. Dabei wird zuerst im Abschnitt 6.1: Datenquellen auf die Datenquellen eingegangen. Dort werden diese beschrieben und näher erläutert. Der Abschnitt 6.2: Archivierung umfasst unter anderem die Beschreibung des Datenbankschemas sowie den Datenimport und -export. Außerdem wird auf die erarbeiteten Prozesse in Apache NiFi eingegangen. Im Abschnitt 6.3: Kommunikationskomponente wird auf die Umsetzung der Schnittstelle eingegangen und wie die Messaging System eingebunden wurden. Des Weiteren wird der Prozess für die Verarbeitung der Anfragen beschrieben. Abschließend wir im Abschnitt 6.4: Data Science auf die PMML-Modellerstellung eingegangen. Es wird zwischen der Steuerung und der Modellierung des Prozesses differenziert.

6.1. Datenquellen

In dem Big Data-Archiv müssen diverse Datenquellen zur Speicherung und Abfrage von Daten bereitliegen. Für die Auswertung werden aktuelle und historische Wetterdaten zum Power-Forecast genutzt und anhand von Smart-Meter Daten des Stromnetzes Lastprognosen ermöglicht. Zusätzlich werden Daten des statistischen Bundesamtes (SMARD-Daten) über realisierte und prognostizierte Energieverbräuche und -erzeugung herangezogen

6.1.1. Wetterdaten

Anhand von aktuellen und historischen Wetterdaten können Prognosen für die Stromerzeugung ermöglicht werden. Dazu dienen in erster Linie Sonnen- und Wind-Daten, um aufgrund dieser Basis Prognosen für Wind- und Solarenergie aufstellen zu können. Diese Daten werden vom Deutschen Wetterdienst bereitgestellt und sind über einen FTP-Server[6] abrufbar. Die Daten werden deutschlandweit an diversen Wetterstationen erfasst. Dabei wird die Möglichkeit geboten, zwischen jährlichen, monatlichen, täglichen und stündlichen Werten auszuwählen. In diesem Anwendungsfall werden stündliche Werte herangezogen. Die Daten auf dem FTP-Server werden somit stündlich aktualisiert. Dazu werden Zip-Files hochgeladen, die Textdateien mit den relevanten Werten enthalten. Zusätzlich sind die Zip-Files mit Metadaten angereichert. Es gibt pro Zip-File eine Produkt-Textdatei mit den Messwerten sowie weitere Text-Dateien, die Stationsmetadaten enthalten. Auch eine Parameterbeschreibung ist dort gegeben. Die Parameterbeschreibungen enthalten Beschreibungen der erfassten Werte sowie Beschreibungen zu den Metadaten der Stationen. Im Datenarchiv werden die Zip-Dateien mit stündlichen Werten zu Windgeschwindigkeiten und Sonnenstunden benötigt, um anschließend die Auswertung der Daten zu ermöglichen.

[6] `ftp://ftp-cdc.dwd.de/pub/CDC/observations_germany/climate/`

6.1.2. Winddaten

Die Wind-Daten des Deutschen Wetterdienstes werden deutschlandweit an diversen Wetterstationen erfasst. Relevant sind dabei vor allem die Produkt-Textdateien, die für die Auswertung relevante Informationen erhalten. Die Datei enthält dabei jeweils folgende Parameter:

- STATIONS_ID - Stationsidentifikationsnummer
- MESS_DATUM_ENDE - Intervallende im Format YYYYMMDDHH
- QN_3 - Qualitätsniveau
- F - mittlere Windgeschwindigkeit in m/s
- D - mittlere Windrichtung in Grad
- eor - Ende data record

Fehlerwerte sind mit -999 gekennzeichnet. Das Qualitätsniveau ist per default auf QN_3 gesetzt und bedeutet älte automatische Prüfung und Korrektur. Im Laufe der Zeit wird das Qualitätsniveau der Messungen überarbeitet, welches allerdings für diesen Anwendungsfall nicht relevant ist. Wichtig sind dabei die Stationsidentifikationsnummer zur geografischen Zuordnung, das Datum und die Uhrzeit sowie die Windgeschwindigkeit und die Richtung.

6.1.3. Sonnendaten

Die Sonnen-Daten des Deutschen Wetterdienstes liegen ähnlich wie der Wind-Daten auf dem FTP-Server in Zip-Dateien vor. Diese Dateien enthalten die Stationsbeobachtungen als Produkt-Textdatei sowie die zugehörigen Metadaten. Dabei ist jedoch lediglich die Produkt-Textdatei relevant, da diese die zur Station gehörigen Beobachtungen den Sonnenstunden beinhaltet. Die Datei enthält die folgenden Parameter:

- STATIONS_ID - Stationsidentifikationsnummer
- MESS_DATUM - Messdatum im Format YYYYMMDDHH
- QN_7 - Qualitätsniveau
- SD_SO - Stündliche Sonnenscheindauer
- eor - End data record

Ähnlich wie bei den Winddaten sind die Fehlerwerte mit -999 gekennzeichnet. Das Qualitätsniveau ist bei diesem Beispiel auf 7 gesetzt, was bedeutet, dass die Aufzeichnung die zweite Prüfung durchlaufen hat und der Wert noch korrigiert werden könnte.

6.1.4. Qualitätsniveau der Aufzeichnungen

In den Produkt-Textdateien steht in den Parametern jeweils der Indikator für das Qualitätsniveau (QN). Dieser kann je nach Aufzeichnung einen unterschiedlichen Wert haben. In den zuvor genannten Beispielen hat das Qualitätsniveau den Wert QN_3 bzw. QN_7. Die vollständige Auflistung des Qualitätsniveaus sieht folgendermaßen aus:

- 1 - nur formale Prüfung
- 2 - nach individuellen Kriterien geprüft
- 3 - alte automatische Prüfung und Korrektur
- 5 - historische, subjektive Verfahren
- 7 - 2. Prüfung durchlaufen, vor Korrektur
- 8 - Qualitätssicherung außerhalb Routine
- 9 - nicht alle Parameter korrigiert
- 10 - Qualitätsprüfung abgeschlossen, Korrekturen beendet

Wie in der Auflistung zu erkennen ist, können die Werte diverse Prüfungen durchlaufen, um als vollständig korrekt anerkannt zu werden. Das Qualitätsniveau von 1-10 ist also steigend, sodass Werte mit QN_10 die beste Qualität haben. Somit werden die erfassten Werte nach dem Upload auf den FTP ggf. noch geändert und das Qualitätsniveau angepasst. Die Änderung benötigt allerdings eine gewisse Zeit, sodass dafür keine Berücksichtigung stattfinden kann und die täglich erfassten Werte unabhängig vom Qualitätsniveau importiert werden müssen.

6.1.5. Smart-Meter-Daten

Als Smart-Meter Daten werden Daten aus einer Niederspannungs-Simulation herangezogen, die selbst über ein Simulations-Framework erstellt werden und darüber hinaus durch die Kooperation mit eStream über eine Schnittstelle im System abgelegt werden. Die Simulation ist Teil des Mosaik-Frameworks zur Generierung von Smart-Meter-Daten in einem Smart-Grid bestehend aus Smart-Meter-Gateways und Smart-Metern. Dabei wird die automatische Erstellung einer Smart-Grid Simulation für verschiedenste Szenarien ermöglicht, die sich beliebig komplex gestalten lassen. Der Fokus dabei wird auf die generierten Messdaten gelegt, die in der Ausgabe der Simulation die Energieverbräuche der Haushalte in einem Grid darstellen. Diese Daten sind für spätere Analysezwecke für Aussagen über prognostizierte Energieverbäuche notwendig. Die erzeugten Ausgaben wurden eingangs im JSON-Format ausgegeben und sind nach einer Umstellung des System als COSEM-formatierte XML-Dateien abrufbar. Topologisch werden die einzelnen Messwerte

der Haushalte mit IDs in dem Format den Smart-Metern zugeordnet, welche wiederum den Smart-Meter-Gateways zugeordnet sind. Somit wird in den Ausgaben jeweils die Zuordnung der Messwerte im jeweiligen Smart-Grid erkenntlich gemacht.

6.1.6. SMARD-Daten

SMARD stellt Strommarktdaten für Deutschland und Europa in Echtzeit zur Verfügung. Dabei werden Daten wie Erzeugung, Verbrauch, Import und Export und Daten zur Regelenergie für unterschiedliche Zeiträume in grafischer und tabellarischer Form bereitgestellt. Für die Anwendungszwecke der Datenanalyse werden die Daten zur Energieerzeugung und für den Verbrauch herangezogen. Dabei kann sowohl auf realisierte, als auch auf prognostizierte Daten zurückgegriffen werden. Vor allem die realisierten Daten stehen für Vergleichszwecke bei der Datenanalyse im Fokus. Dazu werden in 15-minütigen Abständen die jeweiligen Daten in MWh aufgelistet. Die Daten veranschaulichen einerseits den gesamten Energieverbrauch sowie die gesamte Energieerzeugung, lassen aber auch bei der Erzeugung eine Klassifizierung nach den jeweiligen Energieressourcen zu. So lassen sich zu Vergleichszwecken Werte über Wind- und Sonnenenergie heranziehen, die im DAvE-System abgelegt werden.

6.2. Archivierung

In diesem Abschnitt wird die Umsetzung der Archivierungskomponente dargestellt. Dabei wird zuerst auf die Schemadefinition eingegangen. Danach folgt der Datenimport und Datenexport, in dem gezeigt wird, wie die Daten der verschiedenen Datenquellen durch Apache NiFi in Apache Cassandra abgespeichert bzw. abgerufen werden.

6.2.1. Schema-Definition

Wie in den vorherigen Kapiteln erwähnt, ist Cassandra die ausgewählte Datenbank, in der die Daten aus den Datenquellen gespeichert werden sollen. Zuerst wird in Cassandra ein Keyspace erstellt, sozusagen der äußerste Container, in denen sämtliche Tabellen mit Inhalten gespeichert werden können. Es werden bei der Ausführung ein beliebiger Name, eine class und ein replication factor benötigt. Der Name des Keyspaces ist *dave*, hat als class *simple strategy* und einen *replication factor* von 3. *Simple strategy* wird verwendet, wenn lediglich ein Datencenter benutzt wird. Bei mehreren Datencentern würde man eine *NetworkTopologyStrategy* anwenden. Der *replication factor* ist standardmäßig auf 3 eingestellt, welches für unsere Zwecke ausreicht. Dies dient zur Absicherung der Daten, um einen Verlust vorzubeugen. Die Zahl drei bedeutet in diesem Fall eine dreifache Speicherung des jeweiligen Dateninhalts. Ist der Keyspace erstellt, können die benötigten Tabellen erzeugt

werden. Für jede Datenquelle wird dazu eine individuelle Tabelle dem Keyspace hinzugefügt. Danach kann mit der Erstellung von Tabellen für die Dateninhalte begonnen werden.

```
1 CREATE KEYSPACE dave
2 WITH REPLICATION = { 'class' : 'SimpleStrategy', '
      replication_factor' : 3 };
```

Listing 19: Erzeugung des Keyspaces

DWD-Daten Die für uns relevanten Daten des deutschen Wetterdienstes sind aufgeteilt in Klimadaten, Sonnendaten und Winddaten, die jeweils eine eigene Tabelle erhalten. In diesen Tabellen sollen die zur Verfügung gestellten historischen und aktuellen Daten zusammengeführt werden. Damit die Wetterstationen der DWD-Daten lokalisiert werden können, muss zusätzlich noch eine Tabelle mit den gegebenen Informationen zur Beschreibung dieser erstellt werden. Es muss zudem der Primary Key der Tabelle vorher gut ausgewählt sein, um die Tabellen später übersichtlicher zu halten. Dazu haben wir für jede Tabelle einen Partition Key, der zur Datenverteilung über die Nodes hinweg verantwortlich ist, und einen Clustering Key, um die Daten sortieren zu können, ausgesucht. In diesem Fall wird die Stations ID als Partition Key und das dazugehörige Messdatum als Clustering Key verwendet. Dies dient dazu, um nach der Stations ID abfragen und dem Messdatum sortieren zu können. Auch der Tabellenname wird vorher bewusst gewählt, um diese später besser zu identifizieren.

```
# Klimadaten
CREATE TABLE dwd_climate_ger_daily(MESS_DATUM date,
    STATIONS_ID text, QN_3 text, FX text, FM text, QN_4 text
    , RSK text, RSKF text, SDK text, SHK_TAG text, NM text,
    VPM text, PM text, TMK text, UPM text, TXK text, TNK
    text, TGK text, eor text, PRIMARY KEY (STATIONS_ID,
    MESS_DATUM)) WITH CLUSTERING ORDER BY (MESS_DATUM DESC);

# Sonnendaten

CREATE TABLE dwd_sun_ger_hourly(STATIONS_ID text, MESS_DATUM
     timestamp, QN_7 text, SD_SO text, eor text, PRIMARY KEY
     (STATIONS_ID, MESS_DATUM)) WITH CLUSTERING ORDER BY (
    MESS_DATUM DESC);

# Winddaten

CREATE TABLE dwd_wind_ger_hourly(STATIONS_ID text,
    MESS_DATUM timestamp, QN_3 text, F text, D text, eor
    text, PRIMARY KEY (STATIONS_ID, MESS_DATUM)) WITH
    CLUSTERING ORDER BY (MESS_DATUM DESC);

# Stationsbeschreibung

CREATE TABLE stations_ger_description(stations_id text,
    stationshoehe text, geobreite text, geolaenge text,
    stationsname text, bundesland text, PRIMARY KEY (
    bundesland, stations_id, stationsname, geobreite,
    geolaenge, stationshoehe)) WITH CLUSTERING ORDER BY (
    stations_id ASC, stationsname ASC, geobreite asc,
    geolaenge asc, stationshoehe asc);
```

Listing 20: Erstellung der Tabellen für DWD-Daten

SMARD-Daten Bei den SMARD-Daten wurde ähnlich wie bei den DWD-Daten vorgegangen. Hier werden zwei Tabellen für die Stromerzeugung und den Stromverbrauch erstellt. Der Primary Key ist in diesem Fall das Datum und die Uhrzeit.

```
CREATE TABLE dave.smard_stromerzeugung_quarter (
datum date,
uhrzeit text,
biomasse text,
braunkohle text,
erdgas text,
kernenergie text,
photovoltaik text,
pumpspeicher text,
sonstige_erneuerbare text,
sonstige_konventionelle text,
steinkohle text,
wasserkraft text,
wind_offshore text,
wind_onshore text,
PRIMARY KEY (datum, uhrzeit)
) WITH CLUSTERING ORDER BY (uhrzeit ASC)

CREATE TABLE dave.smard_stromverbrauch_quarter (
datum date,
uhrzeit text,
verbrauch text,
PRIMARY KEY (datum, uhrzeit)
) WITH CLUSTERING ORDER BY (uhrzeit ASC)
```

Listing 21: Erstellung der Tabellen für SMARD-Daten

LV-Daten Die Simulationsdaten des Projekts *Netzdatenstrom* werden ebenfalls in Cassandra abgespeichert. Hierfür wurde eine Tabelle erzeugt. Der Primary Key ist in diesem Fall die Spalte *smartmeter* und *timestamp*.

```
CREATE TABLE dave.nds_sim_XML (
smartmeter text,
timestamp bigint,
scaler text,
status text,
unit text,
value text,
logical_name text,
PRIMARY KEY (smartmeter, timestamp)
) WITH CLUSTERING ORDER BY (timestamp DESC);
```

Listing 22: Erstellung der Tabelle für LV-Daten

6.2.2. Datenimport

Im Folgenden werden nun die einzelnen Schritte erläutert, die in Apache NiFi für den Datenimport getätigt wurden. Um in NiFi Daten importieren zu können, müssen unterschiedliche Prozessoren sowie die erstellten Tabellen aus Cassandra genutzt werden. Zuerst wird der Prozess des Imports von DWD-Daten erläutert. An diesem wird genauer gezeigt, welche Prozessoren genutzt wurden und welche Konfigurationen durchgeführt werden mussten. Danach folgen noch die Konfigurationen des Imports der SMARD- und DWD-Daten. Dabei werden nicht mehr die kompletten Prozesse gezeigt, sondern die wichtigsten Einstellungen und individuellen Änderungen der einzelnen Prozessoren.

DWD-Daten-Import Da der deutsche Wetterdienst die Wetterdaten in historische und neueste Daten aufgeteilt hat, wurde auch der Import danach gegliedert. So wurden zuallererst die historischen Daten importiert. Dadurch mussten für die Sonnen-, Wind und Klimadaten Prozesse in Apache NiFi erstellt werden.

DWD-Daten Historisch Wir zeigen nun den kompletten Prozess des Imports der historischen Sonnendaten und geben ergänzend – falls Unterschiede vorhanden – die Einstellungen für die Wind- und Klimadaten an. Per Drag&Drop können die einzelnen Prozessoren auf die Arbeitsfläche abgelegt werden. Für das Herunterladen der Daten auf unseren Server werden drei Prozessoren genutzt. Der erste Prozessor ist der ListFTP Prozessor, der verwendet wird, um auf den FTP-Server des deutschen Wetterdienstes zuzugreifen und damit eine Liste, der dort sich befindenden Dateien, erstellt wird. In diesem wurden bei den Properties der Hostname, Port, Username und der Remote Path angepasst (siehe Abbildung).

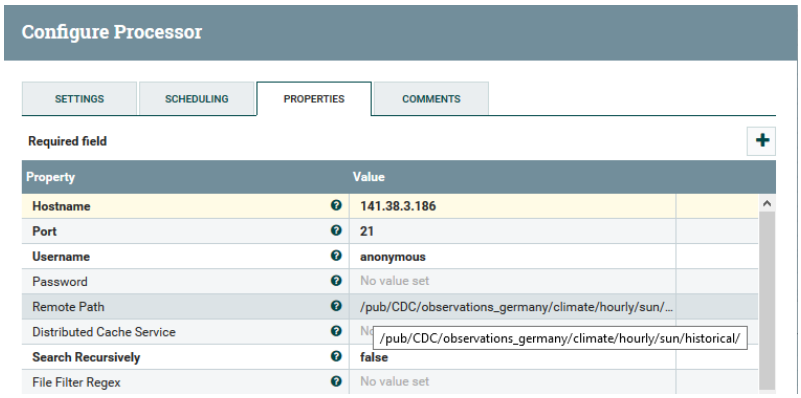

Abbildung 12: Einstellungen ListFTP

Für die Wind- und Klimadaten werden jeweils andere Pfade des FTP-Servers angegeben, die folgendermaßen aussehen:

```
1 # Winddaten
2 /pub/CDC/observations_germany/climate/hourly/wind/historical
    /
3 # Klimadaten
4 /pub/CDC/observations_germany/climate/daily/kl/historical/
```

Listing 23: Pfade für den FTP-Server historische Daten

Da wir nicht alle Dateien des angegebenen Pfades importieren wollen, wird der RouteOnAttribute Prozessor verwendet. In diesem Prozessor muss eine Property hinzugefügt werden, damit nach bestimmten Dateien gefiltert werden kann. In diesem Fall wurde die selbsternannte Eigenschaft *filetofetch* hinzugefügt, damit nur die ZIP-Dateien mit den relevanten Daten heruntergeladen werden.

Abbildung 13: Einstellung RouteOnAttribute

Für die Winddaten bleibt diese Einstellung gleich. Die Klimadaten hingegen müssen nach *tageswerte_ KL* gefiltert werden. Der RouteOnAttribute Prozessor wird mit dem FetchFTP Prozessor verbunden, damit die vorher angelegte Liste mit den ausgewählten Daten gefüllt werden können. Dazu werden bei den Properties wieder der Hostname, Port, Username und der Remote File angegeben. Beim Remote File wird dazu lediglich *$path/$filename* eingegeben. Falls es in diesem Prozessor zu einem fehlerhaften FlowFile kommt, soll dieser ein weiteres Mal den Prozessor durchlaufen. Dies wird mit einem selbstgerichteten Pfeil auf dem Prozessor gelöst. Der erste Teil des Datenimports sieht nun folgendermaßen aus:

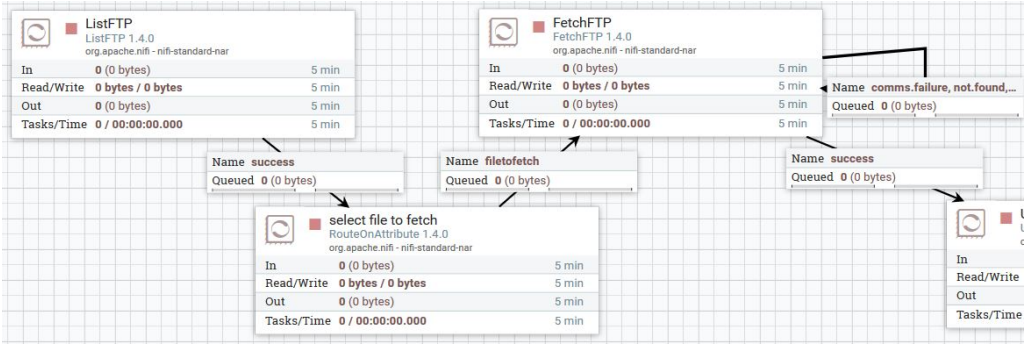

Abbildung 14: 1. Teil DWD-Import

Nachdem die Dateien auf dem Server sind, werden diese nun im UnpackContent Prozessor entpackt. Hier muss ausgewählt werden, welche Formate entpackt werden müssen. In unserem Fall gibt es nur ZIP-Dateien. Danach gilt es die nicht benötigten Dateien mittels eines RouteOnAttribute Prozessors rauszufiltern. In den entpackten Ordnern befinden sich Dateien mit Metadaten, die für die Archivierung nicht mehr benötigt werden. Dazu muss in den Properties eine Eigenschaft hinzugefügt werden. Diese Eigenschaft haben wir *filename* benannt und besagt, dass jeder Dateiname mit dem Inhalt *Metadaten* nicht weitergeleitet werden soll.

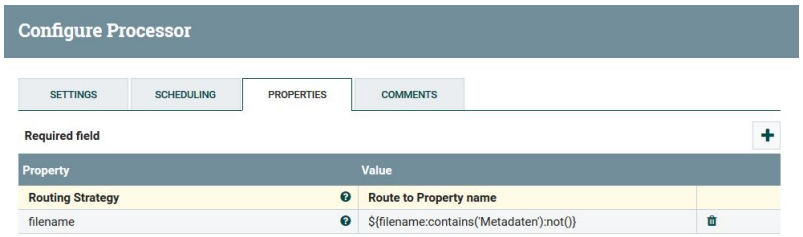

Abbildung 15: Einstellung Entfernen der Metadaten

Im nächsten Schritt wird die herausgefilterte CSV-Datei in ein JSON-Format umgewandelt. Dazu benötigen wir den ConvertRecord Prozessor. Dieser beinhaltet in den Properties einen *Record Reader* und einen *Record Writer*. Für den *Value* dieser Property muss ein Service erstellt werden (siehe Abbildung). Für den *Record Reader* haben wir als Type *CSV-Reader* ausgewählt, in dem eingestellt wird, wie das Schema der CSV-Datei aussehen soll. Hier wird beispielsweise eingestellt, welches Trennzeichen innerhalb des Dokumentes gelten und dass die erste Zeile als Header angezeigt werden soll. In unserem Fall wurde ein Semikolon als Trennzeichen und *Treat First Line as Header* auf *true* gesetzt. Dieser Service wurde *HistoricalCSVReader_without_schema* genannt. Zudem wurde noch ein *RecordWriter* erstellt, dessen Typ *JsonRecordSetWriter* ist. Hier wird lediglich in den Properties die *Schema Write Strategy* in *set 'avro.schema' Attribute* geändert. Diesen *RecordWriter* haben wir *HistoricalJsonRecordSetWriter_without_schema* genannt.

Abbildung 16: Einstellung ConvertRecord

Danach wird ein SplitJson Prozessor angebunden, der das FlowFile in einzelne Zeilen splitten soll, damit diese später problemlos in Cassandra gespeichert werden können. Hierzu muss in den Properties die *JsonPath Expression* in *$.** geändert werden, da der Inhalt des Files eine einfache Struktur aufweist und nicht geschachtelt ist. Da im FlowFile noch Zellen sind, die als Wert -999 aufweisen, wird in einem ReplaceText Prozessor der Inhalt in *Null* geändert. Dafür muss zusätzlich der Wert *Literal Replace* bei der *Replacement Strategy* ausgewählt werden. Der zweite Teil des Datenimports ist in der folgenden Abbildung zu sehen.

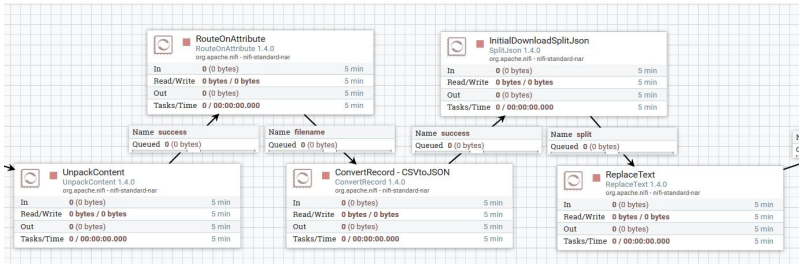

Abbildung 17: 2.Teil DWD-Import

Für den letzten Teil müssen die Daten für den Import in Cassandra vorbereitet werden. Dazu benötigen wir den EvaluateJsonPath Prozessor, damit wir den Inhalt der FlowFiles bestimmten Attributen zuordnen können. In den Properties wird der Wert bei der Einstellung der Destination auf `FlowFile-attribute` gesetzt, damit Apache NiFi neue Attribute erstellen kann. Hierzu können die JsonPath-Ausdrücke verwendet werden, um die Werte des FlowFiles zu extrahieren. Wenn dieser Prozessor durchlaufen wurde, hat das FlowFile die angelegten Attribute hinzugefügt. Die folgende Abbildung zeigt die fünf Attribute der Sonnendaten. Für die Wind- und Klimadaten werden die Attribute entsprechend erstellt. Auch hierbei werden die Spaltennamen der Ausgangstabelle genommen, um an die Werte des FlowFiles zu kommen.

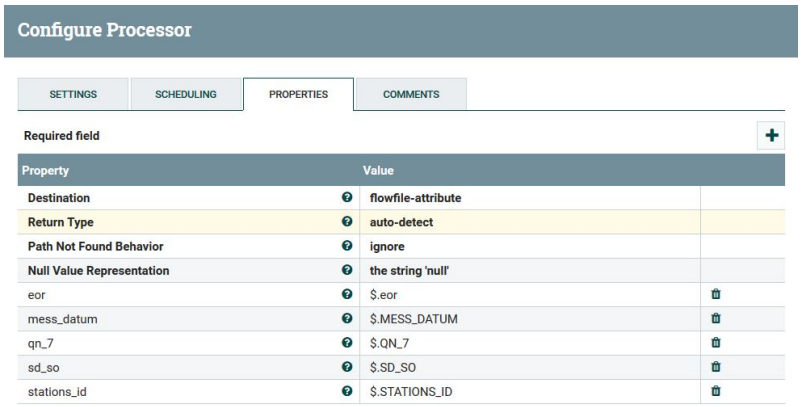

Abbildung 18: Einstellung EvaluateJsonPath

Als nächstes wird das Messdatum einem gewünschten Format angepasst. In einem UpdateAttribute Prozessor fügen wir den Properties einen weiteren Eigenschaft hinzu und nennen diesen *date*. Der Wert dazu ändert das Format für die Sonnen- und Winddaten nach unseren Wünschen um (siehe Abb. 20).

