CARL VON OSSIETZKY universität OLDENBURG

Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Projektgruppe

E-Stream

Big Data Analytics im Smart Grid

vorgelegt von

**Friederike Bruns, Florian Decker, Julian Droste-Rehling,
Eugen Frikel, Marcel Hamacher, Edward Ostertag, Christian Peters,
Jens Plümer, Tim Silhan, Viktor Spadi & Timo Wolters**

Gutachter:

M. Sc. Michael Brand
Dr. Marco Grawunder
Prof. Dr. Sebastian Lehnhoff

Oldenburg, 8. April 2018

1 Zusammenfassung

Durch die Energiewende und der dadurch steigenden Integration von Erzeugern aus erneuerbaren Energien, ist es notwendig, dass Verteilnetzbetreiber intelligent das Stromnetz verwalten können. In diesen sogenannten Smart Grids steigt die Größe der zu verarbeitenden Informationsmenge. Um aus dieser Informationsmenge Einsichten gewinnen zu können und so die Verteilnetzbetreiber bei ihren Entscheidungen unterstützen zu können, ist ein Toolkit notwendig, dass die Datenverabeitung in Smart Grids übernimmt und gewonnene Einsichten sinnvoll visualisiert.

Dieses Dokument stellt E-Stream vor, das eine Lösung darstellt, um Verteilnetzbetreiber darin zu unterstützen, Informationen über Netzzustände zu bekommen. Innerhalb des Systems werden verschiedene Technologien aus laufenden Forschungsprojekten benutzt. Neben der Kooperation mit NetzDatenStrom und ihrer in Entwicklung befindlichen Niederspannungsnetzsimulation auf Grundlage von *mosaik* wird ebenfalls das Datenstrommanagementsystem (DSMS) *Odysseus* benutzt. Das gesamte System besteht somit aus folgenden Teilkomponenten: Server inklusive relationaler Datenbank, Web-Frontend, DSMS, Publish-Subscribe-System für die Datenintegration zwischen internen Komponenten und externen Systemen, sowie eine Anbindung und Simulation von Niederspannungsnetzdaten. Außerdem wurde in Kooperation mit der Projektgruppe DAvE eine gemeinsame Schnittstellendefinition entwickelt, auf Grundlage welcher eine Schnittstelle zur Kommunikation mit einem Langzeitdatenarchiv generiert wird.

Der Server des Systems dient zur Orchestrierung und Verwaltung aller Teilkomponenten. Er ist dafür zuständig, dass die Kommunikation der Teilkomponenten über das Publish-Subscribe-System abläuft, indem das Erstellen, Löschen und Synchronisieren von Publish-Subscribe-Topics verwaltet wird. Weiterhin werden Anfragen in dem DSMS installiert, ausgeführt und für die spätere Wiederverwendung persistiert. Ebenfalls ist der Server dafür zuständig, dass vom Frontend angefragte Daten nach der Verarbeitung durch das DSMS über passende WebSocket-Topics in Echtzeit bereit gestellt werden. Alle Funktionalitäten werden durch ein implementiertes Nutzer- und Rollenmanagement im Server gesichert. Für die Anzeige der Datenverarbeitung wurde ein modernes Web-Frontend entwickelt, das dem Benutzer die Möglichkeit bietet, sich eigenständig frei konfigurierbare Dashboards zu erstellen, um sich die gewünschten Ergebnisse der Datenanalysen mit diversen Visualisierungskomponenten anzeigen zu lassen. Neben der reinen Ergebnisvisualisierung bietet E-Stream ebenfalls eine abstrakte topologische Karte, zur Darstellung von Störungen und weiteren Informationen in einem Niederspannungsnetz.

Dadurch dass Smart Meter Gateway kontinuierlich Daten ihrer dazugehörigen Smart Meter senden, ist es notwendig diesen Datenstrom kontinuierlich zu verarbeiten. Diese Datenstromverarbeitung geschieht in Odysseus. Das Management des DSMS wurde um ein Template-System für eine dynamische Anfragesteuerung erweitert. Resultierend daraus ist es einem autorisierten Verteilnetzbetreiber möglich auf Basis von Smart Meter Gateway Daten dynamisch Analysen zu erstellen und auszuführen und sich die dabei entstehenden Ergebnisse in Form von Charts innerhalb mehrerer Dashboards anzeigen zu lassen, um ihnen einen Mehrwert bieten zu können. Dieser liegt darin, dass durch die Visualisierungen schneller Fehler erkannt und Prognosen erstellt werden können, damit Entscheidungen schneller getroffen werden, um unter anderem die Netzstabilität besser zu gewährleisten.

Inhalt

2 Einleitung

Smart Grids sind eine neue Entwicklung in der Energiewirtschaft, durch welche die Erzeugung, Speicherung und der Verbrauch von Strom intelligent verwaltet und überwacht werden kann. Mit Hilfe dieses intelligenten Stromnetzwerkes können Leistungsschwankungen ausgeglichen werden, die insbesondere durch die Nutzung von regenerativen Energien, die im Zuge der Energiewende immer weiter verbreitetet sind, auftreten. Die Projektgruppe E-Stream hat für die effizientere Nutzung der regenerativen Energien das Ziel, ein Toolkit für die Unterstützung der Netzüberwachung und -steuerung zu erstellen, mit welchem der Netzstatus bestimmt, prognostiziert und visualisiert wird. Hierfür können zum Beispiel auf Basis von aktuellen Echtzeitdaten, in Kombination mit historischen oder netzfremden Daten, Prognosen der Photovoltaik-Leistung getroffen werden.

2.1 Motivation

Im Jahr 1996 wurde von der Europäischen Union die stufenweise Liberalisierung des Elektrizitätssektors beschlossen, welche dann ab 1999 sukzessiv umgesetzt wurde. Durch Öffnung dieses Marktes wurde erstmals Wettbewerb möglich [134]. Um am Markt existieren zu können, ist es wichtig, den Wünschen der Kunden sowie den politischen Forderungen nach sozialer und ökologischer Nachhaltigkeit entgegenzukommen. Beispielsweise beinhaltet die EU-Richtlinie (2009/72 EG), dass ab 2020 80% aller Haushalte mit Smart Metern ausgestattet sein sollen.
Das bedeutet, dass auch in Zukunft mit einem weiteren Ausbau der regenerativen Energien zu rechnen ist.
Hierbei ergibt sich eine Problematik, da die regenerative Energieerzeugung zum Beispiel auf Grund der sich verändernden Wetterlage nicht konstant und deswegen schwieriger steuerbar ist, als die Energieerzeugung durch konventionelle Kraftwerke [175]. Hinzu kommt ein unregelmäßiger Lastverlauf durch die Vielzahl an Konsumenten und der steigenden Anzahl an Erzeugern, sodass die reibungslose und permanente Bereitstellung der erzeugten Energien zu einer immer komplexeren Herausforderung wird. Um Netzzusammenbrüchen vorzubeugen muss mit Hilfe eines intelligenten Stromnetzes, dem Smart Grid, eine Verknüpfung der Stromproduzenten und -konsumenten stattfinden [203, 246]. Daher investieren bereits viele Regierungen weltweilt große Beträge in den Ausbau von Smart Grids [21].
Zur Erfassung der entstehenden Daten werden Smart Meter eingesetzt, die automatisch im Minutenintervall den Verbrauch an ein System senden sollen. Eine effiziente Verteilung kann nur erreicht werden, wenn detailliert und zeitnah bekannt ist, wie viel Energie an welcher Stelle im Netz zu welchem Zeitpunkt erzeugt und verbraucht wird.
Je weiter das Netz also zu Smart Grids ausgebaut wird, desto mehr Daten müssen verwaltet werden. In Los Angeles wird beispielsweise eine Verarbeitungsgeschwindigkeit von etwa 200 Mbps benötigt, wenn jeder Verbraucher etwa 1 KB Daten pro Minute erzeugt [243].

Hierbei besteht das Problem, dass keine Aussage darüber getroffen werden kann, wie viele Daten verarbeitet werden müssen, was die Entwicklung eines geeigneten Softwaresystems erschwert. Um mit einer solch großen Menge an Daten umzugehen, müssen diese aufbereitet und analysiert werden.

Auf Basis der aktuellen Echtzeitdaten, in Kombination mit historischen oder netzfremden Daten können beispielsweise Prognosen getroffen werden. Um diese Daten zugänglich zu machen, ist es

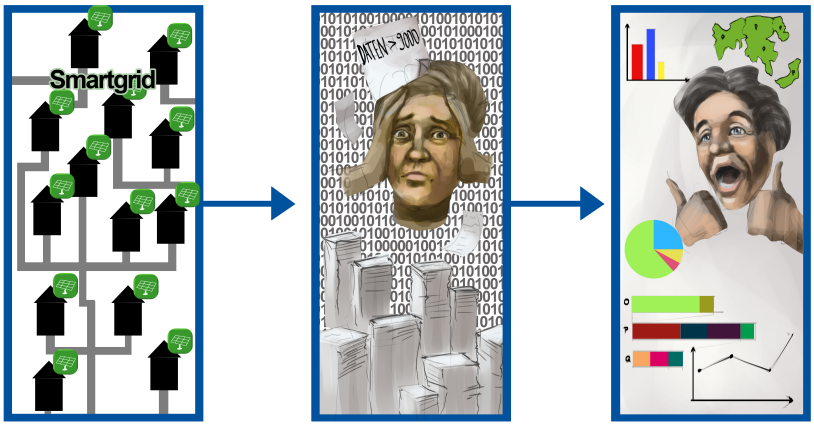


Abbildung 2.1: *Motivation für PG E-Stream*

nötig, eine geeignete Benutzeroberfläche mit aussagekräftigen Diagrammen und Analysemöglichkeiten zu bieten, welche außerdem auch für die Echtzeitdaten genutzt werden können.

Die Entwicklung eines Softwaresystems für die Verarbeitung und Bereitstellung dieser Informationen ist deshalb ein wichtiger Beitrag für die Energiewende. Die Projektgruppe *E-Stream - Big Data Analytics im Smart Grid* bearbeitet diese aktuelle Forschungsfrage und entwickelt ein solches System für die Verbrauchsmessungen von Haushalten.

2.2 Ziele

Grundlegendes Ziel der Projektgruppe E-Stream ist es, die effizientere Nutzung von regenerativen Energien durch die Erstellung eines Toolkits für die Unterstützung der Netzüberwachung und -steuerung voranzutreiben. Mit diesem Toolkit soll der Netzzustand bestimmt, prognostiziert und visualisiert werden können. Auf Basis von aktuellen Echtzeitdaten sollen in Kombination mit historischen oder netzfremden Daten Prognosen getroffen werden können. Hierfür können außerdem bereits bestehende Algorithmen verwendet werden.

Kunden soll es so ermöglicht werden, den eigenen Verbrauch und gegebenenfalls die aktuelle Einspeisung des selbsterzeugten Stroms zu bestimmen. Auf der anderen Seite wird den Operatoren der Energieerzeuger die Möglichkeit geboten, den Gesamtverbrauch und die momentane Energieproduktion zu erfassen. Sollten einzelne Energieerzeuger unerwartet keinen Strom einspeisen, wird dies über die Oberfläche dem Operator mitgeteilt.

Im Detail soll es beispielsweise möglich sein, Smart Meter Daten zu erhalten, auf Plausibilität zu prüfen und topologisch beziehungsweise geografisch zu filtern. Es soll eine Topologie bereitgestellt und visualisiert sowie der prognostizierte Verbrauch und die Erzeugung angezeigt werden können. Rohdaten sollen außerdem tabellarisch dargestellt werden können. So soll ein Nutzer, zum Beispiel ein Verantwortlicher für die Netzüberwachung eines Energienetzes, Dashboards konfigurieren und Anfragen für die Datenverarbeitung starten können, um sich die Daten daraufhin auf unterschiedliche Weise darstellen zu lassen. Als Administrator soll man die Möglichkeit haben, Nutzer und Anfragen für die Datenverarbeitung zu verwalten.

Darüber hinaus soll in Kooperation mit der Projektgruppe DAvE die Verwendung eines Langzeitdatenarchivs umgesetzt werden. So sollen zum Beispiel historische Smart Meter Daten und Wetterdaten angefragt werden können. Außerdem sollen Data Mining Modelle erstellt und abgefragt werden können. Die erhaltenen Rohdaten sollen zudem zur Speicherung an das Langzeitdatenarchiv gesendet werden können.

2.3 Aufbau der Arbeit

Das vorliegende Dokument beinhaltet die Ergebnisse der Projektgruppe E-Stream. So wird in Kapitel 3 eine Übersicht über das Projekt gegeben und auf die grundlegenden Komponenten des Systems, wie beispielsweise Anwendungen (Mosaik, Odysseus und Webvisualisierung), Schnittstellenprotokolle, Datenströme und Vorhersagen eingegangen.

In dem folgenden Kapitel 4 wird das Projektmanagement abgehandelt, wobei nicht nur über das Teammanagement, sondern auch über die Rollenverteilung, die Teamsitzungen, den Zeitplan und die Kooperation mit der Projektgruppe DAvE berichtet wird.
Anschließend wird in Kapitel 5 die Anforderungsanalyse mit einer allgemeinen Beschreibung, den spezifischen Anforderungen und den Schnittstellenanforderungen dargelegt.
Kapitel 6 erörtert zunächst eine Evaluation der möglichen zu verwendenden Tools und geht dann auf die Continuous Integration Umgebung, sowie verschiedene Toolbeschreibungen ein.
Danach wird in Kapitel 7 der Projektfortschritt durch Beschreibung des Projektverlaufes verdeutlicht.
Im Anschluss wird in Kapitel 8 das Produkt beschrieben, welches beispielsweise die Architektur und die Beschreibung der verschiedenen entwickelten Komponenten beinhaltet.
Daraufhin werden in Kapitel 9 die verwendeten Testmethoden und deren Durchführung geschildert.
Zum Abschluss wird in Kapitel 11 ein Ausblick für zukünftige Möglichkeiten in der Weiterarbeit an dem Projekt E-Stream geboten und daraufhin in Kapitel 12 das Fazit der Projektgruppe gezogen.

3 Übersicht des Projektes

Dieses Kapitel gibt eine Übersicht über die verschiedenen Projektbestandteile, welche zur besseren Einordnung in der Abbildung 3.1 dargestellt werden und darauf folgend genauer erläutert werden.

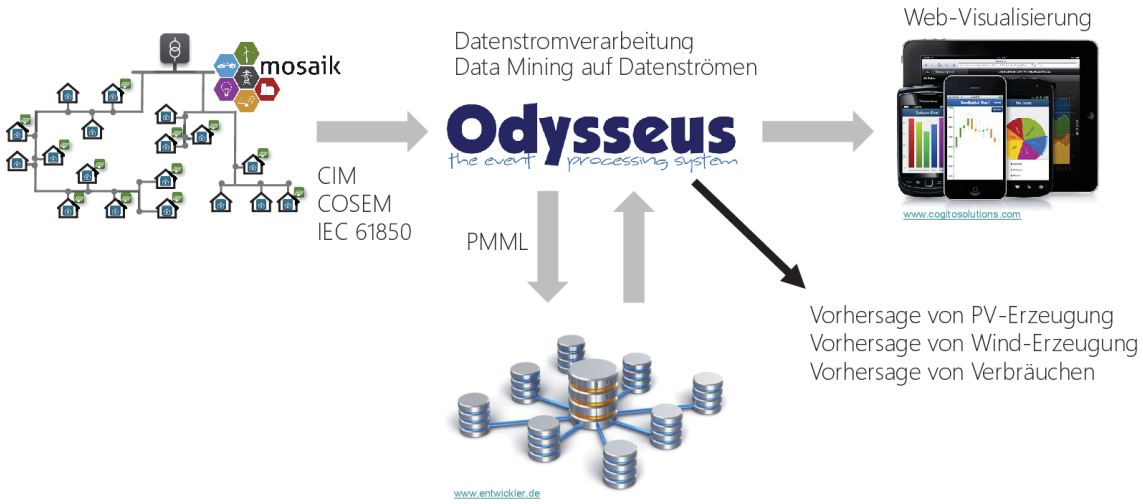


Abbildung 3.1: *Übersicht der Projektbestandteile*

Zunächst werden verschiedene Anwendungen des Projektes betrachtet. Bestandteil davon ist das Framework mosaik für die Smart Grid Simulation und Erstellung von Smart Grid Daten. Eine weitere Anwendung, die in diesem Projekt benutzt und angepasst wird, ist das Datenstrommanagementsystem Odysseus. Diese verarbeiteten Daten werden dann mit Hilfe eines Tools zur Web-Visualisierung dargestellt.

Für die Umsetzung dieses Systems werden neben den Anwendungen auch die verschiedenen Schnittstellenprotokolle vorausgesetzt. Eine benötigte Spezifikation ist das DLMS/COSEM Datenmodell für die Smart Meter. Über diese Schnittstelle werden die benötigten Informationen der Smart Meter der Haushalte an das in diesem Projekt entwickelte System übermittelt. Als weitere Schnittstelle wird PMML als einheitliches Austauschformat verwendet, sodass die Kompatibilität zwischen verschiedenen Programmen maximiert wird. CIM wird benutzt, um ein technologieunabhängiges Modell zu gestalten, sodass Dienste die gleichen Objekte sofort bearbeiten können. Bezüglich Datenströmen wird zum einen auf das Data Mining auf Datenströmen eingegangen und zum anderen auf die Datenstromverarbeitung. Anschließend werden einige Vorhersagemöglichkeiten dargestellt, wobei speziell die Vorhersage von Photovoltaik- und Windenergieerzeugung sowie die Vorhersage von Energieverbräuchen von Haushalten behandelt wird.

3.1 Anwendungen

Zunächst werden alle wichtigen Anwendungen, die in diesem Projekt verwendet werden, genauer beschrieben, denn sie bilden von Anfang an die relevanten Kernstücke dieses Projektes. Denn zu erst werden durch mosaik Smart Grid Daten simuliert und im weiteren werden diese Daten mit Odysseus verarbeitet und zum Schluss werden alle diese unterschiedlichen Daten visualisiert.

3.1.1 mosaik

Dank der Energiewende ist die Stromerzeugung immer öfter dezentral und fluktuierend. Dies muss intelligent gesteuert werden. Um so ein Smart Grid zu optimieren, müssen neue Verfahren entwickelt werden. Auch muss überprüft werden, ob bestimmte Smart Grid Szenarien durchführbar sind. Da diese Überprüfung in der Realität viel zu aufwändig und unter Umständen gefährlich wäre, muss ein anderes Vorgehen verwendet werden.

Beispielsweise können virtuelle Simulationen durchgeführt werden bevor die Szenarien in der Realität umgesetzt werden. Um solche Simulationen durchzuführen wurde das Framework mosaik entwickelt. In dieser Arbeit wird das mosaik-Framework genauer beschrieben.

3.1.1.1 Grundlagen

In diesem Kapitel wird beschrieben was unter einem Smart Grid und einer Simulation verstanden wird. Anschließend wird kurz das mosaik-Framework beschrieben.

Smart Grid

Ein Smart Grid verwendet Informations- und Kommunikationstechnologien, um Strom von zentralen Erzeugungsstationen und verteilten Energieressourcen an Verbraucher zu übermitteln und diese Prozesse mit intelligenten Methoden zu steuern [236].

Simulation

Simulationen können durchgeführt werden, um das Verhalten von komplexen Systemen zu analysieren. Dabei wird versucht durch Ausprobieren und Experimentieren neue Erkenntnisse zu gewinnen. Simulationen haben den Vorteil, dass mit ihnen lange Zeiträume in sehr kurzer Zeit analysiert werden können. Auch werden die Kosten und das Risiko reduziert. Allerdings müssen dafür teilweise auch aufwändige Simulationsmodelle erstellt werden [240].

Co-Simulation

Bei einer Co-Simulation [25] werden verschiedene Werkzeuge verwendet, um eine Simulation auszuführen. Jedes Werkzeug löst dabei selbstständig einen Teil des gesamten Problems. Die einzelnen Werkzeuge werden aneinander gekoppelt und tauschen Daten bzw. Zwischenergebnisse aus.

mosaik

mosaik ist ein am OFFIS[1] entwickeltes Framework, welches zum Ausführen von Simulationen verwendet werden kann. Der Fokus liegt dabei auf Smart Grid Simulationen, aber es kann auch für andere Zwecke eingesetzt werden. Alle Simulatoren werden dabei unabhängig voneinander ausgeführt. Die Aufgabe von mosaik ist das Synchronisieren dieser Simulatoren [259]. Dafür müssen die Simulatoren eine von mosaik bereitgestellte API verwenden.

3.1.1.2 Die Hauptkomponenten von mosaik

mosaik besteht aus vier Hauptkomponenten [258]. Diese werden in diesem Kapitel genauer beschrieben.

Die mosaik Sim API

In der mosaik Sim API ist das Kommunikationsprotokoll zwischen den Simulatoren und mosaik definiert. Dafür stellt mosaik eine Low-Level und eine High-Level API zur Verfügung [263]. Siehe Abbildung 3.2 für einen Vergleich dieser beiden APIs. Bei der Low-Level API wird mit Hilfe von JSON-Nachrichten kommuniziert. Diese Nachrichten werden über Netzwerksockets versendet.

Die High-Level API ist nur für einige Programmiersprachen verfügbar. Diese kapselt die Netzwerkabhängigen Details ab, wodurch der Nutzer sich nicht selbst um die Netzwerkkommunikation kümmern muss. Es reicht nur einige wenige Methoden zu implementieren. Die High-Level API ist allerdings vorerst nur für die Programmiersprachen Python und Java verfügbar. C# und Matlab wird bald verfügbar sein [262].

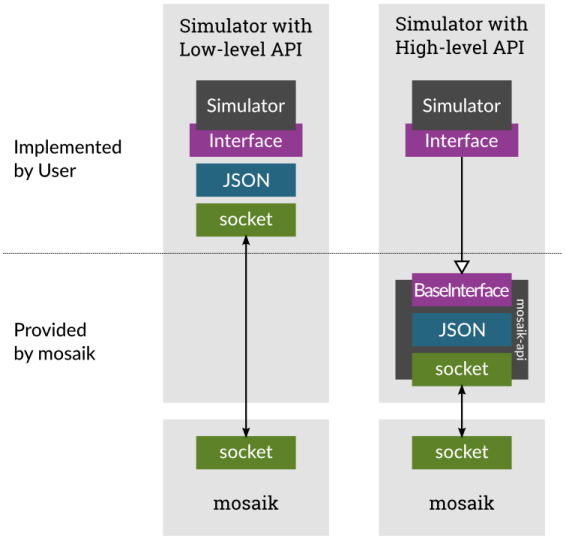


Abbildung 3.2: *Die mosaik Sim API [263]*

[1] https://www.offis.de/

Die API Hier wird nur die High-Level API beschrieben. Die Low-Level API ist ähnlich, allerdings findet dort die Kommunikation mithilfe von JSON statt.

Es folgt eine kurze Beschreibung der wichtigsten Methoden, die die Simulatoren implementieren müssen [262].

init(sid, **sim_params) Initialisiert den Simulator. Dabei wird eine ID, die *sid*, und zusätzliche Parameter, die *sim_params*, von mosaik übergeben. Diese Methode muss die Metadaten *meta* zurückgeben.

create(num, model, **model_params) Erstellt eine Anzahl (*num*) von *model* Instanzen mit den übergebenen Parametern *model_params* und gibt eine Liste mit einer Beschreibung der erstellten Modell-Instanzen zurück.

step(time, inputs) Führt den nächsten Simulationsschritt ab dem übergebenen Zeitpunkt *time* aus. Dazu werden die übergebenen Werte in *values* verwendet. Zurückgegeben wird die neue Simulationszeit, also wann diese Methode von mosaik wieder aufgerufen werden soll.

get_data(outputs) Gibt die Daten zu den übergebenen *outputs* zurück.

Zusätzlich gibt es Methoden, mit denen zum Beispiel der Fortschritt der Simulation zurückgegeben wird oder bestimmte bzw. alle Entitäten (Modell-Instanzen) zurückgegeben werden.

Kommunikation mit dem Simulator Die oben genannten Methoden werden von mosaik in bestimmter Reihenfolge aufgerufen [255]. Im Sequenzdiagramm in Abbildung 3.3 wird diese Reihenfolge anschaulich dargestellt.

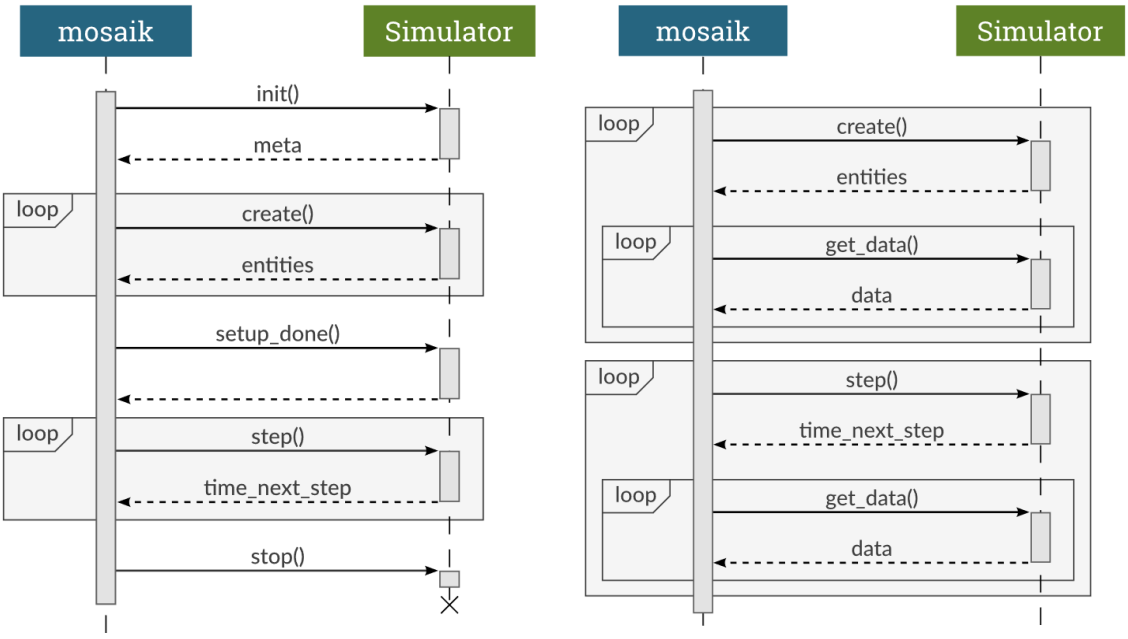
Zuerst wird *init()* aufgerufen, worauf von dem Simulator die Metainformationen zurückgegeben werden. Nach init wird beliebig oft *create* aufgerufen und so beliebig viele Modell-Instanzen erstellt. Wenn alle benötigten Modell-Instanzen erstellt und auch die Beziehungen zwischen diesen definiert wurden, wird *setup_done()* aufgerufen. Dann wird der Simulator gestartet. Es folgen beliebig viele *step()*-Aufrufe, die jeweils die Zeit für den nächsten step()-Aufruf zurückliefern. Wenn die Simulationszeit vorbei ist, wird am Ende noch *stop()* aufgerufen, um den Simulator zu signalisieren, dass dieser beendet werden kann. Diese Reihenfolge wird im Sequenzdiagramm in Abbildung 3.3a dargestellt.

Nach *create()* oder *step()* Aufrufen können auch *get_data()* Aufrufe stattfinden. Dies wird im Sequenzdiagramm in Abbildung 3.3b dargestellt. Zu jeder Zeit können aber auch Methoden wie *get_progress()*, *get_related_entities()*, *get_data()* oder *set_data()* aufgerufen werden.

Die Scenario API

Das Szenario wird in der Programmiersprache Python beschrieben. Das Modellieren bzw. Zusammensetzen des Szenarios besteht aus drei Schritten. Diese sind das Starten des Simulators, das Erstellen der Modelle in diesem Simulator und das Verbinden der Modelle mit den Modellen von anderen Simulatoren [260].

Um ein Szenario zu erstellen, wird die Klasse *mosaik.scenario.World* benutzt. In dieser Klasse sind alle Daten und Zustände, die für das Szenario sowie dessen Simulation benötigt werden. Außerdem befinden sich hier viele Methoden, die für das Ausführen des Simulators und das Definieren der



(a) (b)

Abbildung 3.3: *Die Simulator Aufrufe [255]*

benötigten Datenströme zwischen Simulatoren wichtig sind. Die Simulatoren müssen auch im Code in einer *sim_config* Variable angegeben werden.

Um einen Simulator zu starten, muss *World.start()* aufgerufen und dabei der Name des Simulators übergeben werden. mosaik sucht nach dem passenden Simulator aus der *sim_config* und startet diesen. Diese Methode liefert eine *ModelFactory* zurück. Mithilfe dieser ModelFactory können Modelle erstellt werden. Zusätzlich zum Namen können auch weitere Parameter übergeben werden. Diese werden dann beim *init()*-Aufruf an den Simulator übergeben. Für das Erstellen der Modelle wird die von der *World.start()*-Methode zurückgelieferte *ModelFactory* verwendet. Eine Modell-Instanz ist eine Entität mit einer für diesen Simulator eindeutigen ID. Damit diese ID auch global eindeutig ist, wird die Simulator ID vorangestellt. Diese global eindeutige ID ist die volle ID dieser Entität (*Entity.full_id*).

Damit Simulatoren miteinander kommunizieren, müssen die Beziehungen zwischen Entitäten definiert werden. Für das Verbinden der Entitäten wird die Methode *World.connect()* verwendet. Dabei muss die Quell-Entität und die Ziel-Entität übergeben werden. Die beiden Entitäten müssen zu verschiedenen Simulatoren gehören. Außerdem dürfen dabei keine zirkulären Abhängigkeiten erstellt werden. Das bedeutet es darf zum Beispiel nicht eine Entität A mit der Entität B verbunden werden und danach die Entität B mit der Entität A. Um solche Verbindungen doch zu ermöglichen gibt es eine andere Möglichkeit [260], auf die hier aber nicht weiter eingegangen wird.

mosaik bietet Möglichkeiten an um alle Modellverbindungen einzeln oder auch alle in einem Schritt zu verbinden. Anschließend kann die Simulation mithilfe der *world.run* Methode ausgeführt werden.

Der SimManager

Der Simulations-Manager ist für das Starten und Handhaben der externen Simulator-Prozesse, die an einer Simulation beteiligt sind, sowie für die Kommunikation mit diesen verantwortlich [264]. Es gibt drei Möglichkeiten, um sich mit einem Simulator zu verbinden. In Abbildung 3.4 werden die drei Möglichkeiten anschaulich dargestellt.

Normalerweise werden Simulatoren in einem separaten Sub-Prozess gestartet und kommunizieren dann mit mosaik über Netzwerk-Sockets. Dies hat den Vorteil, dass Simulatoren in beliebigen Programmiersprachen programmiert werden können und dass die Simulationsschritte von verschiedenen Simulatoren auch parallel ausgeführt werden können, wenn diese auch sonst unabhängig voneinander sind. Simulatoren, die in Python 3 implementiert wurden, können aber auch wie ganz normale Python Module importiert und ausgeführt werden. So werden alle API-Aufrufe wie ganz normale Python Methodenaufrufe behandelt. Dadurch kann auf die Socket-Netzwerkkommunikation sowie auf die Nachrichten Serialisierung und Deserialisierung verzichtet werden, wodurch die Performance erhöht wird. Allerdings können Simulatoren so nicht parallel ausgeführt werden.

Wenn Python 3 benutzt wird und der Simulator wenig berechnen muss, kann das Ausführen des Simulators wie ein normales Python Modul gute Ergebnisse liefern. Bei vielen Berechnungen könnte es besser sein, einen eigenen Prozess für diesen Simulator zu starten, wodurch die Berechnungen parallel ausgeführt werden können. In der Praxis kann es auch hilfreich sein beide Möglichkeiten auszuprobieren und auszuwerten, um die beste Performance zu erhalten.

Wenn die beiden oben genannten Möglichkeiten nicht funktionieren, weil man den Simulator nicht selbst starten kann oder will, kann mosaik sich auch zu einer bereits laufenden Instanz eines Simulators verbinden. Für mosaik macht es keinen Unterschied welche der drei Möglichkeiten verwendet wird, da mosaik über eine *SimProxy* auf die Simulatoren zugreift. In der *sim_config* wird angegeben auf welchem Weg auf den Simulator zugegriffen wird.

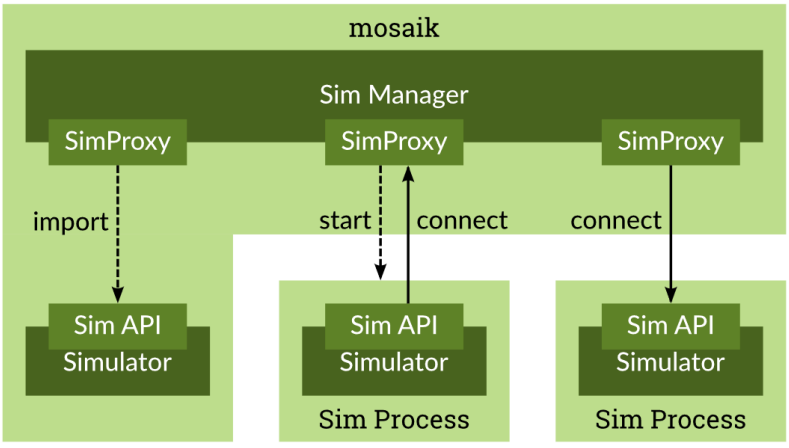


Abbildung 3.4: *Die drei Simmanager Möglichkeiten [264]*

Der Scheduler

Der mosaik-Scheduler wird aktiv, nachdem das Szenario definiert und die Simulation gestartet wurde [261]. Er ist für die Ausführung und die Synchronisation der Simulatoren zuständig. Simulationen werden ausgeführt, indem die Zeitschritte ausgeführt werden.

Zeitschritte in mosaik Die Zeit hat dabei intern keine Einheit, aber nach Konvention werden Sekunden verwendet. Wenn alle an dem Szenario beteiligten Simulatoren mit einer anderen Einheit (z. B. Minuten oder Millisekunden) arbeiten, kann auch eine andere Einheit verwendet werden.

Die aktuelle Simulationszeit der Simulatoren wird von mosaik festgehalten. Am Anfang beginnen alle Simulatoren bei der Simulationszeit 0. Um den nächsten Schritt auszuführen ruft mosaik die *step()*-Methode auf. Dabei wird die aktuelle Zeit übergeben. Diese Methode liefert den Zeitpunkt, an dem die step-Methode dieses Simulators das nächste Mal aufgerufen werden soll. Dadurch muss die Schrittgröße eines Simulators nicht konstant sein, sondern kann während der Simulation variieren. Die berechneten Daten gehören dann zu dem gesamten Intervall.

Synchronisation der Simulatoren mosaik verfolgt Abhängigkeiten zwischen den Simulatoren und lässt sie nur einen Simulationsschritt durchführen, wenn ein anderer Simulator die Daten dieses Simulators benötigt. Wenn die Simulatoren voneinander unabhängig sind, können ihre Simulationsschritte auch parallel durchgeführt werden. Wenn die Daten eines Simulators von keinem anderen Simulator benötigt werden, werden trotzdem solange Zeitschritte ausgeführt, bis die beim Ausführen des Simulators bei der *world.run*-Methode angegebene Endzeit erreicht wurde.

Der Algorithmus zum Ausführen des Simulators funktioniert wie im Folgenden beschrieben [261].

1. Sollte es einen nächsten Schritt geben?

   Ja Weiter mit Schritt 2.

   Nein Der Simulator wird gestoppt.

2. Gibt es Simulatoren, die Daten von uns benötigen?

   Ja Weiter mit Schritt 3.

   Nein Weiter mit Schritt 4.

3. Benötigt ein von uns abhängiger Simulator neue Daten?

   Ja Weiter mit Schritt 4.

   Nein Warte bis ein von uns abhängiger Simulator neue Daten benötigt und gehe dann zu Schritt 4.

4. Sind alle benötigten Eingabedaten von anderen Simulatoren verfügbar?

   Ja Weiter mit Schritt 5.

   Nein Warte bis bis alle Daten verfügbar sind und gehe dann zu Schritt 5.

5. Alle erforderlichen Eingabedaten sammeln

6. Die gesammelten Eingabedaten an den Simulator senden, den Simulationsschritt ausführen und die Zeit für den nächsten Simulationsschritt erhalten.

7. Alle Daten, die von anderen Simulatoren benötigt werden, aus dem Simulator holen.
8. Alle Simulatoren, die bereits auf diese Daten warten, benachrichtigen.

3.1.1.3 mosaik-Simulator mit Odysseus

Odysseus[2] ist ein In-Memory Datenstrommanagement Framework. Es kann verwendet werden, um große Datenmengen in Echtzeit zu verarbeiten. Es kann also auch verwendet werden, um die Ergebnisse von mosaik während einer Simulation zu verarbeiten, zu visualisieren und zu speichern [257].

Beim Verarbeiten von Datenströmen können verschiedene Operatoren, wie SELECT und PROJECT, verwendet werden, um die Daten zu filtern oder eine Aggregation für bestimmte Zeiträume zu berechnen. Es können auch mehrere Operatoren miteinander verbunden werden, um einen so genannten Operator-Graphen zu bilden. Die Daten können zum Beispiel in Listen, Tabellen oder Graphen visualisiert werden. Für komplexere Visualisierungen können Dashboards erstellt und individuell angepasst werden. Außerdem bietet Odysseus auch Konnektoren an, mit den Daten in verschiedene Formate umgewandelt werden können und diese zum Beispiel in Datenbanken abzuspeichern. Dadurch ist Odysseus zum Speichern von Simulationsdaten gut geeignet. Es existiert bereits ein Odysseus-Adapter, mit dem Odysseus in Kombination mit mosaik sehr einfach verwendet werden kann. Dabei gibt es zwei Kommunikationsmöglichkeiten.

RPC

Die Kommunikation über RPCs ist zwar komfortabel, allerdings wartet mosaik dabei immer auf eine Antwort und blockiert. So muss ein Simulator, welcher die SimAPI implementiert direkt beim Start des mosaik-Szenarios gestartet sein. Dadurch kann Odysseus sich nicht jederzeit ein- und ausklinken. Zudem ist eine auf AMQP basierende Anbindung zwar zuverlässig, allerdings ist diese auch langsamer. Zudem ist die Konfiguration durch den Broker aufwendiger [241].