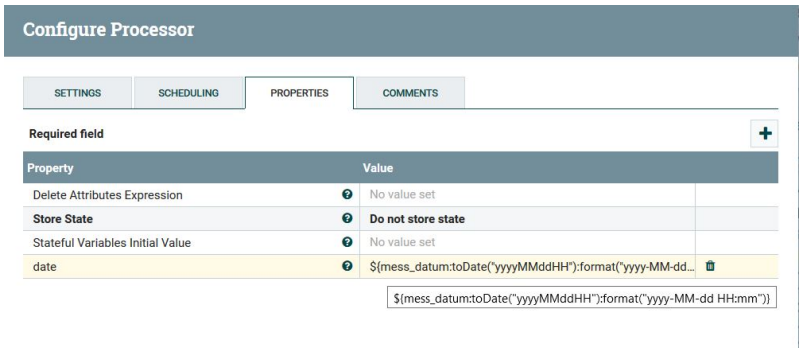

Abbildung 19: Anpassung Messdatum

Da die Klimadaten nur täglich aktualisiert werden, benötigen wir bei der Formatierung des Zeitstempels keine Stundenangabe. Deswegen muss die Angabe bei den Klimadaten folgendermaßen aussehen:

```
1 # Klimadaten
2 date: ${mess_datum:toDate("yyyyMMdd"):format("yyyy-MM-dd")}
```

Listing 24: Zeitstempel Klimadaten

Nach diesem Prozessor ist das FlowFile soweit, dass es in Cassandra eingespeichert werden kann. Dazu benötigen wir zuerst den ReplaceText Prozessor, um ein *CQL INSERT Statement* zu erstellen, und den PutCassandraQL Prozessor für die Verbindung zu Cassandra. Das hinzugefügte CQL-Statement sieht je nach Datensatz folgendermaßen aus:

```
# Sonnendaten
insert into dwd_sun_ger_hourly(STATIONS_ID, MESS_DATUM, QN_7
    , SD_SO, eor ) values ('${stations_id}', '${date}', '${
    qn_7}', '${sd_so}', '${eor}' )

# Winddaten
insert into dwd_wind_ger_hourly(STATIONS_ID, MESS_DATUM,
    QN_3, F, D, eor ) values ('${stations_id}', '${date}', '
    ${qn_3}', '${f}', '${d}', '${eor}' )

# Klimadaten
insert into dwd_climate_ger_daily(STATIONS_ID, MESS_DATUM,
    QN_3,  FX,  FM,QN_4, RSK,RSKF, SDK,SHK_TAG,  NM, VPM,
    PM, TMK, UPM, TXK, TNK, TGK,eor ) values ('${stations_id
    }', '${date}', '${qn_3}', '${fx}', '${fm}', '${qn_4}', '
    ${rsk}', '${rskf}', '${sdk}', '${shk_tag}', '${nm}', '${
    vpm}', '${pm}', '${tmk}', '${upm}', '${txk}', '${tnk}',
    '${tgk}', '${eor}')
```

Listing 25: CQL INSERT Statements

In dem PutCassandraQL Prozessor werden lediglich der Username `cassandra`, das Passwort `cassandra`, der keyspace `dave` und die Cassandra Contact Points (127.0.0.1:9042) eingestellt. Zur Fehlervermeidung lassen wir fehlerhafte FlowFiles mittels eines selbstgerichteten Pfeiles noch einmal den Prozessor durchlaufen. Sind alle Prozessoren miteinander verbunden, kann der gesamte Prozess mit dem Startbutton ausgeführt werden. Es ist zudem möglich, diesen jederzeit zu stoppen, den Status einzusehen und gegebenenfalls in die einzelnen FlowFiles zu schauen. Der dritte Teil des Datenimports sieht nun folgendermaßen aus:

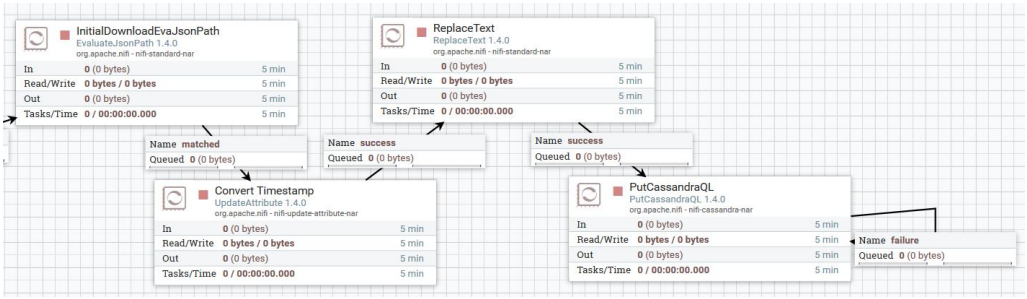

Abbildung 20: 3.Teil DWD-Import

Aktuelle DWD-Daten Nachdem die historischen Daten gespeichert wurden, werden nun die aktuellen Wetterdaten der Tabelle hinzugefügt. Dazu wurde der gleiche Prozess wie vorhin beschrieben, für die Sonnen-, Wind-, und Klimadaten genutzt. Der einzige Unterschied zwischen den Prozessen ist der Pfad, von wo die Daten abgerufen werden sollen. Da aber

auch die Daten des Deutschen Wetterdienstes täglich aktualisiert werden und diese in unserer Datenbank ebenfalls aktuell gehalten werden sollen, benötigen wir zusätzlich zu dem einmaligen Import der historischen und der aktuellen Daten einen immer wiederkehrenden Import der neuen Daten. Dieser Prozess ist nahezu identisch zu dem oben beschriebenen historischen Datenimport. Der wesentliche Unterschied besteht darin, dass aufgrund des Filterns ein zusätzlicher Prozessor hinzugefügt wird, einige Einstellung geändert werden müssen und der gesamte Prozess so eingestellt wird, dass dieser zu einer bestimmten Uhrzeit täglich neu aktiviert wird. Wie diese tägliche Aktivierung genau funktioniert wird im letzten Abschnitt erläutert.

Für die genaue Konfiguration des Prozesses empfiehlt sich den oberen Abschnitt zu den historischen Datenimport begleitend mit der nachfolgenden Beschreibung anzusehen. Damit dieser für die aktuellen Daten komplettiert werden kann, sind dazu noch einige kleine Änderungen vorzunehmen. Zur genaueren Einordnung wird der Prozess nochmals kurz beschrieben.

Wie bei der oberen Beschreibung teilen wir den Gesamtprozess wieder in drei Teilprozesse und erklären anhand dessen die zu den Sonnen-, Wind- und Klimadaten verschiedenen Einstellungen. Zu Beginn des Prozesses ist die einzige Änderung zu dem Import der historischen Daten die Datenquelle der aktuellen Daten. Da die DWD-Daten, wie oben beschrieben, in historische und aktuelle Daten aufgeteilt werden müssen, benötigen wir einen anderen Pfad in den Properties des ListFTP Prozessors. Dieser ist sowohl für die Sonnen-, Wind- als auch Klimadaten unterschiedlich.

```
1 # Sonnendaten
2 /pub/CDC/observations_germany/climate/hourly/sun/recent/
3 # Winddaten
4 /pub/CDC/observations_germany/climate/hourly/wind/recent/
5 # Klimadaten
6 /pub/CDC/observations_germany/climate/daily/kl/recent/
```

Listing 26: Pfade für den FTP-Server aktuelle Daten

Die Konfiguration der anderen Prozessoren im ersten Teil bleiben identisch, sodass nach Abschluss des FetchFTP Prozessors die gewünschten Daten erfolgreich auf dem Server gespeichert wurden, um dann damit weiterarbeiten zu können.

Im zweiten Teil sind die Prozesse und auch die Einstellungen je nach Datenart wie bei dem historischen Datenimport. Hier geht es darum, dass die Daten, nachdem sie entpackt wurden, in das JSON Format gewandelt werden, um diese dann in einzelne Zeilen zu splitten, damit sie später problemlos in unsere Tabelle in Cassandra eingefügt werden können. Nach dem Teilen in einzelne Zeilen müssen noch wiederkehrende Werte, die Nullwerte aussagen (hier: -999) durch *Null* ersetzt werden. Im letzten Teil muss nun ein Prozessor hinzugefügt werden, der dafür sorgt, dass die Zeile mit dem Datum eines davorliegenden Tages importiert wird. Da die Klimadaten täglich, die Sonnen- und Winddaten stündlich aufge-

nommen und täglich auf dem FTP-Server des DWD hochgeladen werden, wollen wir diese in unserer Datenbank aktuell halten. Dazu benötigen wir zwischen dem *EvaluateJsonPath* Prozessor und dem Konvertieren des Zeitstempels einen Prozessor, der ein Attribut hinzufügt, mit dem sich der Datensatz nach einem bestimmten Datum filtern lässt. Dieser Prozessor bewirkt, dass das FlowFile nur die Daten weiterschickt, die als Messdatum einen vorherigen Tag aufweisen. Da auf dem FTP-Server die Daten des davorliegenden Tages täglich zur Mittagszeit hochgeladen werden, wir aber bereits in der Nacht aktualisieren, benötigen wir zwei Tage vor der Systemzeit, welches durch das Subtrahieren von 172800000 Sekunden von der Systemzeit realisiert wird. Da wir die aktuellen Datensätze den bereits importierten historischen Daten hinzufügen, verläuft das Einfügen der Daten in Cassandra dann wieder identisch zu den historischen Daten, weshalb die letzten beiden Prozessoren gleich konfiguriert bleiben können. Nachdem die Prozessoren nun so eingestellt sind, dass die einzelne Zeile des vorherigen Tages in die Datenbank gespeichert wird, benötigen wir noch die Einstellung, dass die Prozessoren täglich aktiviert werden. Dazu gibt es in jedem Prozessor die Möglichkeit eine bestimmte Startzeit einzustellen. Standardmäßig wird jeder Prozessor auf eingehende Flowfiles reagieren. Es reicht in unserem Fall also aus, wenn nur der erste Prozessor (hier: ListFTP) aktiviert wird, der dann beginnt, das Flowfile an den nächsten Prozessor zu schicken. Ab da wird der ganze Prozess automatisch durchlaufen, bis die gesamte Datenmenge gespeichert ist. Unter dem Reiter *Scheduling* im ListFTP-Prozessor kann mittels der Funktion *CRON* dem Prozessor ein Startzeitpunkt angegeben werden. CRON ist eine zeitbasierte Ausführung von Prozessen in Unix und unixbasierten Betriebssystemen und dient genau für unsere Zwecke. Der Prozess der Klimadaten startet um 0 Uhr, die Sonnendaten um 2 Uhr (siehe folgende Abb.) und die Winddaten um 4 Uhr. Der gewählte Zeitraum ist aufgrund der Vermeidung von Engpässen auf dem Server für die jeweiligen Sonnen, Wind und Klimadaten über die Nacht verteilt. Damit diese Funktion in NiFi aber auch greifen kann, müssen alle Prozessoren im gesamten Prozess bereits gestartet sein und durchgängig laufen.

```
1 ${mess_datum:contains("${now():toNumber():minus(172800000):
      format('yyyyMMdd')}")}
```

Listing 27: Filtern des Messdatums nach bestimmten Datum

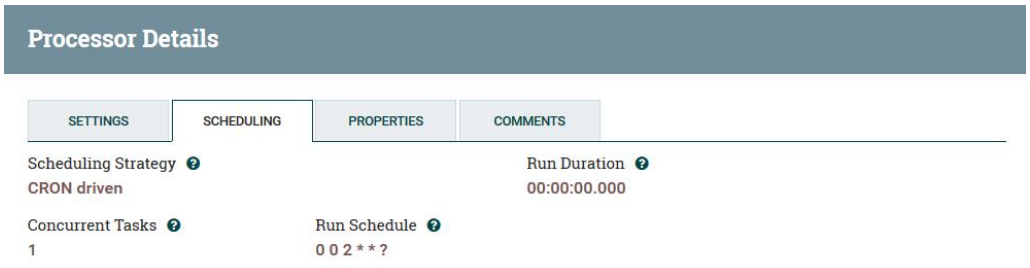

Abbildung 21: Scheduling Strategy

SMARD-Daten Der Import der SMARD-Daten ist ähnlich aufgebaut wie die Prozesse der DWD-Daten. Da aber die Webseite der Bundesnetzagentur[7] keinen FTP-Server besitzt, werden die beiden Dateien manuell auf den Server heruntergeladen. Es wurden hier zwei Prozesse erstellt, die die selben Prozessoren verwenden. So wurden die letzten zwei Jahre (01.01.2016 - 01.01.2018) des realisierten Stromverbrauchs und der Stromerzeugung in Deutschland in Cassandra abgespeichert. Damit die Daten in Cassandra archiviert werden können, muss von NiFi die CSV-Datei abgerufen werden. Dies geschieht durch den GetFile Prozessor. In diesem kann der Pfad des Servers sowie der Dateiname angegeben werden. Die folgende Abbildung zeigt die Konfiguration des Prozessors für den Stromverbrauch. Für den Prozess der Stromerzeugungsdaten wurde nur der *File Filter* abgeändert, um an die andere CSV-Datei zu gelangen.

Abbildung 22: Konfiguration GetFile Prozessor

Da die Daten nicht entpackt werden müssen, folgt nach dem GetFile Prozessor direkt die Umwandlung der Datei von JSON zu CSV und anschließend das Splitten in einzelne Zeilen. Hier sind die Prozessoren und Einstellungen die gleichen wie beim DWD-Datenimport. Die restlichen Schritte wie die Vorbereitung der Daten, das Umwandeln des Zeitstempels, die Erstellung des CQL-Statements und das Verbinden zur Datenbank benutzten auch die gleichen Prozessoren wie beim Import der Wetterdaten, allerdings müssen diese anders konfiguriert werden. Die folgende Abbildung zeigt die Konfiguration für den EvaluateJsonPath Prozessor. Für die Daten der Erzeugung müssen in diesem Prozessor andere Attribute angegeben werden.

[7] https://www.smard.de/blueprint/servlet/page/home/downloadcenter/download_marktdaten/

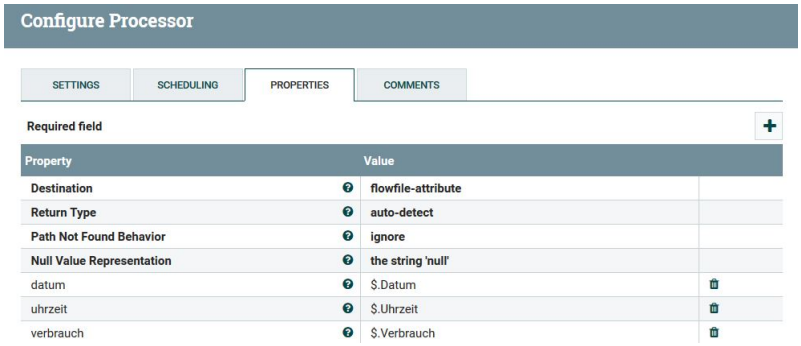

Abbildung 23: Konfiguration EvaluateJsonPath Verbrauch

Abbildung 24: Konfiguration EvaluateJsonPath Erzeugung

Das Anpassen des Messdatums wird auch hier durch ein UpdateAttribute Prozessor durchgeführt. Die Konfiguration ist bei den beiden Prozessen gleich.

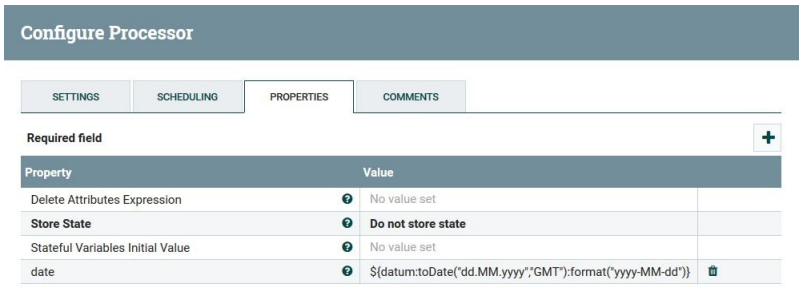

Abbildung 25: Anpassen des Datums bei SMARD-Daten

Die Konfiguration des folgenden ReplaceText Prozessor, der das CQL-Statement beinhaltet, ist wiederum unterschiedlich, da für den Stromverbrauch und der Stromerzeugung

jeweils eine Tabelle erstellt wurde und diese unterschiedliche Attribute besitzen.

```
# Verbrauch
insert into smard_stromverbrauch_quarter(Datum, Uhrzeit,
    Verbrauch) values ('${date}', '${uhrzeit}', '${verbrauch
    }')

# Erzeugung
insert into smard_stromerzeugung_quarter(Datum, Uhrzeit,
    Biomasse, Wasserkraft, Wind_Offshore, Wind_Onshore,
    Photovoltaik, Sonstige_Erneuerbare, Kernenergie,
    Braunkohle, Steinkohle, Erdgas, Pumpspeicher,
    Sonstige_Konventionelle) values ('${date}', '${uhrzeit
    }', '${biomasse}', '${wasserkraft}','${windoffshore}','$
    {windonshore}','${photovoltaik}','${sonstige_erneuerbare
    }','${kernenergie}','${braunkohle}','${steinkohle}','${
    erdgas}','${pumpspeicher}', '${sonstige_konventionelle
    }')
```

Listing 28: CQL INSERT Statements SMARD

Zum Abschluss wird beiden Prozessoren wieder der PutCassandraQL Prozessor verwendet, um eine Verbindung zu Datenbank herzustellen. Diese besitzt die selben Einstellungen wie beim DWD-Datenimport.

Smart-Meter-Daten Import Der in Abbildung 26: Dataflow: Import der Smart-Meter-Daten Dataflow zeigt den Import von Smart-Meter-Daten in Cassandra auf.

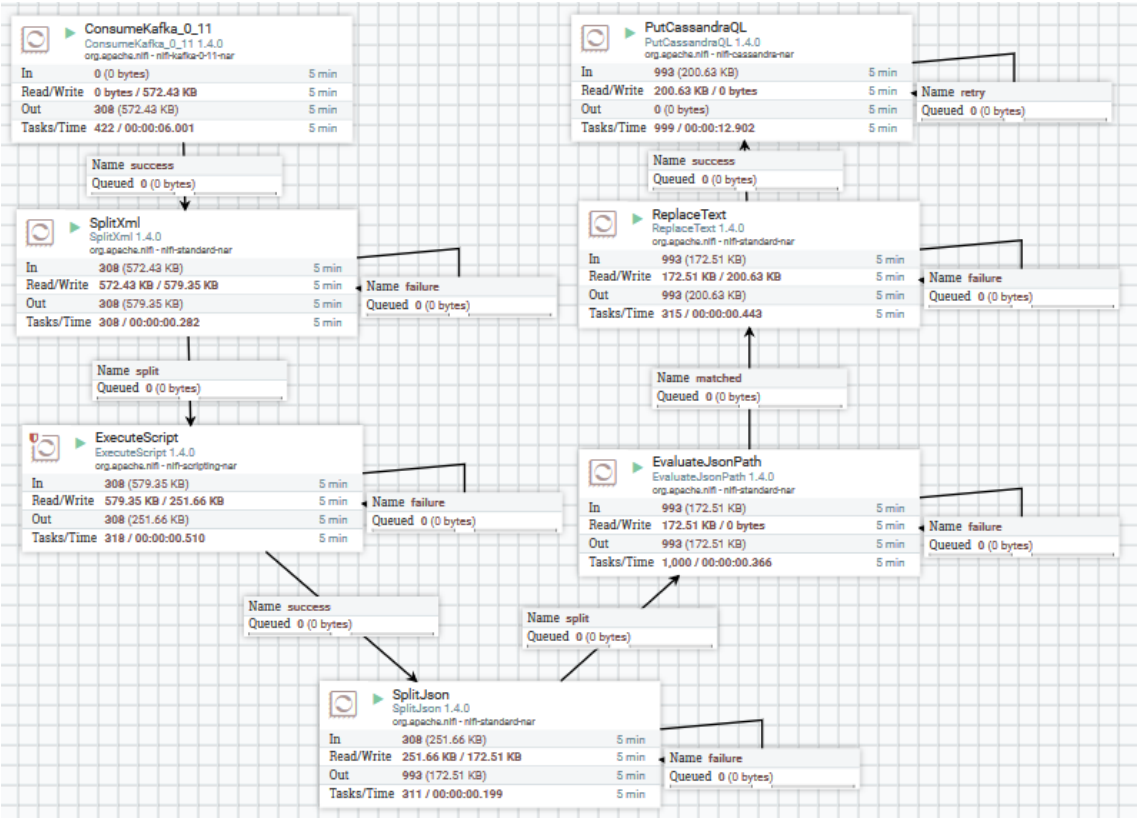

Abbildung 26: Dataflow: Import der Smart-Meter-Daten

Zuerst werden die Daten, welche in einem Kafka Topic hinterlegt sind abgerufen. Für den Abruf der Daten aus einem Kafka Topic wird der Prozessor *ConsumeKafka_0_11* verwendet. Folgende Prozessor Eigenschaften (properties) sind anzupassen:

- **Kafka Brokers:** Die URI zu den abzurufenden Kafka Brokern z. B. 134.106.56.15:29092 (Kafka Broker der Partnergruppe E-Stream)
- **Topic Name(s):** Die Namen der abzurufenden Kafka Topics
- **Group ID:** Diese werden genutzt, um die Kafka Consumer zu identifizieren

Der entsprechende Konfigurationsdialog ist in Abbildung 27: Import der Smart-Meter-Daten: ConsumeKafka Konfiguration dargestellt.

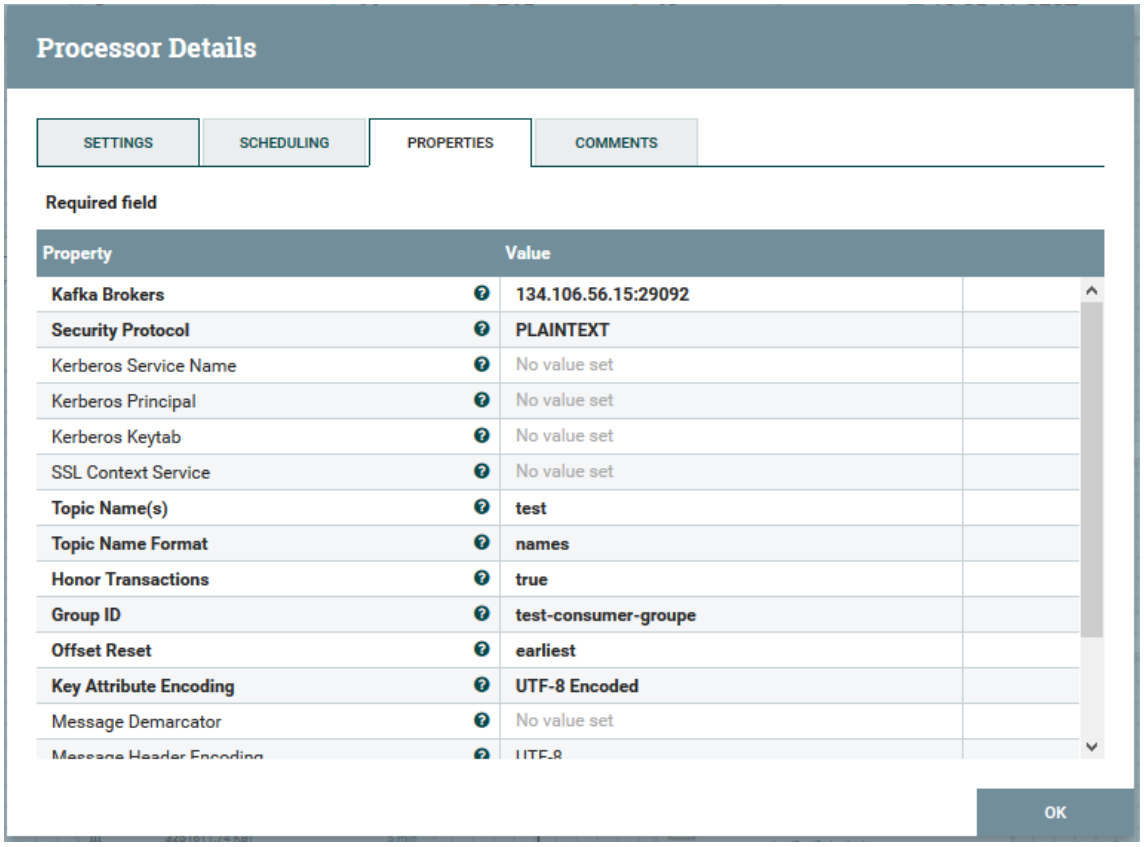

Abbildung 27: Import der Smart-Meter-Daten: ConsumeKafka Konfiguration

Smart-Meter-Daten werden in einer COSEM Struktur, welches eine XML Struktur vorweist, übermittelt. So werden die Daten aus dem *ConsumeKafka_0_11* Prozessor an einen *SplitXML* Prozzesor übergeben und dort die einzelnen Datensätze in sperate XML Dateien (FlowFiles) geteilt. Die einzige Property, welche in diesem Prozessor zu konfigurieren ist lautet:

- **Split Depth:** Angabe der Stelle in der XML Struktur, an welcher die Datei geteilt werden soll. Der Wert *1* bedeutet, dass die XML im Wurzel Element geteilt wird.

In diesem Fall ist der Wert der *Split Depth* auf *1* gesetzt. Im Anschluss werden die, aus dem *SplitXML* Prozessor resultierenden FlowFiles, in das JSON-Format konvertiert. Hierzu wird der *ExecuteScript* Prozessor eingesetzt. Dieser Prozessor führt ein in der Programmiersprache *Groovy* geschriebenes Skript aus.

Ein wichtiger Teil des Skriptes, um es für den *ExecuteScript* Prozessor lauffähig zu machen ist der Befehl in Zeile 1 des Listings 29: ExecuteScript, Framework. Dieser Befehl ermöglicht es auf die FlowFiles innerhalb einer NiFi Session zuzugreifen. Die nachfolgenden Codezeilen

```
1 flowFile = session.get()
2 if (!flowFile) return def text = 'Hello world!'; // Cast a
      closure with an inputStream and outputStream parameter
      to StreamCallback
3  flowFile = session.write(flowFile, { inputStream,
       outputStream ->
4  text = IOUtils.toString(inputStream, StandardCharsets.UTF_8
       ) outputStream.write(toJsonBuilder(text).toPrettyString
       ().getBytes(StandardCharsets.UTF_8))
5   }as StreamCallback)
6    session.transfer(flowFile, REL_SUCCESS)
```

Listing 29: ExecuteScript, Framework

binden den Input Stream ein transformieren diesen und geben ihn als Output Streamaus , welcher im JSON Format ist. Hierzu wird die Methode `toJsonBuilder`, welche innerhalb dieses Skipts definiert ist.

In Abbildung 28: Import der Smart-Meter-Daten: ExecuteScript Konfiguration wird die entsprechende Konfiguration des *ExecuteScript* Prozessor innerhalb dieses Dataflows verbildlicht.

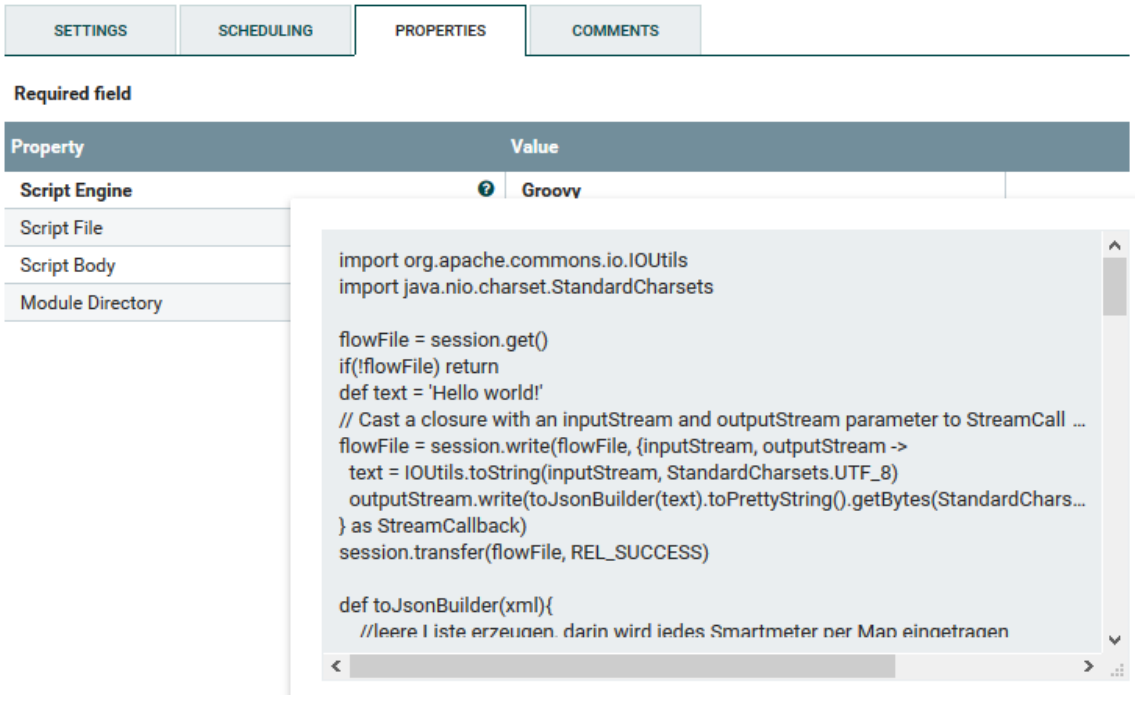

Abbildung 28: Import der Smart-Meter-Daten: ExecuteScript Konfiguration

```
insert into nds_sim_XML(smartmeter, timestamp, scaler,
    status, unit, value, logical_name) values ( '${sm}', ${
    time}, '${scaler}', '${status}', '${unit}', '${value}',
    '${logical_name}')
```

Listing 30: NiFi, Insert XML

Im nächsten Schritt werden die Arrays innerhalb der resultierenden FlowFiles in einzelne JSON Fragmente geteilt. Zu diesem Zweck wird der *SplitJson* Prozessor gebraucht, wie auch bereits in den vorherigen Dataflows des Imports beschrieben wurde. Die Property **JsonPath Expression** wurde in diesem Fall auf *$.** gesetzt.

Die resultierenden FlowFiles des *SplitJson* Prozessor werden an einen *EvaluateJsonPath* weitergeleitet. Dieser Prozessor wurde bereits in den vorangegangen Dataflows beschrieben. In Abbildung 29: Import der Smart-Meter-Daten: EvaluateJsonPath Konfiguration werden die gesetzten Attribute aufgezeigt.

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Destination	**flowfile-attribute**
Return Type	**auto-detect**
Path Not Found Behavior	**ignore**
Null Value Representation	**empty string**
logical_name	$.logical_name
scaler	$.scaler
sm	$.ldev_id
status	$.status
time	$.capture_time
unit	$.unit
value	$.value

Abbildung 29: Import der Smart-Meter-Daten: EvaluateJsonPath Konfiguration

Diese Attribute werden im Folgenden *Replace Text* Prozessor eingesetzt. Jener Prozessor wird an dieser Stelle genutzt, um das CQL Query zu definieren, mit welchem die einzelnen Datensätze in eine Cassandra Tabelle eingefügt werden. Die konfigurierte Porperty ist in diesem Fall **Replacement Value** und der Wert (value) wird mit dem zu nutzendem CQL Query gesetzt. Das genutzte Query ist in 30: NiFi, Insert XML zu sehen.

Der Dataflow endet mit einem *PutCassandraQL* Prozessor, welcher ein CQL Query erwar-

tet. Das CQL Query wird wie beschrieben vom *Replace Text* Prozessor geliefert. Hier wird konfiguriert wie der *PutCassandraQL* Prozessor eine Verbindung zu Cassandra aufbaut. Entscheidende Properties für diesen Fall sind:

- **Cassandra Contact Points:** Eine Liste der Kontaktpunkte, der anzusprechenden Cassandra Nodes
- **Keyspace:** Der anzusprechende Cassandra Keyspace
- **Client Auth:** Einstellung, wie die Client Authentifizierung der anzusprechenden Cassandra Nodes gesetzt ist
- **Username:** Nutzername für den Zugriff auf das Cassandra Cluster
- **Password:** Passwort für den Zugriff auf das Cassandra Cluster
- **Consistency Level:** Das in den anzusprechenden Tabellen gesetzte Consistency Level

In der Abbildung 30: Import der Smart-Meter-Daten: PutCassandraQL Konfiguration ist die Konfiguration der Cassandra Datenbank innerhalb der DAvE Architektur dargestellt.

SETTINGS | SCHEDULING | PROPERTIES | COMMENTS

Required field

Property	Value
Cassandra Contact Points	**127.0.0.1:9042**
Keyspace	dave
SSL Context Service	No value set
Client Auth	REQUIRED
Username	cassandra
Password	Sensitive value set
Consistency Level	**ONE**
Character Set	**UTF-8**
Max Wait Time	**0 seconds**

Abbildung 30: Import der Smart-Meter-Daten: PutCassandraQL Konfiguration

6.2.3. Datenexport

In diesem Abschnitt wird der Export der in DAvE befindlichen Daten erläutert. Für den Datenexport wird wie auch beim Import Apache Nifi genutzt. Um in NiFi Daten exportieren zu können, müssen unterschiedliche Prozessoren sowie die erstellten Tabellen aus Apache Cassandra genutzt werden. Es wird zunächst der Dataflow für den Export der DWD-Daten erläutert. An diesem wird genauer gezeigt, welche Prozessoren genutzt werden und welche Konfigurationen durchgeführt wurden. Anschließend folgt die Erläuterung des Dataflow für den Export der Smart-Meter-Daten. Dabei werden nicht mehr die kompletten Prozesse gezeigt, sondern die wichtigsten Einstellungen und individuelle Änderungen der einzelnen Prozessoren.

Datenexport im JSON-Format In diesem Abschnitt wird der Datenexport, welcher im JSON-Format übermittelt wird, erläutert. Die Use Cases der DAvE-Architektur beinhalten die Abfrage von historischen Wetterdaten, das Ergebnis solch einer Abfrage kann über den folgenden Dataflow eingeholt und übermittelt werden. Mit dem in Abbilung 31: Dataflow: Datenexport im JSON-Format dargestellten Dataflow können jedoch alle Daten innerhalb des DAvE-Systems abgefragt werden, sofern die Ausgabe im JSON-Format vom System vorgesehen ist. So kann der Dataflow auch bei möglicher Erweiterung mit anderen Datenquellen für den Export genutzt werden.

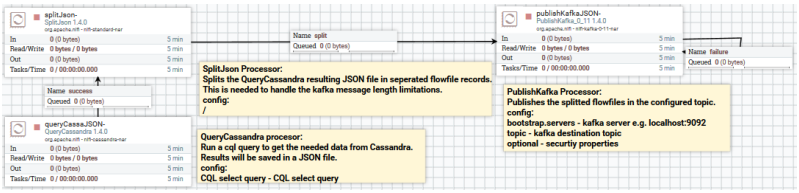

Abbildung 31: Dataflow: Datenexport im JSON-Format

Der dargestellte Dataflow beginnt mit einem *QueryCassandra* Prozessor, dieser dient der Abfrage von Daten innerhalb einer Cassandra Datenbank mittels eines CQL Querys. Die Propertys dieses Prozessors gleichen im weitesten den Propertys des *PutCassandraQL* Prozessors. Folgende Propertys unterscheiden sich:

- **CQL select query:** Angabe der zu tätigenden CQL Abfrage
- **Output Format:** Angabe des Ausgabe Formats des Resultats. Es stehen das JSON-Format und das Avro-Format zur Auswahl

In Abbildung 32: Datenexport im JSON-Format: QueryCassandra Konfiguration werden die in DAvE genutzten Einstellungen dargestellt, die *CQL Query* Property ist auftragsspezifisch und wird somit dynamisch konfiguriert.

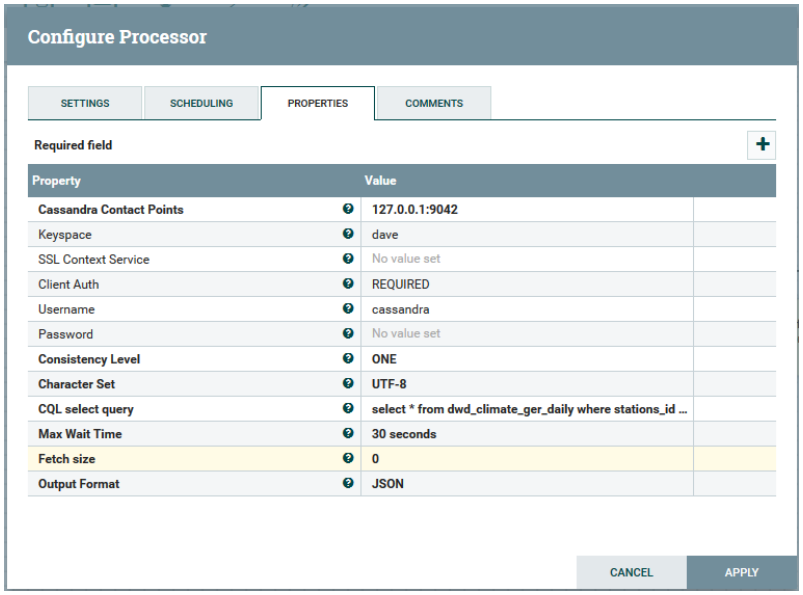

Abbildung 32: Datenexport im JSON-Format: QueryCassandra Konfiguration

Das Ergebnis aus der CQL Select Anfrage ist in diesem Fall im JSON-Format und ein einzelnes JSON-Array, welches alle Datensätze beinhaltet. Damit jeder Datensatz als einzelne Kafka Message übermittelt werden kann, wird ein *SplitJson* Prozessor genutzt, welcher bereits Abschnitt 6.2.2: Datenimport erläutert wird. Die für diesen Dataflow entscheidende Property ist die **JsonPath Expression**. Das JSON-Array, welches mit diesem Prozessor geteilt werden soll wird mit *result* indiziert, daher ist der Wert der Property *$.results* . In Abbildung 33: Datenexport im JSON-Format: SplitJson Konfiguration ist die gewählte Konfiguration aufgezeigt.

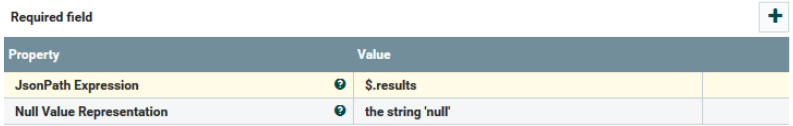

Abbildung 33: Datenexport im JSON-Format: SplitJson Konfiguration

Abschließend werden die aus dem *SplitJson* Prozessor resultierenden FlowFiles an einen *PublishKafka* Prozessor weitergeleitet. Dieser Prozessor dient dem Senden von Messages, welche den Inhalt von FlowFiles beinhalten, an Apache Kafka. Die Propertys dieses Prozessors decken die nötigen Informationen für die Verbindung mit einem oder mehreren Kafka Brokern:

- **Kafka Brokers:** URI der anzusprechenden Kafka Broker

- **Topic Name:** Topic Namen der anzusprechenden Kafka Broker

Diese Propertys sind auftragsspezifisch und werden dementsprechend dynamisch angepasst.

Datenexport im COSEM-Format In diesem Abschnitt wird der Datenexport, welcher im COSEM-Format übermittelt wird, erläutert. Die Use Cases der DAvE-Architektur beinhalten die Abfrage von historischen Smart-Meter-Daten, das Ergebnis solch einer Abfrage kann über den folgenden Dataflow eingeholt und übermittelt werden. In Abbilung 34: Dataflow: Datenexport im COSEM-Format ist der hierfür genutzte Dataflow dargestellt.

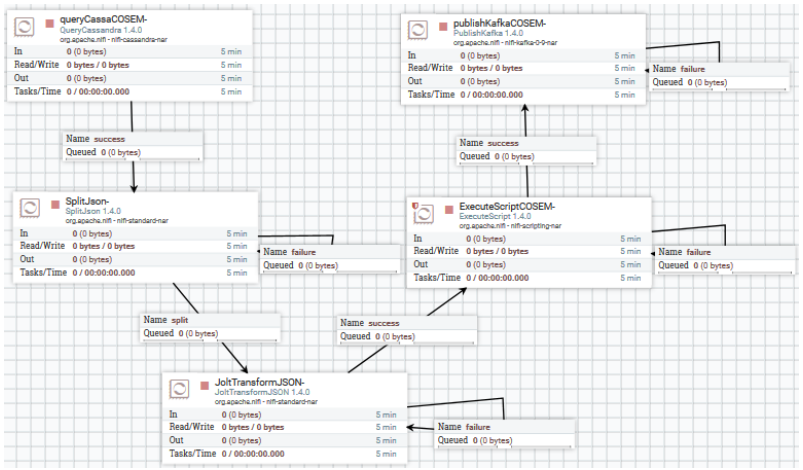

Abbildung 34: Dataflow: Datenexport im COSEM-Format

Dieser Dataflow unterscheidet sich nur an wenigen Stellen vom Dataflow, welcher für den Datenexport im JSON-Format genutzt und in Abschnitt 6.2.3: Datenexport im JSON-Format beschrieben wird. Die beiden Dataflows sind grundlegend identisch, außer der Konvertierung der Daten vom JSON-Format in das COSEM-Format. Die Konvertierung in das COSEM-Format ist notwendig, da dieses Format der Standard für die Kommunikation über Smart-Meter-Daten, ist. Es werden im Folgenden lediglich die abweichenden Prozessoren erläutert. Der Dataflow beginnt mit einem *QueryCassandra* Prozessor gefolgt von einem *SplitJson* Prozessor. Die FlowFiles aus dem *SplitJson* Prozessor werden an einen *JoltTransformJSON* Prozessor weitergeleitet. Dieser Prozessor wird an dieser Stelle genutzt, um das eingehende flache JSON-Array in ein verschachteltes JSON-Array zu transformieren. Auf diese Weise wird das JSON-Array für die Konvertierung in das COSEM-Format vorbereitet. Hierzu wurde die Property *Jolt Specification* konfiguriert, in dieser wird ein Skript eingestellt, welches die vorzunehmende Transformation definiert.

Nach der Transformierung des JSON Arrays werden die resultierenden FlowFiles an einen *ExecuteScript* Prozessor übergeben. Dieser Prozessor wird in diesem Dataflow genutzt, um

ein Python-Skript auszuführen, welches die FlowFiles vom JSON-Format in das COSEM-Format überführt.

Abschließend werden die FlowFiles als Message an einen oder mehrere Kafka Broker mittels eines *PublishKafka* Prozessors gesendet. Die Konfiguration und Beschreibung dieses Prozessors ist im Abschnitt 6.2.3: Datenexport im JSON-Format beschrieben.

6.3. Kommunikationskomponente

In diesem Abschnitt wird auf die Umsetzung der Kommunikationskomponente eingegangen, dazu wird zuerst das PMML-Auftragskonzept vorgestellt. Während dessen wird die Komponente zu den anderen abgegrenzt. Im Anschluss wird die RESTful API vorgestellt und uum Schluss wird die Struktur und das Verhalten näher beschrieben.

6.3.1. Konzept

Für die Kommunikation mit externen Systemen wurde für DAvE eine RESTful Schnittstelle definiert. Diese beinhaltet Funktionen um diverse Prozesse zu starten. Für das Konzept mussten die Use Cases weiter ausformuliert werden. Das Ergebnis sind zwei unterschiedliche Prozesstypen.

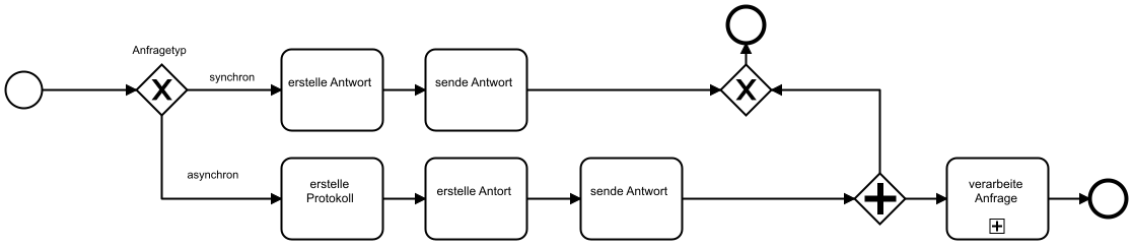

Abbildung 35: schematische Darstellung zur Beantwortung einer Anfrage

Die Abbildung 35: schematische Darstellung zur Beantwortung einer Anfrage zeigt schematisch das zugrunde-liegende Konzept zur Beantwortung einer Anfrage. Eine Anfrage markiert ein Start-Ereignis. Die Anfrage wird immer sofort beantwortet. Dabei wird zwischen einem synchron und einem asynchronen Prozess unterschieden. Beide Prozesstypen erwidern unverzüglich die Anfrage. Im Falle eines synchron Prozesses wird eine Anfrage direkt mit einem validen Ergebnis beantwortet. Die Verarbeitung geschieht sequenziell. Die synchronen Prozesse sind:

- Die Abfrage nach allen Wetterstationen innerhalb Deutschlands.
- Die Anfrage nach dem Prozess-Status einer asynchronen Anfrage.

- Die Anfrage von angebotenen PMML-Modellen.
- Die Anfrage nach einem speziellen PMML-Modell durch die *modelId*.

Bei den asynchronen Prozessen verläuft die Erwiderung der Anfrage und die Verarbeitung dieser parallel. Die Antwort für einen asynchronen Prozess beinhaltet immer eine *Unified Resource Identification* (URI) und eine Topic-Bezeichnung. Die Topic-Bezeichnung ist im gesamten System eindeutig und einzigartig. Die asynchronen Prozesse sind:

- Die Anfrage nach historischen Daten.
- Die Anfrage nach der aktuellstem PMML-Modell anhand der PMML-Struktur
- Der Auftrag nach einem neu-berechneten PMML-Modell.

Die Antwort einer Anfrage umfasst zwei Teile. Es wird eine URI und eine Topic-Bezeichnung erzeugt. Die URI setzt sich aus dem Host und einem Port zusammen. Der Port wird durch den ausgewählten Messaging System definiert. Der Topic ist eine UUID. Anhand dieser Informationen kann das PMML-Modell oder die Ergebnismenge über einen Broker abonniert werden.

Die Topic-Bezeichnung ist gleichzeitig ein interner einzigartiger Schlüssel. Es werden alle asynchronen Prozesse protokolliert, um den Verarbeitungsfortschritt festzustellen und bei bedarf diese neu zu starten. Anhand des Protokolls kann der Verarbeitungszustand abgefragt werden. Für die Verarbeitung einer Data Mining Anfrage sind die Status abfragen essentiell.

Das Konzept zur Verarbeitung einer Data Mining Anfrage wird in der Abbildung 36: Schematische Darstellung der Anfrageverarbeitung einer Data Mining Anfrage schematisch dargestellt. Die Steuerung wird im Abschnitt 6.4.1: Steuerung näher erläutert. Im ersten Schritte wird die Anfrage entgegengenommen. Eine Anfrage besteht aus einer Broker-Angabe und einem PMML-Modell. Das PMML-Modell muss als Base64 kodiert sein, damit wird der Anfrageprozess angestoßen. Der nächste Schritt ist die Protokollierung und das erzeugen eines Topics auf einem MB. Dann wird eine Antwort mit den Informationen, wo das Ergebnis hinterlegt sein wird, zurückgeschickt. Als nächstes findet die Protokollierung statt. Im Anschluss wird der Auftrag in eine Auftragstabelle gespeichert. Das Starten einer periodischen Abfrage über den Status des Auftrags ist der Abschluss dieses Prozessteils.

Der zweite Teil beginnt mit einer Statusänderung des Auftrags. Der Status wird durch ein Long-Datentyp repräsentiert. Dabei entsprechen negative Zahlen einem Fehlercode. Positive Werte werden als EPOCH interpretiert und damit geben sie den Zeitpunkt der Fertigstellung an. Bei einer erfolgreichen Beendigung wird das PMML-Modell als Base64 kodiert und übertragen.

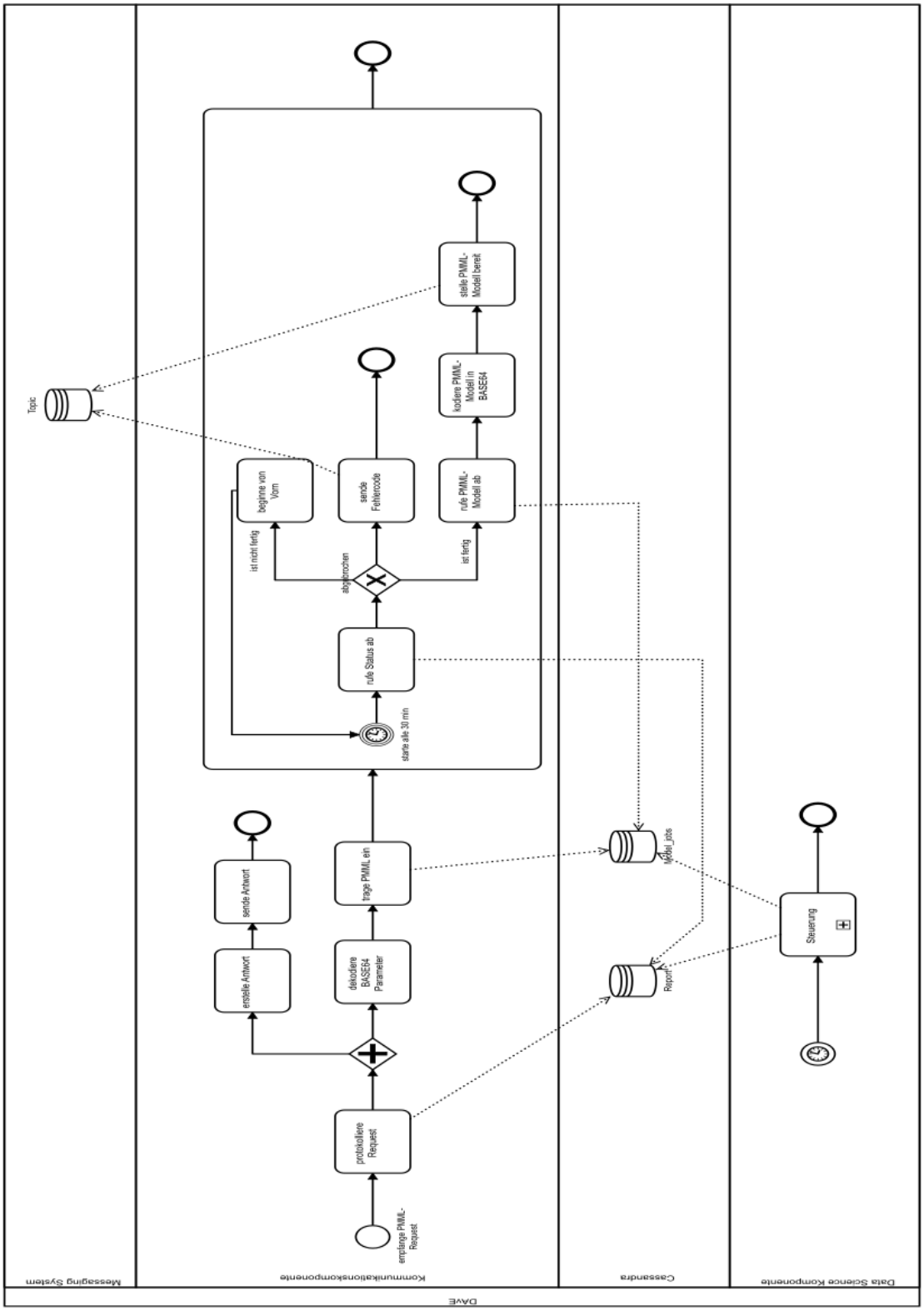

Abbildung 36: Schematische Darstellung der Anfrageverarbeitung einer Data Mining Anfrage

```
1 Create Table if not exists report (jobid uuid primary key,
      startTime bigint, finishTime bigint, parameter text,
      endpoint text, error text);
```

Listing 31: CQL DDL zur Erzeugung der Report-Tabelle

6.3.2. Umsetzung

Die Umsetzung beinhaltet die Struktur sowie das Verhalten der Kommunikationskomponente. Begonnen wird mit einer Erläuterung der RESTful Schnittstelle. Im Anschluss wird die Struktur dargelegt. Die Kommunikationskomponente beinhaltet die Message Broker und die RESTful Schnittstelle. Für die Beschreibung der Schnittstelle wurde das Tool *Swagger* genutzt.

Swagger ist ein Online Editor für die textuelle Beschreibung einer OpenAPI Spezifikation (OAS) in JSON oder Yaml. Aus dieser textuellen Beschreibung kann, in unterschiedlichen Sprachen, testbarer Quellcode generiert werden. Des Weiteren lässt sich eine Schnittstelle darin weiterentwickeln.

RESTful Schnittstelle Die RESTful Schnittstelle kategorisiert die Endpunkte in zwei Gruppen. Die erste Gruppe *historicData* beinhaltet sämtliche Endpunkte bezüglich historischer Daten. Die zweite Gruppe *DataMining* umfasst die Daten Analyse Funktionen. Die Definition wurde Anhand der Use Cases abgeleitet. Die Schnittstelle ist nicht durch eine Authentifizierung geschützt. Die Endpunkten erwarten immer ein JSON-String. Das JSON-Format wurde als Übertragungsformat ausgewählt.

Für die Protokollierung der asynchronen Prozesse wurde eine Tabelle erstellt. Das Listing 31: CQL DDL zur Erzeugung der Report-Tabelle zeigt den dazugehörige CQL DDL Befehl. Die *jobid*-Attribut ist in der Tabelle der primary key.

Die Gruppe *historicData* umfasst acht Endpunkte mit drei Get-Methoden und fünf POST-Methoden. Die Endpunkte *.../data/weather/topic* und *.../data/smgw/topic* erwarten eine Topic-Bezeichnung. Die Antwort ist eine Zustandsbeschreibung über den Fortschritt eines Auftrags. Der Endpunkt *.../data/weather/weatherStations/* liefert eine Liste mit allen zur Verfügung stehenden Wetterstationen innerhalb Deutschlands. Der Endpunkt *../data/smgws/* bietet die Möglichkeit archivierte Smart-Meter-Daten anzufragen. Die Endpunkte für Informationen über Wetterdaten sind erreichbar unter *.../data/weather/climate/*, *.../data/weather/sun/* und *.../data/weather/wind/*. Die Endpunkte für Wetterinformationen erwarten eine Koordinate. Als Antwort wird eine Adresse des MB auf dem die Daten hinterlegt sind und die nächste Wetterstation zurückgeliefert.

Das Listing 32: Antwort, Station zeigt das Antwortschema für die Anfrage aller zurverfügungstehenden Wetterstationen. Es wird eine Liste erstellt. Elemente dieser Liste sind

```
[
  {
    "stationId": "string",
    "name": "string",
    "coor": {
      "lon": 53.14118,
      "lat": 8.21467
    },
    "alt": 0,
    "state": "string"
  }
]
```

Listing 32: Antwort, Station

```
"URI":"localhost:9092",
"Topic":"9bd2380a-1b30-4d89-85b1-139322bec43e"
```

Listing 33: asynchrone Antwort

```
"URI":"localhost:9092",
"Topic":"9bd2380a-1b30-4d89-85b1-139322bec43e",
{
    "stationId": "string",
    "name": "string",
    "coor": {
      "lon": 53.14118,
      "lat": 8.21467
    },
    "alt": 0,
    "state": "string"
  }
```

Listing 34: Antwort, Wetter

die Wetterstationen mit der Stationsidentifikationnummer, Namen, Koordinaten und dem Bundesland. Das Listing 33: asynchrone Antwort zeigt das Antwortschema einer Asynchronen Anfrage. Wie bereits beschrieben enthält dies eine URI und eine Topic-Bezeichnung. Das Listing 34: Antwort, Wetter zeigt das Antwortschema für die Anfrage nach Wetterinformationen, es zeigt zu der gewöhnlichen asynchronen Antwort noch zusätzlich die nächste Wetterstation.

Die *DataMining* Kategorie umfasst fünf Endpunkte. Diese beinhaltet drei GET-Methoden und zwei POST-Methoden. Die Post-Methoden starten asynchrone Prozesse. Als Input wird ein PMML-Modell übergeben. Der Endpunkt *../datamining/submission/* wird genutzt, um ein übergebenes PMML-Modell neu zu berechnen. Der Endpunkt *.../datamining/latests/* erwartet ebenfalls ein PMML-Modell, dabei werden die vorhandenen PMML-Modelle abgeglichen und dasselbe PMML-Modell mit der aktuellsten Version zurückgeliefert. Für den asynchronen Prozess *../datamining/submission/* wird wiederum ein Endpunkt bereitgestellt, um den Fortschritt abzufragen.