ZeroMQ

Bei ZeroMQ findet die Kommunikation direkt statt. Es wird also auf einen zwischengeschalteten Server verzichtet. Dies bietet eine geringere Latenz und die Konfiguration ist einfacher, da kein Server vorhanden ist. Allerdings kann so nur kommuniziert werden, wenn beide Kommunikationspartner gleichzeitig verfügbar sind, was aber bei der Kommunikation zwischen mosaik und Odysseus vernachlässigbar ist. Die verschiedenen Kommunikationsschemata von ZeroMQ bieten zudem Möglichkeiten für verschiedene Anwendungsfälle [241].

mosaik mit Java

Um einen Simulator in der Programmiersprache Java zu implementieren muss zuerst das *mosaik-api-java* Repository [3]geklont werden. Dort befindet sich die High-Level mosaik API für Java [256]. Der eigene Simulator muss von der abstrakten Klasse *de.offis.mosaik.api.Simulator* abgeleitet werden. Der Simulator muss die Methoden *init()*, *create()*, *step()*, und *getData()* implementieren. Diese machen das gleiche wie die gleichnamigen Methoden in Python. Die Modelle können auch in Java implementiert

[2] `http://odysseus.informatik.uni-oldenburg.de`
[3] `https://bitbucket.org/mosaik/mosaik-api-java`

werden und müssen in der Metadaten Definition in dem JSON-String angegeben werden. Alles andere wird ähnlich wie auch mit Python gemacht. Aus diesem Grund wird hier nicht näher darauf eingegangen.

3.1.1.4 Zusammenfassung

Das mosaik Framework kann verwendet werden, um mit möglichst wenig Aufwand eine Smart Grid Simulation durchzuführen. Dazu muss ein Szenario erstellt und ein Simulator implementiert werden. Zudem ist mosaik durch die Low-Level API unabhängig von einer Programmiersprache und durch die High-Level API mit höheren Programmiersprachen verwendbar. Durch den bereits existierenden Odysseus-Adapter kann mosaik auch in Kombination mit Odysseus mit wenig Aufwand verwendet werden.

3.1.2 Odysseus

In intelligenten Energienetzen, sogenannten Smart Grids, müssen große Datenmengen in kurzer Zeit verarbeitet werden, um die Netze entsprechend des aktuellen Netzzustandes steuern zu können. Dabei müssen nicht nur Daten von klassischen Energieerzeugern und –verbrauchern verarbeitet werden, sondern auch von sehr flexiblen Erzeugern wie beispielsweise Windkraft- und Solaranlagen sowie flexiblen Verbrauchern wie Energiespeicher oder elektrischen Fahrzeugen [89]. In einem flächendeckenden Smart Grid müssen somit kontinuierlich Daten verarbeitet und ausgewertet werden.

Klassische Datenbankmanagementsysteme (DBMS) sind für die Verarbeitung solcher Datenströme jedoch kaum geeignet. Daher sind Datenstrommanagementsysteme (DSMS) entwickelt worden, welche über entsprechende Mechanismen mit kontinuierlichen Datenströmen arbeiten können, ohne die Daten zunächst in einer Datenbank zu speichern [110]. Im Folgenden soll daher das Datenstrommanagementsystem Odysseus vorgestellt werden, welches sich insbesondere durch seine große Flexibilität als Grundlage für die Verarbeitung von Datenströmen in Smart Grids eignet. Odysseus wurde an der Abteilung Informationssysteme am Department für Informatik der Universität Oldenburg entwickelt und ist bereits mit seinen Grundfunktionen ein funktionierendes DSMS, kann jedoch durch seine modulare Struktur auch als Framework zur Entwicklung hoch angepasster DSMS gesehen werden.

Zunächst wird in dieser Arbeit daher in Unterunterabschnitt 3.1.2.1 die Architektur von Odysseus näher betrachtet sowie ein Überblick über die Benutzeroberfläche gegeben. Anschließend wird in Unterunterabschnitt 3.1.2.2 erläutert, wie Odysseus gesteuert werden kann. Dabei werden insbesondere die Anfragesprachen CQL und PQL sowie Odysseus Script zur Steuerung von Odysseus vorgestellt. In Unterunterabschnitt 3.1.2.3 werden Funktionen gezeigt, die sich in Odysseus zusätzlich aktivieren lassen, wodurch Odysseus zu einem individuellen DSMS aufgebaut werden kann. Unterunterabschnitt 3.1.2.4 zeigt anschließend Möglichkeiten, die bestehenden Funktionen zu erweitern und grundlegend neue Funktionen zu implementieren. Abschließend folgt in Unterunterabschnitt 3.1.2.5 eine Zusammenfassung und ein Fazit.

3.1.2.1 Aufbau

Odysseus basiert auf einer Client-Server-Architektur, sodass es mit Odysseus Server, dem eigentlichen Verarbeitungssystem, und Odysseus Studio, der Benutzeroberfläche, zwei wesentliche Komponenten gibt.

Odysseus Server dient dabei insbesondere der eigentlichen Verarbeitung von Datenströmen, Anfragen sowie der Bereitstellung weiterer Komponenten (siehe Unterunterabschnitt 3.1.2.3). Damit ist Odysseus Server auch der Ausgangspunkt jeglicher Entwicklung zusätzlicher Komponenten (siehe Unterunterabschnitt 3.1.2.4).

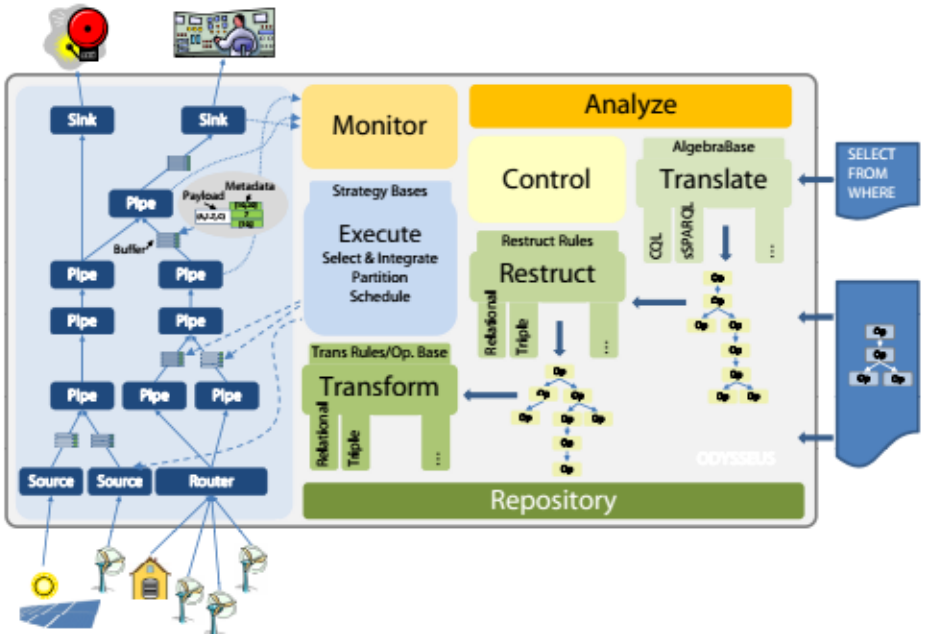


Abbildung 3.5: *Übersicht der Odysseus Architektur [34]*

Abbildung 3.5 zeigt die schematische Struktur von Odysseus. Dabei ist auch die Position von Odysseus zwischen den Datenquellen (unten) und Anwendungen (oben) zu erkennen. Neben der Anfrageverarbeitung und -verwaltung übernimmt Odysseus dabei insbesondere auch die Aufgabe der Übersetzung von formulierten Anfragen in ausführbare Anfragepläne. Hieran sind vorwiegend drei Komponenten beteiligt: eine Übersetzungskomponente (Translate), eine Restrukturierungskomponente (Restruct) sowie eine und eine Transformationskomponente (Transform) [34].

Die Translate-Komponente dient dabei der Übersetzung von Anfragen in eine logische Algebra-Basis. Das heißt, die Anfrage wird durch einen entsprechenden Parser in ihre logischen Operatoren zerlegt und ein logischer Anfrageplan erstellt. Die Restruct-Komponente kann diesen logischen Anfrageplan nun auf Basis vorhandener Regeln so neu ordnen, dass ein möglichst effizienter Plan zur Verarbeitung der Daten erfolgt. Der restrukturierte, logische Anfrageplan muss abschließend von der Transform-Komponente in einen physischen Anfrageplan übersetzt werden. Da ein logischer Operator auf unterschiedliche Arten implementiert sein kann, gibt es auch hier entsprechende Regeln zur Transformation. Insgesamt wird also eine vom Anwender formulierte Anfrage zunächst in einen logischen Plan übersetzt, die logischen Operatoren ggf. neu geordnet und schließlich ein physischer Anfrageplan erzeugt [34].

Die beteiligten Komponenten sind dabei in Odysseus flexibel anpassbar und erweiterbar. Für viele Anwendungsfälle sind die vorhandenen Sprachen, Operatoren und Regeln jedoch umfangreich genug, sodass nur wenige Anwender hiermit konfrontiert werden sollten.

Odysseus Studio beinhaltet als Benutzeroberfläche von Odysseus insbesondere den Editor zur Verwendung der in Unterunterabschnitt 3.1.2.2 genannten Anfragesprachen. Er unterstützt übliche Funktionen wie Autovervollständigung und Syntax-Highlighting. Zusätzlich bietet Odysseus Studio einen grafischen Editor zur Definition von Anfragen, mit welchem Operatoren per Drag'n'Drop kombiniert und so zu einer PQL-Anfrage verknüpft werden können. Operatoren können zudem mit entsprechenden Eigenschaften (z.B. Prädikat für Select-Operator) versehen werden und es findet eine Prüfung auf Vollständigkeit und Ausführbarkeit der so erzeugten Anfrage statt. Die so erzeugte Anfrage kann schließlich auch textuell nachbearbeitet werden [116].

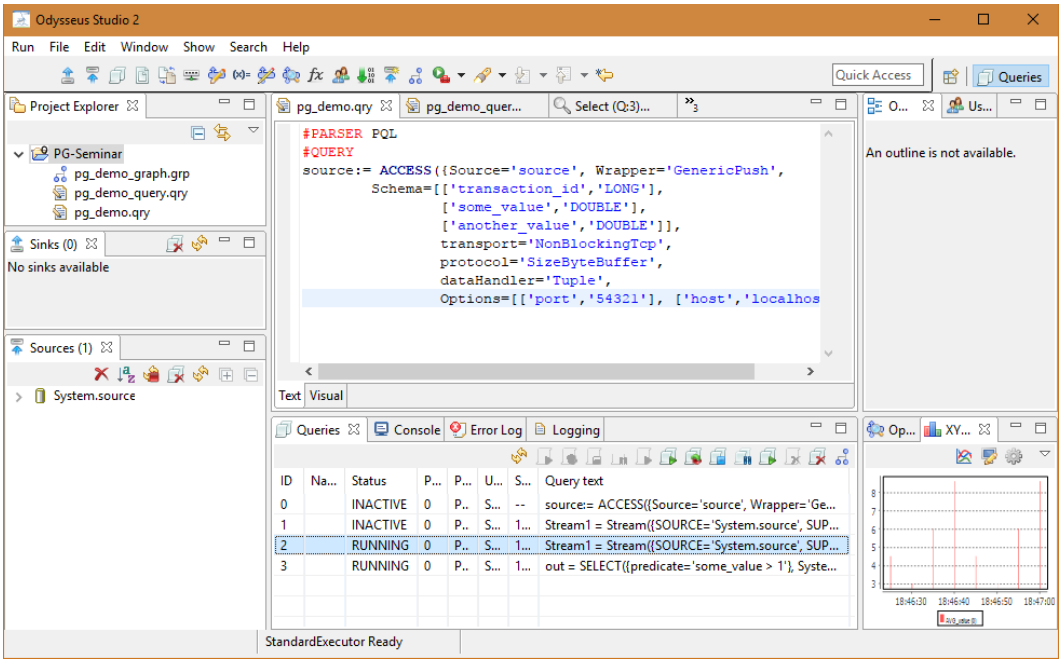


Abbildung 3.6: *Benutzeroberfläche von Odysseus*

Darüber hinaus bietet Odysseus Studio mit dem Project Explorer eine Projektverwaltung zur Organisation verschiedener Projekte, Übersichten zu Datenquellen und –senken sowie der installierten Anfragen. Eine weitere wichtige Eigenschaft von Odysseus Studio ist die Visualisierung der aus den Anfragen erzeugten Anfragepläne. Insbesondere bei der Verwendung sehr komplexer Anfragen bietet diese Funktion im Fall unerwarteter Ergebnisse einen Ausgangspunkt zur Fehlersuche. Dabei lassen sich die Operatoren im Anfrageplan auswählen und weitere Informationen anzeigen [116].

Die aus den Anfragen erzeugten Ergebnisse können in Odysseus Studio auf verschiedene Arten dargestellt werden. Neben einer Listen- und Tabellenform bietet Odysseus verschiedene Diagramme zur Darstellung von Ergebnissen an. So lässt sich die Oberfläche von Odysseus dann insgesamt auch sehr frei gestalten und statt einer wie in Abbildung 3.6 dargestellten eher entwicklungszentrierten Perspektive auch eine ergebniszentrierte Perspektive erzeugen, in welcher Ergebnisse aus Anfragen in Tabellen, Diagrammen, usw. während des laufenden Betriebs dargestellt werden. Natürlich sind auch Mischformen der Perspektiven möglich.

Odysseus muss jedoch nicht zwangsläufig über Odysseus Studio gesteuert werden, sondern es lassen sich auch ein Webservice Interface sowie eine REST-Schnittstelle verwenden. Hierdurch ist es möglich

einen eigenen Client zu entwickeln, bzw. die Verwaltung von Anfragen in bestehende Applikationen zu integrieren. Diese Funktionen müssen allerdings zunächst als zusätzliches Feature aktiviert werden (siehe Unterunterabschnitt 3.1.2.3) [117].

3.1.2.2 Steuerung

Zur Verwaltung von Datenströmen, kontinuierlichen Anfragen sowie von Datensenken, stehen in Odysseus verschiedene Anfragesprachen zur Verfügung. Insbesondere sind hier CQL (Continuous Query Language) [15], PQL (Procedural Query Language) [14] sowie Odysseus Script zu nennen.

Odysseus Script dient hierbei als grundlegende Sprache innerhalb von Odysseus. Hierdurch kann das DSMS gesteuert und Anfragesprachen wie CQL oder PQL verwendet werden. Neben der Deklaration von Variablen, Konstanten, Kommentaren oder auch des Steuerflusses (Control Flow) bietet Odysseus Script insbesondere Befehle (Commands) zur Installation oder Steuerung von Anfragen. Mit dem Rautezeichen (#) wird ein Befehl eingeleitet und dieses vom Namen und ggf. weiteren Parametern des Befehls gefolgt. So lässt sich mittels *#STARTQUERY qName* beispielsweise eine (bereits gespeicherte) kontinuierliche Anfrage des Namens *‚qName‘* starten. Entsprechende Befehle zum Stoppen von Anfragen (*#STOPQUERY*), Hinzufügen (*#ADDQUERY*), Pausieren (*#SUSPENDQUERY*) sowie Fortsetzen von Anfragen (*#RESUMEQUERY*) werden auf identische Art verwendet. Dabei bietet Odysseus Script eine Vielzahl verschiedener Befehle zur Steuerung, von welchen an dieser Stelle noch *#PARSER* genannt sei, mit welchem sich der Parser, z. B. CQL und PQL, als Parser für nachfolgende Befehle, beziehungsweise Anfragen, festlegen [117].

Die bereits erwähnte CQL (Continuous Query Language) ist eine auf SQL (Structured Query Language) basierende Sprache. Dabei unterscheidet sich CQL im Wesentlichen durch die Möglichkeit der Festlegung von Zeitfenstern von SQL. Dies ist aufgrund der Unbegrenztheit von Datenströmen notwendig, da andernfalls beispielsweise bei aggregierenden Operatoren, keine feste Definition der zu verwendenden Daten vorhanden wäre [15]. Wie auch eine SQL-Anfrage besteht ein CQL-Statement somit aus einem *SELECT*-, *FROM*- und *WHERE*-Teil und kann um *GROUP BY* sowie *HAVING* ergänzt werden. Das zu verwendende Zeitfenster kann dabei im *FROM*-Teil der Anfrage angegeben werden [15].

Während CQL somit beschreibt, was innerhalb der Anfrage gemacht werden soll, wird mit PQL beschrieben, wie etwas innerhalb der Anfrage gemacht werden soll. In CQL wird bei der Deklaration der Anfrage also eher das erwartete Ergebnis beschrieben, während in PQL dagegen eher ausgedrückt wird, wie die Anfrage bearbeitet werden soll und bietet damit sehr viel mächtigere Möglichkeiten die Anfrage zu kontrollieren.

```
1 window = TIMEWINDOW({
2          size = [5, 'MINUTES'],
3          advance = 1
4          }, input)
5
6 output = SELECT({
7          predicate='price > 100'
8          }, window)
```

Listing 3.1: *Kontinuierliche Anfrage in PQL*

Listing 3.1 zeigt, wie ein Zeitfenster sowie eine kontinuierliche Anfrage auf einem Datenstrom deklariert werden können. So wird zunächst das Zeitfenster mittels des *TIMEWINDOW*-Operators

festgelegt, welches im Beispiel den Namen *window* trägt. Dem Operator folgen die Parameter, die ihn spezifizieren. In diesem Fall die Größe (5 Minuten) sowie die Verschiebung des Fensters (1). Anschließend wird festgelegt, auf welchen Eingangsdatenstrom sich das Zeitfenster bezieht, hier also einen Datenstrom namens *input*.

Mit dem Namen *output* wird in Listing 3.1 anschließend die eigentliche Anfrage definiert, welche vom Operator *SELECT* eingeleitet wird. In diesem Beispiel sollen als Parameter alle Elemente des Datenstroms ausgewählt werden, bei welchen das Prädikat *price > 100* erfüllt ist. Auch hier muss angegeben werden, auf welchen Datenstrom sich die Anfrage bezieht. In diesem Fall also auf das zuvor angelegte Zeitfenster *window*.

3.1.2.3 Features

Odysseus basiert auf dem OSGi-Standard . Hierbei handelt es sich um eine Reihe von Spezifikationen, welche zu einem Java-basierten Softwaresystem auf Basis von Komponenten, auch Bundles oder Plug-ins, führen. Dabei sind die Implementierungen der einzelnen Bundles völlig unabhängig voneinander und eine Kommunikation zwischen ihnen findet nur über definierte Services statt [216].

Diese Architektur bietet insbesondere für Odysseus diverse Vorteile. Da Odysseus selbst als „Framework für die Entwicklung von DSMS“ [34] entwickelt wurde, ist eine hohe Anpassbarkeit an den jeweiligen Verwendungszweck des DSMS erforderlich. Die OSGi-Architektur ermöglicht es, mit Odysseus ein sehr spezifisches DSMS aufzubauen, welches spezielle Anwendungsfälle abdeckt, dabei jedoch ohne unnötige Funktionen auskommt. Zu den Grundfunktionen von Odysseus gehören dabei unter anderem die Komponenten zur Übersetzung, Restrukturierung und Transformation von Anfragen (siehe Unterunterabschnitt 3.1.2.1), eine Benutzerverwaltung sowie natürlich die gesamte Anfrageverarbeitung.

Ein sehr mächtiges Werkzeug, welches in Odysseus standardmäßig enthalten ist, sind die MEP-Funktionen und Operatoren. Diese können zur Filterung oder Verarbeitung von Daten innerhalb von Anfragen verwendet werden. Zu den grundsätzlichen Operatoren gehören hier mathematische Funktionen (Addition, Subtraktion, Modulo, etc.) sowie logische Verknüpfungen (and, or, not, xor) und Vergleichsoperatoren. Zudem gibt es diverse weitere Möglichkeiten, Daten mit Hilfe der MEP-Funktionen zu bearbeiten. So sind diverse Konvertierungen von Datentypen (z. B. Konvertierung eines Integer-Werts in eine String-Repräsentation) ebenso möglich wie Hashing-, Verschlüsselungs- bzw. Prüfsummen-Operationen. Darüber hinaus gibt es diverse weitere Funktionen, welche jedoch teilweise als zusätzliches Plug-In (bzw. Feature) installiert werden müssen [117].

Zusätzlich zu den Grundfunktionen gibt es bereits diverse zusätzliche Bundles, die als Feature in Odysseus eingebunden werden können und welchen eine kleine Auswahl an dieser Stelle vorgestellt werden soll. So können diverse Wrapper zur Anbindung verschiedenster Datenquellen und Protokolle verwendet werden. Für den Smart Grid Kontext steht beispielsweise ein Protokoll-Handler zur Verfügung, über welchen eine Verbindung zum Simulations-Framework mosaik hergestellt werden kann. Hierdurch können die in der Simulation erzeugten Daten mit Odysseus analysiert oder visualisiert werden. Da viele der in Odysseus genutzten Operatoren auf Tupel-basierten Daten arbeiten, mosaik die Daten jedoch im JSON Format sendet, müssen die Daten unter Umständen vor der Verarbeitung zunächst transformiert werden . Hierzu gibt es das KeyValue Feature, mit welchem Daten, die als Schlüssel-Wert-Paar (Key-Value Pair) vorliegen, in schemabasierte Tupel transformiert werden können.

Das Plug-In bietet jedoch auch selbst Operatoren wie eine Selektion, Projektion, usw., sodass eine Transformation je nach beabsichtigter Weiterverarbeitung nicht unbedingt notwendig ist [117].

Neben vielen weiteren Protokoll-Handlern sind auch diverse Transport-Handler verfügbar, welche für die Kommunikation zwischen Odysseus und Datenquellen oder –senken zuständig sind. Hierzu gehören beispielsweise TCP-, HTTP- oder POP3-Transport-Handler. Auch ein entsprechender Handler für die Kommunikation mit Apache Kafka[4] steht zur Verfügung. Zusammen mit den Daten-Handlern, z.B. dem standardmäßig aktivierten Tuple data handler oder auch dem KeyValueObject data handler, lassen sich so diverse Datenquellen und –senken anbinden und die jeweils speziellen Anforderungen zu Verbindungen und Formaten berücksichtigen [117].

Wie in Unterunterabschnitt 3.1.2.1 bereits erwähnt bietet Odysseus Studio diverse Möglichkeiten zur Darstellung von Ergebnissen. Diese können mit dem Dashboard Plug-In erweitert werden. So ist es hiermit möglich, weitere Darstellungsarten zu verwenden oder auch mehrere Anzeigen, bzw. „DashboardParts“, auf einem einzelnen Dashboard anzuzeigen [117].

Ein weiteres relevantes Plug-In ist das Database Feature. Zwar unterscheiden sich die Anwendungsbereiche von DSMS und DBMS gewöhnlich, doch sind natürlich Situationen denkbar, in welchen bei der Datenstromverarbeitung auch in Datenbanken gespeicherte Daten einbezogen werden sollen. Hierbei kann es sich beispielsweise um historische Daten oder Informationen zum Wetter handeln. Das Database Feature ermöglicht dabei die bidirektionale Kommunikation mit relationalen Datenbanksystemen. Somit können Daten aus DBMS ausgelesen werden und Daten in DBMS gespeichert werden. Das Plug-In unterstützt derzeit MySQL, ProstgreSQL, Oracle und Derby Datenbanken. Datenbanken können so als Datenquelle installiert werden. Mit dem *DBEnrich*-Operator ist es außerdem möglich, Informationen aus einer Datenbank per JOIN-Operation mit den Daten eines eingehenden Datenstroms zu verbinden [117].

So wie Datenströme mit Odysseus um Informationen aus Datenbanken ergänzt werden können, sind auch Informationen aus anderen Quellen möglich. So bietet der *WSEnrich*-Operator beispielsweise die Möglichkeit, Informationen über eine REST- oder SOAP-Schnittstelle zu beziehen und den empfangenen Daten hinzuzufügen [117].

Zusammen mit Odysseus kommen auch diverse Datengeneratoren. Diese stellen zwar nicht direkt eine Erweiterung dar, bieten jedoch insbesondere zu Entwicklungs- und Testzwecken sinnvolle Funktionen, indem sie Datenquellen simulieren. So kann ein entsprechender Generator beispielsweise in festgelegter Geschwindigkeit Zufallszahlen erzeugen. Wird der Generator als Datenquelle in Odysseus eingebunden, lässt sich z. B. die Funktionaltität von Anfragen testen, auch wenn keine reale Datenquelle zur Verfügung steht. Insbesondere lässt sich auch das Verhalten bei sehr großen oder sehr kleinen Datenmengen testen und Parameter oder Fenstergrößen in Anfragen damit optimieren [117].

3.1.2.4 Erweiterung

Zwar lassen sich viele Anwendungsszenarien bereits mit den vorhandenen Grundfunktionen oder zusätzlichen Features abdecken, doch können spezielle Anforderungen die Entwicklung individueller Funktionen erfordern. Aufgrund der Verwendung des OSGi-Frameworks ist die Entwicklung eigener Features für Odysseus möglich, ohne bestehenden Code ändern zu müssen.

[4] `https://kafka.apache.org/`

Häufig ist es für bestimmte Anwendungsfälle jedoch schon ausreichend, bestehende Funktionen zu erweitern. Beispielsweise können die im vorigen Abschnitt vorgestellten MEP-Funktionen erweitert werden, indem eine eigene Funktion implementiert wird, welche dann genau wie die Vorhandenen verwendet werden kann. Hierzu muss lediglich eine neue Java-Klasse erstellt werden, welche die *AbstractFunction*-Klasse erweitert. Über den parametrisierten Aufruf des Konstruktors der Oberklasse lassen sich das Schlüsselwort der neuen MEP-Funktion sowie weitere Parameter festlegen. In Listing 3.2 ist beispielhaft zu sehen, wie im Konstruktor der Super-Konstruktor mit den entsprechenden Parametern aufgerufen wird. In diesem Fall würde die Funktion über das Schlüsselwort *grid* aufgerufen und zwei Eingabewerte entgegennehmen, deren Datentyp Double ist. Anschließend wird der Typ des Rückgabewerts angegeben, hier ebenfalls Double. Der folgende boolsche-Wert gibt an, ob der Rückgabewert eine Konstante ist, sodass die Funktion nicht bei jedem Aufruf erneut berechnet werden muss. Die beiden letzten Werte geben die (erwartete) Komplexität der Berechnung an, sodass Odysseus die Ausführungsreihenfolge optimieren kann, falls mehrere Funktionen verwendet werden. Die in Listing 3.2 gezeigte *getValue()*-Funktion dient dann den eigentlichen Berechnungen, für welche der Operator geschrieben wird. Hier kann mittels *getNumericalInputValue* auf die übergebenen Werte zugegriffen werden. Außerdem kann über entsprechende Funktionen auf Metadaten und weitere Werte des Datenstroms zugegriffen werden [117].

```
1  public class GridCalc extends AbstractFunction<Double> {
2
3      public static final SDFDatatype[][] accTypes =
4          new SDFDatatype[][] {{ SDFDatatype.DOUBLE }, { SDFDatatype.DOUBLE }};
5
6      public GridCalc() {
7          super("grid", 2, accTypes, SDFDatatype.DOUBLE, true, 1, 2);
8      }
9
10     @Override
11     public Double getValue() {
12         double a = this.getNumericalInputValue(0);
13         double b = this.getNumericalInputValue(1);
14
15         // some special calculations
16         // ...
17
18         double result = a * b;
19
20         return result;
21     }
22 }
```

Listing 3.2: *Implementieren einer neuen MEP-Funktion*

Für noch komplexere Aufgaben oder ganz bestimmte Anwendungsfälle kann es erforderlich sein, einen ganz neuen Operator zu implementieren, welcher sich dann beispielsweise in PQL verwenden lässt. Hierzu ist es erforderlich, einen logischen Operator und mindestens eine Implementierung des physischen Operators (siehe Unterunterabschnitt 3.1.2.1) zu erstellen und ggf. Restrukturierungs- und Transformationsregeln für den Operator anzulegen. Für den logischen Operator muss die Klasse *AbstractLogicalOperator* erweitert und mindestens das Interface *ILogicalOperator* implementiert werden. Für den physischen Operator wird die generische Klasse *AbstractPipe<R,W>* erweitert [117].

Da diverse Schritte für die Implementierung notwendig sind, wird an dieser Stelle jedoch auf eine weitere Erläuterung verzichtet und stattdessen auf die Dokumentation im Odysseus Wiki[5] verwiesen.

In Unterunterabschnitt 3.1.2.3 wurde bereits gezeigt, dass für die Anbindung verschiedener Datenquellen bereits diverse Wrapper als installierbare Features gibt. Dennoch sind Anwendungsfälle vorstellbar, welche neue Protokolle voraussetzen. Bei der Implementierung sind zunächst zwei grundsätzliche Varianten zu unterscheiden: GenericPull und GenericPush, während beim ersteren Odysseus die Daten selbst aktiv nachfragt, werden die Daten beim zweiten Ansatz aktiv durch die Datenquelle gesendet. Die Unterschiede sind bei der Implementierung zu berücksichtigen. Zudem muss nicht der gesamte Wrapper in jedem Fall neu implementiert werden, sondern Transport-, Protokoll- und Daten-Handler können unabhängig voneinander entwickelt werden. Sendet eine Datenquelle über TCP also in einem bestimmten Protokoll, welches Odysseus nicht bekannt ist, so kann der vorhandene TCP-Transport-Handler genutzt werden und es muss lediglich ein passender Protokoll-Handler implementiert werden, welcher die Daten in eine Tupel- oder Key-Value-Repräsentation transformiert [117].

Zu beachten ist, dass auch der umgekehrte Weg notwendig sein kann. Sollen die Daten beispielsweise mit einem anderen Programm weiterverarbeitet werden, kann es erforderlich sein diese in einem individuellen Format zu senden. Auch in diesem Fall kann ein Protokoll-Handler entwickelt werden, sodass die Transformation der Daten direkt in Odysseus erfolgen kann und keine weiteren Zwischenschritte notwendig sind.

Neben der gezeigten Erweiterung bestehender Funktionen ist auch die Entwicklung neuer Funktionen von Grund auf möglich. Hierfür kann ein neues Plug-In Projekt angelegt und an passender Stelle in die Ordnerstruktur von Odysseus eingefügt werden. Dabei kann das neue Plug-In über Services mit anderen Erweiterungen oder dem Core-System kommunizieren und selbst eigene Services anbieten, wobei entsprechende Schnittstellen über eine XML-Datei definiert werden können. Sowohl die Erweiterung bestehender Funktionalitäten als auch neue Plug-Ins müssen jedoch einem Feature zugeordnet werden, damit sie in die Startkonfiguration von Odysseus aufgenommen werden können. Dabei ist es möglich diese einem bereits vorhandenen Feature zuzuordnen oder ein neues Feature-Projekt zu erstellen, falls es keine sinnvolle Kombination mit bestehenden Features gibt [117].

3.1.2.5 Zusammenfassung

In diesem Abschnitt wurde das Datenstrommanagementsystem (DSMS) Odysseus vorgestellt. Es konnte dabei insbesondere gezeigt werden, dass Odysseus sich durch seine Flexibilität als Framework zur Entwicklung individueller DSMS für spezifische Anwendungsfälle eignet. So bieten schon die Anfragesprachen CQL und PQL vielfältige Möglichkeiten zur Modellierung der Anfrageverarbeitung, welche sich durch zusätzlich installierbare Features stark erweitern lassen.

Darüber hinaus wurde dargestellt, wie die bestehenden Funktionen bei Bedarf erweitert werden können. Hierdurch ist die Verarbeitung von Daten aus diversen Datenquellen möglich, da individuelle Transport-, Protokoll- und Daten-Handler auf einfache Weise zu Odysseus hinzugefügt werden können. Aufgrund der auf dem OSGi-Standard basierenden Architektur ist zudem auch die Integration völlig neuer Funktionen möglich, ohne den Code des bestehenden Systems ändern zu müssen.

Insgesamt erfüllt Odysseus damit die Anforderungen, die an ein DSMS für die Verarbeitung von Datenströmen in Smart Grids gestellt werden können. Insbesondere die Abdeckung diverse Protokolle

[5] https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Creating+new+Operators

und die Möglichkeit zur Einbindung neuer Funktionen und Schnittstellen ermöglichen auch einen langfristigen Einsatz von Odysseus, da an neue Anforderungen einfach angepasst werden kann.

3.1.3 Webvisualisierung

Deutschlandweit werden jährlich rund 600 Mrd. kWh Strom und zwischen $817 - 978 Mrd.m^3$ Gas verbraucht [74, 75] und dabei wird immer mehr auf erneuerbare Energien gesetzt. So erreichte der Anteil der erneuerbaren Energien 2015 ungefähr 35% an der öffentlichen Nettostromerzeugung [46]. Durch die schwankende Verfügbarkeit dieser meist wetterabhängigen Energiequellen und dem unregelmäßigen Lastverlauf durch die Vielzahl an Konsumenten und der steigenden Anzahl an Erzeugern, wird die reibungslose und permanente Bereitstellung zu einer immer komplexeren Herausforderung. Um durch reale Echtzeitanalysen und Prognosen des Energienetzes und nicht nur durch Extrapolation aufgrund von Vorgangsjahren, Lastspitzen und Tiefs frühzeitig erkennen und prognostizieren zu können, werden viele Echtzeitdaten benötigt. Um unter anderem die Stromdaten von Verbrauchern zu erhalten, sollten sogenannte Smart Meter [6] verwendet werden. Je mehr Haushalte über ein Smart Meter verfügen, desto besser können die Modelle für die Analysen verwendet werden. Um eine maximale Anzahl an Installationen zu erreichen, verabschiedete der Bundestag am 26.08.2016 das Gesetz zur Digitalisierung der Energiewende[7] und hat damit einhergehend über das Messstellenbetriebsgesetz einen Rollout-Verlaufsplan erstellt. Der Verlaufsplan, bei dem zwischen Pflichteinbau und optionalem Einbau unterschieden wird, beginnt 2017 und soll bis zum Jahre 2032 abgeschlossen sein. Unter die optionalen Einbauten, fallen alle Letztverbraucher mit einem Stromverbrauch bis einschließlich $6.000 kWh$ pro Jahr und Anlagenbetreiber einer EEG- oder KWKG-Anlage mit einer installierten Leistung von 1 bis $7 kW$ [45].

In Kombination mit den anderen vernetzten Stromerzeugern, Speichern, elektrischen Verbrauchern und einem automatischen Last- und Ressourcenmanagement wird von einem Smart Grid gesprochen. Ein Smart Grid bietet den Konsumenten diverse Vorteile. Diese reichen von dem Wegfall der jährlichen Zählerablesung, bis hin zur Optimierung von Stromerzeugung und -handel für eine deutliche Reduzierung der Stromrechnung. Um die entstehenden Daten der Smart Meter und eines solchen vernetzten Smart Grids zugänglich zu machen, ist es von Nöten eine geeignete Benutzeroberfläche mit aussagekräftigen Diagrammen und Analysemöglichkeiten zu bieten, welche auch für Echtzeitdaten genutzt werden können. Dies soll im Folgenden beleuchtet werden.

3.1.3.1 User Centered Design (UCD)

In diesem Abschnitt wird beschrieben, wie bei der Erstellung darauf geachtet werden kann, dass ein Anwender einer grafischen Benutzeroberfläche seine Aufgaben bzw. sein Vorhaben zielführend, effektiv und mit einer gewissen Zufriedenstellung erledigen kann. Diese Eigenschaften werden zu-

[6] Smart Meter sind sogenannte intelligente Messsysteme (Zähler), die über ein Gateway verfügen, um Daten über das Internet an die Messstellenbetreiber zu übertragen.

[7] Online verfügbar: `https://www.bmwi.de/Redaktion/DE/Downloads/Gesetz/gesetz-zur-digitalisierung-der-energiewende.pdf?__blob=publicationFile&v=4`. Letzter Zugriff: 25.04.2017

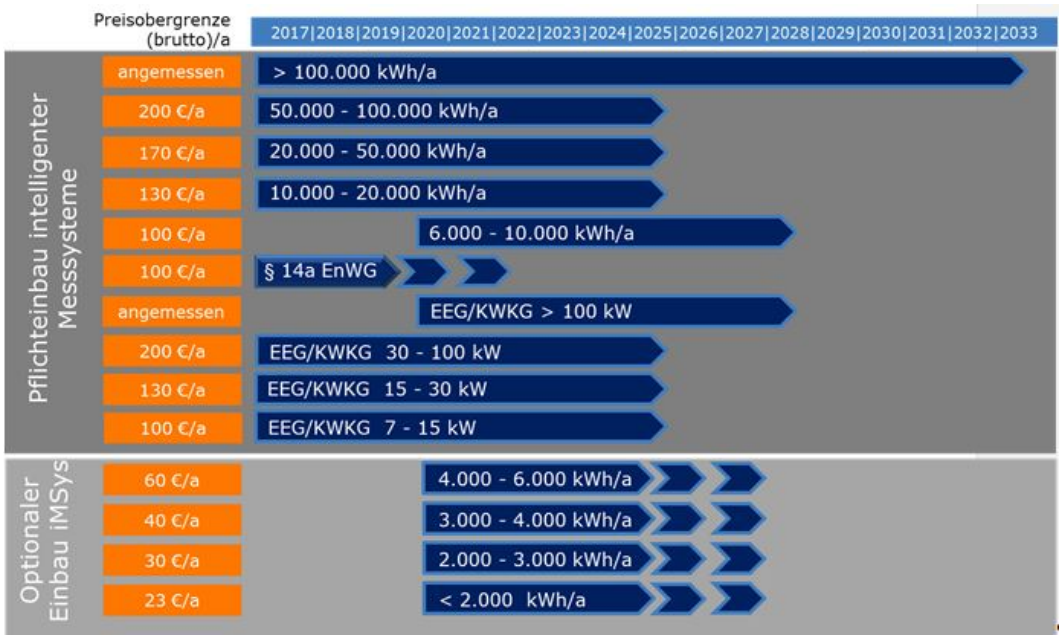


Abbildung 3.7: *Smart Meter Rollout-Verlaufsplan* [8]

sammengefasst als Usability[9] beschrieben. Für die Erstellung einer Web-Applikation sollten dabei bestimmte Richtlinien beachtet werden und Prozesse wie der UCD-Prozess zum Einsatz kommen.