Die Endpunkten emph.../datamining/models/ und *.../datamining/latests/modelid* gehören zu einem zweistufigen Anfrageprozess. Dabei wird zuerst eine Liste mit allen angebotenen PMML-Modellen angefragt. Diese Liste ist im Listing 35: Antwort, Modelle gezeigt. Die Listenelemente enthalten Informationen über die Modell-Id., ein Name sowie eine Beschreibung und den Zeitpunkt der letzten Aktualisierung. Die zweite Stufe ist die konkrete Anfrage eines Modells via Modell-Id. Dieser synchrone Prozesse erzeugt die Antwort wie in Listing 36: Base64 kodiertes PMML-Model. Das Objekte ist wie bereits erwähnt in einem JSON-Format und beendet damit die Anfrage.

Struktur und Verhalten der Kommunikationskomponente Die Schnittstelle umfasst unterschiedliche Bestandteile. Der Source Folder *gen* enthält Klassen die durch das Tool Swagger CodeGenerate erzeugt wurden. Dieser besteht aus den Packages *io.swagger.api* und *io.swagger.model*. Das Package *io.swagger.api* beinhaltet wiederum die gesamte Schnittstelle und Endpunkte. Das Package *io.swagger.model* beinhaltet die Datenobjekte, die über die Schnittstelle übertragen werden. Der SourceFolder *src/main/java* enthält die gesamte Geschäftslogik für die Verarbeitung der Anfragen. Das Package *io.swagger* enthält ebenfalls Swagger generierten Code. Die Verarbeitungslogik befindet sich im Package *de.uniol.vlba.dave*. Die notwendigen Konfigurationsdateien befinden sich im Source Folder *src/main/resources*. Die Testklassen sowie notwendige Konfigurationen für den Testumbegung befinden sich in den Source Folder *src/test*.

```
[
{
        "modelId": 1,
        "name": "linear regression on nds data",
        "discription": "regression",
        "lastbuild": 1521950417
    },

    {
        "modelId": 2,
        "name": "SVR on smard data",
        "discription": "regression",
        "lastbuild": 1521952926
    },
                ...
                ,
    {
        "modelId": 6,
        "name": "linear regression on nds data for estream",
        "discription": "regression",
        "lastbuild": 1521953017
    }
]
```

Listing 35: Antwort, Modelle

```
{
    "pmml": "PD94bWwgdmVyc2lvb ... BNTUw+Cg=="
}
```

Listing 36: Base64 kodiertes PMML-Model

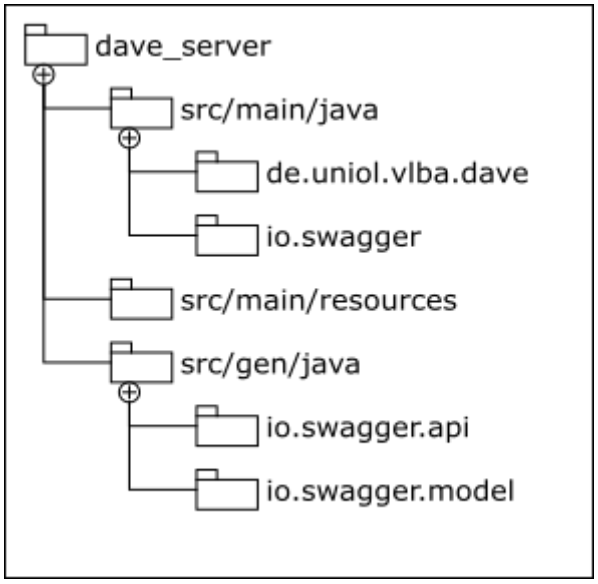

Abbildung 37: Package-Struktur der Kommunikationskomponente

Im Folgenden wird das Verhalten exemplarisch für ein asynchron und einen synchronen Prozess aufgezeigt. Der synchrone Prozessablauf wird beispielhaft an dem Prozess zur Abfrage der PMML-Modelle gezeigt. Das Diagramm 38: Prozessablauf zur Abfrage von PMML-Modellen zeigt der Vorgang zur Anfrageverarbeitung. Ein Nachfrager ruft den Endpunkt *.../datamining/latests/modelid* auf. Dann wird eine Datenbankverbindung zu Cassandra erzeugt. Die vorher erzeugte CQL-Query wird gesendet. Aus der Ergebnismenge wird eine Antwort erzeugt. Diese wird dann an den Nachfrager weitergeleitet.

Das Diagramm39: Prozessablauf zur Abfrage von Smart Metern zeigt das Verhalten bei einer asynchronen Verarbeitung. Dort wird die Anfrage nach Smart Meter exemplarisch vorgeführt. Auch hier wird der Prozess durch eine Nachfrage angestoßen. Während der Erzeugung der Antwort parallelisiert sich der Verarbeitungprozess. Der Status kann während und nach der Prozess abgefragt werden, damit ist diese Abfrage beendet. Das abrufen des Topics kann nun zu jeden Zeitpunkt erfolgen.

Im nachfolgenden Abschnitt wird die Anfragebearbeitung mittels Apache NiFi beschrieben.

Anfragebearbeitung in Apache NiFi Wie bereits in der Technologieauswahl in Abschnitt 4.2 beschrieben wurde, wird das Dataflow Management in DAvE über Apache Nifi betrieben. Aufträge, welche über DAvE's RESTful-Schnittstelle eingehen, müssen automatisiert in Apache NiFi bearbeitet werden. Apache NiFi bietet für den Zugriff und die Steuerung neben der Web UI auch eine REST API[8] an. In diesem Abschnitt wird die

[8] https://nifi.apache.org/docs/nifi-docs/rest-api/index.html

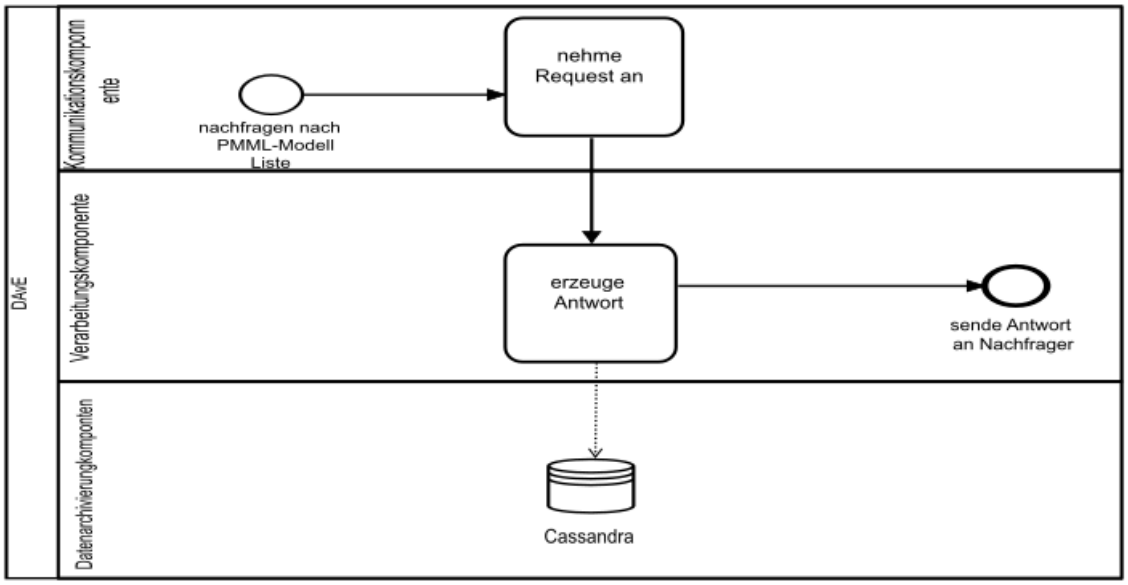

Abbildung 38: Prozessablauf zur Abfrage von PMML-Modellen

Steuerung von Apache NiFi in Kommunikation mit DAvE's RESTful-Schnittstelle erläutert.

In einem ersten Schritt wurde Apache NiFi's REST API von einigen Mitgliedern der Projektgruppe DAvE untersucht. Es fiel auf, dass die einzelnen Methoden dokumentiert sind, aber keine offiziellen Beispiele für dessen Einsatz bereitgestellt werden. Die Herleitung, welche Folge von Methodenaufrufe getätigt werden müssen, um Aktionen wie das Erstellen und Starten eines Dataflows zu bewerkstelligen, ist herausfordernd. Eine Eigenentwicklung eines Programmes, welches die an DAvE gestellten Anfragen in Apache NiFi bearbeitet, erwies sich somit als zu zeitintensiv für die begrenzte Dauer des Projektes.

Die Projektmitglieder recherchierten, ob bereits Lösungen existieren, welche eine erleichterte programmatische Steuerung Apache NiFi's ermöglichen. Als Ergebnis der Recherche wurde das Programm „nifi-config"entdeckt, welches über das freie GitHub Repository von Hermann Pencolé verfügbar ist (vgl. [Her18]). Im nächsten Abschnitt wird das erwähnte Programm erläutert.

Nifi-config Programm In diesem Abschnitt wird das Programm *nifi-config* beschrieben. Für ein besseres Verständnis, der von diesem Programm gebotenen Funktionen wurde ein Aktivitätsdiagramm erstellt. Das in Abbildung 40 dargestellte Diagramm bildet einen Workflow ab, welcher mit dem *nifi-config* Programm umgesetzt werden kann.

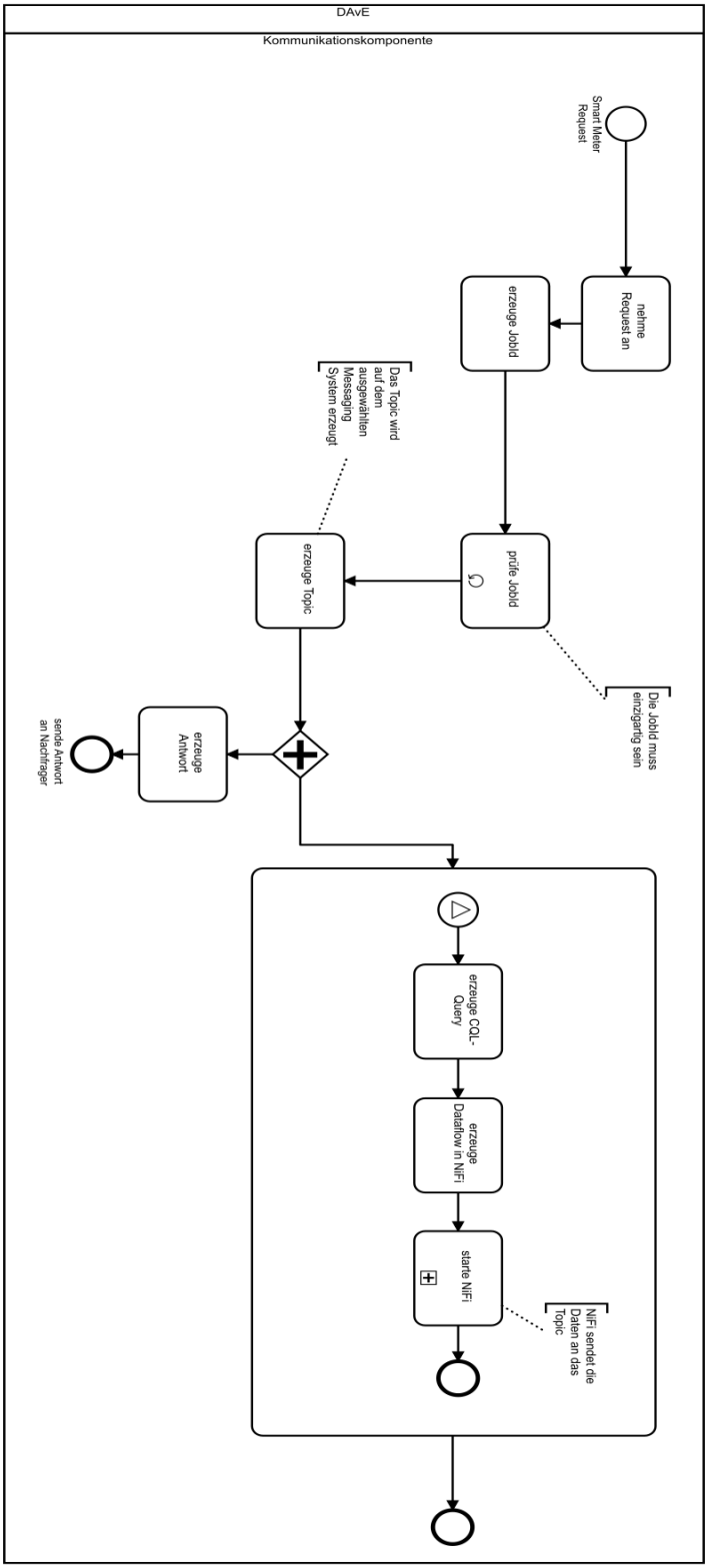

Abbildung 39: Prozessablauf zur Abfrage von Smart Metern

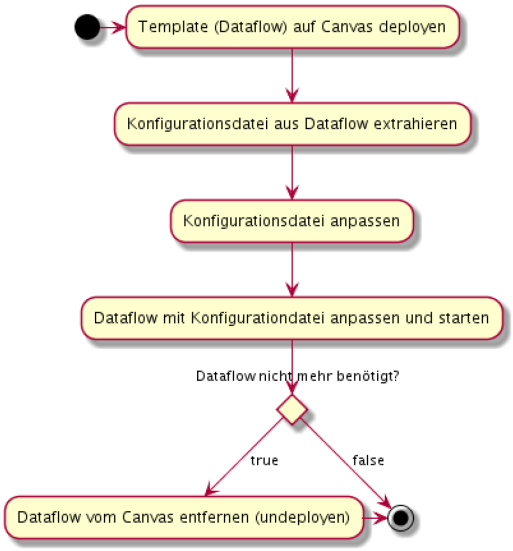

Abbildung 40: Nifi-Config: Workflow

Das *nifi-config* Programm bietet die Möglichkeit bestehende Templates auf das NiFi Canvas zu platzieren (deployen), außerdem kann eine Konfigurationsdatei aus Dataflows extrahiert werden. Diese Konfigurationsdateien werden im JSON-Format abgelegt und bieten die Möglichkeit, Einstellungen eines jeden Processors innerhalb des Dataflows zu bearbeiten.

Hat man die Konfigurationsdatei seinen Anforderungen entsprechen angepasst, kann man mit dem *nifi-config* Programm diese Konfigurationsdatei nutzen, um den entsprechenden Dataflow im Betrieb, zu konfigurieren. Die Anpassungen werden vorgenommen und die Standardeinstellung des Programmes sieht vor, dass der konfigurierte Dataflow im Anschluss gestartet wird. Außerdem können mit dem Programm Dataflows vom Nifi Canvas entfernt werden (undeployed). Die Steuerung von Apache NiFi mittels dieses Programmes setzt voraus, dass die genutzten Dataflows und darin befindlichen Prozessoren eindeutig benannt sind (vgl. [Her18]).

Die Funktionalitäten des vorgestellten Programmes sind ausreichend für die in DAvE angestrebten Zwecke, da DAvE feste Use Cases erfüllt. Jeder Use Case wird durch einen festen Dataflow abgedeckt. Einzig einige Einstellungen in bestimmten Prozessor Konfigurationen unterscheiden sich je Anfrage. Dementsprechend genügt es, wenn vorher festgelegte Templates genutzt werden, welche die Use Cases abdecken. Und die Dataflows dann auftragsspezifisch konfiguriert werden.

Das Bedienen des *nifi-config* Programmes und das Konfigurieren der extrahierten Konfigurationsdateien sollte automatisiert ablaufen. Zu diesem Zweck wurde in Eigenentwicklung ein Programm in Java geschrieben, welches im nächsten Abschnitt beschrieben wird.

Eigentwicklung: DAvE NiFi-Steuerung In diesem Abschnitt wird ein Programm erläutert, welches das im vorherigen Abschnitt 6.3.2 beschriebene Programm *nifi-config* automatisiert bedient und nötige Vorverarbeitungen vornimmt. Das Programm wird im Folgenden als *Nifi-Strg* beannt und ist in DAvE's Schnittstelle im Paket `de/uniol/vlba/dave/nifi` implementiert.

Für ein besseres Verständnis, der von diesem Programm *Nifi-Strg* gebotenen Funktionen, wurde ein Aktivitätsdiagramm erstellt, welches in Abbildung 41 zu sehen ist.

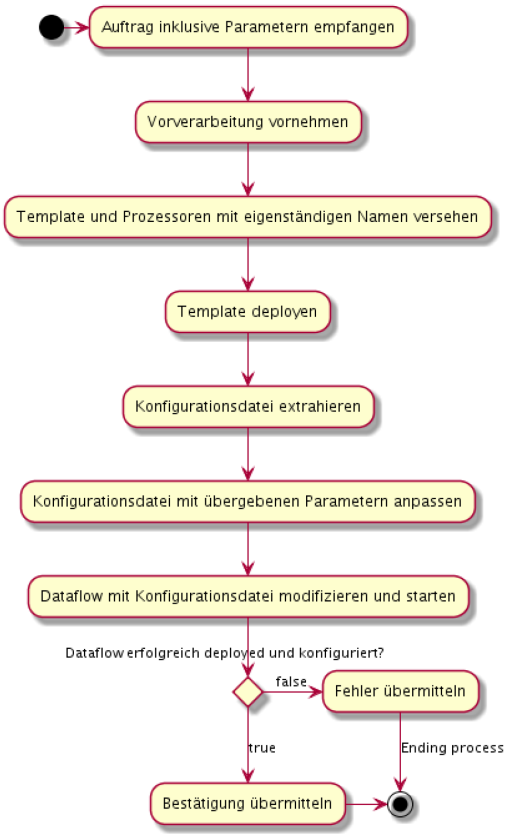

Abbildung 41: Nifi-Steuerung: Workflow

Der abgebildete Workflow fasst die Arbeitsschritte des Programmes zusammen. Über DAvE's RESTful-Schnittstelle werden Anfragen an DAvE übermittelt. Für die Bearbeitung dieser Anfragen innerhalb Apache NiFi's wird das Programm *Nifi-Strg* angesprochen und der Auftrag mit den nötigen Parametern übergeben. Im Anschluss wird eine Vorverarbeitung unternommen, in dieser werden die übergebenen Parameter z. B. genutzt, um ein CQL Query zu bilden oder Zeitangaben in ein passendes Datumsformat konvertiert.

Jede Anfrage wird mittels einer *UUID* eindeutig unterschieden. Die Anfragen beziehen

sich auf einen der Use Cases von DAvE. Für diese Use Cases sind jeweils vorgefertigte NiFi Templates hinterlegt. Die UUID wird in *Nifi-Strg* genutzt, um das Template und darin definierte Prozessoren mit eindeutigen Bezeichnern zu versehen. Hierzu wird mittels eines XML-Parsers an die ursprünglichen Bezeichner, die UUID angehangen.

Mit diesem Schritt wird gewährleistet, dass eindeutige Bezeichner genutzt werden, welches eine Voraussetzung für die Nutzung des Programmes *nifi-config* ist (siehe Abschnitt 6.3.2). In einem nächsten Schritt wird eine Funktion des Programmes *nifi-config* angesprochen, mit dieser wird das Template, welches den abgezielten Use Case abbildet, auf das NiFi Canvas deployed.

Danach wird aus dem platzierten Template bzw. dem darin enthaltenen Dataflow eine Konfigurationsdatei im JSON-Format extrahiert. Mittels eines JSON-Parsers werden die in der Anfrage übermittelten Parameter in den entsprechenden Prozessoren übernommen. Darauf folgend wird eine weitere Funktion des Programmes *nifi-config* angesprochen, um mit der angepassten Konfigurationsdatei den Dataflow zu modifizieren und mit den gesetzen Einstellungen zu starten.

Wenn Fehler innerhalb dieser Folge von Aktivitäten aufgetreten sind, werden diese in einem Log festgehalten und mit DAvE's RESTful-Schnittstelle wird dem Anfragesteller eine Fehlermeldung übermitteln, andernfalls wird eine Erfolgsmeldung übermittelt.

Nachfolgend wird auf die Struktur der verwendeten Klassen des Programmes *Nifi-Strg* eingegangen. Die Struktur ist in Abbildung 42 veranschaulicht.

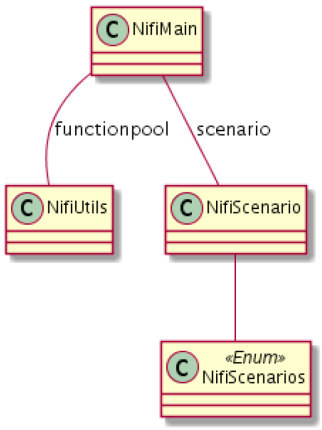

Abbildung 42: Nifi-Steuerung: Klassen Struktur

Die Klasse `NifiUtils` enthält alle Methoden, welche für die Vorverarbeitung der Daten und das Steuern des Programmes *nifi-config* genutzt werden.

Die Klasse `NifiMain` stellt die Hauptklasse dar, diese wird innerhalb DAvE's RESTful

```
session.execute("CREATE TABLE if not exists dave.model_list
    (model_id int, description text, features set<text>,
    target text,method text,script_name text, tables set<
    text>, model text, model_type text, generation_date
    bigint, PRIMARY KEY (model_id));")


session.execute('CREATE TABLE if not exists dave.model_jobs
    (job_id uuid, input_pmml text, output_pmml text, method
    text, features set<text>, target text, start_time bigint
    , end_time bigint,model_type text, model_id int, PRIMARY
     KEY (job_id, start_time));')
```

Listing 37: Erstellen der Tabelle model_jobs und model_list

Schnittstelle angesprochen. Innerhalb dieser Klasse wird die Vorverarbeitung vorgenommen, die beinhaltet z. B. das generieren eines CQL Queries aus den in der Anfrage gelieferten Information. Hierfür werden Methoden aus der `NifiUtils` Klasse verwendet. Die Klasse `NifiScenario` setzt den in Abbildung 41 dagestellten Workflow, ab dem Schritt *Template und Prozessoren mit eigenständigen Namen versehen* um. Hierzu werden die entsprechenden Methoden aus der `NifiUtils` Klasse, in der richtigen Folge aufgerufen. Die Klasse `NifiMain` setzt diesen Ablauf nach der Vorverarbeitung in Gang.

Ein Löschen der Dataflows ist im aktuellen Entwicklungsstand nicht möglich, da Apache NiFi nicht für das einmalige Ausführen von Dataflows vorgesehen ist. Aus diesem Grund ist eine Abfrage, ob ein Durchlauf im Dataflow terminiert ist, erschwert. Das Löschen eines Dataflows ohne solch eine Abfrage stellt die Gefahr dar, das Dataflows vorzeitig entfernt werden und somit die Daten noch nicht vor dem Löschen erfolgreich verarbeitet wurden.

6.4. Data Science

Zur Umsetzung der spezifizierten Data Science Use Cases wurde eine Konzept für die Umsetzung der hierfür erforderlichen Komponenten entwickelt. Dieses wird in der Abbildung 43: Konzept des PMML Prozesses ausführlich dargestellt.

6.4.1. Steuerung

Um die Generierung von neuen Modellen zu steuern wurde ein aus mehreren Python Skripten bestehendes Steuerungssystem entwickelt. Zur Speicherung der Zwischenstände und Daten werden die Tabellen `model_list` und `model_jobs` genutzt. Die Erstellung von diesen wird im Listing 37: Erstellen der Tabelle model_jobs und model_list dargestellt.

Zu Beginne wird zunächst die Tabelle `model_list` mit allen angebotenen Modellen gefüllt,

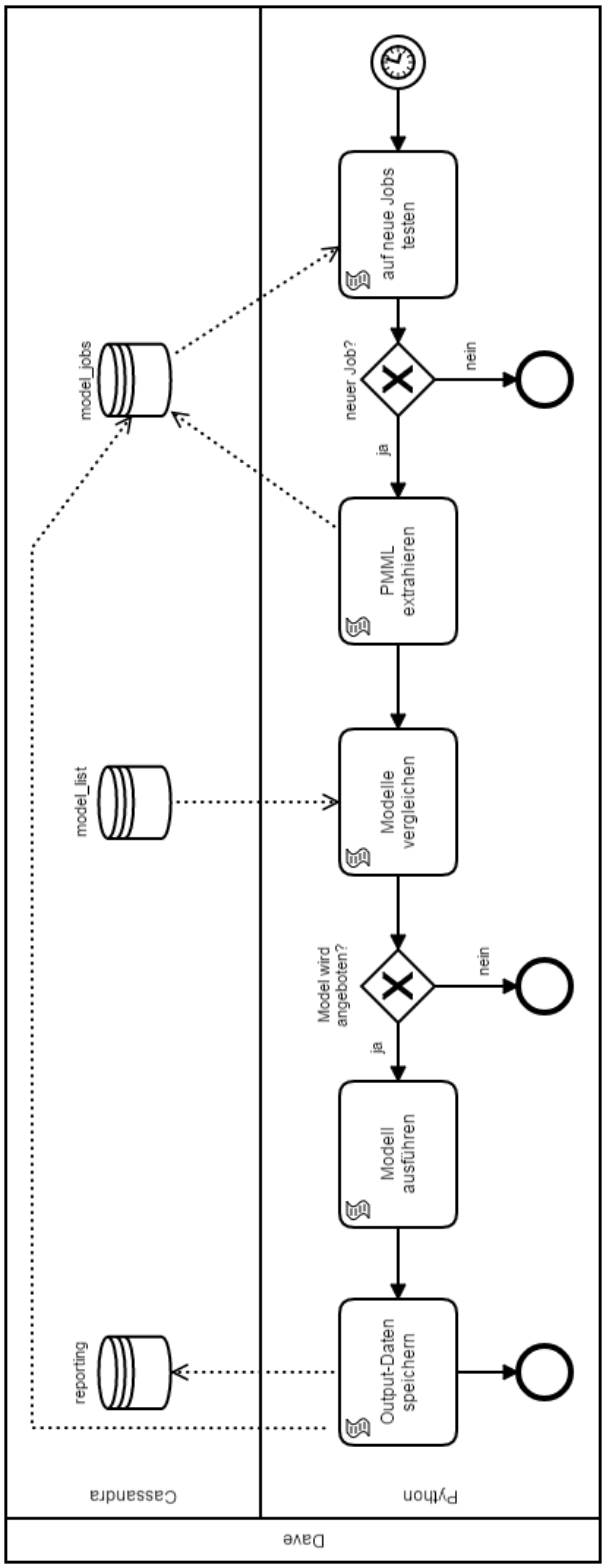

Abbildung 43: Konzept des PMML Prozesses

```
1
2 */1 * * * * sudo -H python3 /home/hduser/models/
      pmml_script_launcher.py
3
4 0 6 * * Sun sudo -H python3 /home/hduser/models/
      update_models_in_model_list.py
```

Listing 38: Cronjobs zur Steuerung der Data Science Funktionen

```
1 def latest_jobs(session, latesttime):
2 rows = session.execute(
3 "SELECT * FROM dave.model_jobs WHERE start_time > " + str(
      latesttime) + " allow filtering")
4 jobs = pd.DataFrame(list(rows))
5 return jobs
6
7 def load_latest_job_time():
8 try:
9 latesttime = pickle.load(open("time.pickle", "rb"))
10 except (OSError, IOError) as e:
11 latesttime = 0
12 pickle.dump(latesttime, open("time.pickle", "wb"))
13 return latesttime
```

Listing 39: Lesen der noch nicht bearbeiteten Anfragen

dies erfolgt über das Python-Skript `initialize_model_list`. Um zu prüfen, ob neue Modelle angefragt wurden wird dabei jede Minute mithilfe eines Cronjobs (siehe 38: Cronjobs zur Steuerung der Data Science Funktionen) das Python-Skript `pmml_skript_launcher` gestartet. In diesem wird zunächst eine Session zur Verbindung mit der Cassandra Datenbank erzeugt. Anschließend prüft das Skript mit den in einem pickle File gespeicherten Startzeiten der vorherigen Modelle, ob es in der Tabelle `model_jobs` Modellanfragen gibt die noch nicht bearbeitet wurden. Falls dies der Fall ist wird die letzte Startzeit dieser Modelle wieder in das pickle File geschrieben und die Verarbeitung der Anfrage gestartet. Der Ablauf ist im Listing 39: Lesen der noch nicht bearbeiteten Anfragen dargestellt.

Bei der Verarbeitung werden zunächst aus den in der Tabelle `model_jobs` befindlichen PMML Dateien die zur Unterscheidung relevanten Werte extrahiert. Da es in scikit-learn keine Funktionalität gibt, um PMML Modelle zur importieren und zu parsen wurde dafür eine Methode entwickelt, die die Bibliothek lxml in Kombination mit regulären Ausdrücken nutzt. Der Code zur Extraktion wird im Listing 40: Parsen der PMML Datei) dargestellt. Damit erfüllt das System die Anforderung DAN-04.

Die extrahierten Werte sind die Methode, der Modeltyp, die Feature Felder und das Target Feld. Diese Werte werden um die Anfragen zu protokollieren in der Tabelle `model_jobs` gespeichert. Nach dem Extrahieren der Werte werden diese genutzt, um zu prüfen, ob die

```
def pmml_extractor(pmml):

pmml_xml = ElementTree.fromstring(
pmml, parser=ElementTree.XMLParser(encoding='utf-8'))
# read method
method = ''
modeltag = pmml_xml.xpath('./*[re:match(local-name(), "\w*[m
    ,M]odel|NeuralNetwork")]',
namespaces={'re': "http://exslt.org/regular-expressions"})

model_type = re.sub(r'\{[^)]*\}', '', modeltag[0].tag)
for datafield in modeltag:
method = datafield.attrib['functionName']
# read target and feature columns
fields = []
target = ''
features = []
for datafield in pmml_xml.iter('{*}MiningField'):
fields.append(datafield.attrib['name'])
if 'usageType' in datafield.attrib and datafield.attrib['
    usageType'] in ["predicted", "target"]:
target = datafield.attrib['name']
else:
features.append(datafield.attrib['name'])

return method, target, features, model_type
```

Listing 40: Parsen der PMML Datei

angefragten Modelle angeboten werden, also in der Tabelle `model_list` vorhanden sind. Die angebotenen Modelle sind in der aktuellen Version Lineare Regression, Support Vector Regression, Random Forest Regression und Neuronale Netzwerke. Falls das Modell nicht vorhanden ist, wird in der Tabelle *report* zu dem Job der Zeitstempel -2 gesetzt. Falls das entsprechende Modell angeboten wird, wird dessen Ausführung gestartet. Alle Modellgenerierungen sind in dem separatem Python-Skript `ML_Script` definiert. Hier werden die Modelle generiert und danach als Ausgabe PMML in der Tabelle `model_jobs` gespeichert. Dadurch erfüllt das System die Anforderung DAN-03 und DAN-07. Zusätzlich wird als Indikator, dass die Aufgabe abgeschlossen ein Endzeitstempel in der Tabelle `model_jobs` und in der Tabelle `report` gespeichert.

Neben dem Anfragen von Modellen über das Senden von PMML Modellen besteht auch die Möglichkeit Modelle über Ihre ID anzufragen. Dazu werden in der Tabelle `model_list` PMML Modelle vorgehalten. Diese werden durch das über einen Cronjob wöchentlich ausgeführte Skript `update_models_in_model_list` aktuell gehalten.

6.4.2. Modellierung

In diesem Abschnitt werden die entwickelten Modelle und die zugrundeliegenden Methoden vorgestellt. Im Rahmen des Projektes wurden verschiedene Modelle entwickelt und getestet. In die finale Version wurden insgesamt sechs Modelle integriert. Diese basieren sowohl auf historischen Energiedaten, als auch auf Smart-Meter-Daten aus Mosaik. Damit erfüllt das System die Anforderungen DAN-05 und DAN-06. Zudem wird durch die Implementierung von verschiedenen Algorithmen Anforderung DAN-01 ebenfalls erfüllt.

Lineare Regression Die lineare Regression geht von einem linearen Zusammenhang zwischen den Eingabewerten und den resultierenden vorhergesagten Werten aus. Lineare Modelle sind einfach und verständlich und bieten häufig eine adäquate Beschreibung des Einflusses der Eingabewerte auf den Ausgabewert. Für Vorhersagen können sie manchmal bessere Ergebnisse liefern als elaborierte, nicht-lineare Modelle (vgl. [HTF09], S. 43). Die allgemeine Formel für die lineare Regression ist die folgende:

$\hat{y}$ = w[0] * x[0] + w[1] * x[1] + ... + w[p] * x[p] + b

Dabei bezeichnen x[0] bis x[p] die Features (in diesem Fall existieren p Features), w und b sind gelernte Parameter des Modells und $\hat{y}$ ist die Vorhersage, die das Modell macht (vgl. [Mü16], S. 45).

Insgesamt wurden zwei Modelle mithilfe des linearen Regression umgesetzt. Für die Erzeugung beider Modell wurden die Netzdatenstromdaten aus der Tabelle `nds_sim1` genutzt.

Im ersten Modell wurden zunächst die Felder `timestamp` und `value` in numerische Werte

```
1 df[['timestamp','value']] = df[['timestamp','value']].apply(
      pd.to_numeric)
2 df['timestamp'] = pd.to_datetime(df['timestamp'],unit='s')
3 df['dummy_time'] =pd.to_numeric(df.timestamp.map(lambda x: x
      .strftime('%H%M')))
4 df['dummy_day'] = df['timestamp'].dt.dayofweek
```

Listing 41: Extrahieren von Wochentag und Uhrzeit aus dem timestamp

konvertiert. Daraufhin werden die Daten auf ein einzelnes Smart-Meter beschränkt. Im Anschluss wir ein scikit-learn LinearRegression Modell initialisiert. Dieses wird mit dem `timestamp` als Feature und dem `value` als Target trainiert. Im Anschluss wird das Modell unter Zuhilfenahme der sklearn2pmml Bibliothek als PMML exportiert und in der Datenbank gespeichert. Dieses Modell ist sehr einfach, bietet dafür aber auch keine gute Vorhersagegenauigkeit.

Im zweiten Modell, welches mit der linearen Regression umgesetzt wurde, wurden aus dem `timestamp` der Wochentag und die Uhrzeit als einzelne Features extrahiert (siehe 41: Extrahieren von Wochentag und Uhrzeit aus dem timestamp). Diese werden anschließend zur Vorhersage genutzt. Dadurch bietet das Modell eine deutlich bessere Vorhersagegenauigkeit als das rein mit dem `timestamp` arbeitende.

Support Vektor Machine Support Vektor Machines (SVM) sind eine Erweiterung von linearen Modellen, bei der komplexere Modelle möglich sind. Ein lineares Modell lässt sich durch das Hinzufügen weiterer Merkmale flexibler machen, beispielsweise durch die Quadrierung eines Merkmals. Allerdings bedeutet das Hinzufügen neuer Merkmale einen erhöhten Rechenaufwand. Daher implementiert eine SVM den sogenannten Kernel-Trick, mit dem man das Modelle in einem höher dimensionalen Raum trainieren kann, ohne die neue Repräsentation explizit zu berechnen (vgl. [MGR17], S. 88ff.).

In DAvE wird die Regressionsfunktion der SVM auf dem SMARD-Datensatz[9] angewendet. Die zu vorhersagende Größe ist die Stromerzeugung durch Photovoltaik. Aus dem Datum und der Uhrzeit der Rohdaten werden die Merkmale 'minute', 'hour', 'month' und 'day' extrahiert, um die Anzahl der Merkmale zu erweitern (siehe Listing 42: Extrahierung Input-Daten SVM).

Die SVM wurde mithilfe der Klasse `GridSearchCV` entwickelt und optimiert, die im Modul `model_selection` implementiert ist. Diese Klasse kombiniert die Gittersuche mit der Kreuzvalidierung. Bei der Kreuzvalidierung werden die Daten wiederholt aufgeteilt, um die Auswertung der Verallgemeinerung stabiler zu machen. So werden bspw. bei der 3-fachen Kreuzvalidierung die Daten in drei gleich größe Teile (Folds) aufgeteilt. Anschließen werden mehrere Modelle erstellt. Beim ersten Modell stellt der erste Fold den Testdatensatz und

[9] Detailliert beschrieben wird der SMARD-Datensatz unter dem Kapitel 6.1.6: SMARD-Daten

```
# extract input data
data['minute'] = data['uhrzeit'].dt.minute
data['hour'] = data['uhrzeit'].dt.hour
data['month'] = data['datum'].dt.month
data['day'] = data['datum'].dt.day

# create feature and target set
X = data[['minute', 'hour', 'month', 'day']]
y = data['photovoltaik']
```

Listing 42: Extrahierung Input-Daten SVM

```
# parameters to be tested
# for development only
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100], 'gamma':
    [0.001, 0.01, 0.1, 1, 10, 100]}

# train models
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
grid_search = GridSearchCV(SVR(kernel='rbf'), param_grid, cv
    =3, verbose=3)
grid_search.fit(X_train, y_train)

# display results of GirdSearchCV
results = pd.DataFrame(grid_search.cv_results_)
display(results.head(120))
```

Listing 43: Auszug Entwicklung SVM-Modell

die übrigen Folds den Trainingsdatensatz dar. Für jedes Modell wird anschließend die Genauigkeit berechnet, woraus am Ende ein Durchschnittswert gebildet wird (vgl. [MGR17], S. 236). Mithilfe der Gittersuche ist es möglich die optimalen Parameter eines Modells zu ermitteln. Hierbei wird für jede Kombinationen der Parameter ein Modell erstellt (vgl. [MGR17], S. 246f.). Das Listing 43: Auszug Entwicklung SVM-Modell zeigt die Parameter, für die - in jeder möglichen Kombination - ein Modell erstellt erstellt wurde. Die Ergebnisse des der `GridSearchCV` werden in dem Attribut `cv_results_` gespeichert. Für jede Kombination der Parameter werden umfangreiche Informationen gespeichert, wie z.B. die durchschnittliche Genauigkeit (Score) auf den Test- und Trainingsdaten. Um die besten Parameter bzw. den besten Score zu ermitteln, können die Attribute `best_params_` bzw. `best_score_` ausgelesen werden. In unserem Fall sind die besten Parameter C=100 und gamma=0.1, mit einem Score von 85.5% auf den Testdaten. In dem Prototypen wird die Gittersuche mit Kreuzvalidierung nicht mit jeder Anfrage gestartet, da die Berechnung aller Kombinationen viel Zeit in Anspruch nimmt. Daher wird das SVM-Modell mit den oben genannten besten Parametern verwendet, wie es in Listing 44: Lernen des SVM-Modells dargestellt ist.

```
# use model with best score
from sklearn2pmml import PMMLPipeline
model = SVR(C=100, gamma=0.1)
pipeline = PMMLPipeline([("SVR", model)])
pipeline.fit(X_train, y_train)
```

Listing 44: Lernen des SVM-Modells

```
# aggregate input data
sum_per_hour = data2.groupby(['datum', 'hour', 'month', 'day
    ', 'year'], as_index=False).aggregate(sum)

# create feature and target set
X = sum_per_hour[['hour', 'month', 'day', 'year']]
y = sum_per_hour['photovoltaik']
```

Listing 45: Preprocessing Random Forest

Random Forest Random Forests sind im Wesentlichen eine Menge von Entscheidungsbäumen[10], wobei sich jeder Baum in gewissem Maße von den übrigen unterscheidet. Dadurch wird das Hauptproblem des Overfittings von Entscheidungsbäumen umgangen. Die Idee eines Random Forest ist es, dass jeder Baum eine gute Vorhersage treffen kann, aber voraussichtlich einen Teil der Daten overfittet. Bei der Konstruktion mehrerer Bäume kann durch Mitteln der Ergebnisse das Overfitting reduziert werden. Die Unterschiedlichkeit der einzelnen Bäume wird dadurch erreicht, dass beim Aufbauen der Bäume ein Zufallselement verwendet wird. Variieren kann hierbei die Auswahl der Datenpunkte und die Auswahl der zu testenden Merkmale bei jeder Teilung (vgl. [MGR17], S. 80).

In DAvE wird der Random Forest ebenfalls auf den SMARD-Daten, zur Vorhersage der Stromerzeugung durch Photovoltaik, angewendet. Anders als bei der SVM, wird die Stromerzeugung für jede Stunde summiert betrachtet. Diese Aggregation ist notwendig, da die PMML-Datei sonst eine Größe erreichen würde, in der sie als String nicht in Apache Cassandra speicherbar wäre. Stattdessen wird das Jahr als zusätzliches Merkmal heran gezogen (siehe Listing 45: Preprocessing Random Forest).

Der Random Forest ist in der Klasse `RandomForestRegressor` im Modul `ensemble` implementiert. Der Parameter `n_estimators` gibt an, wie viele unterschiedliche Bäume erzeugt werden. Je höher dieser Wert ist, desto robuster wird das Modell. Allerdings wird auch mehr Speicher und Rechenzeit benötigt (vgl. [MGR17], S. 85). Während der Entwicklungsphase hat sich gezeigt, dass `n_estimators=10` bereits sehr gute Ergebnisse liefert. Eine Erhöhung dieses Wertes verbessert die Genauigkeit nur sehr geringfügig. Zudem wird dadurch die Größe der späteren PMML-Datei gering gehalten. Der Parameter `max_depth`

[10] Auf das Prinzip der Entscheidungsbäume wird in der angehängten Seminararbeit *Methoden von Data Mining und Machine Learning* näher eingegangen

```
# fit model
from sklearn.ensemble import RandomForestRegressor
from sklearn2pmml import PMMLPipeline
forest = RandomForestRegressor(n_estimators=10, max_depth
    =20, random_state=2)
pipeline = PMMLPipeline([("RandomForest", forest)])
pipeline.fit(X_train, y_train)
```

Listing 46: Trainieren des Random Forests

gibt die Tiefe eines Baumes an. Bäume ohne Begrenzung der Tiefe sind sehr anfällig für Overfitting und können schlechter auf neue Daten verallgemeinern (vgl. [MGR17], S. 72f.). Daher werden die in Listing 46: Trainieren des Random Forests dargestellten Parameter für das Trainieren des Random Forests verwendet. Der Score auf den Testdaten beträgt ca. 94,6%.

Neuronales Netz `scikit-learn` verwendet als neuronales Netz u.a. die Methode des mehrschichtigen Perzeptrons (englisch *multilayer perceptron, MLP*). MLPs lassen sich als Verallgemeinerung von linearen Modellen ansehen, wobei mehrere Verarbeitungsschritte zu einer Entscheidung führen. Die Vorhersage $\hat{y}$ einer linearen Regression errechnet sich, wie im vorherigen Abschnitt beschrieben, durch die gewichtete Summe der Eingabemerkmale x[0] bis x[p] mit den Koeffizienten w[0] bis w[p] als Gewichte. Bei einem MLP wird das Berechnen einer gewichteten Summe nun mehrfach wiederholt. Abbildung 44: Beispiel neuronales Netz zeigt beispielhaft ein neuronales Netz mit einer verborgenen Schicht. Jeder Knoten (Neuron) auf der linken Seite steht für ein Merkmal der Eingabe und die Verbindungslinien geben die erlernten Koeffizienten als Gewichtung eines jeden Merkmals an. Die verborgene Schicht kann als Zwischenschritt betrachtet werden, wo an jedem Neuron dieser Schicht eine gewichtete Summe der Eingabewerte berechnet wird. Der Knoten auf der rechten Seite stellt die Ausgabe dar und berechnet sich wiederum durch die gewichtete Summe der verborgenen Schicht (vgl. [MGR17], S. 99f.).

In DAvE wird das neuronale Netz auf zwei unterschiedliche Datensätze angewendet. Zum einen - wie die beiden Modelle zuvor auch - auf dem SMARD-Datensatz zur Vorhersage der Stromerzeugung aus Photovoltaik, zum anderen auf den simulierten Smartmeter-Daten. Beim Letzteren wurde im Voraus prototypisch ein Smartmeter festgelegt. Bei beiden Modellen war im Preprocessing eine Skalierung der Daten notwendig, denn neuronale Netze erwarten, dass alle Merkmale der Eingabe in ähnlicher Weise variieren (vgl. [MGR17], S. 108). Realisiert wurde dies mithilfe des `StandardScaler` von `scikit-learn`, was in Listing 47: Skalierung der Daten für das neuronale Netz abgebildet ist. Der `StandardScaler` stellt sicher, dass jedes Merkmal den Mittelwert 0 und die Varianz 1 aufweisen. Dadurch wird sichergestellt, dass alle Merkmale in der gleichen Größenordnung liegen (vgl. [MGR17], S.125).
Auf den SMARD-Daten wird ein neuronales Netz mit zwei verborgenen Schichten mit je-

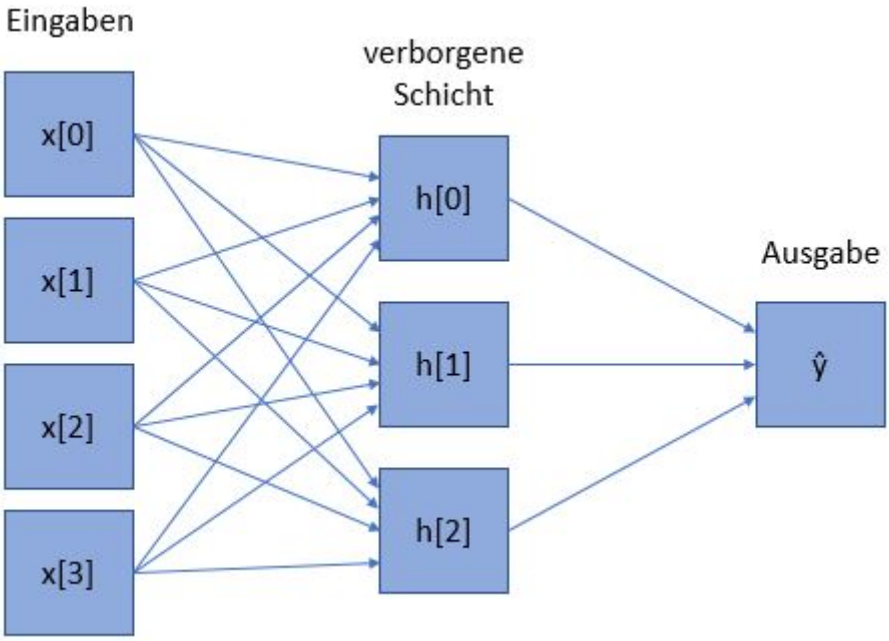

Abbildung 44: Beispiel neuronales Netz. Quelle: [MGR17], S. 100

weils 15 Neuronen angewendet. Die Genauigkeit auf den Testdaten beträgt ca. 90%.
Auf den Smartmeter-Daten wird ein neuronales Netz mit zwei verborgenen Schichten mit jeweils fünf Neuronen angewendet. Die Genauigkeit beträgt ebenfalls ca. 90% auf den Testdaten.

```
1 from sklearn.preprocessing import StandardScaler
2 scaler = StandardScaler()
3 # Fit only to the training data
4 scaler.fit(X_train)
5 X_train = scaler.transform(X_train)
6 X_test = scaler.transform(X_test)
7 X_train = pd.DataFrame(X_train,columns=['minute', 'hour', '
      month', 'day', 'year'])
8 X_test = pd.DataFrame(X_test,columns=['minute', 'hour', '
      month', 'day', 'year'])
```

Listing 47: Skalierung der Daten für das neuronale Netz

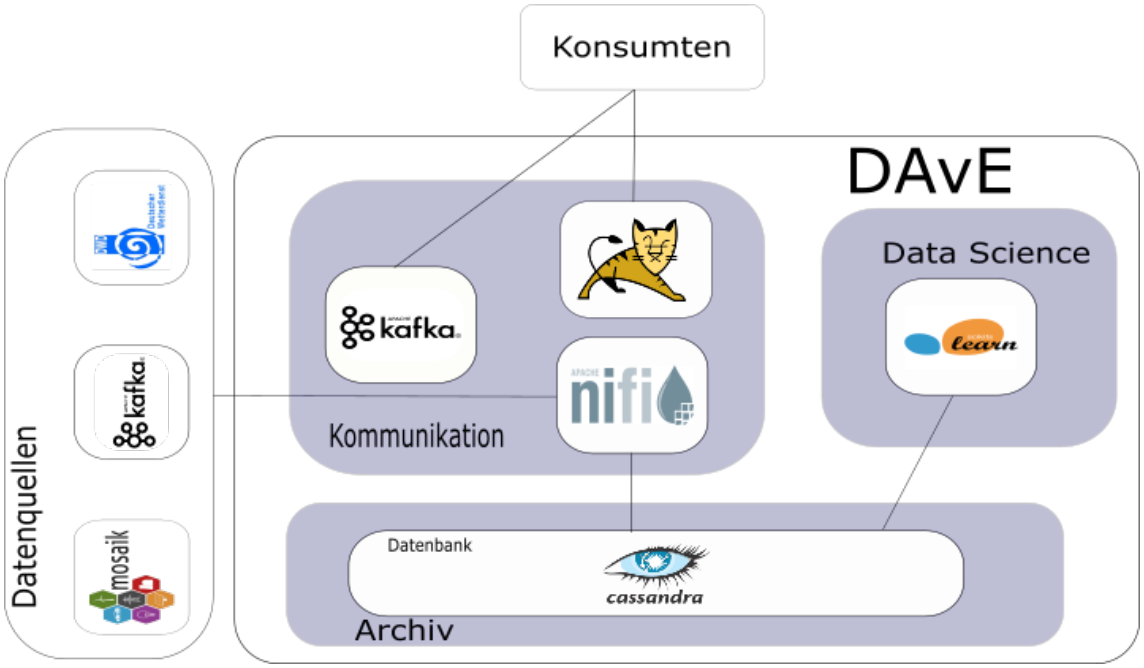

Abbildung 45: Gesamarchitektur DAvE

7. Gesamtarchitektur

In diesem Abschnitt wird abschließend zur Realisierung die Gesamtarchitektur visualisiert und erläutert. Dazu gehört dass alle ausgewählten Software-Dienste und -Systeme verortet werden sowie die Interaktion dieser Komponenten untereinander beschrieben wird. Die Gesamtarchitektur ergibt durch den Zusammenschluss dieser ausgewählten Subsysteme.

In Abbildung 45: Gesamarchitektur DAvE ist das Gesamtsystem visualisiert. Wichtig ist, dass das System DAvE als ein Zusammenschluss von diversen Subsystemen verstanden wird. Die Systemgrenzen des Systems DAvE sind der Abbildung zu entnehmen. Als Archivierungs-, bzw. Datenbanksystem wird Apache Cassandra eingesetzt. Innerhalb dessen werden jegliche zu archivierenden Daten persistent gespeichert. Das Subsystem ist geclustert, sodass es auf die Gesamtmenge und den Durchsatz der Archivdaten angepasst werden kann.

Bei dem Subsystem des Bereichs Data Science handelt es sich um eine Python3-Umgebung mit der Machine Learning Bibliothek `scikit-learn`.

Das Bereitstellen von Daten aus dem Datenbanksystem wird mit dem Message Broker System Apache Kafka realisiert. Wie das Apache Cassandra auch das Kafka Subsystem geclustert werden, um den Ansprüchen von Big Data gerecht zu werden.

Ein essentielle Aufgabe übernimmt das System Apache NiFi. Dieses Dataflow Management Subsystem ist dafür zuständig dass Importe und Exporte sowie Transformation der Archivdaten asynchron durchgeführt werden. Im Falle des Imports wird das Topic eines externen Datenerzeugers abonniert und alle neuen Inhalten in das interne Zielformat des Datenbanksystems überführt und in dieses archiviert. Werden diese archivierten Inhalte angefragt und externen Konsumenten zur Verfügung gestellt werden, werden die Daten durch einen Export-Prozess in Apache NiFi in das ursprüngliche Format überführt und auf einen Kafka Topic im DAvE-System exportiert.

Um entsprechende Anfragen an das DAvE-System zu stellen, wird eine RESTful-Schnittstelle eingesetzt. Darüber können z.B. die archivierten Smart Meter Daten für einen bestimmten Zeitraum angefragt werden. Bei der Plattform der Webserver-Komponente handelt es sich um einen Apache Tomcat Webserver, ein Subsystem. Die RESTful-Schnittstelle ist als Java Servlet auf dem Webserver realisiert. Dabei handelt es sich um eine Eigenentwicklung basierend auf dem Swagger-Framework, genauer den Werkzeugen Swagger Editor und Swagger Codegen (siehe hierfür den Unterabschnitt 6.3.2: Umsetzung).

Im Kontext der Gesamtarchitektur sind darüber hinaus ebenfalls Datenquellen und Konsumenten zu nennen. Zu den Datenquellen gehört z.B. das Topic eines Message Broker eines externen Datenerzeugers welcher dort die Daten hinterlegt oder auch der FTP-Server des Deutschen Wetterdiensts. Wie bereits in diesem Abschnitt erwähnt, extrahiert DAvE mit dem Subsystem NiFi aus diesen Quellen Daten und archiviert diese.

Konsumenten können Anfragen über die bereits beschriebene RESTful-Schnittstelle stellen. Bei den Anfragen kann es sich zum einen um das Bereitstellen von archivierten Daten oder zum anderen um das Erzeugen eines Machine Learning Modells handeln. Die Ergebnisse dieser Anfragen werden in der Regel ebenfalls per Apache NiFi asynchron auf das Message Broker Subsystem Kafka innerhalb des DAvE-Systems bereitgestellt und können dort von den Konsumenten abgefragt werden.

8. Test und Evaluation

Das Ziel der Projektgruppe bestand darin, ein möglichst fehlerfreien Prototypen zu entwickeln. Aus diesem Grund wurde sich frühzeitig dazu entschieden ausführlich zu testen. Es wurde beschlossen, alle selbst entwickelten Komponenten mittels Komponententests auf korrekte Implementierung zu überprüfen. Darüber hinaus wurde ein Testkonzept zur Überprüfung der logischen Funktionalität der einzelnen Komponenten und des Gesamtsystems entwickelt.

8.1. Testumgebung

Um das Archivieren von Smartmeterdaten im COSEM-XML-Format testen zu können, war es notwendig den LV-Simulator auf dem Clustersystem auf der dafür vorgesehenen VM *mosaik-VM* zu installieren, zu erweitern und in Betrieb zu nehmen. Im Folgendem werden die notwendigen Schritte zur Installation, die Erweiterungsprogrammierung und die Inbetriebnahme inklusive der zugrundeliegenden Konfiguration erläutert.

8.1.1. Installation LV-Simulator

Um die Softwaresimulationssoftware (siehe 4.5.2: Low Voltage Simulator) in der Systemumgebung auf der VM *mosaik-vm* (siehe 11: Systeminfrastruktur des DAvE-Cluster) in Betrieb zu nehmen, waren diverse Installationsschritte und nachträgliche Programmerweiterungen notwendig (diese werden in 8.1.2: Erweiterungsprogrammierung LV-Simulator beschrieben). Die notwendigen Installationsschritte werden in diesem Abschnitt vorgestellt.

Bei der Installation traten unvorhersehbare Schwierigkeiten auf. Der Dokumentation auf der Projektwebseite folgend (siehe [OFF18], Menüpunkt Wiki → Installation) kam es beim Testen des LV-Simulators, auch ohne selbstgeschriebene Erweiterungen, zu Laufzeitfehlern. Diese konnten darauf zurückgeführt werden, dass bei der Installation des abhängigen Python-Paktes `mosaik` das darin benötigte und automatisch mitinstallierte Paket `networkx` mit einer falschen Version geladen worden ist. Der dokumentierten Installation mussten also noch weitere Schritte folgen. Diese sind im Listing 48: Auflösen Versionsprobleme des Pakets networkx zu sehen. Mit der Zeile 1 wird das Paket `mosaik` inkl. des Paktes `networkx` in einer falschen Version installiert. Darauffolgend muss dieses wieder deinstalliert werden. Mit dem dritten Befehl wird dann `networkx` mit expliziter Angabe der zu installierenden Version geladen.