UCD-Prozess

Bei dem UCD-Prozess wird versucht, dass die potentiellen Anwender der zu entwickelnden Benutzeroberfläche bis zu einem bestimmten Grad in den gesamten Entwicklungsprozess mit eingebunden werden. Der UCD-Prozess ist ein iterativer Vorgang, welcher aus vier Zwischenschritten besteht (vgl. Abbildung 3.8) und durch die nutzerzentrierte Evaluation verfeinert wird.

Specify Context of Use In diesem Schritt, welcher immer den Einstiegspunkt der Entwicklung auszeichnet, soll zunächst der Verwendungskontext festgestellt werden. Hierfür muss sichergestellt werden, wer die Stakeholder und Benutzer des Systems sind, welche Fertigkeiten, Erfahrungen, Bildung, etc. die Nutzer besitzen und wie sie normalerweise die auf das System zugeschnittenen Aufgaben erledigen.

Specify Requirements Während dieses Schrittes sollen zunächst die Bedürfnisse und Anforderungen der Benutzer erfasst werden. Hierfür gibt es diverse Strategien, welche von simpleren Surveys bis hin zu ethnografischen Studien reichen. Mit den erhobenen Benutzeranforderungen und dem vorher erfassten Verwendungskontext können dann die Anforderungen an das zu entwickelnde System abge-

[9] The extend to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use [251].

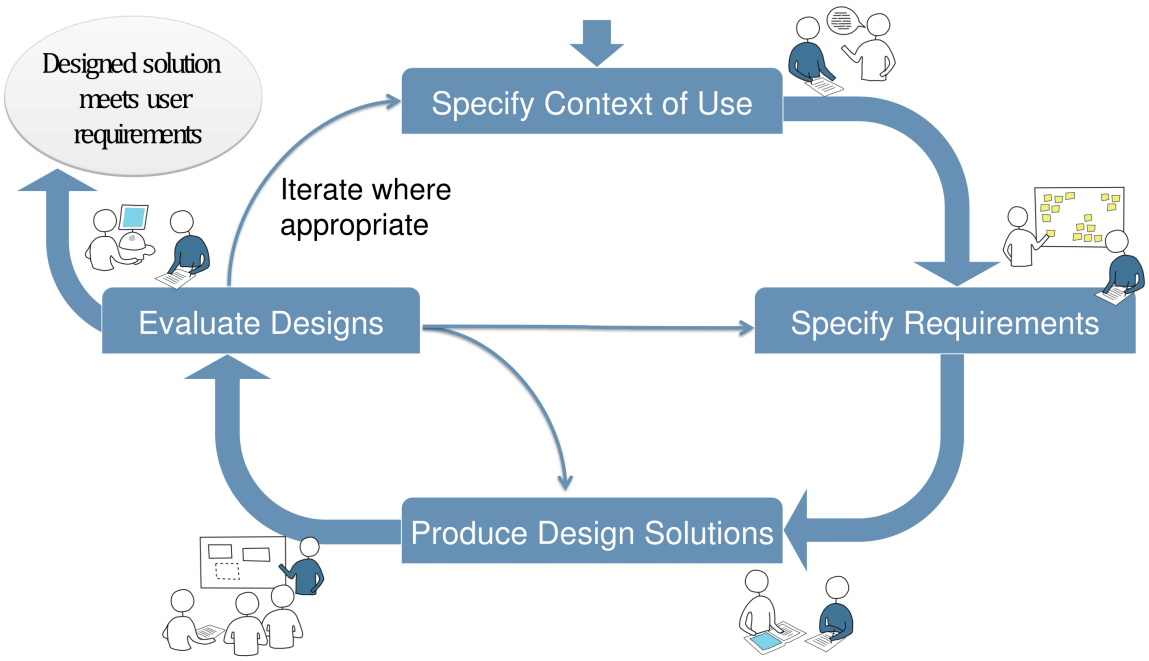


Abbildung 3.8: *User Centered Design Prozess* [10]

leitet werden. Dabei sollte bestehendes Wissen über Benutzerschnittstellen, Standards und Richtlinien, wie zum Beispiel die „Shneiderman's Eight Golden Rules of Interface Design"[11], beachtet werden.

Produce Design Solutions In diesem Schritt sollen die erstellten Anforderungen genutzt werden, um zu Beginn durch Kreativitätstechniken Lösungsvorschläge zu generieren. Sie sollten anfänglich horizontaler Natur sein (Lateral Thinking) und aus Low-Fidelity Prototypen bestehen. Im Bereich der Web-Anwendungen eignen sich hierfür Mockup Tools. Nach weiteren Iterationen des UCD-Prozesses und Evaluationen der vorherigen Designlösungen, sollen diese auf eine Lösung reduziert und dann detaillierter beschrieben werden (Vertical Thinking).

Evaluate Designs Im letzten Schritt einer UCD-Prozess Iteration wird der erstellte Prototyp bzw. das Produkt evaluiert. Dabei gibt es zwei Herangehensweisen. Die erste bezieht den Nutzer nicht mit ein. Dabei können Techniken wie die des Cognitive Walkthroughs zum Einsatz kommen, bei dem mehrere UI Experten das System evaluieren. Es kann außerdem eine heuristische Evaluation anhand der „Nielsen Rules" getätigt werden oder eine Model based Evaluation, wie GOMS oder KLM. Eine weitere Variante ist es den Nutzer einzubeziehen und qualitative oder quantitative Tests durchzuführen. Falls das erwünschte Ergebnis erreicht wurde, kann der UCD-Prozess verlassen werden. Ansonsten muss das System überarbeitet werden und dazu kann nach der Evaluationsphase zu jedem anderen Zwischenschritt gesprungen werden (siehe Abbildung 3.8).

[11] Shneidermans 8 goldenen Regeln lauten: 1. Konsistenz anstreben, 2. Abkürzungen (Shortcuts) für erfahrene Benutzer bereitstellen, 3. Informatives Feedback anbieten, 4. Design von Dialogen zur Verdeutlichung der Abgeschlossenheit, 5. Einfache Fehlerbehandlung anbieten, 6. Möglichkeit zur Stornierung anbieten, 7. Benutzerkontrolle und 8. Kurzzeitgedächtnis entlasten [242, vgl. Abschnitt 3.3.4 der 6. Edition]

3.1.3.2 Zusammenfassung

Hier wurde das User Centered Design und der damit einhergehende Prozess beschrieben. Bezogen auf einen Teil des Kontextes dieses Projekts, der Web Visualisierung von Smart Meter Daten, können die Benutzer im Bereich der Verteilnetzbetreiber und der Product Owner festgelegt werden. Die Aufgaben der Benutzer bestehen dabei aus der Überwachung und Analyse von Smart Meter Daten mithilfe der grafischen Benutzeroberfläche.

3.1.3.3 Web-Visualisierung

Um die erhaltenen Rohdaten der Smart Meter visualisieren zu können, muss zu Beginn betrachtet werden, welche Daten zur Verfügung stehen. Ein Smart Grid ist fähig im Bereich der Niederspannungversorgung, also auf Ebene der Endkonsumenten als Verbraucher und Kleinerzeuger, in einem bestimmten Zeitintervall durch eingebaute Smart Meter Pakete im COSEM Format zu senden. Ein Paket umfasst unter anderem Daten wie einen Zeitstempel, die Seriennummer des Smart Meters und den aktuellen Zählerstand. Das Intervall, in dem die Pakete versendet werden, ist dabei technisch gesehen variabel. Es ist aber standardmäßig im 15 Minuten Takt eingestellt. Außerdem liegt eine komplette Topologie des Niederspannungsnetzes mit den Seriennummern der Smart Meter sowie Verteilungsanlagen vor, welchen GPS-Daten zugeordnet wurden. Daraus lassen sich Anforderungen an die Bibliotheken zur Visualisierung aufstellen. Mit den zeitlich datierten Zählerstanddaten sollte es möglich sein, Time Series Charts bzw. Line Charts mit Annotationen zu erstellen. Dabei sollte darauf geachtet werden, dass es möglich ist, mehrere Datensätze einfügen zu können, um Realdaten von Prognosedaten trennen zu können. Da es sich bei den zeitlichen Daten auch um Datenströme handelt, sollten die Diagramme aktualisierbar sein und eine gewisse Performanz bieten. Mit der vorhandenen Topologie sollte es möglich sein, diese grafisch darstellen zu können, zum einen in Form einer geografischen Karte mit Hilfe von GPS Koordinaten und zum anderen als einfaches zusammenhängendes Netz. Somit können Störlokalisierungen betrieben und verdeutlicht oder auch Plausibilitätsprüfungen visualisiert werden. Da es sich bei diesen Daten jedoch teilweise um sensible haushaltsbezogene Daten handelt, wäre eine Kartentechnologie wie OpenStreetMap von Vorteil, welche auch ohne Internetverbindung betrieben werden kann. In diesem Abschnitt werden verschiedene JavaScript-Diagrammbibliotheken vorgestellt, welche für die Visualisierung von Rohdaten, der prognostizierten und den analysierten Daten existieren. Anschließend folgt eine Evaluation in Form einer Zusammenfassung, bei der passend zum Kontext eine Bewertung der JavaScript-Diagrammbibliotheken vorgenommen wird.

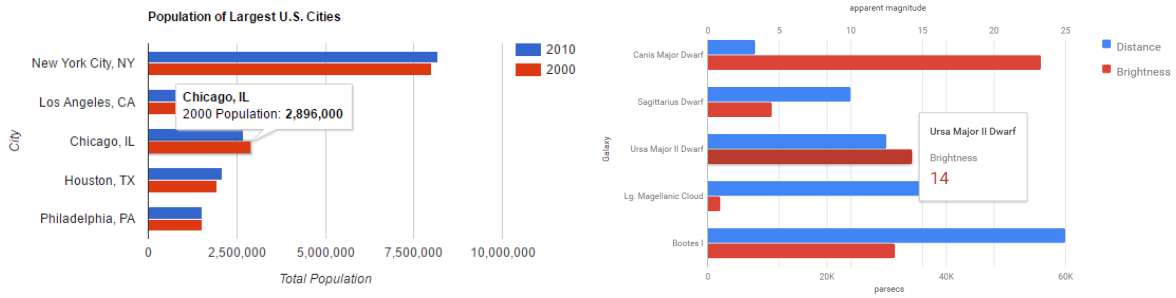
3.1.3.4 JavaScript-Diagrammbibliotheken

In diesem Abschnitt wird ein Auszug an Javascript-Diagrammbibliotheken vorgestellt, die für die Benutzung im Umfeld der Smart Meter Daten Visualisierung in Frage kämen. Die folgenden nennenswerten Bibliotheken, welche jedoch nicht in Betrachtung gezogen wurden, da sie kostenpflichtig sind, lauten: Plotly[12] und CanvasJS[13]

[12] Online: `https://plot.ly/`. Letzter Zugriff: 24.04.2017
[13] Online: `http://canvasjs.com/`. Letzter Zugriff: 24.04.2017

Google Charts

Google Charts[14] bietet eine Vielzahl an verschiedenen Diagrammtypen. Diagramme lassen sich dabei über die Daten in Form einer Google eigenen DataTable und eines Optionsparameters in Form einer SVG oder VML rendern. Durch den Optionsparameter ist es außerdem möglich, die Diagramme an das Material UI anzupassen (Siehe Abbildung 3.9). Es existiert jedoch eine lange Liste an Optionen, die bisher nur im klassischen Stil umgesetzt wurden[15].



(a) *Standard Design* (b) *Material Design*

Abbildung 3.9: *Unterschied der Google Chart Stile* [16]

Für die Visualisierung von Smart Meter Daten über einen zeitlichen Verlauf bietet Google Charts einfache Line Charts, komplexere Annotation Charts, Area Charts oder auch Calendar Charts an (siehe Abbildung 3.10).

Um topologische Visualisierungen zu erstellen, bietet Google Charts die Möglichkeit Google Maps Karten stilisiert darzustellen und mit Markierungen zu versehen (Siehe Abbildung 3.11). Hierfür können GeoMaps, Maps, IntensityMaps oder GeoCharts verwendet werden, je nachdem welcher Detaillierungsgrad und Stil dargestellt werden soll.

Es ist auch möglich eigene Chart Typen zu erstellen[17] jedoch erfolgt die komplette Implementierung manuell und es existieren keine Hilfsklassen.

ChartJS

ChartJS steht derzeit in Version 2.5 zur Verfügung und besteht aus acht vorgefertigten responsiven Chart Typen, welche auf einen HTML Canvas gezeichnet werden. Die vorgefertigten Chart Typen sind Line, Bar, Radar, Polar Area, Pie / Doughnut, Bubble und Scatter Charts. Die Line Charts können gleichzeitig durch Stilisierung als Area Chart verwendet werden und sehen den Google Charts sehr ähnlich (Siehe Abbildung 3.10a). Seit Version 2.0 ist es auch möglich eigene Chart Typen zu erstellen und bestehende zu erweitern. Dabei existiert für jeden Chart Typ eine Update Funktion, um

[14] Online: `https://developers.google.com/chart/`. Letzter Zugriff: 24.04.2017

[15] Fehlende Option Features für das Material Design. Online: `https://github.com/google/google-visualization-issues/issues/2143`. Letzter Zugriff: 24.04.2017

[17] Dokumentation zum Thema „Creating Chart Types“. Online: `https://developers.google.com/chart/interactive/docs/dev/`. Letzter Zugriff: 25.04.2017

(a) *Line Chart* (b) *Annotation Chart* (c) *Area Chart*

(d) *Calendar Chart*

Abbildung 3.10: *Charts für Zeitreihenanalyse*

Abbildung 3.11: *GeoChart mit Markierungen*

zwischen den alten und den neuen Daten eine Animation durchzuführen. ChartJS bietet jedoch keine topologischen bzw. kartenbasierten Visualisierungsmöglichkeiten.

Smoothie Charts

Smoothie Charts[18], aktuell in Version 1.28, ist eine sehr kleine[19] Bibliothek, die dazu dient, Echtzeitdaten in Form von Streams o.ä. schnell visualisieren zu können ohne das Diagramm komplett aktualisieren zu müssen. Jedoch existiert durch die Größe und den Zweck der Bibliothek nur ein einziges Design (Siehe Abbildung 3.12).

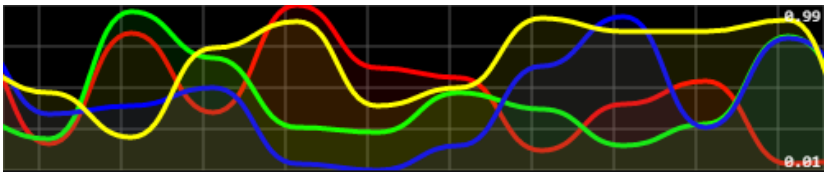


Abbildung 3.12: *Smoothie Chart Beispiel*

D3

D3 (Data-Driven Documents)[20] steht derzeit in Version 4.8.0 zur Verfügung. D3 beinhaltet keine vorgefertigten Diagrammtypen, sondern bietet die Möglichkeit durch deklarative Programmierung Boilerplate Code zu minimieren und dabei alle Funktionalitäten, die einem auch die standardmäßige W3C DOM API bietet, zu gewährleisten und somit jegliche Art von Diagramm zu kreieren (Siehe Abbildung 3.13).

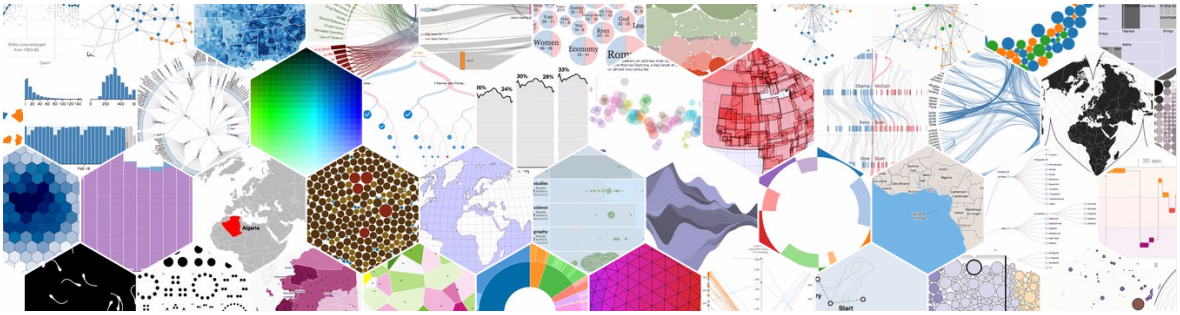


Abbildung 3.13: *Ausschnitt der vorgestellten Beispielcharts auf der Hauptseite von D3*

D3 hat 63.624 Stars und 16.733 Forks auf GitHub (Stand: 25.04.2017) und hat somit die größte Community aller JavaScript-Diagrammbibliotheken. Dadurch existieren eine Vielzahl an Beispielen und Hilfestellungen[21] und Wrapperbibliotheken, die es ermöglichen, standardmäßige Diagrammty-

[18] Online: `http://smoothiecharts.org`. Letzter Zugriff: 25.04.2017

[19] Version 1.28 mit Googles Closure Compiler auf Simpler Einstellung verkleinert auf 2.74KB gzipped und 8.48KB uncompressed.

[20] Online: `https://d3js.org/`. Letzter Zugriff: 25.04.2017

[21] Galerien für D3 Graphen: `https://github.com/d3/d3/wiki/Gallery` und `https://bl.ocks.org/mbostock`.

pen basierend auf D3 zu erstellen. Nennenswerte Wrapperbibliotheken sind C3[22], d3fc[23], NVD3[24], Cubism.js[25], MetricsGraphics.js[26] und Rickshaw[27]. Mit d3fc, Cubism.js und Rickshaw ist es explizit möglich, Echtzeitdaten effizient darzustellen. Für d3fc existiert ein nicht interaktives Beispiel einer Streaming Financial Chart, bei der dynamisch Daten nachgeladen werden und das Diagramm sich aktualisiert (Siehe Abbildung 3.14).

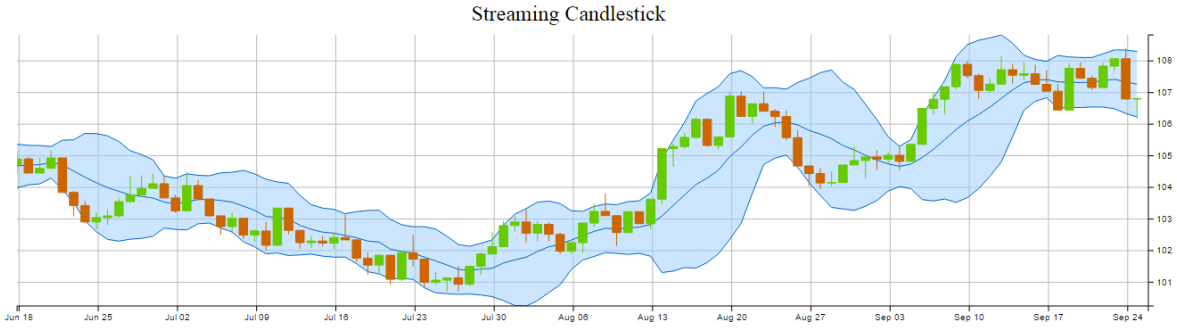


Abbildung 3.14: *Screenshot eines Streaming Financial Chart Beispiels von d3fc* [28]

Cubism.js dagegen bietet eine einzige Art von interaktiver Area Chart, die speziell für die Darstellung von zeitlich basierten Daten genutzt werden kann (Siehe Abbildung 3.15).

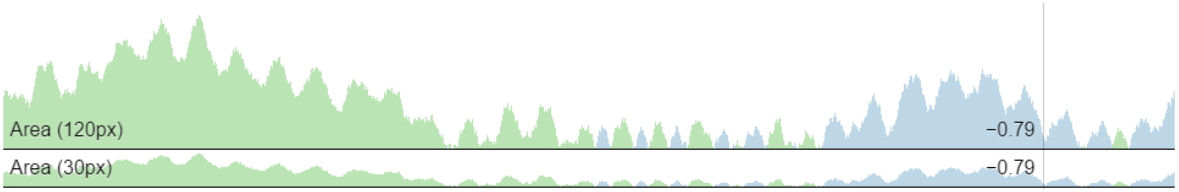


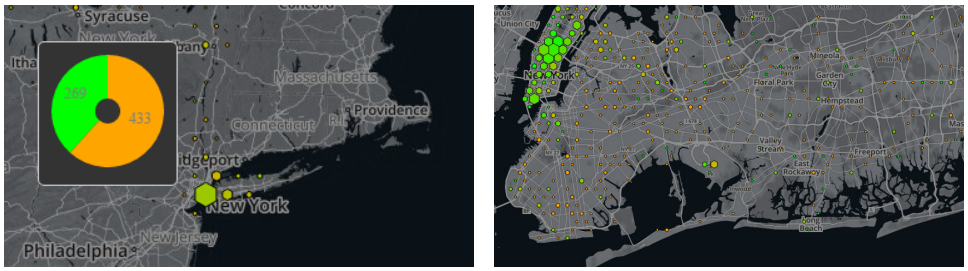
Abbildung 3.15: *Screenshot einer Time Series Visualization Chart von Cubism.js* [29]

D3 bietet gegenüber Google Charts die Möglichkeit, neue Datensätze in die aktuelle Visualisierung zu integrieren, ohne ein komplettes Rerendering aller Daten durchzuführen. Dies führt zu besserer Performanz bei Datenupdates. Für das Aktualisieren von Daten existiert auch ein General Update Pattern[30].

[22] Online: `http://c3js.org`. Letzter Zugriff 25.04.2017
[23] Online: `https://d3fc.io`. Letzter Zugriff 25.04.2017
[24] Online: `http://nvd3.org`. Letzter Zugriff 25.04.2017
[25] Online: `http://square.github.io/cubism/`. Letzter Zugriff 25.04.2017
[26] Online: `https://www.metricsgraphicsjs.org`. Letzter Zugriff 25.04.2017
[27] Online: `http://code.shutterstock.com/rickshaw/`. Letzter Zugriff 25.04.2017
[30] D3 General Update Pattern, Part 3. Mike Bostock, Online: `https://bl.ocks.org/mbostock/3808234`. Letzter Zugriff: 30.04.2017

Leaflet

Leaflet[31] steht derzeit in Version 1.0.3 zur Verfügung. Es ist eine sehr klein gehaltene Bibliothek zum Darstellen von OpenStreetMaps. Dabei ist es möglich jegliche Art der Stilisierung vorzunehmen, die Karte mit eigenen Layern zu überlagern und auch mit der Map zu interagieren. Die Interaktionen reichen von einfachen Klick Events bis hin zu Mobile-Multi-Touch-Zoom Events. Alle normal gewohnten Interaktionsanwendungen von OpenStreetMaps selbst sind auch möglich. Leaflet besitzt eine ausgezeichnete Dokumentation mit Verwendungsbeispielen und der kompletten Referenz[32]. Leaflet könnte in Kombination mit D3, wie in Abbildung 3.16 zu sehen, dazu genutzt werden, Störallokation des Niederspannungsnetzes zu betreiben.



(a) *Zoomstufe 6 mit Mouse Hover* (b) *Zoomstufe 11*

Abbildung 3.16: *Hexbins with D3 and Leaflet Maps* [33]

3.1.3.5 Zusammenfassung

Zusammenfassend lässt sich sagen, dass Google Charts im Bereich der Datenvisualisierung für die Darstellung von Smart Meter Daten eine Vielzahl an schon existierenden Diagrammen bereitstellt. Bei den geografischen Karten ist dies hingegen problematischer, da hierfür Google Maps verwendet wird, welches wie auch die Google Charts allgemein nicht Offline betrieben werden können, da die Nutzungsbedingungen es untersagen[34]. Dieses Problem löst Leaflet, welches mit OpenStreetMaps arbeitet. Außerdem ist eine Kopplung von Leaflet und D3, wie in Abbildung 3.16 zu sehen ist, unproblematisch. D3 bietet mit den existierenden Wrapperbibliotheken für alle Standard Charts und den Wrapperbibliotheken für Echtzeitdaten alle Voraussetzungen, um als Visualisierungsgrundlage für Smart Meter eingesetzt zu werden. Außerdem können mit Google Charts keine topologischen Zusammenhänge außerhalb des geografischen Kontextes dargestellt werden. Diese Aufgabe kann hingegen D3 mit dem Force-Directed Graph Layout übernehmen. ChartJS bietet viel Potential im Bereich der einfachen Datenvisualisierung, jedoch existiert keine Möglichkeit geografische bzw. toplogische Zusammenhänge zu visualisieren. Eine Anbindung an Leaflet wie mit D3 ist derzeit nicht möglich. Falls darüber hinaus eine Visualisierungsmöglichkeit benötigt wird, um im Millisekundentakt Datenpakete in Echtzeit zu visualisieren, wäre es möglich auf Smoothie Charts zurückzugreifen.

[31] Online: `http://leafletjs.com`. Letzter Zugriff: 25.04.2017

[32] Dokumentation für Leaflets Version 1.0.3. Online: `http://leafletjs.com/reference-1.0.3.html`. Letzter Zugriff: 25.04.2017

[34] Vgl. Abschnitt „Can I use charts offline?“ Online: `https://developers.google.com/chart/interactive/faq`. Letzter Zugriff: 30.04.2017

3.1.3.6 Fazit

In diesem Abschnitt wurde zunächst die Motivation erläutert, wieso das Thema der Smart Meter Visualisierung aktuell immer mehr an Bedeutung gewinnt. Danach wurde mit dem User Centered Design und dem einhergehenden Prozess aufgezeigt, was bei der Visualisierung einer grafischen Benutzeroberfläche zu beachten ist. Daran anschließend wurden diverse JavaScript-Diagrammbibliotheken und deren Features und Möglichkeiten zur Visualisierung von Smart Meter Daten vorgestellt. Wie bereits im Unterunterabschnitt 3.1.3.5 zusammengefasst, sollte für die Visualisierung der Smart Meter Daten D3 und die dazu passenden Wrapperbibliotheken wie d3fc verwendet werden, um die Diagramme zu erstellen. Leaflet in Kombination mit D3 sollte dafür verwendet werden, um topologische und geografische Zusammenhänge zu visualisieren.

3.2 Schnittstellenprotokolle

3.2.1 Common Information Model (CIM)

Im Zuge der Deregulierung des Energiemarktes bestand mehr denn je der Bedarf nach dem Austausch von Informationen zwischen den Akteuren in der Energiewirtschaft. Da aber Unternehmen nicht zwangsweise das gleiche Protokoll zum Datenaustausch benutzen, noch die Daten auf gleiche Weise verarbeiten, kommen Probleme zwischen Systemen auf, wenn diese nun zusammen interagieren müssen. Wenn z.B. unterschiedliche Services untereinander kommunizieren müssen, muss die Schnittstelle für die Kommunikation definiert werden, damit die Services die gleichen Objekte gleich bearbeiten können.

Deshalb begann in den 1990er Jahren das Electric Power Research Institute (ERPI) mit der Ausarbeitung eines offenen Standards, des CIMs, welches jetzt bei der International Electrotechnical Commission (IEC) untergebracht ist und kontinuierlich weiterentwickelt wird [193].

Das Ziel des CIMs ist es ein technologieunabhängiges Modell zu gestalten, dass die Modellierung sowohl von Stromversorgungssystemen, dem Energiemarkt selbst als auch die Kommunikation zwischen den einzelnen Entitäten auf dem Energiemarkt abdeckt.

Die vorliegende Arbeit gibt eine erste Einführung zu CIM, um so ein Verständnis für die Komplexität und die Möglichkeiten, die es bietet, zu schaffen. Dabei wird ein Überblick über die Module und Standards gegeben und anschließend wird gezeigt, wie man durch das Erstellen eines CIM-Profils einen gewünschten Anwendungsfall modellieren kann und anschließend die Daten für diesen Anwendungsfall in einem oder zwischen mehreren bestehenden Systemen mittels verschiedener Technologien für den Datenaustausch nutzen kann.

3.2.1.1 Das Common Information Model

Wie schon in der Einleitung erwähnt, kann das Common Information Model der IEC unter anderem für einen standardisierten Datenaustausch bezüglich des Stromnetzes im Energiemarktbereich, einer Modellierung der Netztopologie und zur Integration von Energieinformationssystemen benutzt werden. Dabei ist das CIM nicht ein einzelnes Dokument oder ein einzelner Standard, sondern es ist vielmehr eine Sammlung von mehreren IEC Standards. Die drei Standardsammlungen, die das CIM ausmachen sind das IEC 61970, das IEC 61968 und das IEC 62325.

Die Standards in IEC 61970 befassen sich mit den Komponenten der Elektroversorgung. Einer der Kernpunkte im 61970 Standard ist die Norm 61970-301, die ein unabhängiges Datenmodell liefert, dass die Beziehungen der Geräte untereinander im Netz und die Komponenten an sich beschreibt. Das IEC 61970 behandelt auch den Aufbau von Energiemanagementsystemen (EMS) und deren APIs. Weiterhin beinhaltet diese Reihe an Standards die Common Interface Specification (CIS) und die Generic Interface Defintion (GID). Eine Übersicht der Pakete des Datenmodells für das IEC 61970-301, was die Grundlage des CIM-Datenmodells darstellt ist in Abbildung 3.17 zu sehen.

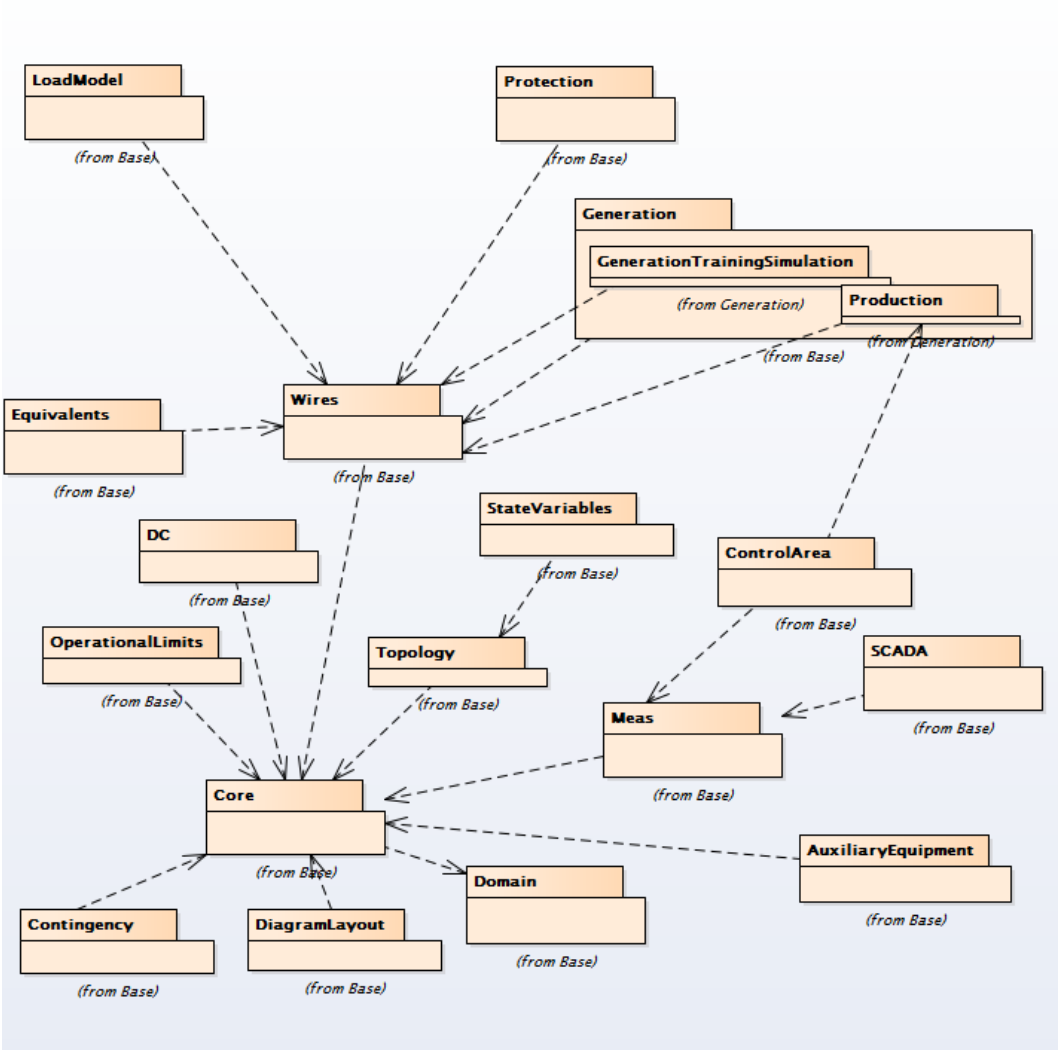


Abbildung 3.17: *Paketübersicht des und Abhängigkeiten IEC 61970-301 aus CIM 16*

Die Standardsammlung IEC 61968 wurde entwickelt, um das IEC 61970 zu erweitern und den Datenaustausch bei Verteilnetzbetreibern zu modellieren. Das IEC 61968-11 Datenmodell hat das CIM mit Klassen erweitert, um den Datenaustausch zwischen Verteilnetzbetrieben zu vereinheitlichen. Wie man in Abbildung 3.18 sehen kann, haben die Pakete des Datenmodells einen Schwerpunkt auf Verteilung, Zählung, Kunden und auch Klassen zur Erweiterung des Assetmanagements. Außerdem

sieht man in Abbildung 3.18 die Abhängigkeiten zu dem Datenmodell IEC 61970-301, gekennzeichnet durch die Beschriftung *(from Base)*.

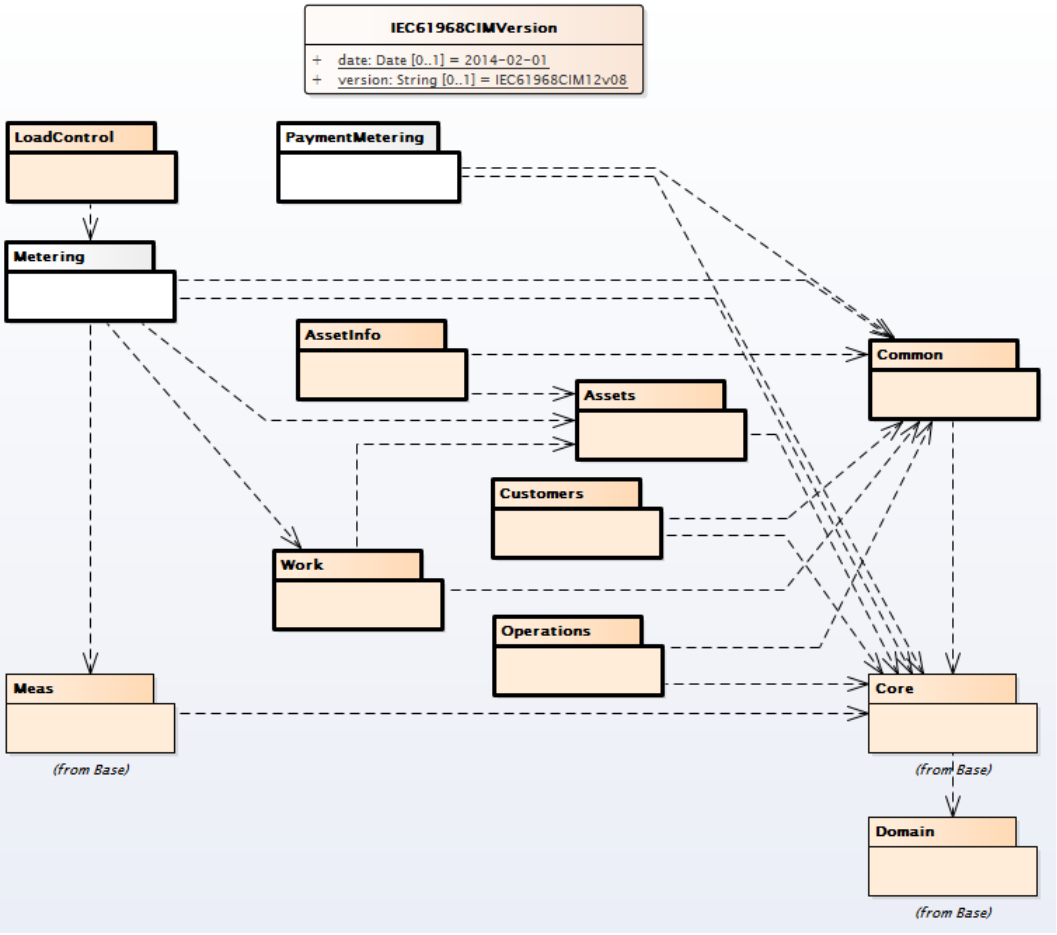


Abbildung 3.18: *Paketübersicht und Abhängigkeiten des IEC 61968-11 aus CIM 16*

Die dritte Standardreihe, IEC 62325, wurde entwickelt, um wiederum die beiden vorher genannten Standardreihen zu erweitern. Sie beinhaltet eine Menge an Standards, die die Kommunikation am Energiemarkt zwischen den Akteuren beschreibt.

Im IEC 62325-301 Datenmodell gibt es drei Pakete, die Erweiterungen für die bisherigen Pakete sind und in Abbildung 3.19 abgebildet sind. Die Pakete *MarketCommon*, *MarketOperations* und *MarketManagement* behandeln den Energiemarkt. Mit *MarketManagement* ist es möglich ein Profil des Energiemarkts zu erzeugen (siehe CIM Profile) und *MarketOperations* kann ein nordamerikanischer oder europäischer Energiemarkt modelliert werden [254].