Die vollständige Installation, bzw. die notwendigen Schritte zur Installation sind im Listing 49: Installation LV-Simulator angegeben. Als Voraussetzung sei angemerkt, dass die Python-Pakete `pip` und `virtualenv` bereits auf der VM installiert worden sind. Beginnend mit der Zeile 1 wird zunächst ein neues Verzeichnis */opt/nds* erzeugt. Mit dem Befehl in

```
1 pip install mosaik
2 pip uninstall networkx
3 pip install networkx==1.11
```

Listing 48: Auflösen Versionsprobleme des Pakets networkx

```
1  sudo mkdir /opt/nds
2  cd /opt/nds
3  sudo sudo chwon -c benutzer:gruppe nds
4  git clone link_zum_nds_repo netzdatenstrom_dave
5  virtualenv -p /usr/bin/python3.4 .virtualenvs/lv
6  source .virtualenvs/lv/bin/activate
7  pip install xlrd
8  pip install numpy
9  pip install requests
10 pip install mosaik
11 pip uninstall networkx
12 pip install networkx==1.11
13 cd netzdatenstrom_dave/LV-Simulation/lv_simulation/
14 python src/simulation_main.py ../simulation_scenarios/DAvE/
       scenario_config_simple_consoleoutput.json
```

Listing 49: Installation LV-Simulator

Zeile 3 wird der Verzeichniseigentümer auf den Benutzer geändert, welcher alle weiteren Schritte vornimmt und den Simulator startet. Dabei steht *benuter* und *gruppe* stellvertretend für den entsprechenden Benutzer und die dazugehörige Benutzergruppe. Der nächste Befehl klont das git-Repository des LV-Simulators und legt es im Verzeichnis *netzdatenstrom_ dave* an. Wie auf der Projektwebseite des LV-Simulators empfohlen wird (siehe [OFF18], Menüpunkt Wiki → Installation), wird mit dem Befehl der Zeile 6 eine virtuelle Python-Umgebung mit der Python-Version 3.4 eingerichtet. Nachdem in diese mit dem nächsten Befehl gewechselt worden ist, können dann die benötigten Python-Pakete in die virtuelle Python-Umgebung *lv* installiert werden. Die Probleme mit der Versionsnummer des Pakets `networkx` wurden in diesem Abschnitt bereits erläutert. Mit dem Befehl der Zeile 14 wird der LV-Simulator mit einer einfachen Konfiguration und Ausgabe auf dem Terminal gestartet.

8.1.2. Erweiterungsprogrammierung LV-Simulator

Da die Projektgruppe einen direkten Zugriff auf das Entwicklungsrepository hatte und der LV-Simulator laufend erweitert worden ist, haben wir uns dazu entschieden dieses auf ein eigenes Repository manuell zu spiegeln. Dies hat die Vorteile, dass zum einen die Hoheit des Repositorys bei der Projektgruppe liegt und somit Zugriffsschwierigkeiten darauf vermie-

den werden und zum anderen, dass Programmerweiterung und das Fixieren[11] von Entwicklungsständen möglich sind ohne sich mit den Verantwortlichen des Original-Repositorys absprechen zu müssen. Diese Anpassungen können dennoch bei Bedarf problemlos in das Original-Repository überführt werden.

Durch die Anforderung bzw. der Schnittstellendefinition, dass von DAvE zu archivierende Daten auf einem Messaging System vorliegen müssen, musste der LV-Simulator dahingehend erweitert werden sodass dieser die erzeugten simulierten Smartmeterdaten auf einen zu konfigurierenden Messaging System, in diesem Fall Apache Kafka, weiterleitet.

Dafür war es notwendig eine neue Output-Klasse zu schreiben, welche mit Hilfe der Kafka API[12] für Python die generierten Datensätze auf einen Kafka Topic schreibt. Neben dem Schreiben dieser Klasse und der Einbindung dieser in den bestehenden Quellcode, war es ebenfalls notwendig eine neue Konfigurationsdatei inklusive neuer Konfigurationsparameter zu erstellen. Diese enthält zusätzlich die Informationen der Adresse des Kafka Brokers inklusive des Ports sowie den Namen des Ziel-Topics. Darüber hinaus ist in der neue Output-Klasse anhand eines Parameters zu steuern ob die Smartmeterdaten im JSON -oder COSEM-XML-Format erzeugt werden sollen. Ein Auszug der beschriebenen Erweiterung ist in 50: Auszug Kafka-Output Klasse, LV-Simulator zu sehen.

In Zeile 7 wird ein Produzent für Nachrichten des konfigurierbaren Kafka Brokers `bootstrap-servers` erzeugt. Der Ziel-Kafka Topic wird aus dem Parameter `topic` übernommen. Ebenfalls wird in Zeile 9 der Parameter `output_type` gelesen, welche dazu führt dass die Smart-Meter-Daten im JSON- oder COSEM-XML-Format an das Kafka Topic gesendet werden. Diese Fallunterscheidung und das Senden der Nachrichten ist den Zeilen 15 bis 19 zu entnehmen.

8.1.3. Inbetriebnahme LV-Simulator

Nachdem die Installation und die Erweiterungsprogrammierung durchgeführt wurde, konnte der LV-Simulator zu Testzwecken in Betrieb genommen werden. Da es für die Tests der Archivierung keine Bedeutung hat ob und welche Struktur der Generierung der Smartmeterdaten zu Grunde liegt, wurde eine simple Konfiguration des Simulators verwendet. Wie bereits im Abschnitt 8.1.2: Erweiterungsprogrammierung LV-Simulator erwähnt, waren zusätzliche Konfigurationsparameter anzugeben, damit die Ausgabe des Simulators im COSEM-XML-Format auf ein spezifizierten Kafka Broker erfolgt. Die dafür notwendigen Zeilen in der Konfiguration sind im Listing 51: Auszug Konfiguration, LV-Simulator zu sehen.

Beim Ausführen des LV-Simulators ist darauf zu achten dass, wie im Listing 49: Installation LV-Simulator in Zeile 14 sichtbar, das Ausführungsverzeichnis `/LV-Simulation/lv_simulation` ist, da die Konfiguration auf weitere Dateien im relativen Pfad `data/` verweist.

[11] gemeint ist das Setzen von tags im git-Repository
[12] siehe `https://pypi.python.org/pypi/kafka-python`

```
class Kafka(Simulator):
    def init(self, sid, output_options, start_difference):
        self.step_size = output_options['step_size']
        self.start_difference = start_difference
        if start_difference == 0:
            self.first_step = False
        self.producer = KafkaProducer(bootstrap_servers=
            output_options["bootstrap_servers"])
        self.topic = output_options["topic"]
        self.output_type = output_options["output_type"]
        return self.meta

    def step(self, time, inputs):
        for console_id, smgw in inputs.items():
            for smgw_id, smgw_cosem in smgw['message'].items
                ():
                if self.output_type.upper() == "JSON":
                    self.producer.send(self.topic, json.
                        dumps(smgw_cosem).encode())
                elif self.output_type.upper() == "XML":
                    cosem_xml_message = cosem_xml.
                        convert_json_to_xml(smgw_cosem)
                    self.producer.send(self.topic,
                        cosem_xml_message)

        if self.first_step:
            self.first_step = False
            return time + self.start_difference
        return time + self.step_size
```

Listing 50: Auszug Kafka-Output Klasse, LV-Simulator

```
  "output": "Kafka",
  "output_options":
  {
    "step_size": 900,
    "bootstrap_servers": "master.local:9092",
    "topic": "LV_SIM_XML",
    "output_type": "xml"
  },
```

Listing 51: Auszug Konfiguration, LV-Simulator

8.2. Evaluationskonzept

Für die Abnahme des System ist es notwendig, dass die definierten Anforderungen erfüllt werden und eine fehlerfreie Nutzung gewährleistet wird. Dazu wurde ein Testplan mit unterschiedlichen Testarten definiert, was die Überprüfung der Funktionen unterstützt, wodurch mögliche Fehler identifiziert werden können. Mit den Tests sollen zum einen die Anforderungen an das System geprüft werden, darüber hinaus werden die abgebildeten Anwendungsfälle des Systems für einen fehlerfreien Durchlauf erprobt. Die gewählten Testarten werden im Folgenden kurz dargestellt:

- **Komponententest** Es werden alle Komponenten des Systems getestet. Dazu werden Datenimport, Archivierung, Datenanalyse und Schnittstellen separat voneinander betrachtet und auf die jeweiligen Funktionen geprüft, sodass mögliche Fehler spezifisch zur Komponente identifiziert werden können.

- **Integrationstest** Nach der Prüfung der einzelnen Komponenten muss das Zusammenspiel der Komponenten geprüft werden. Es gilt das Zusammenspiel und die Funktionen an Schnittstellen und Übergabepunkten auf Fehler zu prüfen.

- **Systemtest** Nachdem die einzelnen Komponenten sowie deren Zusammenspiel geprüft wurden ist es notwendig, einen Test des gesamten Systems durchzuführen. Ziel ist es Grundfunktionalitäten des Systems zu überprüfen. Es geht hier nicht um eine inhaltliche Überprüfung des System, sondern darum festzustellen, ob die Infrastruktur für das Gesamtsystem ausreichend konzipiert ist.

In den nachfolgenden Tabellen werden die entsprechenden Tests inklusive der entsprechenden Ergebnisse zusammengefasst. Die Tabellen geben einen Überblick über die durchgeführten Tests der Projektgruppe.

8.3. Komponententest

In diesem Abschnitt werden die definienerten Komponententests vorgestellt. Diese befinden sich in den Tabellen 12: Komponententests I und dieser folgenden. Die Tests werden durch ein Code, eine Zugehörigkeit und eine Beschreibung definiert. Zusätzlich wird ein erwartetes Ergebnis definiert, dass anhand der Anforderungen formuliert wurden. Das tatsächliche Ergebnis beschreibt formal das Ergebnis des Systems. Es wurden insgesamt 31 Komponententests durchgeführt.

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
T01	Import / Archivierung	Historische Wetterdaten archivieren	Die historischen Wetterdaten werden durch Apache NiFi vom FTP Server des deutschen Wetterdienstes heruntergeladen und in Apache Cassandra abgespeichert.	Die historischen Daten befinden sich in der Datenbank.	Die Daten können erfolgreich vom FTP-Server abgerufen und danach in Apache Cassandra abgespeichert werden.
T02	Import / Archivierung	Aktuelle Wetterdaten archivieren	Die aktuellen Wetterdaten werden durch Apache NiFi vom FTP Server des deutschen, Wetterdienstes heruntergeladen und in Apache Cassandra abgespeichert.	Die aktuellen Daten befinden sich in der Datenbank.	Die Daten können erfolgreich vom FTP-Server abgerufen und danach in Apache Cassandra abgespeichert werden.
T03	Import / Archivierung	Autmatischer Import der aktuellen Wetterdaten	Die Wetterdaten werden automatisch zu einem festgelegten Zeitpunkt von Apache NiFi abgerufen und in Apache Cassandra abgespeichert.	Die aktuellen Daten befinden sich in der Datenbank.	Die aktuellen Wetterdaten werden täglich erfolgreich in Apache Cassandra aktualisiert.
T04	Import / Archivierung	Datensimulation	Die Simulation wird auf ihre Funktionalität geprüft. Zu diesem Zweck wird die Simulation gestartet und die Ausgabe überprüft.	Die simulierten Daten werden auf der Konsole ausgegeben.	Die simulierten Daten werden erfolgreich auf der Konsole ausgegeben.
T05	Import / Archivierung	Archivierung simulierter Daten	Die Daten aus der Simulation werden nach ihrer Erstellung über ein Skript automatisch in Apache Cassandra importiert.	Die simulierten Daten befinden sich in der Datenbank.	Die Daten werden aus der Kafka Topic glesen und erfolgreich in Apache Cassandra gespeichert.
T06	Import / Archivierung	Daten auf Kafka Topic schreiben	Selektierte Daten werden aus dem Archiv auf das Kafka Topic geschrieben.	Datensatz liegt auf dem Kafka Topic.	Die selektierten Daten werden erfolgreich auf das Kafka Topic geschrieben. Die Daten können über einen Kafka Consumer abgerufen werden.Voraussetzung: Die angefragten Daten liegen im Archiv vor.

Tabelle 12: Komponententests I

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
T07	Import / Archivierung	Daten von Kafka Topic lesen und archivieren	Datensätze werden durch Apache NiFi vom Kafka Topic gelesen und in Apache Cassandra abgespeichert.	Die Daten sind in Apache Cassandra abgelegt.	Es können erfolgreich Datensätze durch Apache NiFi vom Kafka Topic abgerufen und in der passenden Cassandra Tabelle abgespeichert werden. Erfolgreich, getestet mit Testdaten von E-Stream.
T08	Import / Archivierung	Automatischer Import der Daten vom Kafka Topic	Die Datensätze werden automatisch durch Apache NiFi vom Kafka Topic gelesen und in Apache Cassandra gespeichert.	Die Daten sind in Apache Cassandra abgelegt.	Die Daten sind in Apache Cassandra gespeichert.
T09	Schnittstelle	Erreichbarkeit der Endpunkte prüfen	Über ein cURL Statement werden alle Enpunkte angesprochen.	Es erfolgt eine Rückmeldung aller Endpunkte.	Jeder Endpunkt gibt ein korrektes Ergebnis zurück.
T10	Schnittstelle	Statusabfrage Smart Meter	Eine Anfrage wird an den Endpunkt versendet.	Es erfolgt eine Rückmeldung (Läuft; Erfolgreich beendet; Abgebrochen; Job ID exisitert nicht).	Es kann anhand einer Universally Unique Identifier (UUID) der Endpunkt angesprochen werden und erhält einen dieser Rückmeldungen.
T11	Schnittstelle	Anfrage historischer Smart-Meter-Daten	Eine Anfrage wird an den Endpunkt versendet.	Topic Name und URI.	Die Response ist eine URI und eine Topic Bezeichnung die über alle MB einzigartig ist.
T12	Schnittstelle	PMML Modell beauftragen	Eine Anfrage wird an den Endpunkt versendet.	Topic Name und URI.	Die Response ist eine URI und eine Topic Bezeichnung die über alle MB einzigartig ist.

Tabelle 13: Komponententests II

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
T13	Schnittstelle	Statusabfrage einer Wetterdatenanfrage	Eine Anfrage wird an den Endpunkt versendet.	Es erfolgt eine Rückmeldung (Läuft; Erfolgreich beendet; Abgebrochen; Job ID exisitert nicht).	Es kann anhand einer UUID der Endpunkt angesprochen werden und erhält einen dieser Rückmeldungen.
T14	Schnittstelle	Anfrage von Klimadaten	Eine Anfrage wird an den Endpunkt versendet.	Topic Name, URI und Daten der nächst gelegenen Wetterstation (Länge und Breitengrad).	Es kann anhand einer UUID der Endpunkt angesprochen werden und erhält einen dieser Rückmeldungen.
T15	Schnittstelle	Abfrage von angebotenen PMML-Modellen	Eine Anfrage wird an den Endpunkt versendet.	Liste angebotener PMML-Modelle, (Modell ID; Methode; Beschreibung; Daten der Generierung).	Man erhält eine Liste aller angebotenen PMML-Modelle.
T16	Schnittstelle	Abfrage des aktuellsten PMML-Modells eines Typs durch eine vordefinierte ID	Eine Anfrage wird an den Endpunkt versendet.	Das Modell als Jason Objekt im Base64 Format.	Man erhält ein JSON Objekt, mit dem Key-Value Paar pmml:base64 String. Es enthält ein valides PMML-Modell.
T17	Schnittstelle	Erhalt des aktuellsten PMMlL-Modells über ein bereits vorhandenen PMML-Modell	Eine Anfrage wird an den Endpunkt versendet.	Die aktuelle Version eines PMMl-Modells wird über die Schnittstelle empfangen.	Es wird das Übergebene PMML analysiert und neu berechnet, dieses wird dann.
T18	Schnittstelle	Statusabfrage über ein beauftragtes PMMl-Modell	Eine Anfrage wird an den Endpunkt versendet.	Es erfolgt eine Rückmeldung (Läuft; Erfolgreich beendet; Abgebrochen; Job ID exisitert nicht).	Es kann anhand einer UUID der Endpunkt angesprochen werden und erhält einen dieser Rückmeldungen.

Tabelle 14: Komponententests III

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
T19	Schnittstelle	Daten auf Kafka Topic schreiben	Ein Datensatz wird manuell auf das Kafka Topic geschrieben.	Der Datensatz liegt auf dem Kafka Topic.	Datensätze lassen sich manuell auf einen Kafka Topic schreiben.
T20	Schnittstelle	Daten von Kafka Topic lesen	Ein Datensatz wird manuell von dem Kafka Topic gelesen.	Der Datensatz, welcher auf dem Kafka Topic liegt.	Datensätze lassen sich manuell von einem Kafka Topic lesen.
T21	Datenanalyse	Zugriff auf Daten in der Datenbank	Über Jupyter Notebook wird auf einen relevanten Datensatz zugegriffen. Über den Cassandra Driver wird auf einen relevanten Datensatz zugegriffen.	Die Werte aus den Daten werden beim Ausführen übernommen.	Daten werden in ein pandas-Datenframe übernommen.
T22	Datenanalyse	Datenauswahl zeitlich eingrenzen	Über Jupyter Notebook werden relevante Daten mit von bis Werten ausgewählt. Über den Cassandra Driver werden relevante Daten mit von bis Werten ausgewählt.	Bei der Ausführung wird nur auf den eingegrenzten Daten Datenbereich zugegriffen.	Eingegrenzte Daten werden in ein pandas-Datenframe übernommen.
T23	Datenanalyse	SVM-Trainings-/Testdaten erstellen	Die Daten werden gesplittet, danach gefittet und die Testdaten werden erstellt. Die Daten werden gesplittet, danach wird das Modell gefittet und mithilfe der Testdaten evaluier.	Es wurde ein Scorewert für Testdaten und Trainingsdaten erstellt.	Mithilfe der Score-Methode wird die Genauigkeit berechnet.
T24	Datenanalyse	Lineare-Regressions Testdaten erstellen, Lineare-Regressions Trainings-/Testdaten erstellen	Die Daten wurden gesplittet und werden danach gefittet und die Testdaten werden erstellt. Die Daten werden gesplittet, danach wird das Modell gefittet und mithilfe der Testdaten evaluiert.	Es wurde ein Scorewert für Testdaten und Trainingsdaten erstellt.	Mithilfe der Score-Methode wird die Genauigkeit berechnet.

Tabelle 15: Komponententests IV

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
T25	Datenanalyse	Random Forest Trainings-/Testdaten erstellen	Die Daten werden gesplittet, danach wird das Modell gefittet und mithilfe der Testdaten evaluiert.	Es wurde ein Scorewert für Testdaten und Trainingsdaten erstellt.	Mithilfe der Score-Methode wird die Genauigkeit berechnet.
T26	Datenanalyse	Neuronales Netz Trainings-/Testdaten erstellen	Die Daten werden gesplittet, danach wird das Modell gefittet und mithilfe der Testdaten evaluiert.	Es wurde ein Scorewert für Testdaten und Trainingsdaten erstellt.	Mithilfe der Score-Methode wird die Genauigkeit berechnet.
T27	Datenanalyse	Support Vector Machine-Modell erstellen	SVM-Modell Erstellung über Jupyter Notebook ausführen. SVM-Modell-Erstellung über ein Python Script ausführen.	Es wurde ein SVM-Modell (PMML) erstellt.	SVM-PMML wird in Apache Cassandra abgespeichert.
T28	Datenanalyse	Lineare Regression-Modell erstellen	Lineare-Regressions-Modell Erstellung über Jupyter Notebook ausführen, Lineare-Regression-Modell Erstellung über ein Python Script ausführen.	Es wurde ein Lineare-Regressions-Modell (PMML) erstellt.	Lineare Regression-PMML wird in Apache Cassandra abgespeichert.
T29	Datenanalyse	Random Forest-Modell erstellen	Random Forest-Modell Erstellung über ein Python Script ausführen.	Es wurde ein Random-Forest Modell (PMML) erstellt.	Random Forst-PMML wird in Apache Cassandra abgespeichert.
T30	Datenanalyse	Neuronales Netz-Modell erstellen	Neuronales Netz-Modell Erstellung über ein Python Script ausführen.	Es wurde ein neuronales Netz Modell (PMML) erstellt.	Neuronales Netz-PMML wird in Apache Cassandra abgespeichert.

Tabelle 16: Komponententests V

8.4. Integrationstests

Die Integrationstests umfassen werden in diesem Abschnitt vorgestellt. Dazu wurde ebenfalls eine Tabellenstruktur genutzt. Auch hier umfasst die Tabelle einen Testcode, den Testfall, eine Zugehörigkeit und eine Beschreibung. Die erwarteten Ergebnisse werden dann mit den tatsächlichen Ergebnissen verglichen. Das System wird hier als Blackbox betrachtet. Die Tabelle 17: Integrationstests zeigt die Ergebnisse.

8.5. Systemtests

Im folgenden Abschnitt wird auf die Systemtests eingegangen. Diese umfassen 5 Tests um mehrere Bereiche und Merkmale des Gesamtsystems zu prüfen. Die Tabelle 18: Systemtests zeigt die unterschiedlichen Testfälle. Die Beschreibung werden die Testfälle mit dem Ergebnis beschrieben. Teilweise können keine Ergebnisse der Tests vorgestellt werden.

Nr.	Zugehörigkeit	Testfall	Beschreibung	Erwartetes Ergebnis	Tatsächliches Ergebnis
I01	Import / Archivierung	Import von Wetterdaten	Das System lädt zu einem vorher festgelegten Zeitpunkt, die aktuellen Wetterdaten herunter und archiviert sie in Apache Cassandra	Die aktuellen Wetterdaten sind in Apache Cassandra abgelegt	Die aktuellen Wetterdaten werden täglich erfolgreich in Cassandra abgelegt.
I02	Import / Archivierung	Import von Smart-Meter-Daten	Das System lädt die in der Simulation erzeugten Smart-Meter-Daten automatisch in Apache Cassandra ab	Die simulierten Daten sind in Apache Cassandra abgelegt	Die Daten sind in Apache Cassandra abgelegt.
I03	Schnittstelle	PMML-Modell abfragen	Ein aktuelles PMML-Modell wird über Schnittstelle angefragt und an den Anfrager übermittelt	Der Anfrager erhält das angefragte aktuelle PMML-Modell	Der Anfrager erhält das aktuellste PMML-Modell.
I04	Schnittstelle	PMML-Modell beauftragen	Ein PMML-Modell wird über die Schnittstelle beauftragt. Das Modell wird berechnet und im Anschluss, über die Schnittstelle, an den Auftraggeber übermittelt	Der Auftraggeber erhält das PMML-Modell	Der Auftraggeber erhält das neuberechnete PMML-Modell.
I05	Schnittstelle	Klimadaten anfragen	Klimadaten werden über die Schnittstelle anberufen und über die Schittstelle an den Anfrager übermittelt	Der Anfrager erhält die historischen Klimadaten	Der Anfrager erhält die angefragten Daten über ein Messaging System im JSON-Format.
I06	Schnittstelle	Smart-Meter-Daten abfragen	Smart-Meter-Daten werden über die Schnittstelle abgerufen und über die Schnittstelle an den Anfrager übermittelt	Der Anfrager erhält die historischen Smart-Meter-Daten	Der Anfrager erhält die angefragten Daten über ein Messaging System im COSEM-Format.

Tabelle 17: Integrationstests

Nr.	Testfall	Beschreibung
S1	Performance	Verschiedene Funktionalitäten des System werden mehrmals hintereinander ausgeführt. Die Performance muss dabei gleichbleibend sein.
S2	Erreichbarkeit	Da sich das System in einer Testumgebung befindet, ist die Erreichbarkeit unter Realbedingung nicht testbar.
S3	Korrektheit	Die Korrektheit des Systems wird durch das mehrmalige einspielen der gleichen Datensätze überprüft.
S4	Skalierbarkeit	Da sich das System in einer Testumgebung befindet, ist die Skalierbarkeit nicht überprüfbar. Durch den Einsatz einer verteilten Datenbank ist es theoretisch jedoch möglich das System auf eine beliebige Größe zu skalieren.
S5	Sicherheit	Da sich das System in einer Testumgebung befindet, ist die Sicherheit nicht überprüfbar. Bei der Entwicklung eigener Komponenten wurde darauf geachtet, dass Daten durch die Ausführung dieser Komponenten nicht verändert werden können. Bei der Sicherheit eingesetzter Komponenten wie Apache Kafka oder Apache Cassandra kann die Sicherheit von Seiten der Projektgruppe nicht gewährleistet werden.

Tabelle 18: Systemtests

9. Fazit und Ausblick

In diesem abschließenden Kapitel werden im Abschnitt 9.1: Fazit die Erfahrungen der Projektgruppe zusammengefasst und bewertet. Dabei wird auf unterschiedliche Aspekte des Projektes eingegangen. Im nachfolgenden Abschnitt wird ein Ausblick über mögliche Funktionserweiterungen und Verbesserungen für DAvE gegeben. Dabei werden die unterschiedlichen Bereiche von DAvE betrachtet.

9.1. Fazit

Ablauf des Projekts Der Projektablauf gliederte sich in drei Phasen. Diese waren die Seminarphase, die Entwicklungsphase und die Testphase. Im Nachhinein stellte sich heraus, dass einige in der Seminarphase behandelte Themen für die weitere Projektarbeit nicht relevant waren. Zudem erstreckte sich die Seminarphase über einen zu großen Anteil des Projektzeitraumes. Im ersten Abschnitt der Entwicklung fiel die Konzepterstellung und Technologieauswahl sehr zeitintensiv aus. Ursache für diese Verzögerung war die fehlende Erfahrung im Bereich Big Data und den dazugehörenden Technologien. Beim Versuch die ursprünglich ausgewählten Technologien zum Einsatz zu bringen, traten verschiedene Probleme auf, die die Auswahl anderer Technologien notwendig machten. Aufgrund dessen verzögerte sich die weitere Projektarbeit deutlich. Ein weiterer Verzögerungsfaktor war, dass zu Beginn der Technologieauswahl kein einheitliches Testsystem zur Verfügung stand. Aufgrund der gesamten Verzögerung ging viel Implementierungs- und Testzeit verloren, sodass eine neue Meilensteinplanung durchgeführt werden musste. Das agile Vorgehen hat sich im Laufe des Projekts als sinnvolles Vorgehensmodell etabliert. Das Vorgehensmodell Scrum wäre mit einem zu hohen Koordinationsaufwand verbunden gewesen.

Kooperation E-Stream Die Kooperation mit der PG E-Stream hatte positive und herausfordernde Aspekte. Zu den Herausforderungen zählte die Kommunikation. Diese verlief zeitweise stockend. Für die schnelle Kommunikation wurde ein Instant-Messaging Dienst ausgewählt. Dieser wurde im Laufe des Projekts durch E-Mail-Kommunikation abgelöst. Behindert wurde die Kooperation vor allem durch einen unterschiedlichen Projektfortschritt. Zur Verbesserung wäre eine synchronisierte Seminarphase vorteilhaft gewesen. Aufgrund dessen konnten keine gemeinsamen Entscheidungen getroffen werden, stattdessen konnte nur noch auf Entscheidungen der anderen PG reagiert werden. Befruchtend an der Zusammenarbeit, war der andere Blickwinkel auf die gleiche Thematik. Ein Zusammenarbeit der Projektgruppen ist aus unserer Sicht sinnvoll, erfordert jedoch eine genaue Planung und ständige Abstimmung.

Werkzeuge für die Projektorganisation Die Verwaltung der im Projektverlauf entstandenen Dokumente wurde mit dem Organisationswerkzeug *Confluence* realisiert, zu diesen

zählen unter anderem die Sitzungsprotokolle. Um einen Gesamtüberblick über alle Aufgabenpakete zu erhalten, sollte im weiteren Projektfortschritt *Jira* genutzt werden. Dies stellte sich jedoch als zu großer Organisationsaufwand heraus. Der Mehrwert rechtfertigte nicht den Mehraufwand.

Für die Versionierung wurde einer von der Universität bereitgestellter Git Dienst genutzt. Dieser wurde nicht nur für den Quellcode genutzt, sondern auch für die Zusammenarbeit bei der Erstellung der Dokumentation.

Technologien Im Rahmen der Data Science Komponente war die scikit-learn Bibliothek ein zentrales Werkzeug. Diese bietet eine Vielzahl von Modellen. Dadurch ließen sich neue Modelle ohne großen Zeitaufwand implementieren und anwenden. Die Dokumentation war sehr gut gepflegt und beschrieben. Es gibt eine sehr aktive Online-Community, von der viele Fragestellung bereits beantwortet wurden. Dies unterstützt ein sehr effizientes Arbeit mit der Bibliothek. Ein Nachteil ist die fehlende Import-Funktion für PMML-Dateien. Diese hat sich im Projektverlauf als relevant herausgestellt, da diese zur Umsetzung eines Use Cases der PG E-Stream erforderlich gewesen wäre.

Im Rahmen des Datenimports hat sich Apache NiFi als ein mächtiges Werkzeug herausgestellt. Der gesamte Prozess wird über Apache NiFi gesteuert. So konnte bspw. der problematische Datenimport mittels Java von einem FTP-Server durch Apache NiFi ersetzt werden. Apache NiFi bietet unter anderem eine Vielzahl von Funktionen um Daten zu manipulieren. Es ist möglich den Dataflow visuell darzustellen. Die Benutzeroberfläche half bei der Erstellung von Prozessen und der Konfiguration dieser. Die Vielzahl von vordefinierten Prozessoren ermöglichen einen vielseitigen Einsatz sowie die Integration weiterer Module. So ist es auch möglich die Anfrage der historischen Daten mit Apache NiFi zu bearbeiten.

Mit dem DAvE-System wird ein beispielhafter Ansatz geschaffen, die gestiegenen Anforderungen an das Datenaufkommen durch intelligente Messsysteme zu bewerkstelligen. Ferner wird durch die Verarbeitung der Messwerte aus der Niederspannungsebene eine Möglichkeit geschaffen, aus den Daten Mehrwerte und datengetriebene Geschäftsmodelle zu entwickeln. Primär bieten die Big Data Technologien und Data Science Verfahren Lösungen zur Anfrage- und Vermarktungsprognosen oder eine Integration von externen Daten in die Netzbetriebsführung. Dies ermöglicht neben möglichen Kosteneinsparungen eine stärkere Kundenindividualisierung bei der Netzbetriebsführung. Die Daten können für das Netzleitstellenpersonal zusätzlich für neue Netzplanungen oder neue Mehrwertdienste für Industriekunden nutzbar gemacht werden. Darüber hinaus können Reaktionszeiten bei gleichzeitiger Steigerung der Häufigkeit und Komplexität von Eingriffen verkürzt werden.

9.2. Ausblick

Im Laufe der Umsetzung des DAvE-Systems war vor allem die Koordination der verschiedenen Komponenten und Subsystemen im Cluster des Prototypen eine große Herausforderung. Zudem wurden einige Subsysteme installiert, konfiguriert und in Betrieb genommen, jedoch zu einem späteren Zeitpunkt wieder aus Gesamtsystem entfernt. Der Zeitaufwand und die fachliche Herausforderung für diese Aufgaben war signifikant.

Um Probleme in diesem Kontext in Zukunft zu vermeiden empfiehlt es sich Subsysteme mit Hilfe von Containerdiensten zur Verfügung zu stellen. Die Virtualisierungssoftware Docker bietet sich insbesondere an um z. B. den Message Broker oder das Datenbanksystem in einem geclusterten System zu virtualisieren. Dadurch ließe sich Zeit zur Installation der Subsysteme sparen. Allein eine entsprechende Konfiguration der, in Containern bereitgestellten Subsysteme, wäre noch notwendig. Darüber hinaus würde diese Art der Bereitstellung eines komplexen Systems die Portabilität und Stabilität dessen verbessern.

Im Rahmen der Data Science Komponente ließe sich Auswertung über angefragte Modelle die noch nicht angeboten werden erstellen. Auf Basis dieser Auswertungen könnten dann neue Modelle erstellt werden. Zudem wird bisher noch nicht der Score der Modelle mitgeliefert, dieser ließe sich zur Bewertung der Qualität durch den Anwender mit übergeben. Für größere Datensätze bei der Modellerzeugung würde die Möglichkeit bestehen das Trainieren auf mehrere Knoten parallel durchzuführen.

Bei der Benutzung von Apache NiFi ist die Protokollierungsfunktion bisher ungenutzt. Es ist uns zurzeit nicht möglich, automatisch auf Fehlermeldungen oder auf erfolgreichem Datenimporten hingewiesen zu werden. So muss manuell täglich überprüft werden, ob die Daten erfolgreich importiert worden sind oder ob Fehler aufgetreten sind.

Neben den aufgezeigten Anwendungsfällen besteht die Möglichkeit, weitere Nutzen aus den Daten der Energiewende zu schaffen. Durch die Digitalisierung der Stromzähler entsteht auch auf Konsumentenseite der Bedarf, über die verbrauchten Energiemengen informiert zu werden. Die Daten können somit aus dem Archiv abgefragt werden und beispielsweise über ein Letztverbraucher-Portal zur Verfügung gestellt werden.

Literatur

[AVM+12] Albanese, Davide ; Visintainer, Roberto ; Merler, Stefano ; Riccadonna, Samantha ; Jurman, Giuseppe ; Furlanello, Cesare: *mlpy: Machine Learning Python Projekt Homepage*. `http://mlpy.sourceforge.net/`. Version: 2012. – [Online; Stand 31. März 2018]

[Bun16a] Bundesministerium für Wirtschaft und Energie: *Die Digitalisierung der Energiewende*. `https://www.bmwi.de/Redaktion/DE/Artikel/Energie/digitalisierung-der-energiewende.html`. Version: 2016. – [Online; Stand: 28. März 2016]

[Bun16b] Bundestag: *Gesetz zur Digitalisierung der Energiewende*. August 2016. – §60 Nr.3

[Dat18] DataStax, Inc.: *CONSISTENCY/CQL for Apache Cassandra 3.0*. `https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlshConsistency.html`, 2018. – [Online; Stand 15. Februar 2018]

[DLM18] DLMS User Association: *DLMS User Association Homepage*. `https://www.dlms.com/index2.php`. Version: 2018. – [Online; Stand 31. März 2018]

[Ger16] German Watch: *Sechs Thesen zur Digitalisierung der Energiewende: Chancen, Risiken und Entwicklungen*. `https://germanwatch.org/de/download/15649.pdf`. Version: 2016. – [Online; Stand: 28. März 2016]

[Har15] Harrison, G.: *Next generation databases: NoSQL, NewSQL, and Big Data : what every professional needs to know about the future of databases in a world of NoSQL and Big Data*. New York : Apress (IOUG), 2015 (The expert's voice in Oracle)

[Has18] HashiCorp: *HashiCorp Vagrant Homepage*. `https://www.vagrantup.com/`, 2018. – [Online; Stand 15. Februar 2018]

[Her18] Hermann, Pencolé: *nifi-config Github Repository*. `https://github.com/hermannpencole/nifi-config`, 2018. – [Online; Stand 15. Februar 2018]

[HTF09] Hastie, Trevor ; Tibshirani, Robert ; Friedman, Jerome: *The Elements of Statistical Learning*. Springer New York, 2009. `http://dx.doi.org/10.1007/978-0-387-84858-7`. `http://dx.doi.org/10.1007/978-0-387-84858-7`

[Ion15] Ionescu, V. M.: The analysis of the performance of RabbitMQ and ActiveMQ. In: *2015 14th RoEduNet International Conference - Networking in Education and Research (RoEduNet NER)*, 2015. – ISSN 2068–1038, S. 132–137

[JL17] John, V. ; Liu, X.: A Survey of Distributed Message Broker Queues. In: *ArXiv e-prints* (2017), April

[Jup18] Jupyter Notebook: *What is the Jupyter Notebook?* `http://jupyter-notebook-beginner-guide.readthedocs.io/en/latest/what_is_jupyter.html`. Version: 2018. – [Online; Stand 17. Februar 2018]

[KNI18] KNIME AG: *KNIME Homepage*. `https://www.knime.com/`. Version: 2018. – [Online; Stand 31. März 2018]

[Mü16] Müller, Andreas C.: *Introduction to machine learning with Python: a guide for data scientists*. First edition. Beijing ; Boston : O'Reilly Media, Inc, 2016. – ISBN 978–1–4493–6941–5

[MGR17] Müller, Andreas C. ; Guido, Sarah ; Rother, Kristian: *Einführung in Machine Learning mit Python: Praxiswissen Data Science*. Heidelberg : O'Reilly, 2017

[MH13] Moniruzzaman, A. B. M. ; Hossain, Syed A.: *NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison*. 2013

[OAS18] OASIS: *AMQP is the Internet Protocol for Business Messaging*. `https://www.amqp.org/about/what`. Version: 2018. – [Online; Stand 20. März 2018]

[OBALB15] Oussous, Ahmed ; Benjelloun, Fatima-Zahra ; Ait Lahcen, Ayoub ; Belfkih, Samir: *Comparison and Classification of NoSQL Databases for Big Data*. 2015

[OFF18] OFFIS e. V.: *NetzDatenStrom Projektseite*. `https://eprojects.offis.de/redmine/projects/netzdatenstrom`. Version: 2018. – [Online; Stand: 1. Januar 2018]

[Piv18] Pivotal Software: *RabbitMQ Documentation*. `http://www.rabbitmq.com`. Version: 2018. – [Online; Stand 31. März 2018]

[R17] RTools Technology Inc: *RStudio*. `https://www.rstudio.com/`. Version: 2017. – [Online; Stand 20. März 2018]

[Sah15] Sahu, B. K.: *A Real Comparison Of NoSQL Databases HBase, Cassandra & MongoDB*. `https://www.linkedin.com/pulse/real-comparison-nosql-databases-hbase-cassandra-mongodb-sahu/`, 2015. – [Online; Stand 15. Februar 2018]

[Sci17] Scikit-learn developers: *Online Documentation*. `http://scikit-learn.org/stable/`. Version: 2017. – [Online; Stand 31.

März 2018]

[Sky17] Skymind: *What is Eclipse Deeplearning4j?* `https://deeplearning4j.org/`. Version: 2017. – [Online; Stand: 20. März 2018]

[Sol18] Solid IT gmbh: *DB-Engines Ranking - die Rangliste der populärsten Datenbankmanagementsysteme.* `https://db-engines.com/de/ranking`, 2018. – [Online; Stand 15. Februar 2018]

[Sul15] Sullivan, D.: *NoSQL for mere mortals.* Upper Saddle River, NJ : Pearson Education, 2015 (For mere mortals series)

[Ten18] Tensorflow developers: *Projekt Homepage.* `https://www.tensorflow.org/`. Version: 2018. – [Online; Stand 31. März 2018]

[The16] The Apache Software Foundation: *Apache Cassandra Projekt Homepage.* `http://cassandra.apache.org/`, 2016. – [Online; Stand 31. März 2018]

[The18a] The Apache Software Foundation: *Apache HBase Homepage.* `https://hbase.apache.org/`, 2018. – [Online; Stand 15. Februar 2018]

[The18b] The Apache Software Foundation: *Apache Kafka Documentation.* `https://kafka.apache.org/documentation/`. Version: 2018. – [Online; Stand 31. März 2018]

[The18c] The Apache Software Foundation: *Apache NiFi Online Dokumentation.* `https://nifi.apache.org/docs.html`, 2018. – [Online; Stand 15. Februar 2018]

[The18d] The Apache Software Foundation: *Apache Spark Projekt Homepage.* `http://spark.apache.org/`. Version: 2018. – [Online; Stand 29. März 2018]

[Tut18] Tutorials Point Pvt. Ltd.: *Apache Kafka Online Tutorium.* `https://www.tutorialspoint.com/apache_kafka/index.htm`. Version: 2018. – [Online; Stand 31. März 2018]

[Uni17] University of Waikato: *WEKA.* `https://www.cs.waikato.ac.nz/ml/weka/`. Version: 2017. – [Online; Stand: 20. März 2018]

A. Anhang

A.1. Tabelle Auswahl Datenbanken

Vergleich von HBase, Cassandra und MangoDB

	HBase	Cassandra	MongoDB
Description	Wide-column store based on Apache Hadoop and on concepts of BigTable	Wide-column store based on ideas of BigTable and DynamoDB	One of the most popular document stores
Developer	Apache Software Foundation	Apache Software Foundation	MongoDB, Inc
Current release	1.2.6, June 2017	3.11.0, June 2017	3.4.6, July 2017
License	Open Source	Open Source	Open Source “AGPL version3, commercial license available
Language	Java	Java	C++
Fault tolerance	Replication, Partitioning	Replication, Partitioning	Replication
Data Model	Big Table	Big Table and Dynamo	Document Oriented (BSON)
Data Storage	HDFS	Inspired by Amazon’s Dynamo for storing data	Volatile memory, file system
MapReduce	Yes	Yes	Yes
Query Language	API calls REST XML Thrift	API calls CQL Thrift	
Protocol	Custom API, Thrift, Reset	Thrift	TCP/IP

Best for	Real-time access, Do bulk operation (indexing, …)	When you write more than you read (logging) Must use Java	Dynamic queries, Defining indexes, Good performance on a big DB.
Value size max.	2TB	2GB	16MB
Optimized For	Reads	Writes	

A.2. Paper

A.2.1. Data Archive 2.0 - Adaptation of ICT Solutions for the Future Energy Sector Transformation

Data Archive 2.0 - Adaptation of ICT Solutions for the Future Energy Sector Transformation

Keywords: data archive, smart grid, data mining

The ongoing transformation of the energy sector (in German "Energiewende") provides major challenges for energy producers, grid operators and other energy market players, such as field-device manufacturers. One of these challenges is the growing amount of data that is generated by measuring devices, which are going to be the standard for dealing with the energy consumption and production of a single household, such as smart meters. In such a world, a number of new information technologies are emerging with the aim of exploiting the growing amount of data and approaching balance between supply and demand possibilities. In order to address the aforementioned challenge, a conceptual architecture was developed and tested. The architecture supports the generation of prediction models, which are able to make forecasts about energy supply and demand on different levels, through the use of the different methods of machine learning. The architecture consists of three core components:

- Data Archive: a central database for storing all data received and send by the system including the data from multiple heterogeneous external sources as well as models generated by the data mining component.
- Data Mining Component: used to generate various predictions and forecasts based on the data received by the system. This includes forecasts of the future load derived from smart meter data and power generation forecasts from renewable energy sources such as wind and photovoltaic. These models are to be send and received using the Predictive Model Markup Language (PMML), the industry standard for prediction models [1].
- Communication Component: manages the internal and external communication of the system. It processes the data flows between the individual components of the systems and towards external information suppliers and information consumers. It enables the system to receive and process the standard message format for devices communication in smart grids, COmpanion Specification for Energy Metering (COSEM) [2].

In order to prove the feasibility of the architecture a prototypical implementation took place. The proposed architecture stack mainly consists of open source software. For the data archiving component the database system Apache Cassandra has been used. The data mining component has been implemented using Python and the scikit-learn library. The communication component consists of two sub components: Apache Kafka (a message broker) and Apache Nifi (a data flow management system). Following the implementation, the system was tested and the integration of the components was successful. Further tests regarding the performance of the system are necessary.

References

[1] Alex Guazzelli, Michael Zeller, Wen-Ching Lin, Graham Williams, et al. PMML: An open standard for sharing models. *The R Journal*, 1(1):60–65, 2009.

[2] Stefan Feuerhahn, Michael Zillgith, Christof Wittwer, and Christian Wietfeld. Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications. In *Smart Grid Communications (SmartGridComm), 2011 IEEE International Conference on*, pages 410–415. IEEE, 2011.

A.2.2. Datenarchivierung von Energiedaten und Generierung von Vorhersagemodellen

Datenarchivierung von Energiedaten und Generierung von Vorhersagemodellen

Abstract

Die Energiewende stellt die Energiebranche vor Herausforderungen. Um die Daten von Smart Metern zu verarbeiten existieren bereits Ansätze. Die Archivierung und die passenden Analysewerkzeuge können neue Mehrwerte aus diesen Daten generieren. Es soll eine Architektur entwickelt werden, um riesige Datenmengen zu archivieren und daraus mit den passenden Analysewerkzeugen Vorhersagemodelle zu generieren. Das Ergebnis sind Vorhersagemodelle, um bspw. die Erzeugung von Strom besser handzuhaben.

Einleitung

Bis zum Jahr 2032 sollen alle Ferraris-Zähler durch intelligente Messsysteme (Smart Meter) ersetzt werden. Das hat der Bundestag in einem Gesetzesentwurf zur Digitalisierung der Energiewende[1] beschlossen. Die Netzbetreiber sind somit mit der Bewältigung riesiger Datenmengen konfrontiert.

Es existieren bereits Ansätze, um riesige Datenmengen zu verarbeiten, z.B. MapReduce (Rauber & Rünger, 2013). Diese Datenmengen können nach der Verarbeitung archiviert und analysiert werden. Dazu soll eine Architektur entwickelt werden, um die Daten persistent über Jahrzehnte zu archivieren und mit geeigneten Analysewerkzeugen neue Mehrwerte zu generieren. Hierzu sollen erste Vorschläge erarbeitet werden, um die Schnittstellen zwischen den Stromerzeugern, Stromanbietern und Konsumenten zu beschreiben und zu standardisieren. Zusätzlich soll durch die Nutzung geeigneter Analysewerkzeuge Stromerzeuger und -anbieter in die Lage versetzt werden, geeignete Vorhersagemodelle zu erzeugen und zu nutzen.

Solche Vorhersagemodelle geben Auskünfte über den Energieverbrauch oder die Stromerzeugung von unterschiedlichen Zeitperioden. Dabei wird der Energieverbrauch der Haushalte analysiert und einem Verhaltensmuster zugeordnet. Daraus lässt sich eine Nachfragemodell erzeugen, um die erforderliche Menge Strom bereitzustellen. Die Einordnung der Haushalte in Verhaltensmuster muss durch historische Daten trainiert und kontinuierlich verbessert werden. Somit ist die Archivierung ein essenzieller Schritt, um neue Verhaltensmuster zu erzeugen und bestehende zu verbessern.

Die Architektur zur Archivierung und Analyse von großen Datenmengen soll in unterschiedlichen Branchen einsetzbar sein. Die Architektur soll prototypisch im Energiesektor umgesetzt werden und es sollen erste Analysewerkzeuge implementiert werden. Dazu werden unter anderem die Lambda-Architektur, und andere Referenzarchitekturen, bspw. eine Architektur für eine Langzeitarchivierung untersucht und bewertet. Mögliche Analysewerkzeuge sind neuronale Netze, Random Forest, Support vector machine und weitere Algorithmen aus dem maschinellem Lernen. Bei der Konzipierung des Prototyps wird auf getesteten Techniken zurückgegriffen, wie bspw. dem Hadoop Distributed Filesystem (HDFS) und NoSQL Datenbanken (Jiang, 2016). Das Ziel ist, den Prototyp in einem Testverfahren zu erproben und zu bewerten.

[1] „Entwurf eines Gesetzes zur Digitalisierung der Energiewende“, Drucksache 18/7555, 17.02.2016

Literaturverzeichnis

Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified Data Processing on Large Clusters. *Sixth Symposium on Operating System Design and Implementation.* San Francisco.

Jiang, H. W. (2016). Energy big data: A survey. *Access, IEEE, 4*, (S. 3844-3861).

Rauber, T. R., & Rünger, G. (2013). *Parallele Programmierung (3. Aufl. 2013. ed., EXamen.press).* Berlin Heidelberg: : Springer.

Viana, P., & Sato, L. (2014). A Proposal for a Reference Architecture for Long-Term Archiving, Preservation, and Retrieval of Big Data. *2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications*, (S. 622-629).

A.2.3. BUIS Paper DAvE

BUIS-Tage 2018: Projektgruppe DAvE

1 Data-Science zur Energiewende

Durch Datenanalysen Mehrwerte aus der Datenflut der Energiewende generieren.

Zusammenfassung

Das Gesetz der Digitalisierung der Energiewende führt zur Einführung von Smart Metern in Deutschland und zum Aufbau einer digitalen Infrastruktur zwischen Stromerzeugern und -verbrauchern. Hauptaugenmerk liegt auf der Instandsetzung von intelligenten Messsystemen, welche als Kommunikationsplattform mehr Sicherheit gewährleisten können als herkömmliche Lösungen. Dies führt allerdings zu einem erhöhten Datenaustausch, welcher durch die Energiebranche bewältigt werden muss. Das Zusammenwirken von Archivierung und Analysewerkzeugen führt zur einer Handhabung der Datenmengen und ermöglicht zugleich die Generierung von Mehrwerten. Diese Mehrwerte können durch die von DAvE entwickelte Architektur bewerkstelligt werden. Durch Vorhersagemodelle aus den archivierten Daten entsteht die Möglichkeit, dass beteiligte Akteure die Erzeugung und Verbräuche besser kalkulieren und Potentiale für neue Geschäftsmodelle entdecken können.

1.1 Einleitung

Die Digitalisierung der Energiewende führt zum Austausch aller herkömmlichen Ferraris-Zähler durch intelligente Messsysteme bis zum Jahr 2032. Diese Einführung konfrontiert die Netzbetreiber mit der Bewältigung riesiger Datenmengen. Doch die Datenmengen bringen auch neue Chancen für die beteiligten Akteure mit sich. Denn die anfallenden Daten können nicht nur archiviert werden, sondern bieten weitläufige Potentiale zu Datenanalyse. Dazu gilt es eine Architektur zu entwickeln, die Daten persistent über Jahrzehnte archivieren kann und mit geeigneten Analysewerkzeugen neue Mehrwerte durch die Auswertung dieser Daten generieren kann.

Die Architektur zur Archivierung und Analyse von großen Datenmengen wird in unterschiedlichen Branchen einsetzbar sein. Zunächst wird sie prototypisch im Energiesektor umgesetzt und es werden Analysewerkzeuge zur Ableitung von Verhaltensmustern aus den Daten der Haushalte implementiert.

Es muss möglich sein, Stromerzeuger und -anbieter in die Lage zu versetzen, Vorhersagen über Energieverbräuche und Stromerzeugung zu treffen und nutzen zu können. Anhand von generierten Vorhersagemodellen können dazu Auskünfte in unterschiedlichen Zeitperioden getroffen werden. Durch den Datenzuwachs ist es möglich, die Vorhersagemodelle stetig zu optimieren und somit immer bessere Prognosen ableiten zu können. Dies ermöglicht es den Stromerzeugern, die Nachfrage durch Konsumenten zu prognostizieren und somit die bereitzustellende Menge an Strom besser kalkulieren zu können.

Die Umsetzung dieses Vorhabens wird im Rahmen der Projektgruppe DAvE an der Carl von Ossietzky Universität Oldenburg realisiert. Das zugehörige Konzept und dessen Umsetzung werden im Folgenden erläutert.

1.2 Projektgruppenvorstellung

Im Rahmen der Projektgruppe DAvE (Datenarchivierung von Energiedaten) in der Abteilung Very Large Business Applications (VLBA) unter der Leitung von Prof. Dr. Ing Marx Gómez, widmen sich zwölf Masterstudierende aus den Bereichen Informatik und Wirtschaftsinformatik der Konzeption und Entwicklung eines Big Data-Archivs. Im Zeitraum vom 1. April 2017 bis 31. März 2018 wird von der Projektgruppe eine Architektur zur Speicherung und Auswertung von Energiedaten konzipiert und entwickelt, die für Stromerzeuger und –anbieter entscheidende Mehrwerte generieren kann. Dabei steht die Analyse von Netz und Daten im Vordergrund, sodass basierend auf den Erkenntnissen mögliche datengetriebene Geschäftsmodelle aufgezeigt werden können. Die Aufgaben werden von

Teilgruppen mit den Kernkompetenzen Datenbereitstellung und –Import, Datenarchivierung, Datenanalyse und der Erstellung einer Schnittstelle zum Senden und Empfangen von Smart Meter- und Wetterdaten sowie der Bereitstellung von Vorhersagemodellen realisiert. Zusätzlich wurde in der Entwicklungsphase ein Testmanagement eingeführt, welches die Umsetzung während der Entwicklung anhand von Testdrehbüchern prüft. Die Organisation der Projektgruppe wird in Abbildung 1.1 dargestellt.

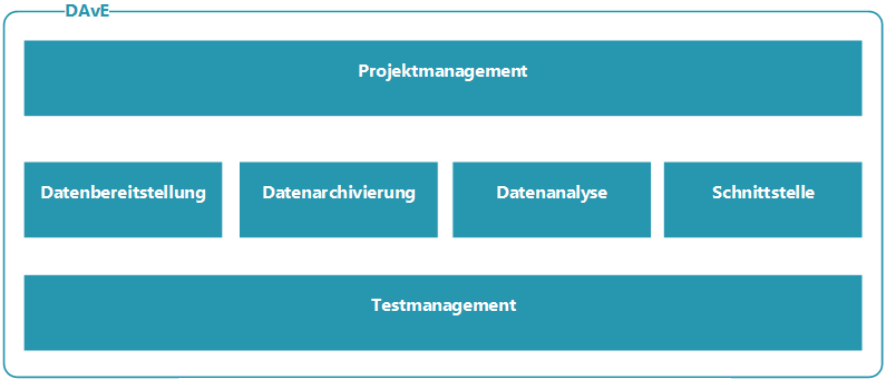

Abb. 1.1: Projektorganisation DAvE

Diese Teilgruppen werden durch Andreas Solsbach, Viktor Dmitriyev und Jad Asswad betreut. Dabei wird durch wöchentliche Abstimmungen der Projektfortschritt kommuniziert und Änderungs- und Optimierungswünsche werden zügig in den Entwicklungsfortschritt eingebunden. Die Arbeitspakete werden anschließend in SCRUM-ähnlichen Iterationen umgesetzt.

1.3 Zielsetzung des Projektes

Das Projekt beschäftigt sich mit der Leitfrage, wie ein datengetriebener Energiemarkt in der Zukunft aussehen kann. Dabei gilt es eine Kombination aus Netz, Daten und Analyse zu schaffen und anhand dieser Erkenntnisse neue Geschäftsmodelle aufzeigen zu können. Es muss die Möglichkeit geschaffen werden, die anfallenden Daten aus Smart Metern und Smart Meter Gateways speichern und archivieren zu können. Dazu wird eine Infrastruktur geschaffen, über die Daten aus Datenquellen entgegengenommen und in einem Archiv abgelegt werden. Bei diesen Daten handelt es sich primär um Datenströme aus einer Niederspannungssimulation von Haushalten. Anschließend muss es ermöglicht werden, die archivierten Daten zur Wissensgenerierung über das Konsumentenverhalten auswerten zu können. Das generierte Wissen wird folglich durch Vorhersagemodelle zur Verfügung gestellt und für Dritte nutzbar gemacht.

Zusätzlich sollen externe Datenquellen in Form von Wetterdaten archiviert werden. Dazu stellt der Deutsche Wetterdienst stündlich Werte bereit, die vom System angefragt und archiviert werden. Diese Daten werden bezüglich Windgeschwindigkeiten und Sonnenstunden ausgewertet und ermöglichen somit eine Prognose für potentielle Energieerzeugung über Wind- und Solarkraft. Zusätzlich ist es möglich, die Daten auch aufgrund ihrer geographischen Informationen auszuwerten, um Hinweise über Standorte zu geben.

Das System muss dabei durch das hohe Datenaufkommen den gestiegenen Anforderungen an die Datenhandhabung und an die Notwendigkeit des schnellen Zugriffs gerecht werden. Durch den Einsatz von Big Data-Technologien und Data Science-Verfahren werden somit Anfragen bezüglich Last- und Vermarktungsprognosen auf hochaufgelösten und umfangreichen Langzeitdaten gewährleistet, die beteiligten Akteuren zukünftig Mehrwerte in den Prozessen ihrer Leistungserstellung ermöglichen können.

Drüber hinaus muss es möglich sein, eine offene und standardisierte Schnittstellenarchitektur zu schaffen, über die ein Datenaustausch zur Archivierung und zur Abfrage von Vorhersagemodellen realisiert wird. Somit ist es Dritten möglich, Daten in das System einzuspeisen und Anfragen zur Auswertung dieser Daten zu stellen. Nach der Auswertung der Daten durch Analysewerkzeuge müssen die erstellten Modelle wieder über die Schnittstelle an jene Anfrager zur Verfügung gestellt werden.

1.4 Definition der Anwendungsfälle

Zur Umsetzung der genannten Ziele ist die Definition von Anwendungsfällen (Use Cases) notwendig, um genau zu definieren welche Funktionen das zu erstellende System erfüllen muss. Dabei wurde sich auf vier Kernfunktionalitäten festgelegt, die im Folgenden erläutert werden.