3.2.1.2 Common Information Model UML

Die CIM-Datenmodelle sind objekt-orientierte Informationsmodelle, die in *Unified Modelling Language* (UML) modelliert sind. Zwar gibt es in dem CIM-Datenmodell keine gemeinsame Oberklasse

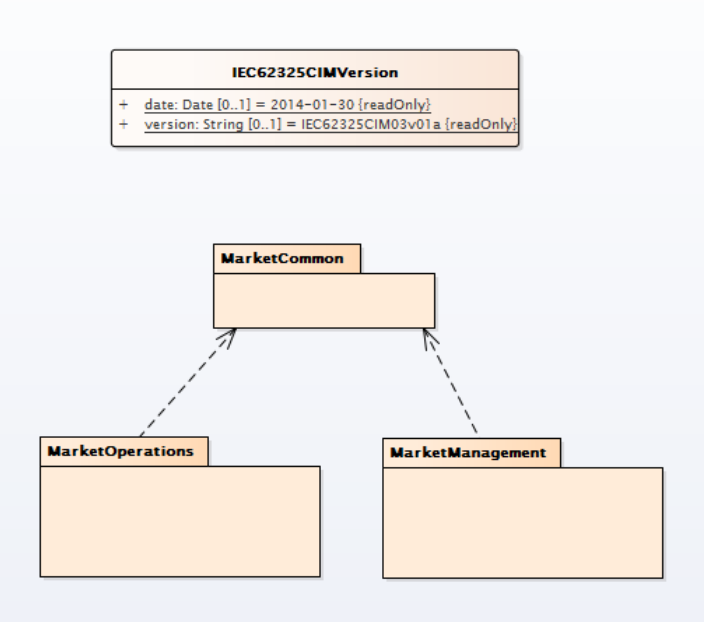


Abbildung 3.19: *Paketübersicht und Abhängigkeiten des IEC 62325-301 aus CIM 16*

von der ausschließlich alle anderen Klassen abgeleitet sein müssen. Jedoch gibt es die Klasse *IdentifiedObject* die als eine mögliche Oberklasse für weitere Vererbung benutzt werden kann. Dadurch haben alle vom *IdentifiedObject* abgeleiteten Klassen die Attribute *name* und *mRID* (Master Resource Identifier) [245]. Von dem *IdentifiedObject* wird nun z. B. eine *PowerSystemResource* abgeleitet und von dieser weiterhin weitere Klassen, bis man eine Klasse für einen konkreten Gegenstand, den man im EMS modellieren möchte, erhält. Somit bekommt man eine Klassenhierarchie, die die notwendigen Attribute beinhaltet und die durch Assoziationen zu anderen Klassen um weitere Attribute erweitert werden kann.

Ein Vorteil ist, den diese Klassenhierarchie bringt, dass Systeme mit einer generischen und von *PowerSystemResource* abgeleiteten Klasse arbeiten können ohne sich dabei auf spezifische Klassen festlegen zu müssen. Zum Beispiel können so beliebige Schalter nach ihrem Zustandsattribut *normalOpen* durchgesucht werden, ohne dass das System von der genauen Art des Schalters wissen muss. Es reicht aus, dass sie eine gemeinsame Oberklasse des CIM haben, die dem System bekannt ist und es so gar nicht mehr die genaue Art des Schalters wissen muss.

Durch die UML-Modellierung entsteht im CIM ein weiterer Vorteil und zwar dass klar zwischen der Functional- und der Asset-Ebene unterschieden werden kann. Das geschieht dadurch, dass Ressourcen auf der Klassenebene der *PowerSystemResource* eine Assoziation zu einer *Asset* Klasse bekommen. Objekte dieser *Asset* Klassen beinhalten Attribute, die für die physikalischen Gegenstände wichtig sind, wie z. B. die Identifikationsnummer dieses Gegenstands, die für die Funktionalität im gesamten System dagegen keine Bedeutung tragen.

Somit wird im CIM ein Objekt sowohl auf der *Functional-* als auch auf der *Asset*-Ebene dargestellt, allerdings spielen dort unterschiedliche Attribute eine Rolle. Auf beiden Ebenen werden zum Beispiel dem Objekt der *PowerSystemResource* eine Identifikationsnummer zugeteilt, allerdings kann die

Identifikationsnummer auf der *Functional*-Ebene die mRID sein und auf der *Asset*-Ebene zusätzlich noch eine Seriennummer des Gegenstands sein.

Durch die Assoziation im Datenmodell des CIM können Wartungsarbeiten an den Gegenständen im Feld durchgeführt werden, z. B. der Austausch von Komponenten oder des gesamten Gegenstandes, ohne dass die Objekte in der Modellierung des Netzes, die von der *PowerSystemResource* Klasse abgeleitet werden, verändert werden müssen, weil im Netz physikalisch eine neue Komponente dazugekommen ist. Es muss lediglich die Relation zu einem neuen Objekt der *Asset* Klasse eingefügt werden, die z. B. als Attribut die aktuelle Seriennummer des Geräts hat. Somit umgeht man das Problem, dass bei dem Austausch einer einzelnen physikalischen Komponente im Feld die Modellierung unnötig überarbeitet werden muss [245].

Außerdem liegt nun ebenfalls bei der Kommunikation eine einheitliche Sprache mit einer einheitlichen Semantik vor. Wenn das CIM-Datenmodell zu Grunde gelegt wird, kann ein konkreter Gegenstand mit einer konkreten Klasse aus dem Datenmodell modelliert werden. Wenn zum Beispiel eine Klasse eine Aggregation aus diversen verschiedenen Klassen ist, ist dieses eindeutig aus dem Datenmodell sichtbar. Durch das Datenmodell und den Beschreibungen in dem Datenmodell schafft man so interdisziplinär eine eindeutige Semantik. Sollten so zwischen zwei verschiedenen Systemen Daten über ein Objekt der Klasse *Switch* zusenden, dann kann davon ausgegangen werden, dass beide das gleiche Konzept eines Schalters verstehen.

3.2.1.3 CIM-Profile

Da das gesamte Datenmodell von CIM komplex und unübersichtlich ausfällt, ist es wünschenswert die Komplexität des CIM reduzieren zu können. Alleine wenn man das Datenmodell in CIM Version 16 betrachtet, gibt es ca. 140 Pakete und mehr als 1000 Klassen mit über 7000 Attributen. Mit neueren CIM-Versionen wird sich diese Zahl vergrößern und genau dafür gibt es CIM-Profile, die nur eine kleine Untermenge des CIM darstellen. Mit den Profilen kann man das CIM auf nur für einen bestimmten Anwendungsfall benötigte Klassen und deren Assoziationen beschränken und muss so nicht mit dem gesamten Datenmodell arbeiten. Welche Klassen für ein Profil genommen werden ist dabei kontextabhängig, das heißt bei einem Profil werden nur die Klassen genommen, die genau jenen Anwendungsfall beschreiben.

Durch das Verwenden von Profilen gibt es einen weiteren Vorteil, der mit dem gesamten CIM Datenmodell nicht umgesetzt werden könnte. Man kann Daten, die von CIM-Profilen abgeleitet werden, auf Interoperabilität und auf Profil-Konformität testen. Da das CIM ein generisches Modell ist, kann und sollte man diese Tests nicht mit dem gesamten Datenmodell durchführen.

Bei den Profilen gibt es neben schon vorgegeben Profilen, die häufige Anwendungsfälle repräsentieren auch die Möglichkeit eigene Profile zu erstellen. Zwei Beispiele für vorgegebene Profile wären *The Common Distribution Power System Model* (CDPSM) was Verteilungsnetzwerke im europäischen Raum modelliert und das *ENTSO-E* Profil, das Datenaustausch zwischen Übertragungsnetzbetreiber in Europa eingesetzt werden kann [179, 85].

Wenn diese Profile nicht den gewünschten Anwendungsfall abdecken oder zu groß sind und man ein eigenes CIM-Profil erstellen möchte, gibt es, wie in Abbildung 3.20 zu sehen, eine vorgeschlagene Reihenfolge an Schritten, die eingehalten werden sollten, um ein verwendbares CIM-Profil zu erstellen.

Zunächst muss man sich auf eine CIM-Version einigen, die als Grundlage für das Profil dienen soll. Ist eine grundlegende CIM-Version beschlossen, muss der Anwendungsfall, für den das Profil erstellt wird, betrachtet werden. Da es keinen Anwendungsfall geben wird, der das gesamte CIM benötigt, müssen nur die Klassen für diesen betrachteten relevanten Fall aus dem CIM herausgesucht werden.

Sind die Klassen für den Anwendungsfall herausgesucht, müssen die etwaigen Assoziationen zwischen den Instanzen der Klassen bestimmt werden. Außerdem muss identifiziert werden, welche Attribute der Klassen benötigt werden und welche optional sein sollten. Reichen die vorgegebenen Attribute des CIMs nicht aus, können auch neue Attribute und Klassen angelegt und das CIM somit erweitert werden. Sind diese Schritte abgeschlossen kann das erstellte Profil getestet und veröffentlicht werden [254].

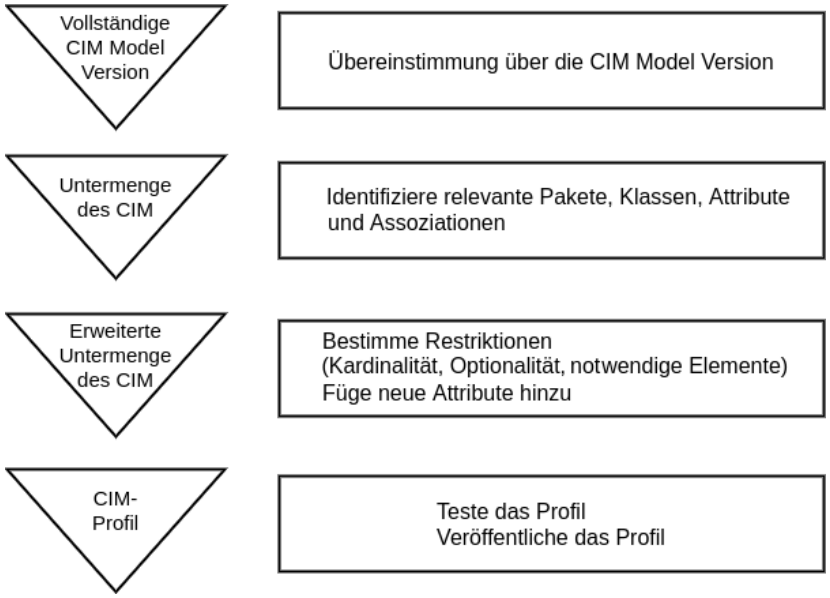


Abbildung 3.20: *Vorgehensweise beim Erstellen von CIM-Profilen [254]*

Praktisch sollte der Prozess ab der Einigung auf eine bestimmte CIM-Version unterstützt durch Tools geschehen. Ein solches Tool, das für die Erstellung von Profilen geeignet ist, ist das kommerzielle Tool *Enterprise Architect* (EA) [84]. Innerhalb des EA ist es beispielsweise möglich aus dem gesamten Datenmodell nur die benötigten Klassen und die gewünschten Attribute auszuwählen. Danach können Assoziationen zwischen den Klassen bestimmt und die Kardinalität festgelegt werden. So erstellt man sich das gewünschte Profil, was anschließend mit der Funktion des *Schema Composer* in diversen Formaten exportiert werden kann. Im EA wird unter anderem ein Export nach RDFS, XSD und JSON unterstützt. Somit erstellt man das Schema des Profils, an die sich spätere Daten halten sollen und gegen das die Daten getestet und validiert werden können.

3.2.1.4 Datenaustausch mit CIM

Informationen in einem Energieinformationssystem können auf unterschiedlichste Weise verarbeitet werden, z. B. kann ein Dienst in dem System, das für die Rechnungen zuständig ist, einen Kunden mit anderen Attributen modellieren, als ein Dienst, der für die Kundenbetreuung zuständig ist. Auch

wenn diese Informationen den gleichen Kunden auf ähnliche Weise beschreiben würden und bei den gespeicherten Informationen Überlappungen vorhanden wären, würde schon ein reiner Vergleich der Daten schwer fallen.

Das muss nicht nur bei dem bloßen Schema der Daten bleiben. Gibt es viele verschiedene Dienste innerhalb eines Unternehmens, die miteinander kommunizieren, müssen diese Dienste Schnittstellen haben, damit die gegenseitige Kommunikation funktioniert. Eine Lösung dafür ist es einen *Enterprise Service Bus* (ESB) zu benutzen, der als Schnittstelle zwischen den einzelnen Diensten dient. Genauer gesagt, können die Daten der Dienste dem CIM-Datenmodell zugeordnet werden und über den ESB an weitere Dienste ausgegeben werden. Somit wäre die Integration zwischen den Diensten gewährleistet.

Damit das funktioniert, können die Daten mit *Extensible Markup Language (XML)* und Resource Description Framework (RDF) serialisiert werden. Die Serialisierung der CIM-Daten nach XML und RDF hat den Vorteil, dass sie sowohl von Maschinen als auch von Menschen lesbar sind und sie auf keinem proprietären Format beruhen, so dass jedes System Nachrichten in den Formaten verarbeiten kann. Das bringt allerdings den Nachteil mit sich, dass die Daten potenziell größer ausfallen, die als XML verschickt werden, da sie mehr Metainformationen haben als Daten, die als *plain text* verschickt werden.

XML Nachrichtenaustausch

Im IEC 61968 wird ein XML-Schema für eine Nachricht empfohlen, das aus einem *Header* und optional *Request*, *Reply* und *Payload* besteht. Ein *Header* besteht wiederum, genau wie die restlichen Bestandteile des Schemas, aus mehreren Attributen. Zu den nicht optionalen Attributen gehören *Verb* und *Noun*. Das *Verb* spezifiziert die auszuführende Aktion, wie z. B. *CREATE, UPDATE, DELTE, GET*, etc. Das *Noun* identifiziert das Subjekt der Aktion. Weitere nicht optionale Attribute sind z. B. *Revision* der Nachricht, ein *Timestamp* oder auch ein *Source* Attribut, dass die Quelle der Nachricht angibt.

Ein *Request* besteht aus *startTime* und *endTime*, das die Zeitdauer für den *Request* angibt. Weitere Attribute sind *Option*, *ID* und *any ##other*, welche benutzt werden können um ein *Request* an individuelle Anforderungen anzupassen.

Nachrichtenteile vom Typen *Reply* haben einen *ReplyCode*, keinen oder mehrere *Error*, *ID* und *any ##other*.

Der letzte Bestandteil der Nachricht, der noch fehlt, ist der *Payload*. Der *Payload* beinhaltet die Nachrichteninformation und besteht aus einem *format* Attribut und einem *any ##other* Element. Das *any ##other* Element ist üblicherweise ein XML-Element, dass aus dem CIM-Datenmodell abgeleitet werden kann. Sollte die Nachricht mit einem Kompressionsverfahren komprimiert werden, wird die Kompression ebenfalls im *Payload* angegeben.

Aus diesen Nachrichtenteiltypen ist es möglich vier Stereotypen von Nachrichtentypen zu erstellen und zwar *RequestMessage*, *ResponseMessage*, *EventMessage* und *FaultMessage* [254]. Die ersten beiden Typen von Nachrichten, *RequestMessage* und *ResponseMessage* haben neben einem *Header* und *Payload* ein *Request*, bzw. ein *Reply* XML-Element. Wohingegen eine Nachricht des Typs *EventMessage* aus einem *Header* und der *Payload* besteht und Nachrichten des Typs *FaultMessage* aus einem *Reply* XML-Element [254].

Beschreibung von Netztopologien

Bis jetzt lag in dieser Arbeit der Schwerpunkt überwiegend auf dem Datenaustausch zwischen Systemen. Jedoch ist es auch möglich mit dem CIM die Topologie eines Netzes zu beschreiben. Dabei kann das Resource Description Framework (RDF) benutzt werden, was ein Modell zum Datenaustausch im Internet ist. Mit dem RDF lassen sich Relationen zwischen Objekten darstellen. Dabei benutzt man Tripel, um die Relationen zwischen Objekten, bzw. Knoten einem Graphen zu beschreiben. Ein Tripel besteht aus einem Subjekt, einem Prädikat und einem Objekt. Das Prädikat ist die Verbindung von Subjekt zum Objekt [227]. Zur Serialisierung kann, wie schon weiter oben bei dem Nachrichtenaustausch, XML benutzt werden.

Mittels der RDF/XML Syntax können so die modellierten Graphen zwischen Systemen ausgetauscht und verarbeitet werden [228]. Das gelingt, da man mit RDF/XML wie bei der UML-Modellierung objektorientiert vorgehen und mittels des RDF-Schemas (RDFS) Klassen definieren kann. Ebenfalls in dem RDFS können die Relationen zwischen den Klassen definiert werden. Damit kann das CIM UML-Datenmodell in ein RDFS konvertiert und von diesem können RDF-Instanzen, die konkrete Objekte in einem EMS beschreiben, abgeleitet werden.

Im IEC 61970-501 und IEC 61970-552 Standard wird die Übersetzung des CIM-Datenmodells in ein RDF-Schema festgelegt und wie aus diesem Schema XML-Instanzen abgeleitet werden können, um Energieerzeuger zu modellieren, beschrieben. Da dieses RDFS eine Untermenge der RDF/XML Spezifikation ist, können vorhandene Tools, die in der Lage sind RDF/XML zu verarbeiten, ebenfalls das RDFS aus 61970-501 verarbeiten [245]. Dadurch erhält man ein Datenaustauschformat, dass von Maschinen und Menschen gelesen werden kann und das CIM UML-Datenmodell repräsentiert.

Unter Verwendung von RDF/XML können die so erstellten Daten leicht unter Verwendung von Namensräumen erweitert und angepasst werden. Die Dateien können mit einer beliebigen Kompression archiviert und es können Differenz-Modelle erstellt werden, so dass nur die Änderungen zwischen zwei Modellen oder zwei Versionen eines Modells sichtbar sind [245].

3.2.1.5 Fazit

Mit dem *Common Information Model* ist eine Reihe an Standards erstellt worden, die es ermöglicht den Markt und die Infrastruktur der Energiewirtschaft zu beschreiben. Darüber hinaus kann man die Standards benutzen, um die Integration von Systemen im Energiesektor zu gewährleisten. Dafür wurde in diesem Abschnitt beschrieben, wie man vom allgemeinen CIM-Datenmodell hin zu einem Profil kommt, mit dem man ein Schema erstellen kann, anhand dessen sich Daten serialisieren lassen.

Ein Vorteil des CIM, ist es, dass es unabhängig von einer bestimmten Plattform ist. Es können beliebige Technologien gewählt werden, mit denen Profile erstellt und der Datenaustausch zwischen Systemen modelliert werden kann. Dennoch gibt es eine klare Tendenz, dass man die Technologien nach klassischem objektorientiertem Design entwirft und XML für den Datenaustausch benutzt, um möglichst standardkonform zu bleiben.

Ferner gibt es auch diverse Nachteile des CIM gegenüber anderen Datenaustauschformaten, so zählt Milano unter anderem die Komplexität des CIM als einen der großen Schwachstellen des CIM auf und dass es zu Duplikation bei den Modellen kommt [196]. Weiter ist die Größe der Dateien bei der Kommunikation 500 bis 1000 mal größer wenn sie nach dem CIM XML-Schema strukturiert sind, als die einer gewöhnlichen Textdatei. Zwar sei die Speicherung dieser Dateien kein Problem mehr, so

aber immer noch die Verarbeitung im Hauptspeicher. Auch diesen Kritikpunkt, der häufig mit CIM aufgeführt wird, da durch Serialisierung von CIM Objekten nach XML, durch die zunehmende Anzahl an Informationen durch die XML-Syntax, viel mehr an Daten versendet werden, als die einfachen Objekte an sich beinhalten würden. Allerdings kann auch diesem Kritikpunkt der Lösungsansatz der Kompression entgegengestellt werden. Heutzutage gibt es viele freie Kompressionsverfahren und viele freie Bibliotheken für diverse Technologien, um mit diesen Kompressionsverfahren zu arbeiten [245].

Ein weiterer Kritikpunkt von Milano ist die hauptsächlich kommerzielle Toolunterstützung des CIM, die zwischenzeitlich durch freie Tools erweitert wurde, wie man bei Uslar et al. sehen kann [254]. Allerdings bleibt das Tool, was am häufigsten für die Arbeit mit dem CIM benutzt wird, der Enterprise Architect. Eine mögliche Ursache dafür kann sein, dass die Weiterentwicklung der freien Tools nicht vorangetrieben wird. Das am häufigsten in der Literatur erwähnte freie Tool, CIMTool, wurde zuletzt vor drei Jahren aktualisiert.[35]

3.2.2 Companion Specification for Energy Metering (COSEM)

Das Smart Grid ist ein Energienetzwerk, welches ein ökonomisch effizientes und nachhaltiges Versorgungssystem darstellt und vielfältige Arten der konventionellen sowie regenerativen Energieerzeugung, intelligente Verbrauchsmessgeräte (Smart Meter) und intelligente Verbraucher (Smart Consumer, Smart Home) in ein Verteilnetz integriert. Die Energieerzeuger sollen so gesteuert werden, dass immer nur so viel Energie produziert, wie benötigt wird. Hierfür berücksichtigt das Smart Grid das Verbrauchs- und Erzeugungsverhalten aller Netzteilnehmer. Die Lastverteilung im Stromnetz kann dadurch besser organisiert werden, was zu einer höheren Netzstabilität und geringeren Verlusten führen soll [198, 284].

Die Informationsübertragung von Verbauchs- und Erzeugungsmessungen ist für die Entwicklung eines Smart Grids von elementarer Bedeutung. Es kann nur dann eine effiziente Lastverteilung erreicht werden, wenn detailliert und zeitnah bekannt ist, wie viel Energie an welcher Stelle verbraucht und erzeugt wird. Entsprechende Sensordaten werden bereits heutzutage im Bereich der Industrie von Großkunden sowie Kraftwerken aufgezeichnet und ermöglichen so eine registrierte Lastgangmessung. Ein völlig neuer Bereich umfasst die Energieverbräuche von Haushaltskunden, welche bislang nicht erfasst werden können, da eine flächendeckende Verteilung von Messgeräten mit der benötigten Auflösung nicht vorhanden ist. Dies soll sich in Zukunft mit der deutschlandweiten Einführung von Smart Metern und dem damit verbundenen Smart Metering ändern.

Ein Smart Meter ist ein digitaler Energiezähler, der mit dem Internet verbunden ist und regelmäßig (im 15 Minuten Takt) Verbrauchsmessungen des Haushaltes an ein Smart Grid übermitteln soll. Für die verschiedenen Energiearten werden unterschiedliche Zähler verbaut, beispielsweise Strom-, Gas- oder Wärmezähler. Im Kontext der Energiewende ist die Integration der regenerativen und verteilten Energiequellen (Distributed Energy Resources, kurz DER) von großer Bedeutung, deren Erzeugungsverhalten ebenfalls durch das Smart Metering erfasst werden soll. Biogasanlagen und Windkraftanlagen werden bereits berücksichtigt. Andere Anlagen, wie private Photovoltaikanlagen oder Blockheizkraftwerke werden nicht erfasst [28]. Für die Projektgruppe ist besonders dieser lokale Bereich im Verteilnetz interessant, da es hierfür keine ausgereiften Systeme für die Datenerfassung und -verarbeitung existieren.

[35] Online: `https://github.com/arnolddevos/CIMTool`. Letzter Zugriff: 30.04.2017

Ein ganz anderer Bereich stellt das Smart Home dar. Das Smart Home bezeichnet die intelligente Überwachung und Steuerung eines Haushaltes. Dieser Bereich könnte ebenfalls für die Projektgruppe von Interesse sein, da für das Smart Grid der Energieverbrauch von sogenannten Controllable Local Systems (CLS) interessant ist. Hierbei handelt es sich um Geräte wie beispielsweise Kühlschränke, Tiefkühltruhen, Waschmaschinen oder Heizungen. Diese Geräte können energiesparend in zeitlicher und mengenmäßiger Weise gesteuert werden. Die Datenerfassung im Smart Home und die Bereitstellung der Daten an ein Smart Grid ist deshalb sinnvoll.

3.2.2.1 Kommunikation im Smart Grid

Das Smart Metering wird durch ein intelligentes Messsystem realisiert, welches sich durch drei wesentliche Komponenten definiert: Smart Meter, Smart Meter Gateway und ein Sicherheitsmodul. Das Smart Meter Gateway (SMGW) stellt die zentrale Kommunikationsschnittstelle für die Datenübertragung zwischen beliebig vielen Smart Metern und einem Server dar. Der Server verarbeitet die Daten vor und schickt diese anschließend zu einem zentralen Knoten, wo die Daten ausgewertet, langfristig gespeichert oder an anderen Stellen im Verteilnetz geschickt werden. Neben einem zentralen Server besteht auch die Möglichkeit, dass ein sogenannter Datenkonzentrator erst Daten zwischenspeichert, aggregiert und dann zu einem Server weiterleitet. Das Sicherheitsmodul ist meistens im SMGW integriert und ist für die Einhaltung von Sicherheitskriterien zuständig, es stellt also sicher, dass Anforderungen an Datenschutz und Datensicherheit eingehalten werden. Ein Messsystem, welches lediglich Messungen durchführt und speichert, aber keinen Zugang zu einem Kommunikationsnetzwerk besitzt, wird modernes Messsystem genannt und stellt in den meisten Fällen einen einfachen digitalen Zähler dar [198, 43].

Die Abbildung 3.21 zeigt alle bisher beschriebenen Komponenten des lokalen Bereichs und deren Kommunikationswege, die alle über das SMGW führen. Hinzu kommen Komponenten, wie Benutzerschnittstellen. Zum Beispiel kann für den Endkunden eine statistisch aufgewertete Visualisierung seiner Daten angezeigt werden oder für den Servicetechniker die Möglichkeit bieten Konfigurationen für Anlagen zu tätigen. Hier ist vor allem die Wartung von Anlagen von Bedeutung [284].

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) beschreibt in der Richtlinie TR-03109 Vorgaben für die Kommunikation im Smart Grid in Deutschland. Unter anderem definiert die Richtlinie drei kommunikations- und sicherheitstechnische Bereiche: LMN, WAN und HAN. Das LMN (Local Metrological Network) beschreibt den Bereich des Smart Metering. Das WAN (Wide Area Network) bietet die Verbindung vom Smart Grid mit den verschiedenen Akteuren, z. B. externe Marktteilnehmer. Das HAN entspricht dem Bereich des Smart Homes. Die Richtlinie gibt für WAN und LMN jeweils einen Protokollstapel an, die auf der Anwendungsschicht des OSI-Modells DLMS/COSEM vorsehen [198, 43]. DLMS/COSEM scheint sich allmählich als Standard für den lokalen Bereich im Smart Grid auch international durchzusetzen. Das Kommunikationsprotokoll ist deshalb für das Projekt besonders relevant. Ein weiteres wichtiges Protokoll, was vor allem im WAN bzw. für die Datenübertragung für Schutz- und Leittechnik verwendet wird, ist IEC 61850. Im folgenden Abschnitt werden diese beiden Protokolle vorgestellt.

Informationstypen für Protokolle auf der Anwendungsschicht

Protokolle für das Smart Grid, die auf der Anwendungsschicht liegen, wie DLMS/COSEM und IEC 61850 müssen unterschiedliche Arten von Informationen spezifizieren und übertragen können. Dabei

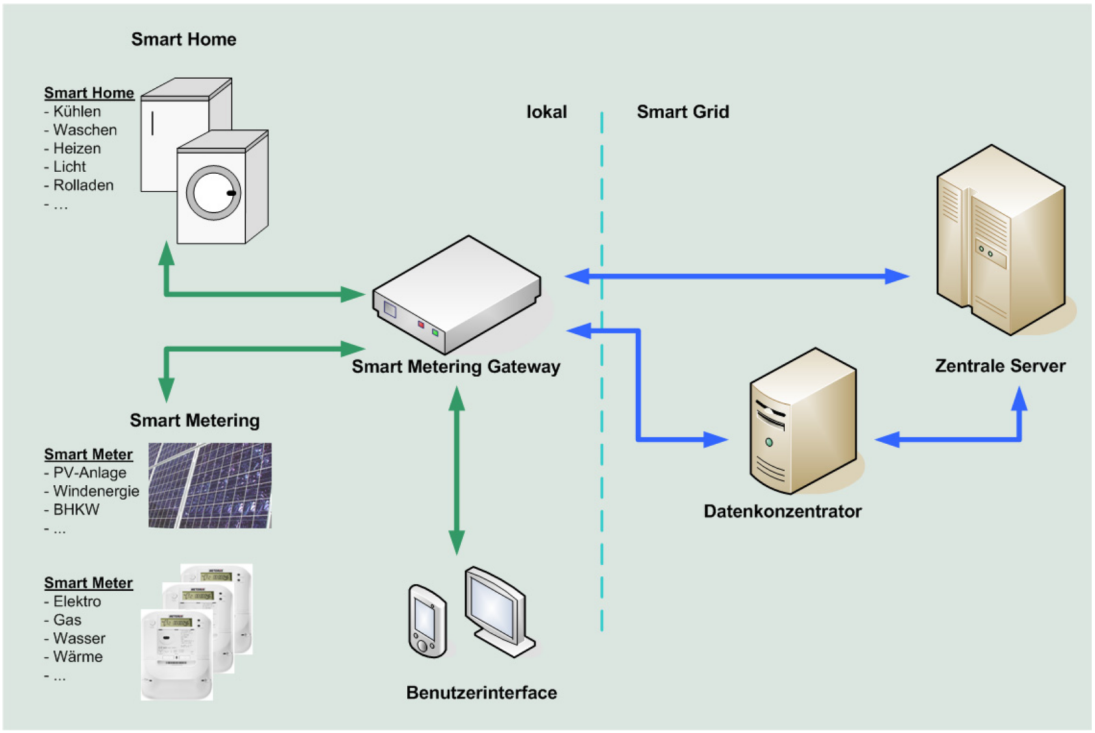


Abbildung 3.21: *Smart Meter Gateway in seiner typischen Umgebung [28]*

muss nicht jede Information von einem Protokoll unterstützt werden. Die folgende Auflistung zeigt die wichtigsten Informationstypen, die auch für einen qualitativen Vergleich unterschiedlicher Protokolle genutzt werden kann [91].

- **Messdaten** - Alle Protokolle müssen Messdaten, wie Energie, Leistung, Spannung oder Volumen übertragen können. Weiter können unterschiedliche Protokolle verschiedene Metadaten unterstützen:
  - **Lastprofil** - Ein Smart Meter könnte Lastprofile speichern, also den zeitlichen Verlauf der abgenommenen Leistung über eine zeitliche Periode. Das Protokoll muss hierfür eventuell Zeitstempel für die Daten definieren.
  - **Digitale Signatur** - Zur Integritätsprüfung der Daten muss ggf. eine Signatur übermittelt werden.
- **Steuerungssignale** - Für die Steuerung von DER oder anderen Geräten müssen Signale übertragen werden können.
- **Zeitsynchronisierungsdaten** - Die zeitliche Synchronisierung von Daten ist notwendig, wenn das Smart Meter Lastprofile unterstützt oder dynamisch zwischen verschiedenen zeitlichen Tarifen wechseln muss.
- **Firmware Updates** - Gateways und Smart Meters sollten durch Software Updates aktualisiert werden können.

- **Tarifinformationen** - Es müssen unter Umständen Informationen zu Tarifen und Abrechnungen übermittelt werden.

3.2.2.2 DLMS/COSEM

DLMS/COSEM ist eine Spezifikation (IEC 62056), welche aus der DLMS und COSEM besteht. COSEM definiert ein Interface Modell für die Beschreibung von Smart Metern. DLMS erweitert COSEM um abstrakte Services, mit denen die Übertragung von Mess- und Tarifdaten sowie Zähler Informationen zum SMGW ermöglicht wird. Das Protokoll befindet sich auf der Anwendungsschicht und kann durch einen Wrapper über TCP/IP oder UDP verwendet werden (vgl. [141]), wobei TCP/IP in der TS-03109 beispielsweise auf dem Protokollstapel für WAN vorgeschrieben wird. Grundlegend basiert die Kommunikation auf einer strikten Client-Server Kommunikation, d. h. bevor Daten übertragen werden können, muss eine Verbindung vom Client ausgehend aufgebaut werden. Besteht eine Verbindung ruft der Client das Interface Modell des Servers ab und erhält so alle notwendigen Informationen. Der Server kann solange weiter unaufgefordert Benachrichtigungen schicken bis die Verbindung aufgelöst wird. Die Richtlinie sieht vor, dass die Verbindung vom SMGW angefordert wird, dieser also den Client repräsentiert während ein Smart Meter den Server darstellt [5, 91].

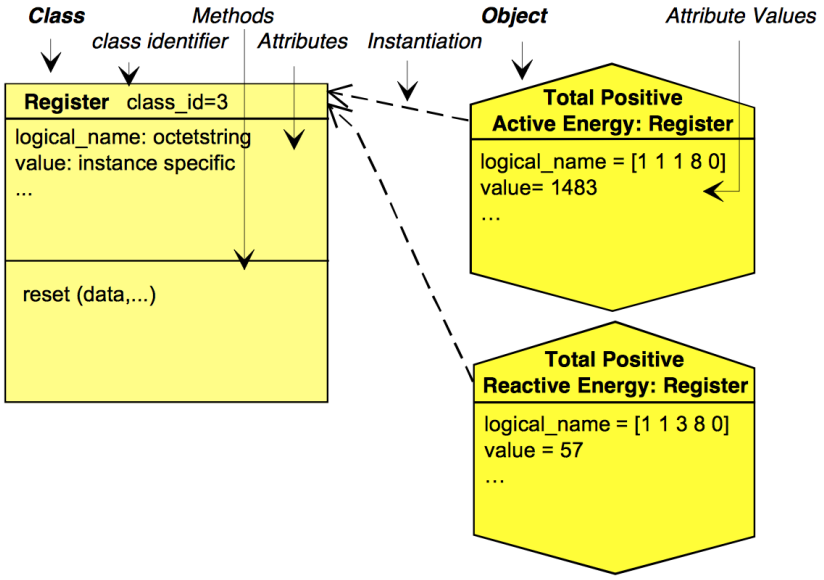


Abbildung 3.22: *Beispiel für eine COSEM Interface Klasse und zwei COSEM Objekten* [36]

Das Interface Modell ist eine abstrakte Beschreibung für Smart Meter und definiert verschiedene Interface Klassen. Eine Interface Klasse beschreibt welche Informationen ein bestimmter Smart Meter besitzt und wie auf diese zugegriffen werden kann. Die Daten werden über sogenannte Attribute deklariert und der Zugriff über sogenannte Methoden beschrieben. Ein logisches Gerät (Logical Device) repräsentiert einen Smart Meter und besitzt eine eindeutige ID, welche sich aus dem logischen Name (*Logical-Name*) und einer Klassen ID (*COSEM-Class-ID*) zusammensetzt. Der Name setzt sich aus OBIS-Kennzahlen (Object Identification System) zusammen, die die Energieart (wie Strom,

[36] Online: `http://dlms.com/documents/Excerpt_BB12.pdf`, 2011, Abgerufen: 28-04-17

	Client	Server
Attribut Service	GET.Request SET.Request	GET.Response SET.Response EVENT.NOTIFICATION.Request
Methoden Service	Action.Request	Action.Response

Tabelle 3.1: *DLMS Service Klassen*

Wasser, Wärme) und die Parameter (z. B. die Wirkleistung) des Gerätes codieren. TR-03109 sieht vor, dass ein solches COSEM Objekt durch XML repräsentiert wird und der Zugriff über HTTP mittels einer RESTful API ausgeführt wird (vgl. BSI TR-03109-1 Anlage II: COSEM/HTTP WebServices[37]).

Der Zugriff auf COSEM Objekte wird durch verschiedene abstrakte Services, die in DLMS spezifiziert sind, beschrieben. Hierbei unterscheidet man zwischen Services, die Attribute ansprechen und welche die Methoden aufrufen. Tabelle 3.1 zeigt eine Übersicht über die verschiedenen Services. Ein Service erhält eine Referenz auf das zugehörige Attribut, dessen Wert abgerufen (`GET`) oder modifiziert werden kann (`SET`). Ein Service, der eine Methode eines COSEM Objektes aufruft, erhält ebenso eine entsprechende Referenz.

Um die Verwendung eines `GET`-Services zu demonstrieren, ist in Listing 3.3 ein einfaches Gerät mit zwei Attributen und einer Methode dargestellt. Das Gerät entspricht der COSEM Klasse Register (`class-id=3`) und ist für die Verbrauchsmessung von Energie (hier Strom) zuständig. Diese Angaben lassen sich über die OBIS-Kennzahl aus der `id` ermitteln. Die zwei Attribute repräsentieren den logischen Namen und den aktuellen Verbrauchsstand. Die angegebene Methode kann für das Zurücksetzen des Zählerstandes verwendet werden.

```
1  <object class-id="3" version="0" id="0101010800FF">
2    <attributes count="2" >
3      <attribute id="1" >
4        <octet-string>0101010800FF</octet-string>
5      </attribute>
6      <attribute id="2" >
7        <Integer8>2047</Integer8>
8      </attribute>
9    </attributes>
10   <methods count="1" >
11     <method id="1" />
12   </methods>
13 </object>
```

Listing 3.3: *COSEM Objekt in XML*

Die Anfrage für den aktuellen Wert des zweiten Attributes kann durch HTTP folgendermaßen ausgedrückt werden, wobei das `PoC`-Element die Wurzel des COSEM Baumes bildet. Dieses Element muss vor der Übertragung vom Server aus bekannt gegeben werden:

```
GET <poc>/cosem/ldevs/3/objects/0101010800FF/attributes/2
```

[37] `https://goo.gl/rKWELW`

Das Ergebnis der Anfrage ist der entsprechende Response-Body:

```
1  <attribute id="2" >
2    <Integer8>2047</Integer8>
3  </attribute>
```

Listing 3.4: *Response-Body einer Get-Anfrage*

Die Spezifikation von DLMS/COSEM wird von der DLMS User Association durch vier Dokumente beschrieben, wobei diese nur in ihrem vollem Umfang durch eine Mitgliedschaft in der DLMS User Association verfügbar sind:

- *Blue Book*[38] beschreibt das COSEM Interface Modell und OBIS-Kennzahlen.
- *Green Book*[39] beschreibt Protokolle auf der Transportschicht für DLMS/COSEM.
- *Yellow Book*[40] behandelt alles was zum Testen der Protokolle benötigt wird.
- *White Book*[41] fasst alle Begriffe und Definition in einem Glossar zusammen.

3.2.2.3 IEC61850

Die Protokollfamilie IEC 61850 ist seit längerem in der Verteilnetzautomatisierung, also der Steuerung von Kraftwerken und DER (vgl. IEC 61850-7), verbreitet. Es ist auch möglich die Spezifikation im Bereich des Smart Metering zu verwenden. Im Vergleich zu DLMS/COSEM treten einige Gemeinsamkeiten in den Vordergrund: IEC 61850 beschreibt die selbe strikte Client-Server Kommunikation und spezifiziert ebenfalls ein Interface Modell für die Modellierung von Daten und Services. Hierbei werden nur Konzepte betrachtet, d. h. das die eigentliche Implementierung durch andere Protokolle umgesetzt wird. IEC 61850 kann als Meta-Protokoll betrachtet werden, dass ebenfalls auf der Anwendungsschicht aufsetzt und den weiteren Protokollstapel bis zur Transportschicht definiert. Die Datenübertragung wird dann durch TCP/IP realisiert [23, 91].