1.4.1 Use Case: Daten archivieren

Das erste Use Case, welches in Abbildung 1.2 dargestellt wird, behandelt die Datenarchivierung des DAvE-Systems. Für dieses Use Case werden die drei Akteure Wetterdienst, Stromerzeuger und Stromanbieter benötigt. Zu Beginn werden unterschiedliche Datenquellen an das System angebunden. Solche Daten wären zum Beispiel Klimadaten vom Deutschen Wetterdienst. Die angekommenen Datenströme werden entgegengenommen und persistent sowie unverändert archiviert. Weiterhin werden die Daten auf Fehler überprüft und sollten Fehler vorhanden sein, sendet das DAvE-System eine Fehlermeldung.
Es kann bei diesem Use Case auch zu Ausnahmenfällen kommen. Solche Ausnahmefälle wären beispielsweise Datendefekte, Kapazitätsauslastungen bei der Verarbeitung oder Speicherauslastungen.

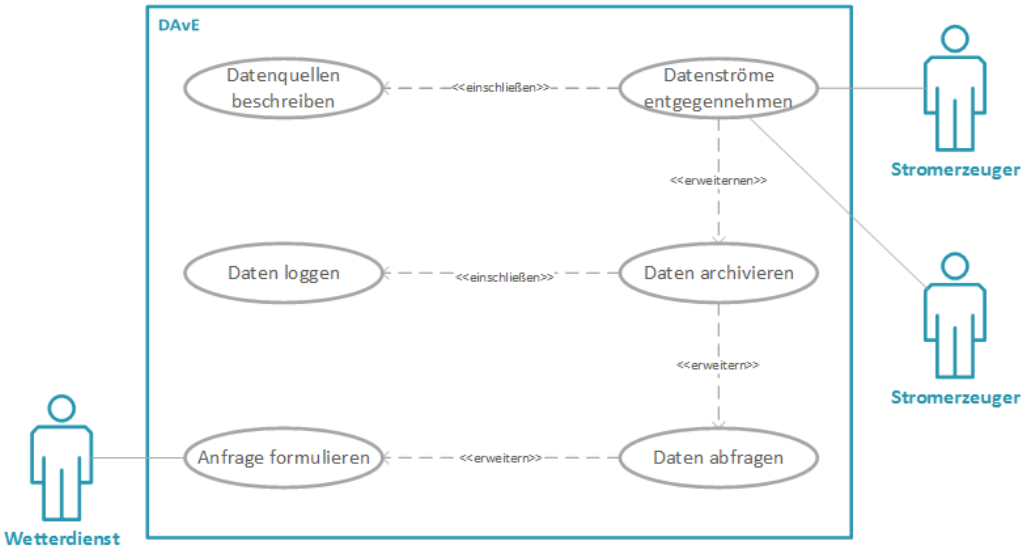

Abb. 1.2: Use Case: Daten archivieren

1.4.2 Use Case: Anfragebearbeitung von Daten

In dem zweiten Use Case, welches in Abbildung 1.3 zu sehen ist, wird die Anfragebearbeitung von Daten verarbeitet. Zuerst stellt der Anfrager eine Anfrage, welche verarbeitet und anschließend geprüft wird. Danach werden gleichzeitig die Rohdaten und Funktionen für die Anfrage vorbereitet. Der nächste Schritt besteht darin, dass die Anfrage ausgeführt wird, sodass im Anschluss daran das Versenden der Daten erfolgen kann.

Zusätzlich zu der allgemeinen Anfrage der Datensätze können auch spezifische Anfragen über diese Datensätze getätigt werden. Ausnahmefälle bei diesem Use Case wären, dass Anfragen nicht valide oder Ergebnismengen zu groß sind, sodass das System dem Nutzer eine Rückmeldung geben muss.

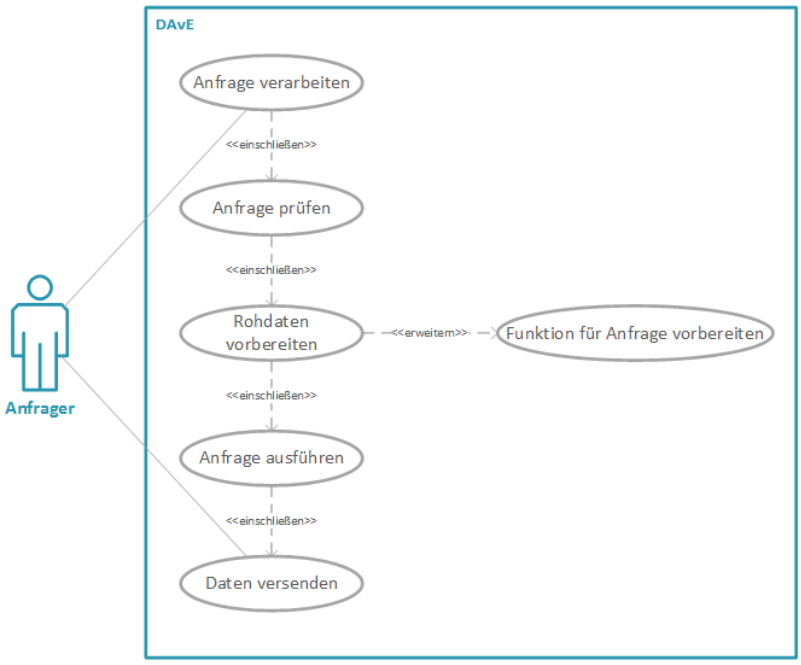

Abb. 1.3: Use Case: Anfragebearbeitung von Daten

1.4.3 Use Case: Modellerzeugung für das Power-Forecasting

Abbildung 1.4 zeigt das dritte Use Case, welches die Modellerzeugung für das Power Forecasting für Windkraft und Photovoltaikanlagen behandelt. Es werden für dieses Use Case die Akteure Stromerzeuger, Stromanbieter sowie Netzbetreiber benötigt. Im ersten Schritt erfolgt die Anfrage der Daten, worauf Daten für das Datenset abgerufen werden. Anschließend wird das Datenset erstellt und eine Energiequelle ausgewählt. Daraufhin findet die Vorverarbeitung des Datensets statt. Auf dieses Datenset erfolgt die Anwendung der Mining-Methode und das daraus entstehende Datenmodell wird evaluiert. Zum Schluss besteht die Möglichkeit das Datenmodell einzusetzen.

Dieses Use Case hat im Vergleich zu den anderen Use Cases auch Ausnahmefälle, denn hierbei kann ein Under- oder Overfitting zu einem Ausfall führen.

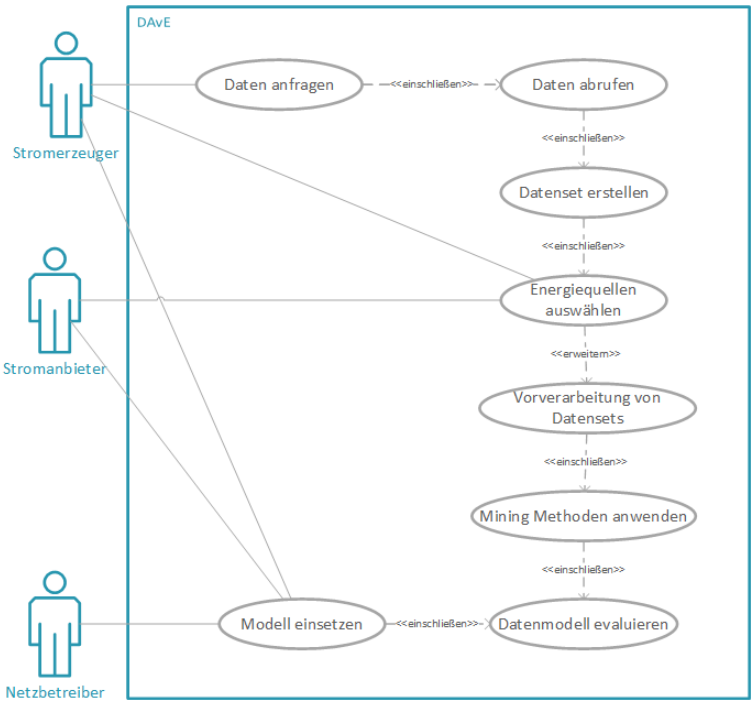

Abb. 1.4: Use Case: Modellerzeugung für das Power-Forecasting

1.4.4 Use Case: Modellerzeugung für das Load-Forecasting

Die Modellerzeugung für das Load Forecasting wird in Abbildung 1.5 dargestellt. Im Vergleich zum vorherigen Use Case werden keine Vorhersagemodelle für die Erzeugung von Windkraft oder Photovoltaikanlagen erstellt, sondern hier liegt das Hauptaugenmerk auf der Prognose der Stromverbrauchsdaten. Die Schrittfolge und Ausnahmefälle sind identisch mit dem Use Case für die Modellerzeugung für das Power-Forecasting.

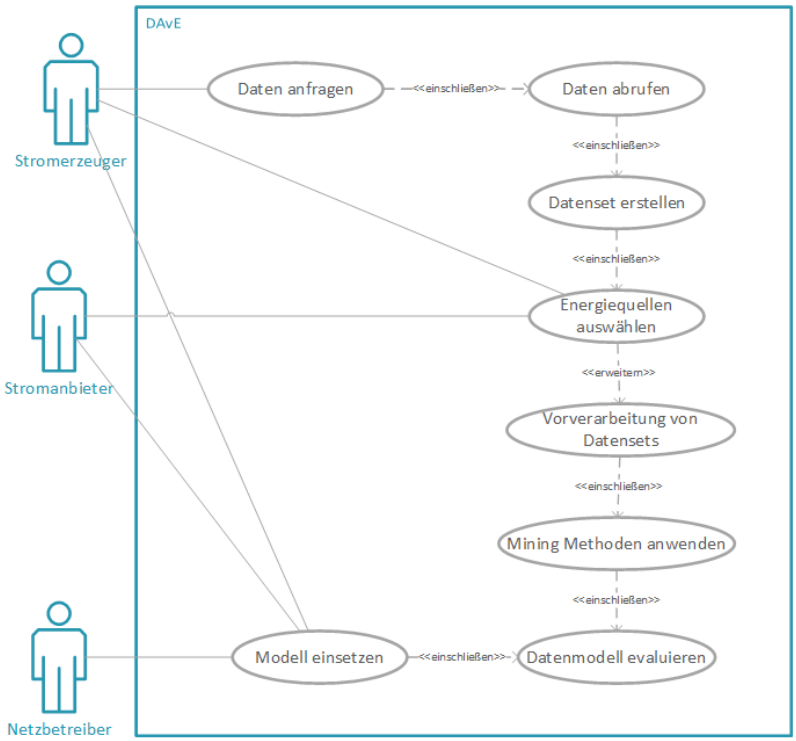

Abb. 1.5: Use Case: Modellerzeugung für das Load-Forecasting

1.5 Konzept zur Umsetzung

Das zu erstellende Konzept bildet den in Abbildung 1.6 dargestellten Workflow ab. Dabei wird der Ablauf von der Datenquelle bis zum Datenexport bzw. zur Modellbereitstellung dargestellt.

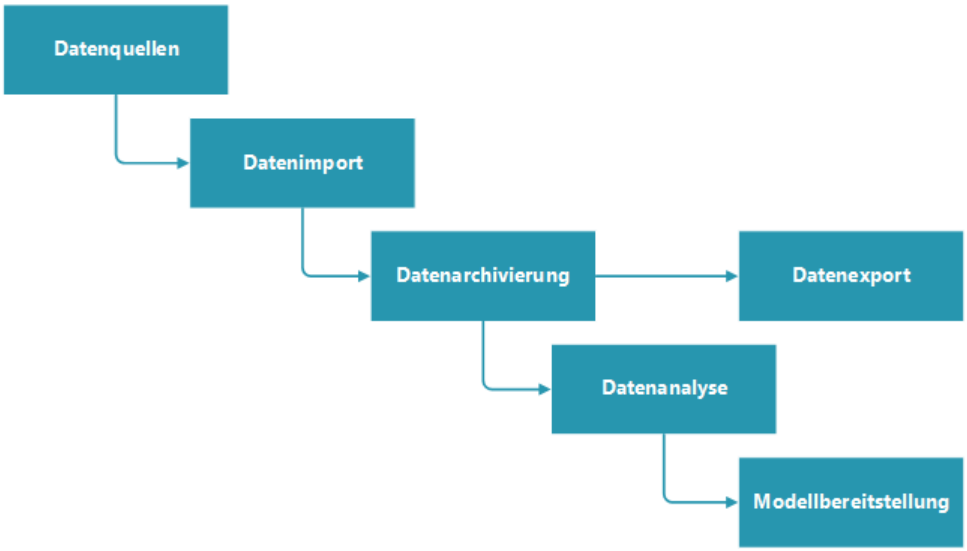

Abb. 1.6: Workflowdarstellung

- **Datenquellen**
  Das System muss bestehende Datenquellen verarbeiten können. Als Datenquellen dienen die Wetterdaten des Deutschen Wetterdienstes und die Verbrauchsdaten von Haushalten aus der Niederspannungssimulation. Die Daten dienen im weiteren Verlauf als Grundlage für die Modellerzeugung.

- **Datenimport**
  Die Daten werden über einen automatisierten Prozess aus den bestehenden Datenquellen importiert. Dazu muss zu einem definierten Zeitpunkt der Import von sich täglich erneuernden Daten in das Archiv durchgeführt werden.

- **Datenarchivierung**
  Die importierten Daten aus den jeweiligen Datenquellen werden im Archiv abgelegt. Dazu wird in Tabellen ein Datensatz erzeugt, der mit neuen Daten aktualisiert wird, ohne die bereits bestehenden Daten zu überschreiben. Die Daten werden dabei nach dem jeweiligen Datum sortiert.

- **Datenexport**
  Nach der Datenarchivierung besteht die Möglichkeit, die sortierten Daten als solche wieder ohne Bearbeitung für einen Datenexport bereitzustellen. Dies gewährleistet für Dritte, Daten im Archiv abzulegen und anschließend wieder auslesen zu können.

- **Datenanalyse**
  Die in das Archiv importierten Daten können im Anschluss für eine Datenanalyse genutzt werden. Dazu lassen sich Analysealgorithmen auf die Datensätze anwenden, die Vorhersagemodelle erzeugen. Diese Modelle können anschließend zu Prognosezwecken verwendet werden.

- **Modellbereitstellung**
  Die erzeugten Modelle können anschließend für Anfragesteller bereitgestellt werden. Dazu ist es möglich, auf die zur Archivierung gesendeten Daten eine Anfrage zur Modellerstellung zu stellen. Diese Anfrage wird im Anschluss bearbeitet und das Modelle wird dem Benutzer zur Verfügung gestellt.

1.5.1 Architektur zur Technologieauswahl

In Abbildung 1.7 wird die DAvE-Architektur dargestellt, welche aus zwei Ebenen besteht. Die unterste Ebene ist die Archivierungsebene, welche die Datenbank enthält. Als Datenbank wird Apache Cassandra verwendet. Auf der obersten Ebene befindet sich die Verarbeitungsebene, welche den Message Broker, das Datenflow Management und das maschinelle Lernen umfasst. Apache Kafka stellt in der DAvE-Architektur den Message Broker dar. Für das Datenflow- Management wird Apache NiFi eingesetzt und für das maschinelle Lernen Scikit-Learn wird Scikit-Learn verwendet. Außerhalb der DAvE-Architektur befinden sich die Datenquellen. Es werden Klimadaten vom Deutschen Wetterdienst sowie die Daten vom Low-Voltage Simulator (LVS) genutzt. Mit dem LVS werden Smart Meter-Daten generiert, welche für die DAvE-Architektur verwendet werden.

Der automatisierte Datenimport des DAvE-Systems wird von NiFi gesteuert. Des Weiteren lädt NiFi die temporären Klimadaten vom Deutschen Wetterdienst über deren FTP-Server herunter und archiviert täglich die aktuellsten Werte in Cassandra. Die Daten vom LVS werden vor der Archivierung in einem eigenen Kafka Broker geschrieben. Dieser wird dann von der NiFi Komponente angenommen, sodass die Daten bzw. Messages, welche sich im Kafka Broker befinden, in Cassandra archiviert werden. Weiterhin steuert NiFi die Datenübermittlung zwischen Cassandra und Kafka.

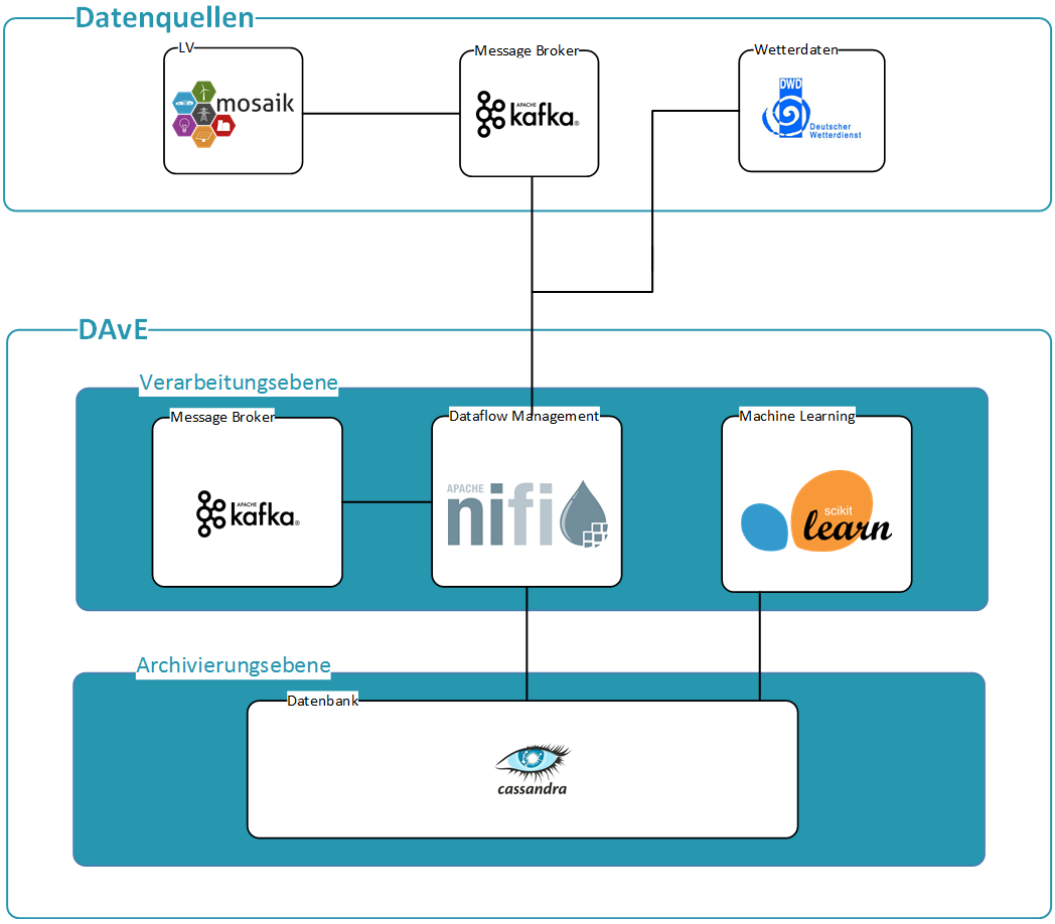

Abb. 1.7: Architektur der Systemkomponenten

Bei einer Datenanfrage wird der NiFi-Prozess gestartet, indem die bestimmten Daten, welche angefragt wurden, abgerufen und verarbeitet werden. Danach werden diese in einem Kafka Topic abgelegt. Der Datenanfrager kann somit auf das Topic zugreifen, um an die Daten zu gelangen.

Das maschinelle Lernen wird eingesetzt, um Modelle zu erzeugen. Dabei werden die archivierten Daten aus Cassandra für die Verwendung der Modelle genutzt. Die erstellen Modelle werden anschließend in ein Predicitive Model Markup Language (PMML)-Modell umgewandelt. Diese werden danach in Cassandra archiviert und können abgerufen werden.

1.5.2 Systemkomponenten

- **Mosaik** ist ein Open Source Framework für die automatische Erstellung von großflächigen Smart Grid Simulationen für unterschiedliche Szenarien. Dabei werden Verbrauchsdaten in einem Netz aus Smart Metern und Smart Meter Gateways generiert. Die Daten werden als COSEM-formatierte XML-Dateien empfangen und archiviert.[1]

- **Apache NiFi** ist ein Framework, mit dem komplexe Datenworkflows umgesetzt werden können. Dazu werden Importmechanismen von Daten aus unterschiedlichen Datenquellen zur Verfügung gestellt und darüber hinaus Transformationsmöglichkeiten bereitgestellt. Über ein User-Interface können die Dataflows durch eine Verknüpfung von Prozessoren konfiguriert, gestartet und überwacht werden.[2]

- **Apache Kafka** ist ein Open Source Projekt für die Verarbeitung von Datenströmen. Dazu wurden Mechanismen zu Speicherung und Verarbeitung von Datenströmen entwickelt und es wird eine Schnittstelle zum Laden und Exportieren von Datenströmen zu Drittsystemen bereitgestellt. Kafka ist ein verteiltes System, welches durch Skalierbarkeit und Fehlertoleranz ausgezeichnet wird und sich deshalb überwiegend für Big-Data-Anwendungen eignet.[3]

- **Scikit Learn** ist eine freie Bibliothek für maschinelles Lernen, die Algorithmen für Klassifizierungen, Regressionen und Clustering bereitgestellt. Es stellt einfache und effiziente Tools für Data-Mining und Data-Analyse zur Verfügung, sodass verschiedene Vorhersagemodelle aus Support-Vector-Machines, Random Forest, Neuronalen Netzen oder der Linearen Regression erstellt werden können.[4]

- **Apache Cassandra** ist ein einfaches, verteiltes Datenbankverwaltungssystem für große und strukturierte Datenbanken. Es zeichnet sich durch eine hohe Skalierbarkeit und Ausfallsicherheit bei verteilten Systemen aus. Cassandra ist darüber hinaus die am meisten verbreitete spaltenorientierte NoSQL-Datenbank. In einer Cassandra Datenbank werden die einzelnen Datensätze in multidimensionalen Hashtables abgelegt, sodass jede Zeile eines Hashtables beliebig viele Spalten haben kann.[5]

1.6 Mehrwerte durch den Einsatz von DAvE

Es wurden schon frühzeitig die Potentiale von Datenanalysen für Unternehmen erkannt. So können vor allem die Prozesse zur Leistungserstellung optimiert werden und somit langfristig kosteneffizienter gestaltet werden. Dadurch werden einerseits Gewinne maximiert und andererseits die Wettbewerbsfähigkeit und der Unternehmenserfolg zunehmend beeinflusst.

Darüber hinaus können Unternehmen durch prognostiziertes Verhaltensweisen bessere Entscheidungen treffen. Vor allem für den Energiesektor können spezifische Mehrwerte generiert werden, von denen Energieerzeuger und -anbieter profitieren können:

- **Bedarfsplanung der Ressourcen**
  Durch die Analyse der Verbraucher und deren Verhaltensweisen kann nahezu genau prognostiziert werden, welcher Haushalt zu einer bestimmten Tageszeit wieviel Energie benötigt. Die historischen Daten lassen sich angereichert mit Erfahrungswerten soweit auswerten, dass ein Verbrauchsmuster erkennbar ist und somit den Bedarf für jede Netzregion spezifisch vorhergesagt werden kann.

- **Effektivitätssteigerung**
  Anhand der analysierten Wetterdaten ist es möglich, nahezu genau vorherzusagen, wann die Windgeschwindigkeiten und Sonnenstunden für eine Stromerzeugung geeignet sind. Dadurch können Stromerzeuger erkennen, zu welchen Tages- und Jahreszeiten mit einer Einspeisung dieser regenerativen Energieträger zu rechnen ist. Darüber hinaus werden durch die Auswertung der geografischen Lage dieser Datenquellen Hinweise auf mögliche neue Standorte zur Energiegewinnung gegeben.

- **Effizienzsteigerung**
  Energieerzeuger und -anbieter können exakt auslesen, zu welchen Tageszeiten in welchem Netz Energie verbraucht wird. Dadurch wird es ermöglicht, die Energieerzeugung genauer steuern zu können und somit auch nur die Energie einzuspeisen, die laut Prognose von den Haushalten verbraucht wird. Dadurch soll eine Überproduktion von Energie vermieden werden.

- **Erstellung von Prognosen**
  Es werden Prognosen erstellt, die über Energieverbräuche von Haushalten informieren und Verhaltensmuster aufzeigen. Zusätzlich dazu werden Auskünfte über erwartete Wetterverhältnisse gegeben, aus denen Prognosen über mögliche Energieerzeugung erstellt werden können. Darüber hinaus können die Prognosen für Markt- und Verhaltensforschung genutzt werden.

- **Transparenz für beteiligte Akteure**
  Durch die Nutzung der anfallenden Daten ist es möglich, für alle beteiligten Akteure mehr Transparenz zu schaffen. Somit wissen Energieerzeuger und -anbieter nahezu genau über das Konsumentenverhalten Bescheid, können sich abstimmen und somit besser auf Bedarfe und Maxima im Verbrauch reagieren.

- **Schnittstellenoffenheit**
  Das System wurde so konzipiert, dass es möglich ist, externe Daten zu archivieren. Aus den archivierten Daten können anschließend Analysemodelle erstellt werden, die über diese Schnittstelle wieder abrufbar sind. Dies wurde bereits in der Praxis erprobt, sodass externe Datenströme mit Verbrauchsdaten von Haushalten archiviert und ausgewertet wurden.

1.7 Ausblick

Mit dem DAvE-System wird ein Ansatz geschaffen, die gestiegenen Anforderungen an das Datenaufkommen durch intelligente Messsysteme zu bewerkstelligen. Ferner wird durch die Verarbeitung der Messwerte aus der Niederspannungsebene eine Möglichkeit geschaffen, aus den Daten Mehrwerte oder gar datengetriebene Geschäftsmodelle zu entwickeln. Zudem ist es immer wichtiger, die Daten nicht nur einfach zu speichern, sondern auch ein Nutzen daraus ziehen zu können. So besteht durch die Digitalisierung der Stromzähler auch auf Konsumentenseite der Bedarf, über die verbrauchten Energiemengen informiert zu werden. Die Daten können somit aus dem Archiv abgefragt werden und beispielsweise über ein Letztverbraucherportal zur Verfügung gestellt werden.

Primär bieten die Big Data-Technologien und Data Science-Verfahren Lösungen zur Anfrage- und Vermarktungsprognosen oder gar eine Integration von externen Daten in die Netzbetriebsführung. Dies ermöglicht neben möglichen Kosteneinsparungen eine stärkere Kundenindividualisierung bei der Netzbetriebsführung. Die Daten können für das Netzleitstellenpersonal zusätzlich für neue Netzplanungen oder neue Mehrwertdienste für Industriekunden nutzbar gemacht werden. Darüber hinaus können Reaktionszeiten bei gleichzeitiger Steigerung der Häufigkeit und Komplexität von Eingriffen verkürzt werden. Da das DAvE-System derzeit nur mit simulierten Messdaten arbeitet, bleibt es spannend zu sehen, wie es sich in Zukunft eine Änderung im Echtbetrieb auswirkt.

1.8 Literatur

[1] OFFIS. *Modulare Simulation aktiver Komponenten im Smart Grid* [Online]. Available: https://www.offis.de/offis/projekt/mosaik.html

[2] Apache Software Foundation. *Apache NiFi* [Online]. Available: https://nifi.apache.org

[3] L. Berle. (2017, October 17). *Apache Kafka – eine Schlüsselplattform für hochskalierbare Systeme* [Online]. Available: https://www.informatik-aktuell.de/betrieb/verfuegbarkeit/apache-kafka-eine-schluesselplattform-fuer-hochskalierbare-systeme.html

[4] D. Cournapeau. *Scikit-learn – Machine Learning in Python* [Online]. Available: http://scikit-learn.org/stable/index.html#

[5] M.Jensing, P. Ghadir. (2015, June 17). *Apache Cassandra – Für Daten ohne Grenzen* [Online]. Available: https://www.innoq.com/de/articles/2015/06/apache-cassandra/