Das Interface Modell von IEC 61850 definiert Daten-Objekte, die sich baumartig aus weiteren Teilen zusammensetzten. Ein logisches Gerät (Logical Device) besitzt einen oder mehrere logische Knoten (Logical Nodes). Ein logischer Knoten ist eine Zusammenfassung von Daten und assoziierten Services, welche sich semantisch auf beispielsweise eine physikalische Energiequelle beziehen. Die Datenelemente eines logischen Knoten besitzen, wie der Knoten selbst, einen eindeutigen Namen und werden durch die Spezifikation IEC 61850-7-3 (Common Data Class, kurz CDC) beschrieben. Die Spezifikation beschreibt Datentypen und Datenstrukturen, welche z. B. Informationen über den Status eines Gerätes oder Anlage sein können. Das abstrakte Datenmodell wird dann auf weitere konkrete Protokolle abgebildet. Solch eine Abbildung könnte folgendermaßen aussehen: Das Manufacturing Message Specification Protokoll (MMS) kann für die Übersetzung des Datenmodells verwendet werden. TCP/IP als Protokoll für die Transport- und Vermittlungsschicht und letztendlich Ethernet für Umsetzung auf der physischen Ebene. Für die Übersetzung könnte auch Simple Object Access Protocol (SOAP) verwendet werden [23]. Abbildung 3.2.2.3 zeigt ein logisches Gerät.

[38] http://dlms.com/documents/Excerpt_BB12.pdf
[39] http://dlms.com/documents/Excerpt_GB8.pdf
[40] http://dlms.com/documents/Excerpt_YB5.pdf
[41] http://dlms.com/documents/White_book_1.pdf

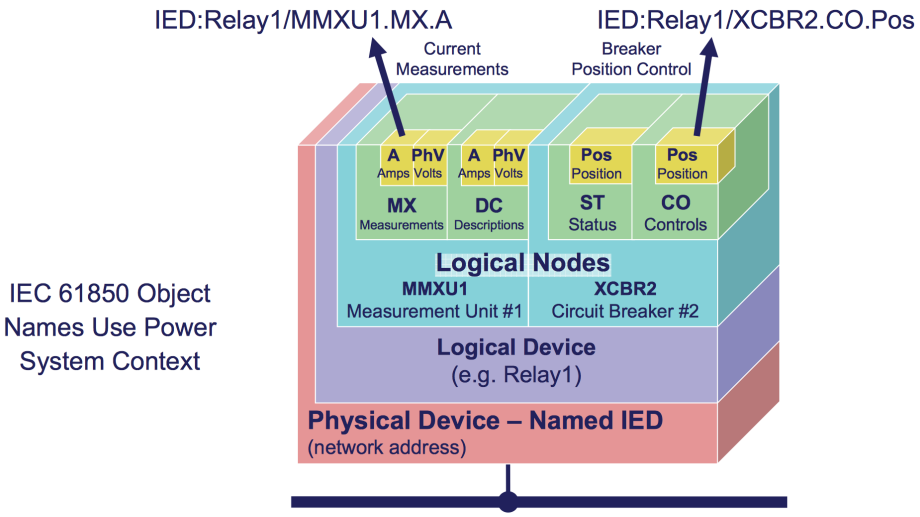


Abbildung 3.23: *Beispiel für ein logisches Gerät in der IEC 61850* [42]

Die Transformation des Datenmodells in ein MMS Objekt, welches in IEC 61850-8-1 spezifiziert ist, wird durch eine Methode umgesetzt. Die Übersetzung erzeugt eine eindeutige und einzigartige Referenz für jedes Datenelement im Datenmodell, wie beispielsweise Abbildung 3.2.2.3 ein bestimmtes Datenmodell zeigt. Abbildung 3.24 zeigt ein MMS Objekt für ein anderes Datenmodell, welches sich aus dem Namen des logischen Gerätes `Relay1` zusammensetzt und einen logischen Knoten mit der Beziehung `XCBCR1` (ist die Klasse für eine Überstromschutzeinrichtung) besitzt.

Das Abstract Communication Service Interface (ACSI) ist ein Modell, welches eine Menge an abstrakten Services sowie deren möglichen Antworten beschreibt. Dies ermöglicht ein einheitliches Verhalten zwischen allen Geräten und Anlagen im Netzwerk. IEC 61850-8-1 beschreibt eine Abbildung der abstrakten Services auf MMS Services. Beispielsweise werden die Dienste, die für die Steuerung einer Anlage verantwortlich sind, auf die `READ` und `WRITE` Services von MMS abgebildet [23]. Die Spezifikation IEC 61850 unterteilt sich insgesamt in zehn Kapitel. Im Folgenden ist eine Auflistung der wichtigsten Kapitel gegeben, die für die Projektgruppe von Relevanz sind.

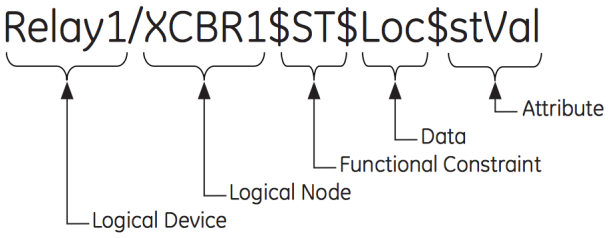
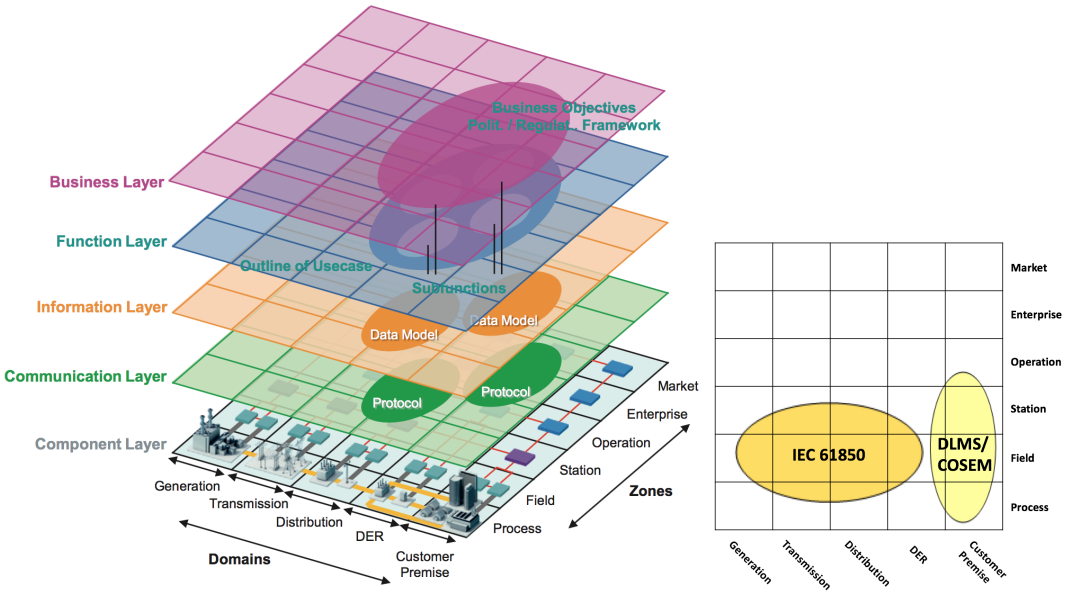


Abbildung 3.24: *Übersetzung eines IEC 61850-8-1 Objektes in MMS [23]*

- IEC 61850-7-2:2010 - Abstract communication service interface (ACSI)
- IEC 61850-7-3:2010 - Common Data Classes
- IEC TS 61850-80-4:2016 - COSEM object model to IEC 61850 data model

- IEC TR 61850-90-8:2016 - Object model for E-mobility
- IEC 61850-8-1:2011 - Mappings to MMS and to ISO/IEC 8802-3
- IEC TR 61850-80-3:2015 - Mapping to web protocols



(a) *Smart Grid Architectur Model* (b) *Einordnung*

Abbildung 3.25: *IEC 61850 und DLMS/ COSEM im SMAG [253]*

Die Abbildung 3.25 zeigt ein Modell, welches von der International Electronical Commission (IEC) zur Beschreibung und Einordnung entsprechender Spezifikationen, Standards und Methoden im Bereich des Smart Grids entwickelt wurde. Das Modell beschreibt Schichten, Domänen und Zonen. Hierdurch lassen sich Schnittstellen und Anwendungsfälle zwischen verschiedenen Spezifikationen beschreiben (vgl. Smart Grid Reference Architecture[43]). Abbildung 3.25b zeigt eine Einordnung der beiden Protokolle in Domänen und Zonen, wobei sich beide auf den Ebenen Communication und Information befinden. IEC 61850 umfasst die Domänen Generation, Transmission, Distribution sowie DER und die Zonen Process, Field und Station. DLMS/COSEM umfasst die Domänen DER sowie Customer Premises und lässt sich den gleichen Zonen wie IEC 61850 zuordnen, wobei hier noch Operation hinzukommt.

3.2.2.4 Zusammenfassung

Es wurde in diesem Kapitel die grundlegenden Konzepte, Ziele und Begriffe vorgestellt, die das intelligente Stromnetz beschreiben. Darunter wurde vor allem auf die Kommunikation zwischen diversen Netzteilnehmern eingegangen und es wurden die Kommunikationsprotokolle DLMS/COSEM und IEC 61850 vorgestellt, die sich allmählich als Standard für die Kommunikation im Smart Grid

[43] `https://ec.europa.eu/energy/sites/ener/files/documents/xpert_group1_reference_architecture.pdf`

herauskristallisieren. Es wurde eine Übersicht über die wichtigsten Begriffe gegeben, die unter anderem Smart Meter, SMGW und Smart Metering umfassen. Hier wurde zwischen einem intelligenten und einem modernen Messsystem unterschieden. Weiter wurden drei Kommunikationsbereiche identifiziert, die als LMN, WAN und HAN bezeichnet werden. Es wurden entsprechende Literaturverweise für eine tiefergehende Recherche zu den beiden vorgestellten Protokollen gegeben. Im Kontext des Smart Metering wurde eine Übersicht über Informationstypen dargestellt, die von Protokollen der Anwendungsschicht unterstützt werden sollten.

DLMS/COSEM wird durch das BSI in der Richtlinie TR-03109 als ein Standard für die WAN-Kommunikation im Smart Metering in Deutschland vorgeschrieben. Es unterstützt ein Interface Modell für die Modellierung von Smart Metern und definiert abstrakte Services für die Übertragung, welche durch HTTP Webservices mittels einer RESTful API umgesetzt werden können. Hierfür wurde ein kurzes Beispiel gezeigt. Die Protokollfamilie IEC 61850 abstrahiert Daten und Services zu einem Protokollstapel für den Bereich des Smart Grid, der für die Steuerung von Energieanlagen verantwortlich ist. Dadurch kann die Spezifikation als ein Meta-Protokoll betrachtet werden, welches durch eine entsprechende Abbildung weitere Protokolle vorsieht. Dementsprechend wurde eine Abbildung von IEC 61850 auf MMS gezeigt.

3.2.3 Predictive Model Markup Language (PMML)

Daten, die auf verschiedenste Wege entstehen, werden typischer Weise nicht nur abgespeichert, sondern dienen in der Regel auch Auswertungszwecken, sodass neues Wissen aus ihnen gewonnen werden kann. Diese großen Daten bezeichnet man als Big Data [276] und sie umfassen Datenmengen, die zu schwach strukturiert, komplex oder schnelllebig sind. Die Gewinnung von Wissen (auch als Data Mining bezeichnet) z. B. zur Trenderkennung ist häufig nur sehr schwer von Menschenhand durchzuführen, da die auszuwertenden Mengen zu groß sind. Vielmehr werden computergestützte Verfahren verwendet, um diese Auswertungen durchzuführen. Diese Verfahren können sehr vielfältig sein, von (z. B. Gesichtserkennung in Bildern, bis zu häufig zusammen gekauften Artikeln in Onlineshops), weshalb die verwendeten Daten hier nicht weiter eingegrenzt werden.

Mit der Zeit haben viele verschiedene Hersteller, darunter auch sehr große Unternehmen wie IBM, SAS und Microsoft, Werkzeuge geschaffen, um Datenmengen verschiedenster Art auszuwerten. Damit gehen auch sehr viele Austauschformate für die Verfahren und Ergebnisse einher, die in der Regel nicht, oder nur eingeschränkt, untereinander kompatibel sind. Aus dieser Notwendigkeit heraus, Verfahren und Ergebnisse darzustellen, entstand die DMG. Dieses unternehmensgetriebene Konsortium erstellte einen kontinuierlich weiter gepflegten Standard, der es ermöglicht die genannte Problematik zu standardisieren. Mittlerweile (Stand April 2017 [222]) unterstützen 35 Software Hersteller das 1997 verabschiedete XML-basierte Austauschformat.

In diesem Abschnitt wird anfangs auf verwandte Themen eingegangen, bevor das Data Mining näher gebracht wird. Anschließend wird die PMML in ihrer Struktur erklärt und im Nachgang anhand eines vollständigen Beispiels gezeigt.

3.2.3.1 Verwandte Themen

PMML gehört zur Klasse der Wissensrepräsentation (engl. knowledge representation and reasoning). Theoretisch gesehen könnte man Wissen, das aus Datenmengen gewonnen wurde, auch in einer Daten-

bank, oder anderen Speichermedien ablegen, um diese auszutauschen. Jedoch müssen Informationen dazu, wie genau die Daten gewonnen wurden, in irgendeiner anderen Form abgelegt sein, was genau die Problematik der Kompatibilität aufwirft.

Es gibt in dieser Domäne weitere Austauschformate, wie z. B. das Portable Format for Analytics (PFA) (ebenfalls von der DMG [235]) oder das Knowledge Discovery Metamodel (KDM) [166]. Diese Modelle unterscheiden sich jedoch im Funktionsumfang gegenüber PMML, da sie sich thematisch weiter abgrenzen (PFA für Analytische Auswertungen und KDM für semantische Integration von Application Lifecycle Management für das Software Mining).

Data Mining

Als Data Mining bezeichnet man die Gewinnung von Wissen aus Datenmengen verschiedener Art. Häufig entstehen z. B. aus Nutzungsdaten von Onlineshops (Kaufvorgänge), oder Sensoren (Wettermessstationen, Verbrauchsmessungen von Haushalten oder GPS-Daten von Fahrzeugen) Meta-Daten, aus denen nützliche Informationen gewonnen werden können. Das Data Mining selbst bezieht sich eigentlich nur auf die Verarbeitung des Prozesses der Informationsgewinnung und hat erst einmal nichts mit der Sammlung von Daten zu tun.

Das Data Mining lässt sich in einen iterativen Prozess, der wie folgt aussieht, eingliedern [86]:

- Erhebung und Selektion der Daten
- Datenbereinigung zur Beseitigung von Inkonsistenzen durch Entfernen oder Ergänzen von unvollständigen Datensätzen
- Transformation in das passende Format
- **Der eigentliche Data Mining Prozess**
- Evaluation der gefundenen Informationen

Das Data Mining stellt sich allgemein verschiedensten Fragestellungen an Datenmengen, in Bezug auf die Informationen, die gewonnen werden sollen. Typische Anwendungsgebiete davon sind die Klassifikation von Elementen, deren Klasse nicht bekannt ist, oder Klassen, die anhand von gleichen Eigenschaften abgeleitet werden können. Das Ableiten der Eigenschaften ist eine weitere typische Aufgabe und wird Clustering genannt. Hier werden Elemente mit ähnlichen Eigenschaften gruppiert und können hinterher unbekannte Zusammenhänge aufdecken.

Zusammenhänge zwischen Elementen können auch durchaus komplizierterer Natur sein und sich gegenseitig bedingen z. B. „Wenn Eigenschaft X und Y gelten, gilt auch mit hoher Wahrscheinlichkeit Z“. Dieses Verfahren wird der Regressionsanalyse zugeordnet. Das ist nur eine kleine Auswahl an Data Mining-Aufgabenstellungen, die mit Hilfe computergestützter Analyse durchgeführt werden können [86].

3.2.3.2 Predictive Model Markup Language (PMML)

Man kann aus den Beispielen heraus sehen, dass das Anwendungsgebiet von Data Mining ebenso schwer einzugrenzen ist, wie die Tatsache, dass die Darstellung des Verfahrens selbst nur sehr abstrakt ist. Die DMG hat sich das Ziel gesetzt ein einheitliches Austauschformat für die Verfahren selbst

sowie die Struktur der benötigten Daten zu pflegen, um die Kompatibilität der verschiedenen Software Hersteller zu maximieren.

PMML wurde in der ersten Version (0.9) von Dr. Robert Lee Rossmann im Jahr 1998 veröffentlicht. Seitdem kümmert sich die DMG um die fortlaufende Weiterentwicklung des Austauschformats. Das Austauschformat folgt einem XML-Schema, welches auf der Homepage von der DMG[44] eingesehen werden kann. Da der Standard mittlerweile (aktuell in v4.3, Stand April 2017) über 700 Sprachelemente verfügt, wird hier des Umfangs halber darauf verzichtet auf alle einzugehen und viel mehr der grundlegende Aufbau gezeigt.

Aufbau

Das verwendete XML-Format ist in Version 1.0. Als Root-Element wird immer PMML mit der Angabe der jeweiligen Version mit Hilfe eines Attributes verwendet. Laut Schemadefinition müssen im PMML-Dokument ein `Header` und das `DataDictionary` vorkommen.

Der `Header` sollte folgende Informationen beinhalten:

- `copyright`: Hier wird die Lizenz des verwendeten Data Mining Verfahrens abgelegt. Diese Information ist seit v4.1 optional
- `description`: Eine textuelle Beschreibung des dargestellten Verfahrens. Laut Konvention sollte es alle Informationen beinhalten, die zur Benutzung wichtig sind
- `modelVersion`: Um ähnliche Modelle voneinander abzugrenzen, wird ein Verfahren versioniert, um sie eindeutig identifizieren zu können
- `Application`: Hinterlegt die Information, aus welcher Anwendung das Dokument heraus erzeugt wurde. Ähnliche Daten können bei gleichen Verfahren anwendungsbedingt unterschiedliche Verfahren hervorbringen. Diese Information dient dem Benutzer zur Unterscheidung und dem Vergleich
- `name`: Name der verwendeten Software
- `version`: Version der verwendeten Software
- `Annotation`: Hier werden weitere Informationen (z. B. zu Veränderung des Verfahrens/Models) in Klartext abgelegt. Zusätzlich kann ein Kindelement Extension mit dem Attribut `name=Äuthor"` verwendet werden, um den Autor des Kommentars anzugeben
- `Timestamp`: Der Erstellungszeitpunkt

Neben dem Header muss noch das `DataDictionary` vorhanden sein. Im `DataDictionary`-Element muss als Attribut `numberOfFields` die Anzahl der existierenden Datenfelder definiert sein. Das `DataDictionary` dient als Definition aller existierenden Datenfelder für das Data Mining-Model und legt die entsprechenden Datentypen und Wertebereiche fest. Das Dictionary selbst ist unabhängig von den letztendlich verwendeten Daten. Da das PMML-Dokument mehrere Mining Modelle beinhalten kann, erlaubt das `DataDictionary` von allen innerhalb eines Dokuments verwendet zu werden und ist somit als globale Definition zu betrachten.

Zu den Kindelementen eines `DataDictionary`-Elements zählt eine Auflistung von `DataFields`, wo der Datentyp und zusätzliche Einschränkungen, wie z. B. das Intervall und die da-

[44] DMG Homepage `http://dmg.org/pmml/v4-3/GeneralStructure.html`

zugehörige Taxonomie und Erweiterungen, angegeben werden können. Zu einem Datenfeld müssen der Name, der Datentyp (auszuwählen aus einer vordefinierten Menge an Basisdatentypen) und die Kategorie (Diskret, Kontinuierlich oder Kategorial) in Form von Attributen angegeben werden. Neben den Datentypen selbst kann zusätzlich eine Hierarchie dargestellt werden, um Zusammenhänge zwischen Feldern darzustellen. Hierzu gibt es eine Taxonomie-Auflistung im `DataDictionary`, die es erlaubt über geschachtelte Elemente mit folgenden Attributen Hierarchien zu erzeugen:

- `childField`: Name des Felds mit dem Kindelement
- `parentField`: Name des Felds mit dem Elternelement
- `parentLevelField`: Legt für das jeweilige Element ein Level fest, typischer Weise beginnt die Aufzählung bei 0. Dieses Feld ist jedoch optional und kann aus der Struktur heraus erzeugt werden.
- `isRecursive`: Gibt an, dass die Struktur rekursiv zu lesen ist. Somit können alle Elemente hintereinander angeordnet sein und über die gerade genannten Elemente hierarchisiert werden.

Folgender Code zeigt eine simple Datenstruktur einer Tabelle mit drei Attributen `userName`, `userAge` und `groupName`. Hier wird einerseits `userAge` mit einer linken Schranke von 18 und einem offenen Intervall nach oben restriktiert. Die Auflistung der `groupName` wird auf drei Werte (Admin, Member und Guest) limitiert. Zusätzlich werden `userAge` und `groupName` als Kindelemente and den Benutzernamen geknüpft, um einen Zusammenhang darzustellen, wie in Listing 3.5 zu sehen.

```
<DataDictionary numberOfFields="3">
  <DataField name="userAge" optype="ordinal" dataType="integer">
      <Interval closure="closedOpen" leftMargin="18"/>
  </DataField>
  <DataField name="userName" optype="ordinal" dataType="string"/>
  <DataField name="groupName" optype="ordinal" dataType="string">
      <Value value="Admin"/>
      <Value value="Member"/>
      <Value value="Guest"/>
  </DataField>
  <Taxonomy name="User-Group">
      <ChildParent childField="userAge" parentField="userName"/>
      <ChildParent childField="groupName" parentField="userName"/>
  </Taxonomy>
</DataDictionary
```

Listing 3.5: *Deklaration des* `DataDictionary`

Neben dem `Header` und `DataDictionary` existiert ein `TransformationDictionary`, welches angegeben werden kann, um zuvor definierte Datenfelder zu transformieren oder sie zu ergänzen. Damit können zusätzliche Daten für das Modell definiert werden, um sie zugänglicher im Sinne des Datentyps zu machen. Üblicherweise bezieht sich das `DataDictionary` auf reale Datenquellen, da in der Taxonomie ein `TableLocator` angegeben werden kann, der sich direkt auf Schema und Entität einer Datenbank beziehen kann. Damit muss die Datenquelle keine unnötig aggregierten Werte enthalten und es können anwendungsspezifisch Werte generiert werden (z. B. für Modelle, die aggregierte oder kommutierte Werte erfordern, der Datenbestand es aber nicht hergibt).

Die Funktionalität des `TransformationDictionary` ist sehr umfangreich und umfasst eine Reihe an Operanden um neue Felder zu erzeugen. Es kann ein abgeleitetes Feld (`DerivedField`)

oder eine Funktion (`DefineFunction`) definiert werden. Abgeleitete Felder müssen eine Funktion angeben, die ein Quellelement in ein neues `DataField` überführt. Diese Funktion ist entweder aus den Standardfunktionen gewählt, oder wird selbst definiert. Eine händisch definierte Funktion (`DefineFunction`) kann verschiedene Parameterfelder erhalten, die wiederum verwendet werden, um Transformationen auf gegebenen Daten durchzuführen.

Eine Funktion erlaubt es, beliebig viele Parameter anzugeben und verschiedene Operationen darauf auszuführen. Das bringt den Vorteil mit sich, dass man Felder, die ähnlich abgeleitet werden, mit der Funktion versieht, um unnötige Redundanzen in der Definition zu vermeiden oder wenn die Standardfunktionen nicht genug Flexibilität haben, die gewünschte Transformation auszuüben. Um eine Funktion aufzurufen, wird ein `DerivedField` mit einem `Apply`-Element versehen, welches den Namen der aufzurufenden Funktion sowie die notwendigen Parameter unter Angabe des `FieldRef`-Elements braucht. Als `FieldRef` können auch bereits abgeleitete Felder verwendet werden. Diese müssen jedoch in der Reihenfolge vor ihrer Verwendung bereits ausgewertet worden sein. Listing 3.6 zeigt die Aggregation über den Durchschnitt des Alters des vorherigen Beispiels und das Auffüllen des Feldes `groupName` mit dem Wert „Guest“, wenn dieser Null ist.

```
1  <LocalTransformations>
2    <DerivedField name="sumAge">
3      <Aggregate field="userAge" function="average" groupField="avgAge"/>
4    </DerivedField>
5    <DefineFunction name="fillGroups">
6      <ParameterField name="my" optype="categorial" dataType="string">
7        <MapValues outputColumn="fullGroup" mapMissingTo="Guest"/>
8      </ParameterField>
9    </DefineFunction>
10   <DerivedField name="nonNullGroup">
11     <Apply function="fillGroups">
12       <FieldRef field="groupName"/>
13     </Apply>
14   </DerivedField>
15 </LocalTransformations>
```

Listing 3.6: *Beispiel einer* `LocalTransformation`

In allen der bisher verwendeten Elemente kann zusätzlich noch eine `Extension` verwendet werden, die es ermöglicht den Inhalt der Elemente zu erweitern. Typischerweise ist eine Extension Zusatzinhalt, der häufig herstellerabhängig ist, und weitere Informationen, z. B. wie mit Werten umgegangen werden soll, zu erweitern. Für ein `DataField` kann außerdem wie in Listing 3.7 eine Formatierung angegeben werden.

```
1 <DataField name="foo" dataType="double" optype="continuous">
2   <Extension name="format" value="%9.2f"/>
3 </DataField>
```

Listing 3.7: *Deklaration eines* `DataFields`

Der Inhalt der Extension ist in der PMML-Dokumentation nicht weiter spezifiziert, sodass eigene Inhalte eingepflegt werden können. Das `Extension` Element soll laut Definition aber an erster oder letzter Stelle in der PCDATA (Inhalt innerhalb eines Elements) eines Elternelements untergebracht werden.

Bisher wurde nur auf den Aufbau von Datenstrukturen und Transformationen innerhalb der PMML eingegangen. Der eigentlich wichtige Teil des PMML-Dokuments sind die Data Mining Schemata, welche in der Regel die entsprechenden Ergebnisse, abhängig von dem verwendeten Algorithmus, angeben. Ein PMML Dokument kann mehr als ein Schema enthalten, jedoch ist im Standard definiert, dass eine Software, die keine Auswahl der enthaltenen Modelle darstellen kann, immer das erste Schema als Data Mining-Model nimmt. Jedes Modell hat ein eigenes Root-Element, welches den Namen und das Suffix „Model“ hat. Tabelle 3.2 zeigt die in PMML v4.3 verfügbaren Modelle, die in ihrer ausführlichen Form in der PMML-Referenz[45] eingesehen werden können.

Data Mining Models in PMML
AssociationModel
BayesianNetworkModel
BaselineModel
ClusteringModel
GaussianProcessModel
GeneralRegressionMode
MiningModel
NaiveBayesModel
NearestNeighborModel
NeuralNetwork
RegressionModel
RuleSetModel
SequenceModel
Scorecard
SupportVectorMachineModel
TimeSeriesModel
TreeModel

Tabelle 3.2: *PMML v4.3 Data Mining Modelle nach [67]*

Bis auf ein paar gleiche Elemente sowie Attribute, hat jedes Data Mining-Modell die Möglichkeit eigene Vorgaben zu schaffen, was die Elemente angeht. Zu den Attributen, die für alle Mining Modelle gebraucht werden, gehören:

- `modelName`: Gibt den Modellnamen zur eindeutigen Identifikation an. Der Bezeichner ist eindeutig und darf nur ein mal innerhalb eines Dokuments vorkommen
- `functionName`: Jedes Mining Modell hat verschiedene Verfahren, mit denen er ausgeführt wird. Ein neuronales Netz kann man z. B. zur Vorhersage von numerischen Werten, oder auch zur Klassifikation benutzen. Deshalb muss diese Information zu einem Algorithmus zusätzlich angegeben werden
- `algorithmName`: Optionaler Wert, der dem Algorithmus einen Namen gibt
- `isScorable`: Viele der Algorithmen sind „Scorable“, was bedeutet, dass man sie anhand eines Richtwerts vergleichen kann. Da fast alle Modelle nur Schätzverfahren sind, bleibt die

[45] PMML Referenz `www.dmg.org/pmml/v4-3/GeneralStructure.html`

Notwendigkeit offen, Verfahren mit einander zu Vergleichen. So können verschiedene Verfahren mit gleichen Daten auf ihre Leistungsfähigkeit überprüft und verglichen werden

Zu den Kindelementen eines Data Mining-Models gehören:

- `MiningSchema`: Durchgangsschema eines jeden Data Mining-Modells. Die gesamte Datenmenge muss durch die darin enthaltenen Operatoren gereicht werden. Es können dafür beliebig viele `MiningFields` definiert werden. Ein `MiningField` dient wiederum anschließend als Datenfeld für den entsprechenden Algorithmus. Mit Hilfe eines solchen Feldes kann in gewisser Weise eine Plausibilitätskontrolle durchgeführt werden, damit keine ungültigen Daten verwendet werden. Als Beispiel hierfür dient Listing 3.8

```
1 <MiningField
2 name="foo"
3 missingValueReplacement="3.14"
4 missingValueTreatment="asMean"
5 />
```

Listing 3.8: *Deklaration eines `MiningFields`*

Das gezeigte `MiningField` namens `foo` hat einen Wert, der bei fehlenden Werten verwendet werden kann, und in diesem Fall eine Interpolation für ungültige Werte. Mit Hilfe von `lowValue` und `highValue` können zum Beispiel Grenzwerte festgelegt werden. Sollte der eingegebene Wert diesen Bereich verlassen, wird er hier mit dem Durchschnitt der anderen Werte ersetzt.

- `Output`: Beschreibt die Ausgabe. Als Kindelemente werden `OutputFields` angelegt, welche erst einmal die gleichen Attribute und strukturellen Eigenschaften besitzen wie die `DataFields`, die im vorherigen Abschnitt bereits eingeführt wurden. Zusätzlich gibt es jedoch noch eine Menge weitere Attribute, welche spezifische Ausgaben des verwendeten Algorithmus enthalten, wie zum Beispiel der `rank` oder `rankBasis`, welcher den ausgegebenen Wert mit einem Rang versieht und Auskunft darüber gibt, wie der Rang entsteht (Auswahl aus: confidence, support, lift, leverage, affinity).
- `ModelStats`: Hier kann eine beliebige Anzahl an Typinformationen abgelegt werden, wie zum Beispiel die `NumericInfo`, das Minimum, Maximum, Durchschnitt, Standardabweichung und die Quantile zu einem angegebenen Feld angibt. Abgesehen von dem Beispiel ist eine große Anzahl an Statistiken in dem PMML-Standard definiert.
- `ModelVerification`: Um das Modell zu verifizieren, können in diesem Element Verifikationsfelder angegeben werden, die zum Beispiel Abweichung zum Trainingsdaten-Set darstellen. Die Informationen, die hier eingebettet werden können, variieren mit den Algorithmen.

3.2.3.3 Beispiel

Um ein vollständiges Beispiel für ein valides PMML Dokument zu bringen wird Listing 3.9 näher betrachtet [221].

In diesem Beispiel wird eine Datenquelle mit vier Spalten beschrieben, welche das Alter, das Einkommen, die Abschläge und die Arbeitszeiten enthält. Im `DataDictionary` sind diese Informationen abgelegt, sodass die Software weiß, welche Spalten aus einer Quelle entnommen werden müssen. Als Mining Modell wird ein Clustering Verfahren definiert, das den K-Means Algorithmus

als Funktion verwendet. Für den K-Means-Algorithmus muss die Clusteranzahl explizit angegeben werden, in diesem Beispiel zwei.

Als Distanzfunktion für den K-Means wird `squaredEuclidean` benutzt, was gleich dem Abstand zwischen zwei Punkten entspricht. Das Data Mining-Schema und die dazugehörigen Werte sowie die Clusterfelder entsprechen dem oben definierten `DataDictionary`. Am Ende der PMML Datei sind die Ergebnisse des Verfahrens in Form von zwei Clustern dargestellt. In diesen sind, wie man an den Attributen ablesen kann, jeweils 1081 und 319 Werte enthalten. Das darin liegende Array beschreibt die konvexe Hülle der beiden Cluster (in diesem Fall die linke obere und die rechte untere Ecke des aufspannenden Rechtecks der Cluster).

```
1  <PMML>
2    <Header copyright="Copyright (c) 2012 DMG" description="KMeans cluster model">
3      <Extension name="user" value="DMG" extender="Rattle/PMML"/>
4      <Application name="Rattle/PMML" version="1.2.29"/>
5      <Timestamp>2012-09-27 13:30:20</Timestamp>
6    </Header>
7    <DataDictionary numberOfFields="4">
8      <DataField name="Age" optype="continuous" dataType="double"/>
9      <DataField name="Income" optype="continuous" dataType="double"/>
10     <DataField name="Deductions" optype="continuous" dataType="double"/>
11     <DataField name="Hours" optype="continuous" dataType="double"/>
12   </DataDictionary>
13   <ClusteringModel
14     modelName="KMeans_Model" functionName="clustering"
15     algorithmName="KMeans: Hartigan and Wong"
16     modelClass="centerBased" numberOfClusters="2">
17     <MiningSchema>
18       <MiningField name="Age" usageType="active"/>
19       <MiningField name="Income" usageType="active"/>
20       <MiningField name="Deductions" usageType="active"/>
21       <MiningField name="Hours" usageType="active"/>
22     </MiningSchema>
23     <ComparisonMeasure kind="distance">
24       <squaredEuclidean/>
25     </ComparisonMeasure>
26     <ClusteringField field="Age" compareFunction="absDiff"/>
27     <ClusteringField field="Income" compareFunction="absDiff"/>
28     <ClusteringField field="Deductions" compareFunction="absDiff"/>
29     <ClusteringField field="Hours" compareFunction="absDiff"/>
30     <Cluster name="1" size="1081">
31       <Array n="4" type="real">
32         40.0074005550416 52168.0953839037 79.8489053345668 41.8205365402405
33       </Array>
34     </Cluster>
35     <Cluster name="2" size="319">
36       <Array n="4" type="real">
37         33.5172413793103 191428.689278997 11.6112852664577 35.6990595611285
38       </Array>
39     </Cluster>
40   </ClusteringModel>
41 </PMML>
```

Listing 3.9: *PMML Beispiel: KMeans cluster model [221]*

3.2.3.4 Fazit

Zusammenfassend ist zu sagen, dass die PMML viele wichtige Aspekte von Data Mining-Modellen gezielt abdeckt und auch für komplexe Datenstrukturen geeignet ist. Durch zahlreiche Erweiterungen bezüglich der einpflegbaren Extensions, können über den Standard hinaus überall softwarespezifische Eigenschaften eingebettet werden, sodass Hersteller von Software weitere Flexibilitäten in Anspruch nehmen können. Als Beispiel für solche Extensions dienen Datenquellen, in denen angegeben wird, aus welcher Datenbank, welcher Tabelle und welcher Spalte die Werte entnommen werden.

Durch die umfassende Definition zahlreicher Algorithmen können sehr komplexe Analysen strukturell dargestellt und verglichen werden. Zudem besteht die Möglichkeit die Ergebnisse in Form von Auflistungen, Arrays, Matrizen und anderen Rohdaten einzubetten. Mit Hilfe dieser Ergebnisse können zum Beispiel verschiedene Algorithmen miteinander verglichen werden, um komplexen Problemstellungen in vielerlei Hinsicht gerecht zu werden.

Die Recherche hat ergeben, dass PMML durchaus der flexibelste und am weitesten entwickelte Standard zum Austausch solcher Informationen ist. Ähnliche Standards haben bei weitem nicht so viele Hersteller, die den Import und Export entsprechender Datenformate unterstützen.

3.3 Datenströme

3.3.1 Datenstromverarbeitung

Täglich werden große Datenmengen in der Netzwerkanalyse oder durch Sensoren in bspw. Handys oder Ampeln erzeugt. Diese Datenmengen können nach [111] nicht durch herkömmliche Datenbankmanagementsysteme (DBMS) verarbeitet werden. DBMS speichern zunächst die gesamte Datenmenge auf Festplatten, um sie anschließend verarbeiten zu können [112], wodurch Ergebnisse entweder gar nicht oder mit einer sehr großen Latenz bereitgestellt werden.

Typische Anwendungsbereiche, in denen dieses Problem auftritt, sind die Verkehrsteuerung oder die Netzwerkanalyse, in der die Auslastung von Straßen oder Leitungen untersucht werden, um das Ampelsystem für einen besseren Verkehrsfluss dynamisch umzustellen oder die Auslastung bestimmter Netzwerk-Knoten festzustellen und die Last im Netzwerk neu zu verteilen [104]. Dafür lassen sich Anfragen wie „Wie viele Autos fahren auf der Hauptstraße im Vergleich zu den Nebenstraßen 1, 2 und 3?“ oder „Wie hoch ist die Auslastung auf den Straßen x, y und z zwischen 18:00 und 18:30“ aufstellen und mit den Vergleichswerten der Anfrage die Ampelzeiten der Kreuzungen der Straßen ermitteln.

Darüber hinaus gibt es immer mehr Sensorik, insbesondere die RFID-Technologie wird immer beliebter. Smartphones zum Beispiel verfügen typischerweise über einen GPS-Empfänger, Barometer, Magnetometer, Beschleunigungs- und Rotationssensoren, die alle zu verarbeitende Daten erzeugen [159]. Da Smartphones häufiger zur Erfassung von sportlichen Leistungen benutzt werden, lassen sich für diese Anfragen wie „Was war die Durchschnittsgeschwindigkeit der letzten 10 Minuten?“ oder „Wie verhält sich der Puls in 5 Minuten-Intervallen beim Zirkeltraining?“ formulieren.

Es gibt auch Anfragen an Systeme, bei denen sich die Anfrageparameter konstant ändern. Ein Beispiel dazu sind spatiotemporale Anfragen, welche von neueren High-Tech Navigationsgeräten ausgeführt werden können. Dies sind Top-k-Anfragen, welche unter anderem die Frage „Welche maximal k Hotels gibt es in einem Umfeld von 5 Kilometern, die ein freies Zimmer haben, und weniger

als 100 Euro pro Nacht kosten?“ beantworten. Befindet man sich dabei auf der Autobahn, muss bei der Verarbeitung die sich ständig ändernde Position berücksichtigt werden [77].

Auch die Überwachung von Vitalwerten in Krankenhäusern oder beim Militär sowie die Überwachung eines Smart Grids, erzeugt konstant große Datenmengen [42].

Im Smart Grid werden unter anderen die Erzeugung und der Verbrauch überwacht, da sowohl ein zu hoher Verbrauch als auch eine zu große Produktion zum Zusammenbruch des Energienetzes führen kann. Zur Erfassung werden Smart Meter eingesetzt, die im Minutenintervall automatisch den Verbrauch an ein System senden sollen. Die Überwachung gestaltet sich im Vergleich zum herkömmlichen Stromnetz problematisch, da mit Smart Metern bspw. in Los Angeles nach [243] eine Verarbeitungsgeschwindigkeit von etwa 200 Mbps benötigen wird, wenn jeder Verbraucher etwa 1 KB Daten pro Minute erzeugt. Die Daten der Smart Meter können dann dazu genutzt werden, den Stromverbrauch vorauszusagen und ein Netzleitsystem kann ggf. entsprechende Gegenmaßnahmen einleiten. Zum Beispiel steigt der Verbrauch morgens schlagartig an, da viele Menschen nahezu gleichzeitig aufstehen. Auch ein Fehlen von Datensätzen kann zur Ausfallerkennung beitragen, wodurch Techniker zur Fehlerbehebung informiert werden, sodass der Ausfall nur eine minimale Zeit andauert [243].

Die Produktion von Energie hat sich im Smart Grid ebenfalls geändert, da neben wenigen zentralen Kraftwerken erneuerbare Energien immer weiter ausgebaut werden. Das schließt On- und Offshore Windparks sowie Solarfarmen, aber auch Photo-Voltaik-Anlagen ein, die neben hydro-elektronischen Kraftwerken wie Tiefseeturbinen oder Biogas-Analagen am weitesten ausgebaut sind. Diese produzieren allerdings nicht wie die zentralen Kraftwerke konstant Energie, sondern sind zum Teil wetterabhängig. Ein Windpark kann bei einem Sturm plötzlich große Energiemengen produzieren, wodurch das Stromnetz instabil werden kann. Im Smart Grid können daher neben den vom Stromnetz erzeugten Daten auch andere Daten zur Vorhersage und damit zur Steuerung integriert werden [249]. Durch die großen Datenmengen, die bei der Verarbeitung benötigt werden, wird die Echtzeitanalyse des Smart Grids unter anderem durch ein DSMS durchgeführt.

Hier wird daher im nächsten Abschnitt das Datenstrommodell näher betrachtet. Der Unterunterabschnitt 3.3.1.2 beschäftigt sich mit der Verarbeitung von Datenströmen. Abschließend fasst Unterunterabschnitt 3.3.1.4 dieses Kapitel zusammen.

3.3.1.1 Datenstrommodell

Als Datenstrom wird eine unendliche Sequenz von total geordneten Elementen bezeichnet [111]. Ein DSMS kann die Elemente eines Datenstroms auf verschiedene Art interpretieren: Sie können als relationales Tupel analog zu einer Relation in einem DBMS oder als heterogene Objekte analog zu Klassen in objektorientierter Programmierung betrachtet werden.

Im relationalen Ansatz muss jedes Tupel ein Schema erfüllen und ein Datenstrom wird als Relation betrachtet, in die nur Elemente hinzugefügt werden [104]. Dies hat den Vorteil, dass die relationale Algebra auch auf Datenströme anwendbar wird und das insbesondere Konzepte und Technologien von DBMS übernommen werden können.

Die Kernunterschiede von DBMS und DSMS werden in Abbildung 3.26 dargstellt. In DBMS (siehe Abbildung 3.26 links) werden Anfragen einmal vom Nutzer ausgeführt und das DBMS antwortet, indem es den aktuellen Zustand des DBMS zurückgibt. Dabei werden Ergebnisse nur dann erzeugt, wenn ein Nutzer oder eine Anwendung diese verlangt. Dies wird im Allgemeinen als *pull*-basiertes

Konzept bezeichnet. Der aktuelle Zustand von DBMS befindet sich normalerweise auf einer Festplatte und muss unter Einhaltung der Transaktions- und ACID-Eigenschaften durchgeführt werden. Zusätzlich legt das DBMS eine Transaktionshistorie (*transaction log*) an, durch die bei einem Ausfall der letzte konsistente Zustand des DBMS wiederhergestellt werden kann [171].

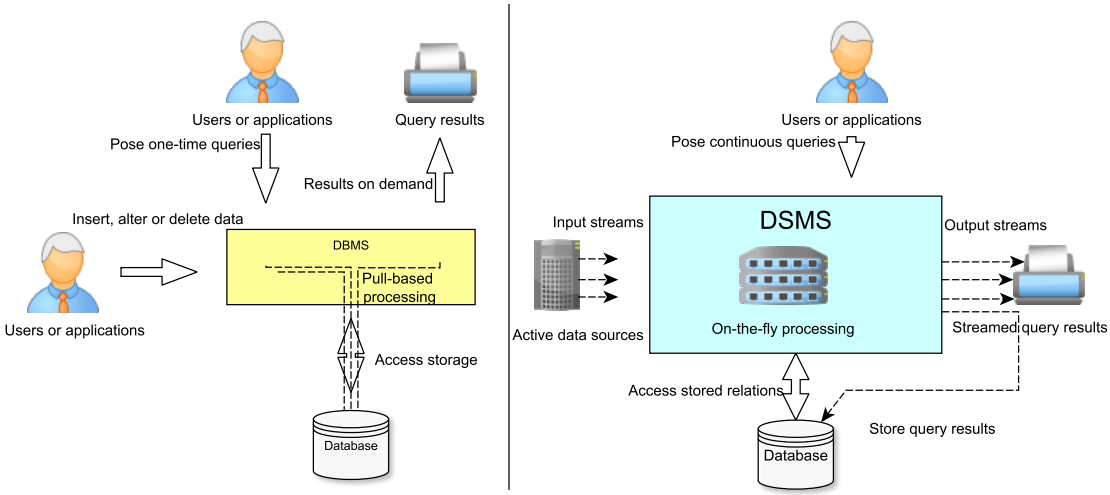


Abbildung 3.26: *Unterschiede in der Verarbeitung von DSMS und DBMS nach [171]*

In DSMS (siehe Abbildung 3.26 rechts) hingegen installiert der Nutzer sogenannte kontinuierliche Anfragen. Die Verarbeitung kann dann entweder durch das Eintreffen von Elementen oder nach einem Zeitintervall erfolgen. Die zeit-getriebene Verarbeitung ist immer dann sinnvoll, wenn Datenströme keinen hohen Durchsatz haben und somit regelmäßig Ergebnisse produziert werden. Die element-getriebene Verarbeitung ist immer dann sinnvoll, wenn viele Elemente eintreffen und das DSMS einen hohen Durchsatz hat, da so die aktuellsten Ergebnisse garantiert werden [104].

Die Elemente werden im DSMS-Kontext von aktiven Datenquelle an das System zur Verarbeitung geschickt, was auch als *push*-Konzept bezeichnet wird. Die Daten verweilen im Arbeitsspeicher, da Lesen und Schreiben von bzw. auf die Festplatte für die Verarbeitung i.d.R. zu lange dauern würde. Allerdings kann das DSMS mit der Festplatte oder mit DBMS interagieren, um zum Beispiel aus IP-Adressen lesbare Namen zu generieren, die eine Interpretation der Daten durch den Nutzer leichter gestalten. Die Ausgaben aus einem DSMS werden wieder als Datenstrom zur Verfügung gestellt und eventuell wird ein Teil der Ergebnisse in einem DBMS persistiert. Werden die Anfragen nicht mehr benötigt, müssen sie explizit vom Nutzer aus dem System entfernt werden [171].

Insgesamt entsprechend DBMS dem *DBMS-passive,Human-active*-Modell und DSMS dem *DBMS-active,Human-passive*-Modell [104].

Alternativ kann ein DS auch aus heterogenen Objekten bestehen. Dabei hat jedes Element eigene Attribute und das DSMS klassifiziert die Elemente anhand ihrer Attribute [138]. Im Bereich der DBMS ist dies mit NoSQL-Systemen zu vergleichen: Der Umgang mit heterogenen Daten wird deutlich erleichtert, da diese nicht in ein allgemeines Schema gebracht werden müssen, aber die Verarbeitung

wird komplexer und dadurch erhöht sich die Latenz. Da Latenz ein kritisches Qualitätsmaß ist, wird dieser Ansatz seltener umgesetzt als die Betrachtung eines Datenstroms durch ein relationales Schema [104].

Ein Problem der Analogie von DBMS und DSMS besteht in der Unendlichkeit von Datenströmen. Zum einen sind Ressourcen zur Verarbeitung in Form von CPU-Leistung und Speicher begrenzt und zum anderen müssen einige DBMS-Operationen wie zum Beispiel die Durchschnittsberechnung oder Joins angepasst werden, da diese durch die Unendlichkeit nicht unangepasst berechnet werden können [210].

3.3.1.2 Datenstromverarbeitung

Operatoren sind die Bausteine zur Verarbeitung von DS. Analog zur Aufteilung in DBMS, kann es auch in DSMS logische und physische Operatoren geben. Eine Anfrage wird in logischen Operatoren formuliert und durch physische Operatoren ausgeführt [171]. Sie können durch das PIPES-Modell dargestellt werden. Innerhalb des Modells gibt es drei Arten von Operatoren: Quellen besitzen keine Eingaben und erzeugen einen Datenstrom als Ausgabe, Operatoren erhalten einen oder mehrere Datenströme als Eingabe und geben einen Datenstrom als Ausgabe heraus und Senken erhalten einen Datenstrom als Eingabe, haben aber keine weiteren Ausgaben. Quellen sind im PIPES-Modell beliebige Sensoren oder andere Erzeuger, auf welche abstrahiert und uniform zugegriffen werden kann. Operatoren können beliebig miteinander vernetzt werden, da sie sowohl Datenströme akzeptieren als auch ausgeben und dienen zur Realisierung von Anfrageplänen. Senken dienen häufig zur Darstellung von Ergebnissen der Anfragepläne [47]. Im Smart Grid sind die Smart Meter Quellen und ein Netzleitsystem sind Senken, während die Anfragen aus Unterabschnitt 3.3.1 aus Operatoren aufgebaut werden.

In der Relationalen Algebra gibt es zum Beispiel „Selektion“ und „Projektion“. Diese werden als zustandslose Operatoren für Datenströme bezeichnet, da sie bei gleicher Eingabe immer die gleichen Ergebnisse produzieren und als mathematische Abbildung eines Datenstroms auf einen modifizierten Datenstrom gesehen werden können, die bei jedem Aufruf des Operators ausgeführt werden. Die Projektion wird in DSMS typischerweise als MAP-Operator bezeichnet [47], da Attribute eines Datenstroms auf Attribute eines sich ergebenen Datenstroms abgebildet werden. Die Selektion wird als FILTER-Operator bezeichnet [104], da Elemente für die Weiterverarbeitung gestrichen werden, wenn sie bestimmte Attribute nicht erfüllen.

Neben den zustandslosen Operatoren gibt es auch zustandsbehaftete Operatoren. Zu diesen gehört zum Beispiel der „Join“. Dieser versucht Verbundpartner zwischen zwei Datenströmen zu finden und speichert daher Elemente beider DS um für später eintreffende Elemente Partner zu finden. Die Ausgabe des Operators hängt vom Zustand des Operators ab, da ein unterschiedlicher Zustand bei gleichen Datenströmen andere Ergebnisse hervorbringt.

Zustandsbehaftete Operatoren in DSMS haben eine zentrale Eigenschaft zu erfüllen: Sie dürfen nicht blockieren. Dadurch, dass zustandslose Operatoren keine Elemente speichern, können sie auch nicht blockieren. Operationen in DBMS wie die Berechnung des Durchschnitts oder bestimmte Join-Implementierungen setzen voraus, dass es eine endliche Menge von Daten gibt, die durchlaufen wird. Im Fall des *Nested Loop Joins* wird das erste Element einer Relation betrachtet und die andere Relation wird vollständig nach Verbundpartnern durchsucht. Dies ist für Datenströme nicht möglich, da sie potentiell unendlich sind und damit die Verarbeitung blockiert wird. Daher ist es unerlässlich, dass Join-Implementierungen für Datenströme diesen nicht blockieren. Mögliche Implementierungen dafür

sind der *Ripple Join* [122], bei dem element-getrieben überprüft wird, ob es in einem Puffer einen Join-Kandidaten gibt. Alternativ bildet der *Hash-Join* Gruppen, sodass Elemente gleicher Hashgruppen Join-Kandidaten sind. Durch die Totalordnung der Datenströme kann in beiden Fällen festgestellt werden, dass bestimmte Elemente keine Join-Kandidaten mehr haben können, wodurch diese beim *Ripple Join* aus dem Puffer entfernt werden, während beim *Hash Join* die vollständige Join-Gruppe entfernt wird [104].

Für Aggregierungsfunktionen wie Durchschnitt oder Summe können partielle Aggregate benutzt werden, sodass die Berechnung der Ergebnisse inkrementell stattfindet und immer den aktuellen Zustand widerspiegeln, statt ein Endergebnis darzustellen. Dies wird auch als teilweise blockierend bezeichnet, da diese Operatoren inkrementell ein Ergebnis liefern, aber auch ein Endergebnis haben können. Zum Beispiel ist die Suche nach dem Element mit dem minimalen Ordnungsattribut des Datenstroms durch die Totalordnung mit dem ersten Element erfüllt. Auf der anderen Seite wird das maximale Element des Datenstroms immer das zuletzt betrachtete Element sein und sich damit bei jedem Eintritt eines neuen Elements ändern [104].

Allerdings kann ein Operator wie das kartesische Produkt nicht über den vollständigen Datenstrom gebildet werden, daher werden endliche Ausschnitte des Datenstroms betrachtet, in denen das kartesische Produkt angewandt werden kann.

Fenster

Als Fenster werden endliche Ausschnitte aus Datenströmen bezeichnet. Ein Fenster kann bspw. über die Anzahl der Elemente oder über einen bestimmten Zeitraum definiert sein [104]. Dadurch haben DSMS in der Verarbeitung im Gegensatz zu DBMS auch einen expliziten zeitlichen Aspekt.

Fenster schreiten zusammen mit dem Datenstrom fort, d. h. neben der Größe wird bei Fenstern angegeben wie sie sich über Zeit zum Datenstrom verhalten. Ein Fenster wird durch die Art charakterisiert wie es sich bei fortschreitendem Datenstrom verhält. Abbildung 3.27 gibt einen Überblick über die verschiedenen Fensterarten und ihrem Verhalten beim Eintreffen neuer Elemente [104].

Das *landmark window* hat einen fest definierten Startpunkt und wächst mit dem Fortschreiten des Datenstroms. Mit einem *landmark window* lassen sich Anfragen wie „Ermittle die Durchschnittstemperatur seit dem 18.03.2017“ realisieren. Für die Implementierung des Fensters werden normalerweise logarithmische Annäherungen der Elemente vorgenommen, damit der Speicherverbrauch des Fensters minimiert wird [104].

Bei einem *sliding window* gibt es weder einen festen Start- noch einen festen Endpunkt. Mit Fortschreiten des Datenstroms ändern sich sowohl Start- als auch Endpunkt. Wenn das Fenster fortschreitet, entfällt typischerweise nur ein Teil der Elemente, d. h. Fenster teilen sich einige Elemente des Datenstroms. Eine Anfrage mit einem *sliding window* wäre „Ermittle die Durchschnittstemperatur der letzten 5 Minuten“ [104].

Wenn sich zwei *sliding windows* beim Fortschreiten keine Elemente mehr teilen, dann werden sie als *tumbling window* bezeichnet, sofern immer noch alle Elemente des Datenstroms in den Fenstern berücksichtigt werden. Dies entspricht Anfragen vom Typ „Ermittle alle 5 Minuten die Durchschnittstemperatur für die letzten 5 Minuten“ [104].

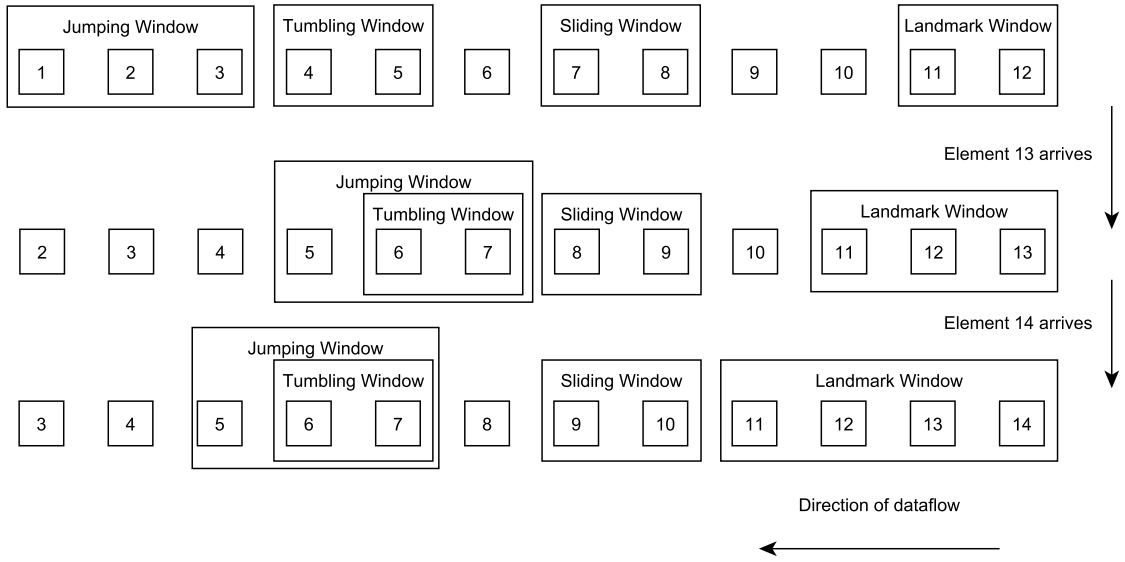


Abbildung 3.27: *Verschiedene Fensterarten nach [104] und [111]*

Wenn beim Fortschritt von *tumbling windows* Elemente übersprungen werden, d. h. der Datenstrom wird nicht mehr vollständig von den Fenstern erfasst, dann werden diese Fenster als *jumping window* bezeichnet. Diese werden in Anfragen wie „Ermittle alle 15 Minuten die Durchschnittstemperatur der letzten 5 Minuten“ benutzt [111].

Alle Fenster können allerdings einen Speicherüberlauf verursachen, da sie entweder viele oder große Elemente speichern können.

Intervall-Ansatz

Der Intervall-Ansatz ist eine Möglichkeit in der Anfrageverarbeitung mit Fenstern umzugehen. Dabei erhält jedes Element im Datenstrom durch das Fenster ein halboffenes Gültigkeitsintervall $[t_s, t_e)$. Die untere Grenze bestimmt sich i.d.R. durch den Zeitstempel des Elements, während die obere Grenze i.d.R. durch das Fenster bestimmt wird. In einem zeit-basierten Fenster ist dies eine Addition auf die untere Grenze. Das Fenster selber besitzt dafür auch ein Gültigkeitsintervall, welches durch neue Elemente fortschreitet [173]. Im Folgenden werden die einzelnen Fensterarten als zeit-basierte Fenster mit dem Intervall-Ansatz betrachtet.

In Abbildung 3.27 ist beim zeit-basiertem *landmark-window* ein Startzeitstempel gesetzt. Alle Elemente, die einen Zeitstempel haben, der größer als der Startzeitstempel ist, befinden sich im Fenster. Wenn die Elemente 13 und 14 am DSMS ankommen, dann müssen sie durch die Ordnung des DS einen größeren Zeitstempel als das Element 12 haben und befinden sich dadurch im Fenster.

Beim *sliding window* entfällt das Element 7 aus dem Fenster, wenn Element 13 ankommt, da die Gültigkeitsintervalle des Fensters und die des Elements sich nicht mehr überschneiden. Auf die obere und untere Grenze des Gültigkeitsintervalls des Fensters wird ein Wert addiert, der kleiner als die Fenstergröße ist. Dadurch ist der Startzeitstempel von Element 9 kleiner oder gleich der oberen Grenze

und befindet sich dadurch im Fenster. Das *sliding window* ordnet dem Element eine obere Grenze zu, die aus der Addition der Fenstergröße mit dem Zeitstempel des Elements entsteht [172].

Das *tumbling window* verhält sich genauso wie das *sliding window*, nur das auf die Grenzen des Gültigkeitsintervalls genau die Fenstergröße addiert wird. Ebenso verhält es sich beim *jumping window*, nur das hier mit einem Wert addiert wird, der größer als die Fenstergröße ist.

Alternativ dazu kann der Positiv-Negativ-Ansatz betrachtet werden. Hier erhält jedes Element durch das DSMS zwei Zustände: Der +-Zustand gibt an, dass das Element neu hinzukommt und der --Zustand gibt an, dass das Element nicht mehr gültig ist. Dadurch wird streng genommen aber jedes Element zwei Mal verarbeitet und alle Operatoren müssen die Zustände berücksichtigen, während die Nutzung von Intervallen im Allgemeinen intuitiver ist [173].

Darüber hinaus gibt es noch weitere Probleme, mit denen ein DSMS umgehen muss.

3.3.1.3 Probleme

Für gewöhnlich gilt in DSMS das *one-pass*-Paradigma. Anders als in DBMS, wo Elemente beliebig oft verarbeitet werden, werden in DSMS nach dem *one-pass*-Paradigma alle Elemente maximal einmal verarbeitet, da eine Mehrfachverarbeitung das *overload*-Problem verschlimmert und ein Zwischenspeichern zur Wiederverarbeitung durch den begrenzten Speicher nicht ratsam ist und ein Festplattenzugriff eine zu hohe Latenz erzeugt [104].

Sollte es trotzdem zur Überlastung kommen, kann ein DSMS nicht blockieren und die gesamte Last verarbeiten. Daher lässt das DSMS einen Teil der Elemente bei der Berechnung unberücksichtigt, wodurch die Ergebnisqualität schlechter wird. Wie ein DSMS die unberücksichtigten Elemente auswählt, wird über sogenannte *load-shedding*-Algorithmen bestimmt [111].

Ein zusätzliches Problem stellen die Datenquellen dar, da ein Datenstrom total geordnet sein soll. Die Quellen können Elemente außerhalb ihrer Reihenfolge an das DSMS schicken (*out-of-order tuple*) oder geben keine explizite Ordnung der Elemente an oder vor. Durch eine explizite Ordnung ist es dem DSMS möglich eine *out-of-order* Verarbeitung durchzuführen, wenn das Tupel noch innerhalb eines Puffers eingeordnet werden kann. Wenn keine explizite Ordnung vorhanden ist, dann stellt das DSMS eine implizite Totalordnung her, die in der Regel von der Ankunftszeit der Elemente am DSMS abhängt [104].

Die Datenerzeugung in Datenströmen ist unstetig. Es können augenblicklich viele Elemente (*overload*) oder über lange Zeit keine Elemente (*starvation*) erzeugt werden. Ein DSMS muss daher sowohl mit *overload* als auch mit *starvation* durch entsprechende Strategien umgehen können, da bei DSMS der Zeitfortschritt durch Datenströme und nicht durch die Systemzeit festgestellt wird. Für zeit-basierte Fenster lösen DSMS dies durch Zeitstempel-Elemente, sogenannte *punctuations*, die keine Daten enthalten. Dies passiert entweder auf Basis von Ereignissen oder nach einem bestimmten Zeitintervall. Wie groß das Zeitintervall ist, ist immer anwendungsabhängig, da dies zusätzliche Last für das DSMS bringt. Dadurch wird das Problem gelöst, dass zeitbasierte Fenster keine Ergebnisse in einer datengetriebenen Verarbeitung erzeugen, da *punctuations* auch nicht durch Filter aus dem Anfragebaum entfernt werden [104].

3.3.1.4 Zusammenfassung

Die Verarbeitung von großen Datenmengen soll zeitnah Ergebnisse produzieren, da ein verspätetes Ergebnis weniger Gewicht besitzt. Im Smart Grid erzeugen Smart Meter große Datenmengen, durch die eine Überlastung oder ein Zusammenbruch des Stromnetzes verhindert werden kann. Dies ist aber nur möglich, wenn zeitnah auf Ereignisse durch Personal reagiert werden kann [249].

Dafür sind herkömmliche DBMS nicht anwendbar, da diese die gesamte Datenmenge speichern und dann verarbeiten. Die Sensordaten sind allerdings potentiell unendlich und die Latenz zur Ergebnisproduktion ist für kritische Szenarien nicht anwendbar [112].

Daher wurde das Datenstrommodell aufgestellt, das die Verarbeitung von potentiell unendlichen Datenmengen durch DSMS ermöglichen soll. In diesem Abschnitt wurde das grundlegende Datenstrommodell betrachtet und mit dem relationalen Modell von DBMS verglichen. Dazu wurden Konzepte beschrieben, die von DBMS übernommen werden konnten, aber auch neue Konzepte eingeführt, die bei der Verarbeitung von Datenströmen auftretenden Probleme lösen sollen [111].

Dazu zählt die potentielle Unendlichkeit von Datenströmen, auf denen lang laufende Anfragen mit endlichen Ressourcen ausgeführt werden [210]. Dafür wurde die Fensterbehandlung durch den Intervall-Ansatz näher betrachtet, der jedem Element und jedem Fenster einer Anfrage eines Datenstroms eine Gültigkeitsdauer zuteilt und somit eine intuitive Verarbeitung von Fenstern ermöglicht [104].

3.3.2 Data Mining auf Datenströmen

Die Menge digital verfügbarer Daten unterliegt einem exponentiellen Wachstum. Eine Studie der EMC Corporation aus dem Jahr 2012 ergab, dass das Datenvolumen im Zeitraum von 2005 bis 2020 um den Faktor 300 wachsen wird, was 2020 einem Datenvolumen von über 5200 Gigabyte pro Person entspricht [103]. Um mit dieser Fülle an Daten umgehen zu können, müssen sie aufbereitet und analysiert werden. Diese Aufgabe der gezielten Extraktion von Wissen aus einer Datenmenge wird Data Mining genannt. Das Data Mining bedient sich verschiedener Konzepte und Technologien aus dem Bereich des Machine Learning und bezeichnet ihre spezielle Anwendung bei der Wissensgewinnung aus großen Datenmengen. Drei Kernbereiche des Data Mining sind Clustering, Frequent Pattern Mining und das Lernen von Klassifikatoren für Vorhersagemodelle [197].

In den vergangenen Jahren hat ein neuer Anwendungsbereich für das Data Mining an Bedeutung gewonnen. Die zu analysierenden Daten liegen häufig nicht mehr gespeichert in einer Datenbank vor, sondern befinden sich auf einem kontinuierlichen Datenstrom. Dadurch ergeben sich neue Herausforderungen, die sich mit den klassischen Data Mining Algorithmen nicht ohne Weiteres lösen lassen. Datenströme haben häufig einen hohen Datendurchsatz und sind potentiell unendlich, was ein Speichern des kompletten Stroms unmöglich macht. Als Konsequenz darf häufig nur ein Mal auf jedes Datenelement zugegriffen werden oder es wird ein Teil des Datenstroms, ein sogenanntes Fenster zwischengespeichert, auf das beliebig zugegriffen werden kann. Hieraus haben sich zwei Klassen von Algorithmen ergeben, die für das Data Mining auf Datenströmen verwendet werden. Zum einen Online-Algorithmen, die auf jedes Datenelement nur ein Mal zugreifen und ihre Berechnungen inkrementell durchführen und zum anderen fensterbasierte bzw. intervallbasierte Algorithmen, die ähnlich zu den klassischen Algorithmen auf einer festen Datenmenge agieren [197]. Im Folgenden wird zuerst auf die Möglichkeiten und Anwendungsbereiche des Data Mining eingegangen und grundlegendes

theoretisches Vorwissen erläutert. Danach werden die genannten Kernkonzepte vorgestellt und mit Beispielalgorithmen veranschaulicht.

3.3.2.1 Data Mining

In der heutigen Gesellschaft werden fast alle Daten digital gespeichert. Die Möglichkeiten Daten zu sammeln und zu speichern entwickeln sich schnell weiter. Betriebe werden digitalisiert und produzieren riesige Mengen an Daten pro Tag, wie zum Beispiel Transaktionsdaten durch Verkäufe, Börsendaten, Forschungsdaten oder personenbezogene Daten. Data Mining stellt die Werkzeuge bereit, mit denen diese Flut an Daten ausgewertet werden kann. Dazu gehört unter anderem die Aggregation von Daten um Metadaten zu erstellen, die einen besseren Überblick über die Gesamtheit der Daten geben. Statistische Methoden werden eingesetzt, um die Datenmenge zu charakterisieren. Dazu gehören Durchschnittswerte, Häufigkeitsanalysen, Ausreißeranalysen und viele weitere statistische Methoden und Kennwerte, die aus einer Menge von Daten Wissen herausziehen. Data Mining versucht auch Wissen aus den Zusammenhängen der Daten zu generieren, das nicht aus den einzelnen Datensätzen zu erkennen ist. Ein Beispiel hierfür ist das Erkennen von Grippewellen mittels Google Suchanfragen. Bestimmte Suchanfragen bilden ein Muster, das auf Grippe schließen lässt. Die Häufigkeit dieses Musters lässt danach Rückschlüsse auf die aktuelle Ausbreitung der Grippe zu. Hier wird durch Aggregation der Daten neues Wissen gewonnen, das in den Daten alleine nicht enthalten ist. Ein weiteres Anwendungsbeispiel ist die Warenkorbanalyse in Onlineshops. Hier werden häufige Muster, auch Frequent Pattern genannt, aus den Warenkorbzusammenstellungen analysiert und für Cross-Selling Zwecke genutzt. Weitere Anwendungsbeispiele finden sich in fast jeder Branche, da durch die Digitalisierung überall große Mengen an Daten anfallen [126, S. 1-5].

Im Folgenden werden einige theoretische Grundlagen, die für das Data Mining benutzt werden, eingeführt. Sie bilden die Grundlage für die Folgekapitel und tragen damit zum Verständnis bei.

Hoeffding Bound

Der Hoeffding Bound ist eine statistische Kenngröße, die vor allem beim Data Stream Mining von Bedeutung ist. Er gibt eine obere Schranke der Abweichung des Stichprobenmittels vom tatsächlichen Mittelwert unter Verwendung eines Vertrauensniveaus an. Für die Berechnung des Hoeffing Bounds ε sind lediglich die Anzahl der Elemente n, der mögliche Wertebereich R dieser Elemente und das Vertrauensniveau δ notwendig. Der Hoeffding Bound wird wie folgt berechnet:

$$\varepsilon = \sqrt{\frac{R^2 \ln(\frac{1}{1-\delta})}{2n}}$$

Mit dieser Schranke kann man beurteilen wie genau die Berechnungen der aktuellen Kenngrößen für den Datenstrom sind und trotz unendlicher Datenmenge Vorhersagen mit fester Vertrauenswürdigkeit treffen [68]. Verwendung findet der Hoeffding Bound unter anderem bei der Change Detection (siehe Unterunterabschnitt 3.3.2.2).

Online Algorithmen

Online Algorithmen versuchen das Problem zu lösen, dass die Daten zu Beginn des Algorithmus nicht alle vorliegen, sondern inkrementell geliefert werden. Oft wird auch davon ausgegangen, dass auf ein Datenelement höchstens einmal zugegriffen werden darf. Sie generieren bei jedem Eintreffen eines

Datenelements einen neuen Output und liefern so inkrementell Ergebnisse. Häufig benutzen Online Algorithmen besondere Datenstrukturen um aggregierte charakteristische Größen der Datenströme zu speichern, die für die Ausführung des Algorithmus notwendig sind [6]. Ein Beispiel für einen Online Algortithmus ist das inkrementelle Berechnen des arithmetischen Mittels bei einem Datenstrom mit numerischen Daten. Anstatt alle Elemente zu speichern wird eine Datenstruktur definiert, die die Anzahl der gelesenen Elemente und deren Summe speichert. Der Output erigbt sich als Quotient der Summe und der Anzahl. Beim Eintreffen eines weiteren Datenelements wird die Datenstruktur aktualisiert und ein neuer Output geliefert.

3.3.2.2 Change Detection

Change Detection in Datenströmen befasst sich mit dem Erkennen statistisch signifikanter Änderungen der Daten im Datenstrom. Eine solche Änderung wird Concept Change genannt, wobei das Concept die Variable der Daten ist, die vorhergesagt werden soll. Wenn sich diese Variable signifikant ändert muss in der Regel auch das Vorhersagemodell geändert werden. Wenn zum Beispiel der Stromverbrauch einer Maschine beobachtet wird und diese Maschine vom laufenden Betrieb in einen Energiesparmodus wechselt, ändert sich der Stromverbrauch. In den Daten wäre dies als Concept Shift zu erkennen. Concept Shift wird für eine abrupte Änderung, Concept Drift für eine graduelle Änderung der Größe verwendet [30, Kap. 2.2]. Im Folgenden wird kurz eine Methode der Change Detection beschrieben, die für das Erkennen der Veränderung auf den Hoeffding Bound aufbaut.

Die hier beschriebene Methode der Change Detection beruht auf einem adaptiven Fenster, woraus sich auch ihr Name ADWIN (**Ad**aptive **Win**dow) ableitet. ADWIN startet mit einem leeren Fenster und fügt dem aktuellen Fenster so lange Datenelemente hinzu, bis es einen Concept Change erkennt. Die Fenstergröße wird maximiert und ein Fenster enthält alle Daten eines Konzepts. Beim Eintreffen eines neuen Datenelements wird dieses zuerst dem aktuellen Fenster hinzugefügt und für jeden möglichen Split Point des Fensters das harmonische Mittel

$$m = \frac{2}{\frac{1}{|W_0|} + \frac{1}{|W_1|}}$$

der beiden Fensterteile W_0 und W_1 berechnet. Danach wird mit Hilfe des Hoeffding Bounds

$$\varepsilon_{cut} = \sqrt{\frac{1}{2m} \ln\left(\frac{4|W|}{\delta}\right)}$$

berechnet. ε_{cut} gibt die maximale Abweichung an, die die Mittelwerte $\hat{\mu}_{W_0}, \hat{\mu}_{W_1}$ der Fenster W_0 und W_1 voneinander haben dürfen, um unter dem gegebenen Vertrauensniveau δ noch zum gleichen Mittelwert zu gehören. Solange es Split Points gibt, für die die Differenz der arithmetischen Mittel der Fensterteile mehr als εcut voneinander abweicht,

$$|\hat{\mu}_{W_0} - \hat{\mu}_{W_1}| > \varepsilon_{cut}$$

wird das älteste Datenelement aus dem Fenster entfernt. Diese Methode Concept Changes zu erkennen benutzt das Vertrauensniveau, das durch den Hoeffding Bound gegeben ist. Es wird festgestellt, ob das aktuelle Datenelement zum gegebenen Vertrauensniveau zum gleichen Konzept gehören kann, wie die restlichen Fensterdaten. Wird diese Bedingung nicht erfüllt, werden so lange die ältesten Elemente aus dem Fenster entfernt, bis alle Daten wieder zum gleichen Konzept gehören. Diese Methode beinhaltet eine Verzögerung beim Erkennen des Concept Change, da erst einige Daten, die nicht zum Konzept

gehören, gelesen werden müssen, damit die Bedingung an die Mittelwerte nicht mehr erfüllt ist. Ein Vorteil dieser Methode ist jedoch, dass das Fenster nach einem Concept Change fast vollständig geleert wird, sodass das neue Konzept in einem fast leeren Fenster beginnt [30, Kap. 4].

3.3.2.3 Supervised Learning

Supervised Learning ist ein Teilbereich des Machine Learning und befasst sich mit dem Lernen von Modellen, die zur Vorhersage von bestimmten Eigenschaften der Datenelemente genutzt werden. Charakteristisch für das Supervised Learning ist ein Trainingsdatenset, für das die Größe, die vorherzusagen ist, bereits bekannt ist. Unsupervised Learning Methoden hingegen benötigen keine Trainingsdaten. Wenn Supervised Learning Vorhersagemodelle diskrete Vorhersagen treffen, heißen sie Klassifikatoren und weisen einem Datenelement ein sogenanntes Label zu. Alle Datenelemente mit dem gleichen Label bilden eine Klasse. Ein Beispiel für einen Klassifikator ist, Kreditanträge in die Klassen „sicher“ und „unsicher“ aufzuteilen. Ist die Vorhersage kontinuierlich heißt das Vorhersagemodell Regressor, da es eine Regression auf den Daten ausführt, um ihnen einen kontinuierlichen Wert zuzuweisen. Ein Beispiel für die Verwendung eines Regressionsmodells ist die Vorhersage des zu erwartenden Gewinns pro Kunde bei einer Verkaufsaktion [126, Kap. 8]. Im Folgenden wird näher auf die Klassifikatoren eingegangen. Die Konzepte lassen sich jedoch häufig auch für Regressoren anwenden, auch wenn die mathematischen Verfahren variieren.

Die Klassifikation von Daten lässt sich in zwei Schritte unterteilen. Zuerst muss der Klassifikator mit Hilfe von Trainingsdaten trainiert werden (Learning Step). Danach kann er im zweiten Schritt zum Klassifizieren von Daten verwendet werden (Classification Step). In der Trainingsphase wird ein Klassifikationsalgorithmus verwendet, um einen Klassifikator zu lernen. Hierbei lernt er von einem Trainingsdatenset, bei dem die Label vorher bekannt sind. Der Algorithmus stellt Verbindungen zwischen den Attributen der Datenelemente und ihrem Label her und erstellt so Entscheidungsbäume oder ähnliche Datenstrukturen, die für die Klassifikation benutzt werden. Die Attribute der Datenelemente werden im Machine Learning Kontext Features genannt. Der Klassifikator kann als eine Abbildung $y = f(\mathbf{X})$ gesehen werden, wobei y das Label und $\mathbf{X}$ ein Vektor mit den Features des Datenelements ist [126, Kap. 8]. Wenn der Klassifikator gelernt wurde, kann er zur Prädiktion von Labels benutzt werden. Es stellt sich jedoch die Frage, wie gut der Klassifikator die Labels vorhersagt. Um diese Frage zu beantworten wird der Klassifikator benutzt, um die Labels von anderen Datenelementen vorherzusagen, für die das Label bereits bekannt ist. Für diesen Schritt sollen die Trainingsdaten vom Lernen des Klassifikators nicht wiederverwendet werden, da dieser in der Regel an die Trainingsdaten *overfitted* ist. Das bedeutet, dass der Klassifikator eventuell Anomalien aus dem Trainingsset erlernt hat, die in der Gesamtdatenmenge nicht vorkommen. Die Genauigkeit der Vorhersage kann danach mit Hilfe von statistischen Kenngrößen angegeben werden. Zwei dieser Kenngrößen sind Precision und Recall. Für ihre Berechnung benötigt man die Anzahl der korrekt und nicht korrekt vorhergesagten Label: True-Positive TP, False-Positive FP, False-Negative FN und True-Negative TN. Nun kann man die Präzision $Precision = \frac{TP}{TP+FP}$ berechnen, die den Anteil der wirklich zu diesem Label gehörenden Daten an der Gesamtmenge der Daten, die diesem Label zugeordnet worden sind, angibt. Des Weiteren kann man den Recall berechnen $Recall = \frac{TP}{TP+FN}$, der angibt wie viele Daten, die eigentlich zu diesem Label gehören, nicht erkannt worden sind. Ist die Genauigkeit ausreichend kann der Klassifikator nun benutzt werden, um neue Daten mit unbekanntem Label zu klassifizieren. Andernfalls sollten weitere oder andere Trainingsdaten zur Verbesserung des Klassifikators benutzt werden [126, Kap. 8].

Die Anwendung des gelernten Klassifikators auf einen Datenstrom ist unproblematisch, da jedes neu ankommende Element den Entscheidungsbaum durchlaufen kann und dementsprechend klassifiziert wird. Beim Lernen des Klassifikators treten jedoch Probleme auf, da viele Algorithmen zur Berechnung der Split Points im Entscheidungsbaum die relativen Häufigkeiten der Ausprägungen der bisher gesehen Features benötigen. Die relativen Häufigkeiten ändern sich im Datenstrom mit der Zeit, da weitere Elemente gelesen werden. Der Algorithmus kann also keinen Split ausführen sobald der Split an einem bestimmten Attribut den größten Informationsgewinn bringt, da die Informationen auf denen diese Annahme beruht, nicht sicher sind. Um hier ausreichende Sicherheit zu schaffen, wird der Hoeffding-Bound ε verwendet. Dadurch kann eine Schranke $G(X_a) - G(X_b) > \varepsilon$, wobei $G(X)$ den Informationsgewinn beim Split von Attribut X angibt, berechnet werden, die mit einer gegebenen Sicherheit angibt, dass der Informationsgewinn beim Split höchstens um ε vom wahren Informationsgewinn abweicht. Damit der Informationsgewinn beim Split des Attributs X

$$G(X) = wgain(T,P) = entropy(T) - \sum_{i=1}^{m} \frac{|P_i|}{|T|} entropy(P_i)$$

$$entropy(T) = -\sum_{i=1}^{n} p_i \log(p_i)$$

mit Grundmenge T und Partitionen P_i, berechnet werden kann, müssen Informationen über die Trainingsdaten gespeichert werden. Für jede Ausprägung jedes Attributs muss pro Label ein Zähler angelegt werden. Beim Eintreffen eines Datenelements werden die entsprechenden Zähler inkrementiert und der Informationsgewinn der beiden Attribute mit dem maximalen $G(X)$ verglichen. Wird die Schranke $G(X_a) - G(X_b) > \varepsilon$ überschritten, wird an dem Attribut mit dem maximalen $G(X)$ gesplitted. Nach dem Split werden die Zähler zurückgesetzt. Bereits aufgeteilte Attribute müssen nicht mehr betrachtet werden. Sind alle Attribute nach diesem Schema aufgeteilt, wird jedem Blatt im so entstanden Entscheidungsbaum das Label zugeordnet, das die höchste Zählersumme erhalten hat. Ein Algorithmus, der in etwa nach diesem Schema arbeitet, ist der Hoeffding-Tree-Algorithmus [140].

3.3.2.4 Unsupervised Learning

Clustering

Eine Methode Wissen aus einer Datenmenge zu gewinnen ist es, die Daten strukturiert zu gruppieren, so dass ähnliche Datenelemente der gleichen Gruppe zugeordnet werden bzw. unterschiedliche Daten nicht in einer Gruppe zusammengefasst werden. Diese Gruppen werden auch Cluster genannt und sind das Ergebnis von Clustering-Algorithmen. Eine gute Aufteilung erkennt man an einer niedrigen intracluster Varianz und einer hohen intercluster Varianz. Der Grad der Ähnlichkeit wird mit Hilfe der Attributwerte bestimmt und basiert häufig auf einem Distanzmaß. Clustering wird außerdem im Bereich des Machine Larning zur automatischen Klassifikation von Daten verwendet, ohne vorher einen Klassifikator gelernt zu haben oder einen Trainingsdatensatz zu benötigen. Diese Form der Klassifikation gehört zum Unsupervised Learning, da die Clusterbildung ohne Aufsicht oder Eingreifen geschieht. Ein Anwendungsbeispiel ist das Erkennen bzw. Klassifizieren von handgeschriebenen Ziffern. Ohne eine Trainingsmenge kann ein Clustering-Algorithmus die Ähnlichkeiten in den verschiedenen Darstellungen der einzelnen Ziffern erkennen und einem Cluster zuordnen. Außerdem dienen Cluster häufig als Startpunkt für andere Machine Learning Algorithmen. Um zum vorherigen Beispiel zurückzukehren könnten verschiedene Darstellungen der gleichen Ziffer unterschieden werden, um dann einen Klassifikator besser trainieren zu können. Des Weiteren können Ausreißer im

Datensatz erkannt werden. Ausreißer haben wenig Ähnlichkeit zu allen anderen Clustern und können daher nicht zugeordnet werden. Outlier Detection wird zum Beispiel verwendet um ungewöhnliche Kreditkartenaktivitäten zu erkennen, wie etwa unregelmäßige, hohe Transaktionen.

Im Allgemeinen werden Clustering-Algorithmen in vier Kategorien eingeteilt. Diese Kategorien sind partitionsbasierte, hierarchische, dichtebasierte und gitterbasierte Algorithmen. Partitionsbasierte Clustering-Algorithmen unterteilen die Daten in eine feste Anzahl von Clustern, die vor dem Beginn des Algorithmus festgelegt wird. Hierbei wird jedem Cluster mindestens ein Datenelement zugewiesen und kein Datenelement kommt in mehreren Clustern vor. Sie basieren in der Regel auf Distanzmaßen und versuchen die Cluster so zu bilden, dass alle Elemente innerhalb des Clusters möglichst nah zusammen und verschiedene Cluster weit auseinander sind. Diese Zuordnung wird meisten iterativ wiederholt, bis ein Gütekriterium erfüllt ist. Ein Vertreter diese Kategorie ist K-Means, der in einer abgewandelten, für Datenströme angepassten Version später in diesem Kapitel vorgestellt wird. Die zweite Kategorie, hierarchische Algorithmen, agglomeriert oder teilt Cluster in jeder Iteration. Am Anfang ist entweder jedes Datenelement ein eigenes Cluster (bottom-up) oder alle Datenelemente sind im gleichen Cluster (top-down). Der Algorithmus teilt oder agglomeriert so lange, bis ein Gütekriterium erfüllt ist oder kein weiterer Schritt mehr möglich ist. Eine weitere Kategorie von Clustering-Algorithmen sind die dichtebasierten Algorithmen. Sie erweitern Cluster so lange, bis nicht genug Punkte in der Nachbarschaft sind. Hat ein Datenpunkt genügend Nachbarn, zählt er als Kernpunkt, hat er einen Kernpunkt in der Nachbarschaft, aber nicht genug Nachbarn um selbst ein Kernpunkt zu sein, zählt er als Randpunkt und gehört noch zum Cluster. Datenpunkte, die keine Kernpunkte in der Nachbarschaft haben, gelten als Ausreißer und werden keinem Cluster zugeordnet. Die letzte hier vorgestellte Kategorie sind die gitterbasierten Clustering-Algorithmen. Diese Algorithmen unterteilen den Datenraum in Zellen, auf denen alle Operationen durchgeführt werden. Sie können mit anderen Clustering-Algorithmen verbunden werden, um die Performanz des Algorithmus zu erhöhen, da ihre Laufzeit lediglich von der Anzahl der Zellen im Gitter abhängt und nicht von der eigentlichen Anzahl an Datenelementen [126, Kap. 10].

Clustering-Algorithmen wurden vor dem Aufkommen von Datenströmen entwickelt und arbeiten normalerweise auf einer festen Datenmenge. Dies führt zu Problemen, wenn Daten aus einem Datenstrom gruppiert werden sollen, da sie am Anfang des Algorithmus nicht alle verfügbar sind und potenziell unendlich sind. Das heißt, selbst wenn ein Algorithmus mit inkrementeller Datenbereitstellung arbeiten kann, treten eventuell Probleme auf, wenn zu viele Datenelemente gespeichert werden müssen. Die Algorithmen müssen daher für die Verwendung auf Datenströmen angepasst werden. Ein Hilfsmittel, das hier vorgestellt wird, ist der Cluster Feature Vector (CFV), der ein Cluster in wenigen Kenngrößen zusammenfasst und nicht alle Datenelemente des Clusters einzeln speichert. Typische gespeicherte Kenngrößen sind die Anzahl der Elemente im Cluster sowie ihre lineare und quadratische Summe. Der Cluster Feature Vektor ist ein Tupel der Form $CFV = (\overline{SS}, \overline{LS}, n)$. Diese Kenngrößen werden zur Ausführung des Algorithmus genutzt und ermöglichen so eine weitaus größerer Menge an Datenelementen zu clustern ohne Speicherprobleme zu bekommen [2, Kap. 7.3.2].

Als Beispiel für einen Clustering-Algorithmus, der für Datenströme angepasst wurde, wird nun Single Pass K-Means vorgestellt. Für die Speicherung der Cluster wird der Cluster Feature Vektor verwendet. Die zentrale Idee des Algorithmus ist es, einen Puffer zu benutzen, in dem Datenelemente komprimiert gespeichert werden. Zuerst wird festgelegt in wie viele Cluster der Datensatz unterteilt werden soll und zufällige Clustermittelpunkte generiert. Dann wird der vorhandene Speicherplatz des Puffers mit Datenelementen gefüllt. Diese Punkte werden wie beim normalen K-Means dem Cluster zugeordnet, dem sie am nächsten sind. Als nächstes wird der Pufferinhalt beim einfachen

Verfahren in einem Schritt und beim verbesserten Verfahren in zwei Schritten komprimiert. Die primäre Kompression ermittelt die Punkte, die mit hoher Wahrscheinlichkeit bei weiteren Iterationen des Algorithmus dem gleichen Cluster zugeordnet bleiben. Hierfür kann z. B. der Abstand des jeweiligen Punktes zum Clustermittelpunkt gewählt werden oder mit Hilfe von statistischen Verfahren ein Worst-Case Szenario simuliert werden, indem die Entfernung zum aktuellen Clustermittelpunkt vergrößert und die Entfernung zu allen anderen Mittelpunkten verringert wird. Wenn der Punkt immer noch dem gleichen Cluster zugeordnet werden würde, gilt er als sicher. Die Punkte, die bei der primären Kompression ermittelt wurden, werden aus dem Puffer entfernt und in sogenannte Discard Sets verlagert. Diese Sets werden wiederum durch CFV repräsentiert. Jedes Cluster hat genau ein Discard Set, das ihm zugeordnet ist. Die sekundäre Kompression wird in dieser einfachen Form des Single Pass K-Means nicht betrachtet. In der nächsten Iteration zählt jedes Discard Set als ein Punkt (Mittelpunkt des Sets). Es wird jedoch mit der Anzahl der Punkte im Set gewichtet. Wenn der Algorithmus mit den Daten im Puffer konvergiert ist, also kein Punkt einem neuen Cluster zugeordnet wurde, werden die Discard Sets mit den anderen Punkten, die diesem Cluster zugewiesen sind aktualisiert, also ihre lineare und quadratische Summe neu berechnet sowie die Anzahl der Punkte aktualisiert und die Punkte aus dem Puffer entfernt. Wenn noch weitere Datenpunkte vorhanden sind, werden sie nun in den Puffer geladen und der Algorithmus wiederholt sich wie beschrieben. Sind keine Punkte mehr vorhanden terminiert der Algorithmus [90].

Frequent Pattern Mining

Frequent Pattern werden vor allem in Recommender-Systemen verwendet, also in Systemen, die, basierend auf vergangenen Transaktionen, Vorschläge für zukünftige Transaktionen machen. Mit ihrer Hilfe lassen sich Assoziationen und Korrelationen zwischen Datenelementen finden. Des Weiteren helfen sie bei der Klassifikation und beim Clustering und sind somit ein wichtiges Werkzeug beim Data Mining.

Ein Frequent Pattern ist ein Muster, das häufig in dem Datensatz vorkommt. Es werden generell zwei Arten von Mustern unterschieden. Zum einen die frequent itemsets, die Muster bezeichnen, bei denen die Datenelemente häufig in der gleichen Transaktion vorkommen und zum Anderen frequent sequential patterns, die eine häufige Abfolge von Datenelementen in verschiedenen Transaktionen bezeichnen. Ein frequent itemset könnten Brot und Aufschnitt beim Einkaufen sein. Sie werden häufig in der gleichen Transaktion gekauft. Ein Beispiel für ein frequent sequential pattern könnte der Kauf eines Computers mit anschließendem Kauf von einem Drucker und Kopfhörern sein. Diese Items sind über mehrere Transaktionen hinweg verbunden. Ein Muster, das beide vorangegangenen Arten beinhaltet, wird als structured pattern bezeichnet [126, Kap. 6].

Im Folgenden wird Apriori als Frequent Pattern Mining Algorithmus kurz vorgestellt und danach auf die Besonderheiten eingegangen, die sich durch FPM auf Datenströmen ergeben. Vorher werden die statistischen Kenngrößen Support und Confidence eingeführt, die bei der Ausführung des Algorithmus notwendig sind.

Support und Confidence beschreiben einen statistischen Zusammenhang zwischen zwei vorkommenden Datenelementen. Hierbei bezeichnet Confidence die Wahrscheinlichkeit, dass, wenn ein bestimmtes Datenelement auftritt, ein anderes bestimmtes Datenelement in der gleichen Transaktion vorhanden ist. Zum Beispiel beim Einkauf eines Computers wird in 50% der Fälle auch Software gekauft. Damit ergibt sich eine Confidence zwischen dem Kauf eines Computers und dem Kauf von Software von 50%. Der Support bezeichnet den Anteil der Transaktionen, in denen beide Datenele-

mente gemeinsam vorkommen an der Gesamtmenge der Transaktionen. Wenn im vorherigen Beispiel in 1% der Transaktionen Computer und Software gekauft wird, ergibt sich ein Support von 1% [126, S. 17-18]. Diese beiden Kennwerte werden zum Filtern von Mustern im Frequent Pattern Mining benutzt. Nur Muster, die eine bestimmte Schwelle beim Support und der Confidence überschreiten, gelten als häufige Muster.

Der erste Schritt beim Apriori Algorithmus ist es, die häufigen 1-itemsets L_1 zu finden. Dafür wird die Datenbank nach Datenelementen durchsucht, die den minimalen Support erfüllen. Danach werden aufbauend auf L_1 die häufigen 2-itemsets L_2 ermittelt. Dieses Schema wird fortgesetzt, bis keine weiteren k-itemsets gefunden werden können. Um die Suche nach den k-itemsets zu verbessern wird die Apriori-Eigenschaft ausgenutzt. Diese besagt, dass alle Teilmengen einer häufigen Menge selbst häufig sein müssen $P(I) > minSup \Rightarrow P(A \subset I) > minSup$ und im Gegenzug alle Erweiterungen einer nicht häufigen Menge nicht häufig sein können $P(I) < minSup \Rightarrow P(I \cup A) < minSup$. Die Aufgabe ist es nun, von der itemset Menge L_{k-1} auf L_k mit $k \geq 2$ zu schließen. Hierfür wird ein Prozess mit zwei Schritten durchgeführt. Im join step wird eine Menge von Kandidaten C_k der Länge L_k generiert, indem die L_{k-1} Menge mit sich selbst vereinigt wird $L_{k-1} \bowtie L_{k-1}$. Hierbei können zwei Elemente l_1 und l_2 aus L_{k-1} vereinigt werden, wenn sie sich ausschließlich in ihrem letzten Element unterscheiden. Damit die Reihenfolge der Elemente in den itemsets eindeutig ist, wird eine lexikographische Ordnung vorausgesetzt. Die itemsets l_1 und l_2 werden dann zu einem itemset der Länge k zusammengefügt, indem alle Elemente inklusive des vorletzten übernommen werden und die letzten Elemente der beiden itemsets in ihrer lexikographischen Reihenfolge angehängt werden. Im prune step wird die Kandidatenmenge bereinigt. Die Menge C_k aus dem vorherigen Schritt ist eine Übermenge von L_k, deren itemsets nicht alle häufig sein müssen, die aber alle häufigen k-itemsets enthalten muss. Um die nicht häufigen itemsets herauszufiltern wird die Apriori-Eigenschaft genutzt. Es kann kein (k-1)-itemset geben, das nicht häufig ist und trotzdem eine Teilmenge von einem häufigen k-itemset ist. Andersherum wird nun geprüft, ob es Teilmengen der Länge k-1 der Kandidaten gibt, die nicht in L_{k-1} vorkommen. Ist dies der Fall wird der entsprechende Kandidat aus C_K entfernt. Sind alle Teilmengen in L_{k-1}, wird der Support des Kandidaten mit Hilfe der Datenbank ermittelt. Ist der Support hoch genug gehört der Kandidat zu L_k, ansonsten wird er verworfen [126, Kap. 6.2].

Arbeitet man auf Datenströmen, können die 1-itemsets nicht einfach gefunden werden, da der Datenstrom nicht durchsucht werden kann. Des Weiteren können nicht alle 1-itemsets gespeichert werden, da es potenziell unendlich viele geben kann. Außerdem muss für die Ermittlung des Supports für k-itemsets eine geeignete Datenstruktur angelegt und gespeichert werden. Die Ermittlung der 1-itemsets kann z. B. mit Hilfe des Frequent-Algorithmus erfolgen [161]. Eine geeignete Struktur für die Ermittlung des Supports der k-itemsets ist ein fensterbasierter FP-Baum [106]. Der Frequent Algorithmus speichert eine feste Anzahl k häufiger 1-itemsets, indem er zuerst k Zähler mit 0 initialisiert. Wenn ein neues Element aus dem Datenstrom ankommt, wird sein Zähler erhöht, falls es bereits einen Zähler hat. Falls es noch keinen Zähler hat, aber noch unbenutzte Zähler zur Verfügung stehen, wird es diesem Zähler zugeordnet. Falls es keinen Zähler hat und keine Zähler frei sind, werden alle Zähler um eins verringert und das entsprechende Element entfernt, falls der Zähler auf 0 ist. Mit dieser Methode können häufige 1-itemsets in Datenströmen ermittelt werden und Concept Changes in den Daten werden berücksichtigt, indem die alten 1-itemsets mit der Zeit vergessen werden [161]. Auf FP-Bäume wird hier nicht weiter eingegangen.

3.3.2.5 Zusammenfassung

Der Bedarf an Data Mining wird immer größer, da mehr digitale Daten produziert und ausgewertet werden müssen. Das Data Mining bedient sich vieler Konzepte des Machine Learning. Wichtige Aufgaben sind das Erkennen von Veränderungen in den Daten sowie das Lernen von Klassifikatoren und Regressoren mit Hilfe von Supervised und Unsupervised Learning Methoden und das Finden von Korrelationen und Assoziationen in den Daten mittels Frequent Pattern Mining. Diese Auswertungen werden auf der Grundlage von statistischen Methoden durchgeführt. Durch den Einsatz des Data Mining auf Datenströmen ergeben sich Probleme, da die Algorithmen nicht auf einer geschlossenen Datenmenge agieren können. Zur Lösung dieser Probleme wurden verschiedene Konzepte, wie z. B. Online-Algorithmen oder Datenintervalle eingeführt. Des Weiteren dienen spezielle Datenstrukturen zur Aggregation von mehreren Datenelementen um den Speicherbedarf zu minimieren. Data Mining auf Datenströmen ist ein aktuelles Forschungsthema und gewinnt mit dem Datenwachstum und der weiten Verbreitung von Sensoren stetig an Bedeutung.

3.4 Vorhersagen

3.4.1 Vorhersage von Photovoltaikenergieerzeugung

Die Energiewende führte im letzten Jahrzehnt zu einem starken, jährlichen Anstieg des Anteils von erneuerbaren Energien an der Stromversorgung in Deutschland, wie Abbildung 3.28 verdeutlicht. Mit einem weiteren Ausbau der regenerativen Energien kann auch in Zukunft gerechnet werden. Dies macht es allerdings erforderlich, dass bei dem Prozess der Energiewende nicht mehr nur steuerbare Erzeuger und vorhersehbare Verbraucher einbezogen werden [44, 189, 219, 233]. Stattdessen soll eine Verknüpfung der Stromproduzenten und -konsumenten durch ein intelligentes Stromnetzwerk, einem Smart Grid, stattfinden, welches für sichere, hocheffiziente und transparente Energiesysteme sorgen kann [203, 246]

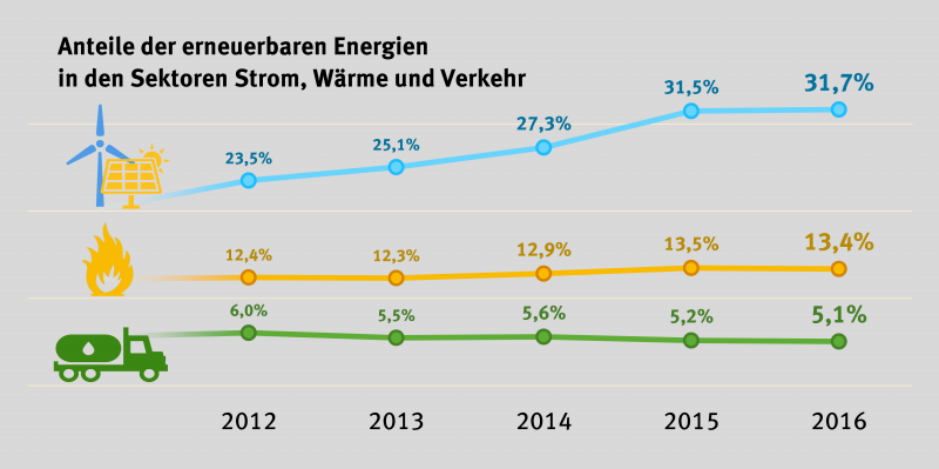
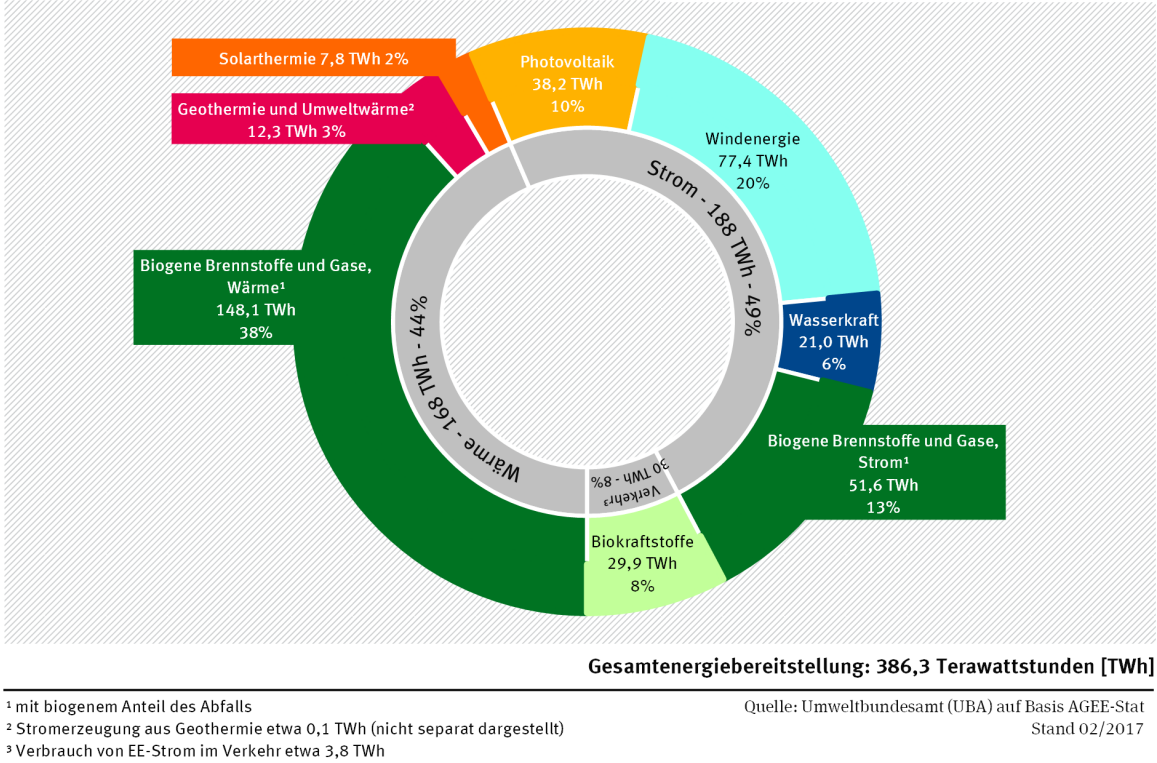


Abbildung 3.28: *Erneuerbare Energien 2012-2016 [44]*

Abbildung 3.29 ist zu entnehmen, dass neben biogenen Brennstoffen und Gasen und der Windenergie, die Photovoltaik (PV) mit 10% der bereitgestellten Energie einen wichtigen Platz im Smart Grid einnimmt.



¹ mit biogenem Anteil des Abfalls
² Stromerzeugung aus Geothermie etwa 0,1 TWh (nicht separat dargestellt)
³ Verbrauch von EE-Strom im Verkehr etwa 3,8 TWh

Quelle: Umweltbundesamt (UBA) auf Basis AGEE-Stat
Stand 02/2017

Abbildung 3.29: *Energiebereitstellung aus erneuerbare Energien 2016 (vorläufige Zahlen) [44]*

Bei der Netzintegration von regenerativen Energien entsteht jedoch eine Problematik:
Die regenerative Energieerzeugung ist zum Beispiel auf Grund von der sich verändernden Wetterlage nicht konstant und somit nicht so steuerbar, wie im Gegensatz dazu die Energieerzeugung durch Kraftwerke [175].

Im Zuge der Energiewende soll die Energieversorgung zukünftig größtenteils auf Wind- und Solarenergie basieren. Das bedeutet jedoch, dass eine mehrfache Menge an PV-(beziehungsweise Wind-)Anlagen installiert werden müsste, um auch an sonnenarmen Tagen den Strombedarf zu decken. Dies würde allerdings an Tagen mit hoher PV-Energieerzeugung zu Überschusssituationen und damit einem Überlastungsausfall des Netzes führen. Um die PV-Ausdehnung nicht zu gefährden, ist eine Einspeisebegrenzung von PV-Systemen in Kombination mit Batteriespeichern notwendig. An sonnenstarken Tagen ist ein solcher Batteriespeicher jedoch frühzeitig geladen, sodass auch dann eine erhöhte Einspeise in das Netz nicht umgangen werden kann. Prognosen der PV-Leistung erhöhen den PV-Eigenverbrauch und verringern so die Netzeinspeiseleistung [275].

Die Projektgruppe E-Stream hat daher für die effizientere Nutzung der regenerativen Energien das Ziel, ein Toolkit für die Unterstüzung der Netzüberwachung und -steuerung zu erstellen, bei welchem der Netzstatus bestimmt, prognostiziert und letztendlich visualisiert wird. Hierfür können zum Beispiel

auf Basis von aktuellen Echtzeitdaten, in Kombination mit historischen Daten oder netzfremden Daten, Prognosen der PV-Leistung getroffen werden.

3.4.1.1 Photovoltaik Stromerzeugung und Vorhersage

Die sogenannten PV-Anlagen lassen sich zwischen Insel- und netzgekoppelten Anlagen unterscheiden. Im Gegensatz zu der netzgekoppelten Anlage, bei der der Betreiber eine Einspeisevergütung erhält, ist die Inselanlage nicht an das öffentliche Stromnetz angeschlossen und dient lediglich dem Eigenverbrauch. Da die netzgekoppelten Anlagen mit 80% den Großteil der PV-Anlagen ausmachen, hat die Stromerzeugung durch PV-Anlagen einen großen Einfluss auf die Spannungsqualität und führt zu dem Bemühen, durch Prognosen die Zuverlässigkeit und Performanz der Solarenergie zu optimieren und die Ressourceneffizienz somit zu erhöhen [189, 219, 229, 233].

Die sogenannten PV-Anlagen ziehen einen Vorteil aus dem photoelektrischen Effekt und können Sonnenlicht in Gleichstrom umwandeln, sodass dadurch der umweltfreundliche Solarstrom erzeugt wird [108]. Bei dem hier ausgenutzten inneren photoelektrischen Effekt werden Halbleiter mit Licht bestrahlt, sodass unterhalb einer Grenzfrequenz die Energie des Photons kleiner ist, als die Bindungsenergie des Elektrons. Diese kann jedoch schon genügen, um das Elektron in einen angeregten Zustand zu versetzen und dadurch in einem Leitungsband Strom erzeugen [218, 250].

Dieser Effekt entsteht in Solarzellen, in welchen einzeln nur sehr wenig Strom fließt. Mehrere Solarzellen werden daher zu einem Solarmodul zusammengefasst und diese dann, zum Schutz vor Störeinflüssen, in eine Kunststoffschicht eingebettet sowie mit lichtdurchlässigem Spezialglas abgedeckt. Die entstandenen PV-Module werden daraufhin zu einem Solargenerator zusammengeschlossen. Der erzeugte Gleichstrom wird mittels eines Wechselrichters in Wechselstrom gewandelt, sodass der Strom zum Eigenverbrauch dienen und der überschüssige Strom in das öffentliche Netz eingespeist werden kann. Für die Einspeisevergütung ist es außerdem notwendig einen Einspeisezähler und einen Bezugszähler, für den etwaigen Zusatzbedarf, zu integrieren [108]. Die schematische Darstellung einer solchen netzgekoppelten PV-Anlage ist in Abbildung 3.30 gegeben.

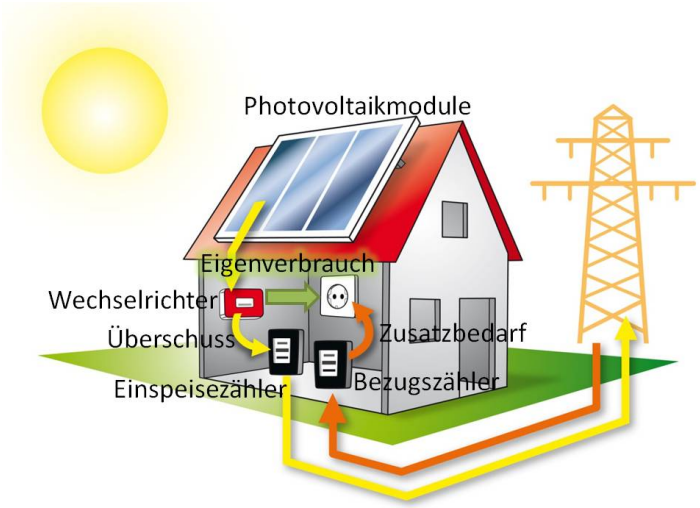


Abbildung 3.30: *Schematische Darstellung einer netzgekoppelten PV-Anlage [231]*

Um nun die PV-Effizienz bezüglich Zuverlässigkeit und Performanz zu optimieren, können Vorhersagen über die PV-Energieerzeugung, mittels Wettervorhersagen und PV-System-Daten, getroffen werden. Demnach werden verschiedene Vorhersagemethoden verwendet, welche physikalischen oder statistischen Ursprungs sein können. Dabei basieren die physikalischen Vorhersagen auf Solar- und PV-Modellen, wo hingegen die statistischen Vorhersagen auf historischen Datenmodellen beruhen [219]. Ein Vergleich der statistischen und physikalischen Prognosen ergab, dass theoretisch bei Ersteren eine höhere Genauigkeit erzielt werden könnte. In der Praxis konnte jedoch gezeigt werden, dass sich beide Methoden nicht in ihrer Qualität unterscheiden [219].

Prognosen können innerhalb eines Tages, beispielsweise 0 bis 6 Stunden im Voraus, oder für den folgenden Tag getroffen werden. Erstere sind dabei momentan von geringerem ökonomischem Interesse, als Vorhersagen für den folgenden Tag. Jedoch werden mit der zunehmenden Sonneneinstrahlung und einer erwarteten Verbesserung der Genauigkeit der Prognosen innerhalb eines Tages im Vergleich zu welchen für den folgenden Tag, die Marktchancen wesentlich erhöht [219].

3.4.1.2 Vorhersagen auf Basis von Solar- und PV-Modellen

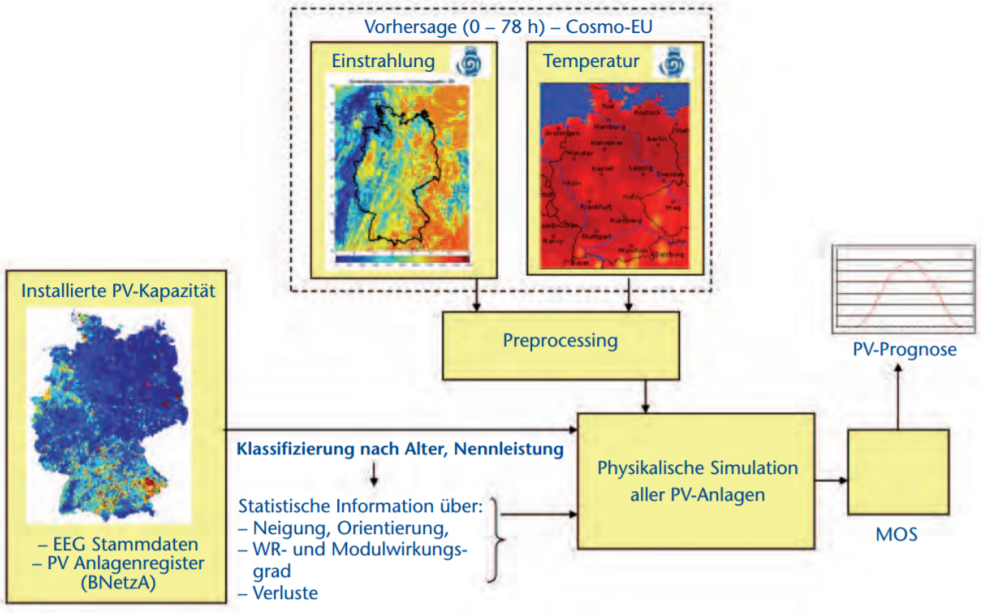


Abbildung 3.31: *Schematische Darstellung der Modellkette von Prognosen [180]*

Abbildung 3.31 macht deutlich, dass nicht alle benötigten Daten für die Prognosen durch Wettervorhersagen erhalten werden können, sondern durch Solar- und PV-Modelle berechnet werden müssen. Diese verknüpfen PV-Systemspezifikationen und vorverarbeitete Wettervorhersagen sowie, je nach Datenverfügbarkeit, historische Daten miteinander. In einer physikalischen Simulation aller PV-Anlagen findet dann eine Modellierung statt und nach einer Korrektur durch ein Model Output Statistic (MOS) kann daraus letztendlich die PV-Prognose abgeleitet werden [180, 219].

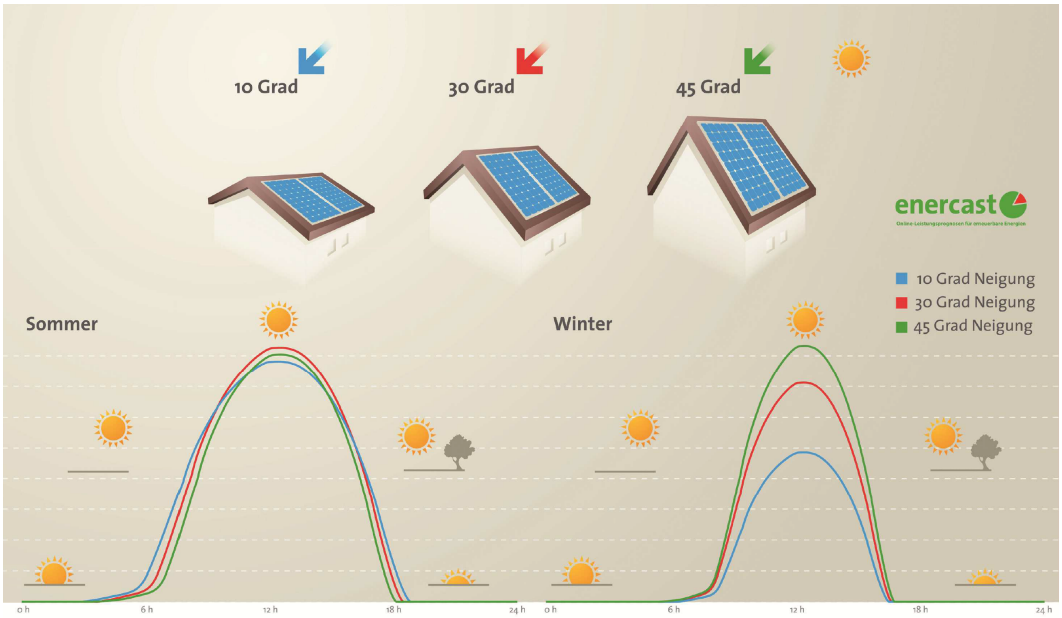


Abbildung 3.32: *Auswirkung des Neigungswinkels auf die Stromproduktion von Solaranlagen [175]*

Für die Erstellung von PV-Modellen werden verschiedene wichtige Variablen benötigt. Die PV-Ausgangsleistung kann durch die Bestrahlungsstärke in der Ebene des PV-Arrays und die Temperatur an der Rückseite der PV-Module beeinflusst werden. Wie Abbildung 3.32 verdeutlicht, besitzen außerdem der Einfallswinkel der Sonneneinstrahlung und die spektrale Verteilung der Bestrahlung, welche in einigen PV-Modellen enthalten sind, eine hohe Relevanz [219]. Außerdem können die Einstrahlungs- und Temperaturdaten in hoher zeitlicher Auflösung verwendet werden, um möglichst schnell Fehlfunktionen und Ausfälle zu erkennen [239]. Mit einer hohen Auflösung kann das Verhalten eines Systems im Kurzzeitbereich gut wiedergegeben werden [27]. Durch die Zeitreihen unterschiedlicher meteorologischer Größen an aussagekräftigen Standorten können dann zusätzlich Aussagen über die Performanz über eine längere Zeit getroffen und somit beispielsweise auch der Einfluss der PV-Stromeinspeisung auf die Strommärkte beurteilt werden [239].

Abbildung 3.33 verdeutlicht hingegen die Fehler, die bei zeitlich zu hoch aufgelösten Leistungen erzeugt werden. In diesem Beispiel ist das Problem, dass wegen eines lokalen Wolkendurchzugs kurzzeitige Schwankungen auftreten (Abbildung 3.33 links). Werden im Gegensatz dazu Mittelwerte über 15 Minuten berechnet (Abbildung 3.33 rechts) können längerfristige Tendenzen abgebildet werden. Allerdings wird dabei der Tagesverlauf durch die Mittelwertbildung falsch beurteilt.

Für Vorhersagen von 6 Stunden bis Tage im Voraus werden numerische Wettervorhersagemodelle verwendet. Die Genauigkeit dieser Methode übertrifft die, der auf rein historischen Daten basierten Methoden. Sie beruhen auf dynamischen Gleichungen, welche die Entwicklung und Atmosphäre anhand von initialen Umständen Tage im Voraus vorhersagen können. Die Gleichungen und Eingaben werden in einem dreidimensionalem Raster diskretisiert, welches sich vertikal über die Erdoberfläche spannt. Die initialen Daten werden beispielsweise von Satelliten erhalten und daraufhin verarbeitet und in dem 3D-Raster interpoliert [219].

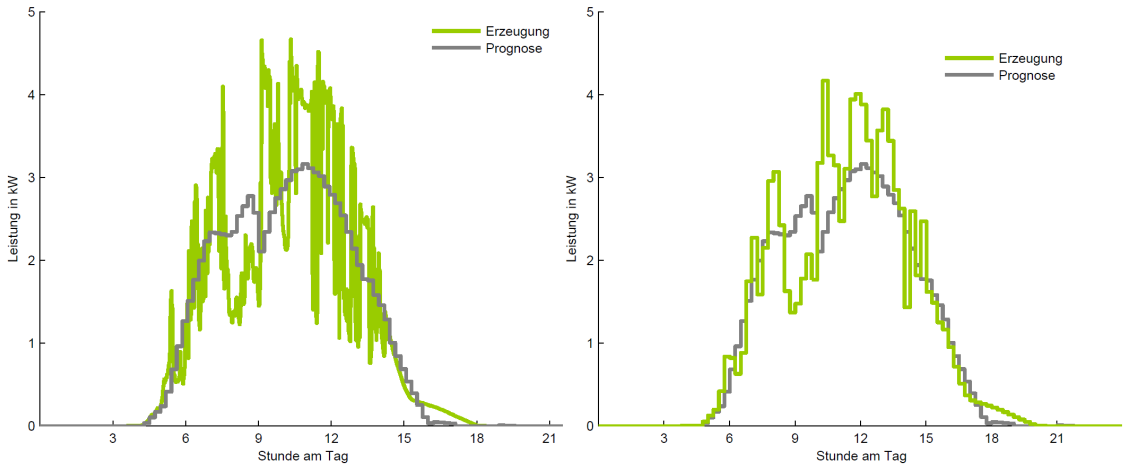


Abbildung 3.33: *Zeitlicher Verlauf des PV-Ertrages eines exemplarischen Tages in minütiger (links) und 15-minütiger Auflösung (rechts) jeweils mit Erzeugungsprognose [27]*

Das PV-Modell nach [239] gibt ein Beispiel, wie sich für jede Station die relativ zur Verfügung stehende Leistung ergeben kann (siehe Gleichung 3.1)). Dieses Modell wird durch folgende Variablen beeinflusst: Für einzelne Stationen werden die Systemleistungen für jede Ausrichtung, Neigung, Montageart und Modultechnologie spezifisch berechnet und anschließend aufsummiert. Jede Spezifikation benötigt dafür die nach einem vorgegebenem Anteil gewichtete Ausrichtung, Neigung, Montageart und Modultechnologie.

$$P_{System}(h,i,t,s,\gamma_E,\alpha_E,G_{Mod}) = \eta_{WR}(i)\cdot l\cdot \eta_{M_{rel}}(G_{Mod},t,s)\cdot G_{Mod}(\gamma_E,\alpha_E) \tag{3.1}$$

Hierbei entsprechen die Variablen $P_{System}(h,i,t,s,\gamma_E,\alpha_E,G_{Mod})$ der Systemleistung, h der Stunde des Jahres, i der Wechselrichterart, t der Modultechnologie, s der Montageart, $\eta_{WR}(i)$ dem relativen Wirkungsgrad des PV-Feldes, G_{Mod} der Globalstrahlung auf das Modul, γ_E der Modulneigung, α_E der Modulazimut und l sonstigen Verlusten.

Vorhersagen der PV-Energieerzeugung können außerdem auf Basis von Satellitendaten und unter Berücksichtigung von Messwerten von Wetterstationen erstellt werden. Außerdem werden zum Ausgleich systematischer Fehler PV-Generatordaten aufbereitet und mit einbezogen [27].

Wird ein totales Himmelsbild verwendet, kann in Realzeit und zusätzlich 10-30 Minuten im Voraus vorhergesagt werden. Dabei werden Bildverarbeitungstechniken und „Wolkentracking“ auf Himmelsfotografien angewandt. Wird eine Abfolge der Aufnahmen erstellt, ist es möglich die zeitliche Entwicklung des Wolkenfeldes zu analysieren. Dies geschieht durch eine Extrapolation der sich bewegenden Wolken ausgehend von den letzten registrierten Bildern. Die Bestrahlungsstärke wird für die augenblicklichen Wolken vorhergesagt und der Schatten dieser Wolken dann zeitlich weiter verschoben werden auf Basis der Wolkendichte und Richtung. Auf Grundlage dieser Informationen können dann wiederrum die Einstrahlungsdaten für die PV-Anlagen abgeleitet werden [29, 219].

Für Satellitenbilder gibt es ähnliche Methoden, denn die Wolken reflektieren Licht zu den Satelliten, sodass sie von ihnen detektiert werden können. Die Berechnung der Menge des zurückgestrahlten Lichts entspricht $Transmissivität = 1 - Reflektivität - Absorptionsfähigkeit$ [219]. Abbildung 3.34

unterstreicht die Annahme, dass die Wolkenform beziehungsweise -eigenschaften sich zwischen zwei Aufnahmen nicht ändern, sodass versucht werden kann die gleichen Eigenschaften in dem nächsten Bild zu finden und daraus die Geschwindigkeit zu berechnen [219]. Wie ebenfalls in Abbildung 3.34 zu erkennen ist, können in Satellitendaten jedoch keine kleineren Wolken beziehungsweise Lücken in dichteren Wolken erfasst werden, sodass dennoch Fehler entstehen können. Die entstehenden Prognosefehler können allerdings durch ein sehr ausgedehntes Netzgebiet ausgeglichen werden [27]. Die geringere räumliche und zeitliche Auflösung führt wahrscheinlich zu den weniger genauen Vorhersagen, als Prognosen durch Himmelsbilder innerhalb einer Stunde. Dennoch können die erhalteten Bewegungsvektoren genutzt werden, um die Ergebnisse von numerischen Modellen zu verbessern. Satellitenbildanalyse ist die häufigste Prognosetechnik für Vorhersagen bis zu 5 Stunden im Voraus [219].

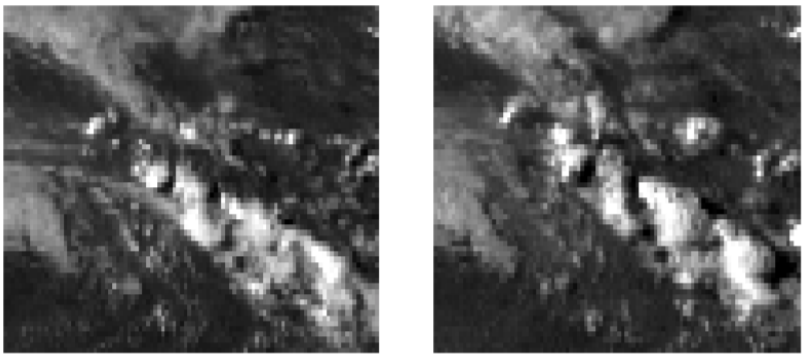


Abbildung 3.34: *Zwei aufeinanderfolgende Satellitenbilder [125]*

Die besten PV-Vorhersagen kombinieren numerische Modelle mit einer Aufarbeitung dieser Vorhersagen, welche nicht direkt in dem Modell enthalten sind, um sie zu verbessern oder um neue Prognosen zu erzeugen. Das bedeutet, die Ergebnisse der numerischen Modelle, werden mit bereits gemessenen Daten verglichen. Dabei gibt es unterschiedliche Methoden, die auch in Kombination angewandt werden können. Zu diesen gehören die Modellausgabestatistiken und räumliche-zeitliche Interpolation und Glättung [219].

3.4.1.3 Vorhersage auf Basis historischer Daten und Qualitätssicherung

Eine Summenleistung über die in einem Gebiet verteilten PV-Anlagen führt zu einer wesentlich geringeren Anzahl an Vorhersagefehlern, mit Ausnahme von Persistenzprognosen und Bewegungsextrapolation. Dies ist darin begründet, dass sich die mittlere Bewölkungssituation innerhalb einer Stunde nur wenig ändert. Eine Prognose mehrere Stunden im Voraus ist auf Grund des stochastischen Anteils bei Vorhersagen auf Basis historischer Daten für einzelne PV-Anlagen mit einer hohen zeitlichen Auflösung nicht zuverlässig [29].

Eine Persistenz-Prognose bezeichnet eine statistische Zeitreihenanalyse und ist damit ein Beispiel für eine stochastische Lerntechnik, bei der Pattern in meteorologischen Daten, Variablen oder sogar Bildern identifizieren werden können. Die Identifikation kann auf Basis der erlernten Pattern aus historischen Daten durch bestimmte Algorithmen stattfinden [219]. Persistenz Prognosen werden als Vergleichsmodell für andere, gegebenenfalls komplexere, Modelle verwendet [163]. Der Vorteil

hierbei ist, dass keine externen Daten benötigt werden. Erzeugt werden sie durch die Analyse der am vorangegangenen Tag registrierten PV-Energieerzeugung [200]. Genauer bedeutet das:
Es wird angenommen, dass zu einem Zeitpunkt $t+1$ der gleiche Wert auftritt, wie zum Zeitpunkt t [163].

Neuerdings wird jedoch auf den Master Optimization Process (MOP) gesetzt, bei dem nicht mehr nur eine einfache Patternerkennung durchgeführt wird. Stattdessen werden evolutionäre Optimierunsschemata, wie beispielsweie einem generischen Algorithmus, eingesetzt. Dabei wird die beste Topologie und der beste Teil einer stochastischen Pattern-Lerneinheit für jedes Mikroklima berechnet [219]. Eine Kombination dieser Vorhersagemöglichkeit mit Echtzeitdaten kann eine hohe Persistenz, konventionelle Regression und Patternerkennungsmethoden und dadurch zuverlässigere Vorhersagen erzeugen. Dabei sind sie robust und vielseitig [219].

Die Qualtität eines Prognosesystems kann für einen bestimmten Prognosehorizont, ein bestimmtes Gebiet mit der dort installierten Leistung und bezüglich eines gewissen Zeitraumes lediglich quantifiziert werden [180]. Zur Bestätigung der Genauigkeit von Prognosen, gibt es unterschiedliche standardisierte Metriken. Dazu gehören nach [219] der Mittelwertfehler (MBE oder Bias, siehe (Gleichung 3.2)), der mittlere quadratische Fehler (MSE), beziehungsweise die Wurzel des mittleren quadratischen Fehlers(RMSE, siehe (Gleichung 3.3)), der absolute mittlere Fehler (MAE, siehe (Gleichung 3.4)) und die Standardabweichung (SDE beziehungsweise σ, siehe (Gleichung 3.5)). Hierbei gilt jeweils: $e_i = y_{i,vorhersage} - y_{i,ueberwacht}$, wobei y_i die i-te Vorhersage ist.

$$MBE = \frac{1}{N}\sum_{i=1}^{N} e_i \tag{3.2}$$

$$RMSE = (MSE)^{\frac{1}{2}} = (\frac{1}{N}\sum_{i=1}^{N} e_i^2)^{\frac{1}{2}} \tag{3.3}$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N} |e_i| \tag{3.4}$$

$$SDE = \sigma = (\frac{1}{N-1}\sum_{i=1}^{N} (e_i - e_i)^2)^{\frac{1}{2}} \tag{3.5}$$

Während MAE den durchschnittlichen Anteil von Vorhersagefehlern angibt, legt RMSE (und MSE) mehr Wert auf den größten Fehler. Die Gleichung (Gleichung 3.6) aus [219] stellt den Zusammenhang zwischen den Metriken dar.

$$RMSE^2 = MSE = MBE^2 + SDE^2 \tag{3.6}$$

Jede dieser Metriken kann als absolut, beziehungsweise normalisiert angesehen werden. Die Nutzung dieser Metriken erzeugt Konfidenzintervalle, aus denen direkt die Signifikanz abgelesen werden kann. Durch die Verwendung dieser Intervalle in Prognosen ist ein Indikator für die erwartete Genauigkeit gegeben [219].

Zusätzlich sollte ein Benchmarking durchgeführt werden, denn die Genauigkeit von Prognosen ist stark abhängig von dem Ort und der Zeit, welche für die Auswertung genutzt wurden. Mit Hilfe des Benchmarking kann außerdem identifiziert werden, unter welchen Bedingungen eine Vorhersage gute oder schlechte Ergebnisse liefert. Dies kann leicht durchgeführt werden, in dem die Metriken für die

Genauigkeit miteinander Verglichen werden, oder indem der sogenannte „*skill score*“ einer gegebenen Metrik berechnet wird, siehe (Gleichung 3.7) [219].

$$skillscore = \frac{Metrik_{Referenz} - Metrik_{Vorhersage}}{Metrik_{Referenz} - Metrik_{perfekteVorhersage}} \tag{3.7}$$

3.4.1.4 Fazit

Durch den steigenden Anteil der nicht steuerbaren regenerativen Energien in der Stromversorgung, treten bei der Netzintegration Probleme auf. Daher ist es sinnvoll mit Hilfe von Prognosen die voraussichtliche Stromerzeugung durch diese Energien vorherzusagen. Damit kann einem Überlastungsausfall vorgebeugt und der Grundbedarf zuverlässig gedeckt werden. Das geplante Toolkit zur Unterstützung der Netzüberwachung der Projektgruppe E-Stream sollte wegen des großen Einflusses der durch PV-Anlagen erzeugten Energien auf das Stromnetzwerk auch Vorhersagen der PV-Energieerzeugung berücksichtigen.

Der Projektgruppe E-Stream bieten sich dabei verschiedene Daten zur Vorhersage der PV-Energieerzeugung an, die in das Toolkit mit einbezogen werden können. Wie aufgezeigt wurde, können Vorhersagen zu Verbesserung der PV-Effizienz innerhalb eines Tages oder aber für den folgenden Tag getroffen werden und verwenden Solar- und PV-Modelle, die PV-Systemspezifikationen, Wettervorhersagen und gegebenenfalls historische Daten miteinander verknüpfen oder die Prognosen basieren stattdessen lediglich auf historischen Daten. Je nach den zur Verfügung stehenden Prognosedaten können auch relativ einfach mehrere dieser Vorhersagen beziehungsweise Modelle verwendet werden. Fest steht, es sollten zusätzlich für die Gewährleistung der Genauigkeit der getroffenen Vorhersagen standardisierte Metriken und das Benchmarking berücksichtigt werden.

3.4.2 Vorhersage von Windenergieerzeugung

Für eine effiziente Nutzung von regenerativen Energien, wie die Windenergie, wird ein intelligentes Stromnetz (Smart Grid) benötigt. Das Smart Grid benötigt allerdings die Information, wann welche Energiequelle zur Verfügung steht und wie viel Energie sie in das Stromnetz einspeist. Daher ist die Wind(-energie)-Vorhersage eine wichtige Informationsquelle für ein Smart Grid.

Dieser Abschnitt gibt einen Überblick über die verschiedenen Methoden, die zur Windenergie-Vorhersage verwendet werden. Neben den möglichen Modellen werden auch die Informationsquellen aufgezeigt, die bei der Prognoseerstellung benötigt werden. Denn neben den offensichtlichen Einflussfaktoren wie zum Beispiel Wind, wird die Energieerzeugung der Windkraftanlagen auch durch weitere nicht so intuitive Faktoren beeinflusst.

Die Verwendung von anderen Informationsquellen ist zum Beispiel für Prognosen interessant, die nicht unbedingt kostenpflichtige Wetterprognose verwenden können. Für die Verwendung von Wetterprognosen für die Windenergieprognose werden hochauflösende Modelle benötigt, die nicht für alle Regionen der Welt kostenlos erhältlich sind. Somit besteht auch Interesse an Prognosemodellen, die ohne Daten aus der Wetterprognose funktionieren.

3.4.2.1 Eingangsinformationen

Für eine Prognose der Windenergie können verschiedene Eingangsinformationen verwendet werden. In dem Paper von Heinermann und Kramer [132] wurden nur die Windgeschwindigkeit und die Energiemesswerte der Windkraftanlagen verwendet, da sie auf keine weiteren Daten zurückgreifen konnten. Weitere Informationen zum Wetter wie zum Beispiel, ob es regnet oder die Sonne scheint, könnten evtl. weiteren Aufschluss geben ob die zukünftige Energieerzeugung konstant bleibt oder sich mit höherer Wahrscheinlichkeit ändert.

Allerdings haben Heinermann und Kramer [132] aus den bestehenden Informationen weitere Eingangsdaten generieren können. Denn bei der Betrachtung der Energiewerte als Zeitreihe, besitzt jeder Wert zusätzlich eine Steigung. Mithilfe der Steigung erhält man die Information, wie stark sich die Energieerzeugung bzw. die Windstärke zu dem Zeitpunkt davor verändert hat [132].

Dies zeigt, dass zwar theoretisch eine Auswahl von Eingangsinformationen vorhanden ist, auf dem die Energievorhersage basieren soll. Trotzdem ist es nicht immer möglich Daten zu bekommen, die man verwenden möchte.

Bei der Auswahl der Eingangsinformationen gibt es außerdem nicht *die* richtige Antwort. Denn jedes Problem und in diesem Fall jeder Windpark verhält sich anders. Somit kann sich die Eingangsinformation von Temperatur, bei Windkraftanlage in der Nähe von Bergen als sehr nützlich erweisen (zum Beispiel wegen der Föhnwinde). Wohingegen bei Windkraftanlagen auf offener Fläche diese Information weniger Relevanz besitzen könnte, sodass sie evtl. eher hinderlich für die Prognose ist.

Abbildung 3.35 zeigt aus welchen Datenquellen optimale Prognosemodelle bestehen. Für eine gute Vorhersage der Energieerzeugung, sollten möglichst alle drei Arten von Eingangsinformationen verwendet werden. Die Ausarbeitung von Heinermann und Kramer [132] verwendete zum Beispiel nur Messwerte der Windkraftanlagen („1. SCADA“) und die Umgebungsdaten der Windkraftanlagen („3. Terrain“). Dies zeigt, dass Prognosemodelle allerdings auch mit weniger Informationsquellen erstellt werden können, dies hat jedoch negative Auswirkungen auf die Prognosegüte [132].

Die Verwendung nur von Messwerten und Umgebungsdaten, ohne Wetterprognosen, wie es Heinermann und Kramer [132] in ihrem Modell gemacht haben, wird nach [107] als Kurzzeitvorhersage definiert. Diese Art von Modellierung soll für Prognosen von bis zu sechs Stunden in kommerziellen Modellen Anwendung finden.

Messwerte / Supervisory Control and Data Acquisition (SCADA)

Als Basis für jede Prognose von technisch beeinflussten Modellen werden anlagenspezifische Messwerte verwendet. Für eine gute Prognosegüte werden viele Referenzdaten aus der Vergangenheit benötigt, somit steigt in der Regel die Prognosequalität je mehr historische Messwertdaten zur Verfügung stehen. Da viele technische Anlagen heutzutage ein SCADA-System besitzen, welches die technischen Prozesse überwacht und protokolliert, verfügen Anlagenbetreiber schon länger über solche Messwerte. Da viele Unternehmen diese Messwerte ohne konkrete Verwendung speichern, können diese nun für Prognosen verwendet werden.

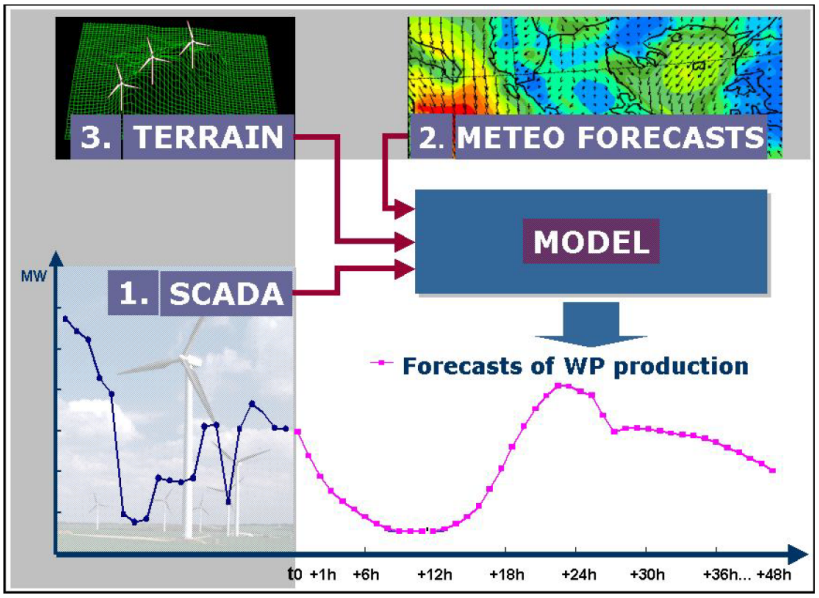


Abbildung 3.35: *Die drei Arten der Eingangsinformationen für die Windenergievorhersage: 1. Anlagendaten, 2. Wetterprognosen, 3. Umgebungsdaten [107]*

Wetterdaten

Als wichtigstes Eingangskriterium für die Windenergievorhersage gilt das Wetter, denn es ist der bestimmende Faktor für den Wind, der wiederum die Rotorblätter in Bewegung versetzt und die Energie erzeugt. Allerdings besitzen Wetterprognosen einen sehr großen Unsicherheitsfaktor, sodass Wetterdaten als Eingangsinformation eine große Fehlerquelle für die Windenergieprognose darstellen.

Daher besitzt die Qualität der Wetterprognose das größte Potential zur Verbesserung der Windenergievorhersage. Denn je besser die Wettervorhersagen sind, desto besser kann die Energieerzeugung prognostiziert werden [107].

Die verwendeten Wettervorhersagen für die Windenergieerzeugung basieren auf sogenannten Numerischen Wetterprognosen (NWP - Numerical weather prediction). Dies ist ein rechnergestütztes Modell, bei dem ein Zustand in der Erdatmosphäre betrachtet und dieser als Anfangspunkt für eine numerische Lösung verwendet wird. Für dieses numerische Rechenmodell wird ein Gitternetz über den zu betrachtenden Bereich gelegt. Dabei werden alle wetterrelevanten Daten wie Temperatur, Luftdruck, Dichte, Windrichtung und Windgeschwindigkeit diskret pro Gitternetz dargestellt. Aufgrund der Möglichkeit der verschieden legbaren Gitternetze werden Wetterprognosen in globale und in lokale Modelle unterschieden [107].

Globale Modelle Als globale Modelle werden Wetterprognosen bezeichnet, bei denen das Gitternetz die komplette Erde umspannt. Diese Modelle verwenden dementsprechend ein grobmaschiges Gitternetz, damit das Prognosemodell noch in akzeptabler Rechenzeit erstellt werden kann. Typische Gitternetzgrößen für globale Modelle sind 10 bis 50 km [107].

Der Deutsche Wetterdienst (DWD) ist der Anbieter für numerische Wetterprognosen in Deutschland. Das globale Modell, welches der DWD anbietet, heißt *ICON* (Icosahedral Nonhydrostatic). Das

Gitternetz dieses Modells besitzt eine Maschenweite von 13 km und teilt die Luft vertikal in 90 Schichten auf. Das Prognosemodell beinhaltet die Wetterparameter Luftdruck, Wind, Wasserdampf, Wolken und Niederschlag. Der Hochleistungsrechner des DWDs benötigt für eine 180-Stunden-Vorhersage eine Stunde [73].

Lokale Modelle Bei lokalen Modellen der Wetterprognose wird nur ein begrenzter Raum bzw. ein bestimmter Ausschnitt betrachtet. Die lokalen Wettermodelle werden auch als *LAM* (limited area model) bezeichnet. Diese Betrachtung von kleineren Gebieten hat mehrere Vorteile. Da die Berechnung von Wettermodellen sehr aufwendig ist, wird durch die Verkleinerung des zu brechenden Gebietes, der Aufwand verringert, sodass das Modell schneller berechnet werden kann. Des Weiteren kann bei lokalen Modellen das Gitternetz feinmaschiger gemacht werden. Es steigert zwar die Genauigkeit des Modells, allerdings werden dadurch die Modelle wieder aufwendiger. Die Gitternetzgrößen für lokale Modelle sind in der Regel bis zu 15 km groß [107].

Der Deutsche Wetterdienst bietet auch zwei lokale Wetterprognosemodelle an:

- **COSMO-EU**: Das Regionalmodell COSMO-EU (Consortium for Small-scale Modeling) ist das lokale Wettermodell für Europa und besitzt eine Gitternetzmaschenweite von 7 km und eine vertikale Aufteilung der Luft in 40 Schichten. Für die Ränder Europas werden die Werte des globalen Modells ICON verwendet.
- **COSMO-DE**: Des Weiteren wird ein noch hochauflösenderes Regionalmodell nur für Deutschland angeboten, COSMO-DE. Hier beträgt die Maschenweite nur 2,8 km. Dieses Modell hat einen Vorhersagehorizont von 27 Stunden. An den Rändern des Deutschland-Modells werden die Werte des COSMO-EU Modells verwendet.

Umgebungsdaten

Als letzte Informationsquelle für Energievorhersagen von Windkraftanlagen können die örtlichen Begebenheiten (Umgebungsdaten) verwendet werden. Dies beinhaltet zum einen umliegende Berge und Täler und zum anderen die Höhenposition der eigenen Windkraftanlage. Grundsätzlich hat sich gezeigt, je komplexer die Beschaffenheit der Umgebung ist, desto größer waren die Abweichungen der Prognosen. Windparks in einer relativen flachen Umgebungen sowie bei Offshore Parks haben in den Vergleichen bessere Prognosen geliefert [107].

3.4.2.2 Vorhersagehorizonte

Als Vorhersagehorizont wird die Zeitspanne bezeichnet, die aussagt wie weit der prognostizierte Wert in der Zukunft liegen soll. Da in der Regel diskrete Zeitreihen vorhergesagt werden sollen, beinhaltet der Vorhersagehorizont die Anzahl der prognostizierten Werte. Je nach Granularität der Zeitreihe, kann sich die Anzahl der Zeitpunkte und Prognosewerte stark unterscheiden.

- **Kurzzeitvorhersage**: Bei einer Kurzzeitvorhersage für die Windkraftenergie wird häufig ein Zeitraum von bis zu zehn Stunden betrachtet [107].
- **Langzeitvorhersage**: Bei der Langzeitvorhersage hingegen wird ein Zeitraum von bis zu drei Tagen betrachtet [107].

Dieser große Vorhersagehorizont der Langzeitvorhersage kann allerdings nicht nur durch die Leistungsinformationen der Vergangenheit erreicht werden, hierbei müssen zusätzliche Informationen über das Wetter verwendet werden [107]. Des Weiteren ist zu beachten, dass man bei der klassischen Wettervorhersage einen Zeitraum von bis zu drei Tagen häufig als Kurzzeitprognose bezeichnet. Da heutzutage bei Wettervorhersagen mit einem größeren Horizont als drei Tagen die Prognosegüte sich enorm verringert, ist es nicht sinnvoll auch bei der Windenergieprognose einen größeren Vorhersagehorizont zu wählen.

3.4.2.3 Methoden für Kurzzeitvorhersagen

Es existieren verschiedene Methoden, die Windenergie vorherzusagen. Die meisten hier vorgestellten Methoden basieren auf statistischen Modellen, die eine große Menge an historischen Daten benötigen. Mithilfe der historischen Daten werden Modelle erzeugt, mit denen Prognosen erstellt werden können. Diese Prognosemodelle benötigten je nach Modellierung unterschiedliche (aktuelle) Eingangsinformationen, um mit dem Modell kontinuierlich neue Prognosewerte erzeugen zu können.

Räumlich-Zeitliche Regression

Eine Windenergie-Vorhersage kann durch die Betrachtung von Daten aus der Vergangenheit erstellt werden. Mit dieser Zeitdatenreihe kann ein Vorhersagemodell, durch das Trainieren auf einem gegebenen Datensatz, erzeugt werden. Zur Erstellung der Regression muss der Vorhersagehorizont λ festgelegt werden. Dieser gibt an, wie viele Zeitschritte der Wert in der Zukunft vorhergesagt werden soll.

Die Eingangsdaten bestehen in dem Beispiel für eine Windenergieprognose aus [132] aus Leistungsdaten p von einzelnen Windkraftanlagen, die zusammen in einem Windpark stehen. Der Windpark besteht aus m Windkraftanlagen, die in einem Vektor zusammengefasst werden. Die Zeitreihe geht μ Zeitschritte in die Vergangenheit zurück. Somit stellt $p_i(t)$ den Messwert der Windkraftanlage i zum Zeitpunkt t da.

Bei der Erstellung der Regression in Gleichung 3.8 wird der zu prognostizierende Wert p_j festgelegt. In [132] definiert j zwar eine explizite Windkraftanlage aus dem Windpark, jedoch kann der Wert p_j auch die Summe des gesamten Windparks darstellen. Allerdings nur unter der Annahme, dass bei der Erstellung bzw. Trainings der Regression die Summe aller Anlagen als Zielwert definiert wurde.

$$\begin{pmatrix} p_1(t_0-\mu) & \cdots & p_1(t_0) \\ \vdots & \ddots & \vdots \\ p_m(t_0-\mu) & \cdots & p_m(t_0) \end{pmatrix} \rightarrow p_j(t_0+\lambda) \tag{3.8}$$

In [132] hat sich gezeigt, dass die Windkraftanlagen immer eine ähnliche Energieerzeugung wie die ihrer benachbarten Windkraftanlagen aufzeigen (siehe Abbildung 3.36). Des Weiteren zeigte sich in der Analyse von Heinermann und Kramer [132], dass sich die vermutete Korrelation zwischen der Windgeschwindigkeit und der Energieerzeugung in den Messwerten widerspiegelt.

Durch das Hinzufügen der örtlichen Informationen bzw. die Leistungsdaten von Windkraftanlagen in der Nähe, wurden die Vorhersagen signifikant verbessert. Im Gegensatz zu den Prognosen, die nur auf den Informationen der eigenen Anlage beruhten.

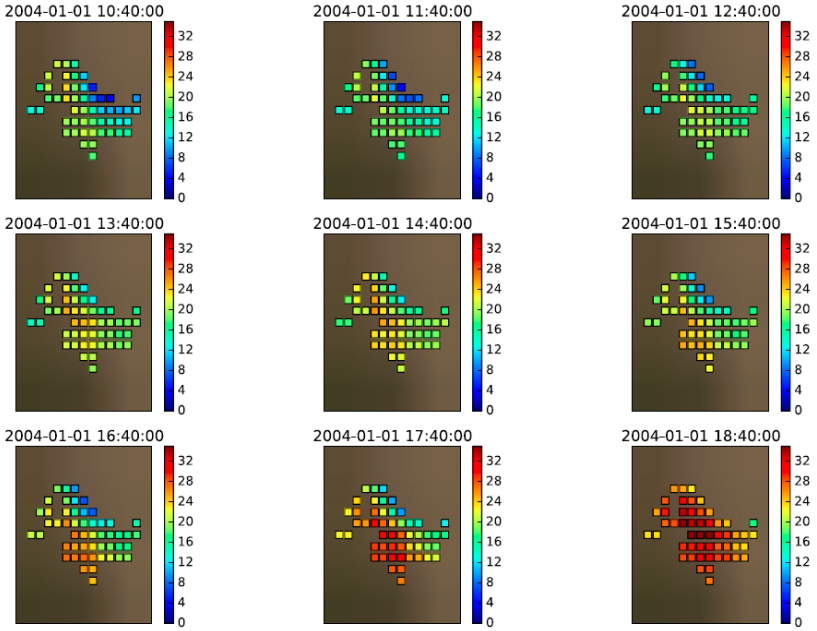


Abbildung 3.36: *Windgeschwindigkeitsmessungen eines Windparks in der Nähe von Reno (USA), Angabe der Geschwindigkeit in Farbe [132]*

Nächste-Nachbarn Regression

Eine alternative Prognosemöglichkeit besteht in der Verwendung der k-nächste-Nachbarn Klassifikation bzw. Regression. Bei dieser Prognose wird der Mittelwert der umliegenden benachbarten Werte (in diesem Fall die Leistung der Windkraftanlage) gebildet.

In der folgenden Gleichung 3.9 wird der Mittelwert über die K Nachbarn gebildet, die in einer vorher definierten Distanz zu dem zu bestimmenden Objekt stehen. Als Abstandsmaß wird in der Regel der Euklidischer Abstand verwendet [131].

$$f_{KNN}(x) = \frac{1}{K} \sum_{i \in N_K(x)} y_i \tag{3.9}$$

Somit liegt der Fokus bei dieser Regressionsart auf dem örtlichen Zusammenhang (Abstand zu den Nachbarn). Bei der Räumlich-Zeitlichen Regression wird zusätzlich der zeitliche Verlauf mit betrachtet.

Support Vector Regression

Die Support Vector Regression, die auf der Support Vector Machine basiert, wurde von Vapnik [266] entwickelt. Bei diesem Algorithmus werden die gegebenen Daten mithilfe des *Kernel-Tricks* in eine höhere Dimension überführt, um später eine linear trennbare Lösung zu erhalten. Die Dimensionsänderung wird in dem Hilbertraum $H \subseteq R^X = \{f : X \to R\}$ durch den *Kernel-Trick* $k : X \times X \to R$ durchgeführt.

Für eine Support Vector Regression kann folgende Formel definiert werden:

$$min_{f\in H, b\in R} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(X_i + b)) + \lambda \|f\|_H^2 \tag{3.10}$$

Mit y_i wird die Klassenzugehörigkeit definiert. Die Funktion $L : R \times R \to [0, \infty)$ bewertet das Ergebnis von der Funktion f mit Hilfe der Klassenzugehörigkeit y_i. Für die Berechnung von f werden die Trainingsdaten X_i und ein Offset b (Bias) benötigt. Zum Schluss wird noch eine Bewertung der Komplexität addiert. Diese besteht aus einer quadrierten Norm $\|f\|_H^2$.

Der Parameter λ ist ein Regularisationsparameter, der dabei hilft einen Kompromiss zwischen einem Overfitting und einer zu großen Komplexität der Lösung zu finden. Als Overfitting wird beim Machine Learning das Auswendiglernen bezeichnet. Dies bedeutet, dass eine Lösung zu lange auf einem Trainingsdatensatz optimiert wurde und damit eine sehr gute Lösung nur für diesen einen Datensatz existiert. Sobald andere Daten verwendet werden, ist das Ergebnis der Prognose schlecht. Die gefundene Lösung kann nicht mehr generalisieren [132].

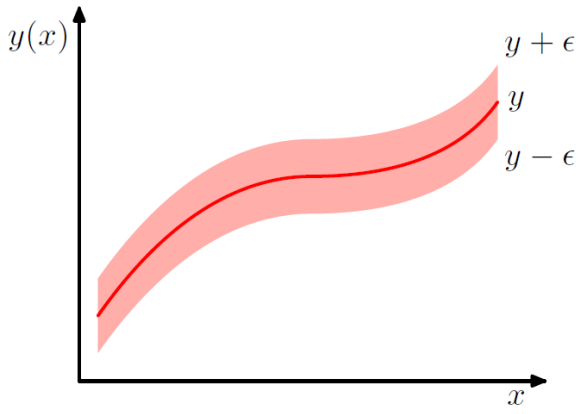


Abbildung 3.37: *Schematische Support Vector Regression y in einem Lösungsraum $y \pm \varepsilon$ [31]*

Mithilfe dieser Machine Learning Methode kann nun die Leistungskennlinie einer Windkraftanlage durch die Support Vector Regression prognostiziert werden. In Abbildung 3.37 wird so eine schematische Regressionslinie y dargestellt. Sie hat den maximalen Abstand zu den Rändern der roten ε-Röhre und ist damit die optimale Lösung für den gegebenen Lösungsraum $y \pm \varepsilon$ [31].

Random Forest Regression

Die Random Forest Regression basiert auf mehreren Entscheidungsbäumen, die jeweils mit zufälligen Werten aus den gegebenen Windparkdaten erstellt werden. Bei Random Forest wird eine große Anzahl an unterschiedlichen Entscheidungsbäumen kombiniert. Diese Entscheidungsbäume werden nicht optimiert („beschnitten“, engl. pruned), sondern sie werden voll ausgebaut verwendet. Bei der Erstellung der einzelnen Bäume werden nur N Merkmale zur Verfügung gestellt. Durch die zufällige Zusammenstellung von verschiedenen Entscheidungsbäumen wird die Prognosegüte durch ihre starke Diversität verbessert [132].

Ein weiterer Vorteil für den Einsatz von Random Forest ist, dass die Komplexität der Aufgabe, bei Verwendung unterschiedlicher Methoden, überschaubar bleibt. Des Weiteren ist dieses Konzept dadurch sehr gut parallelisierbar, was bei vielen Daten von großem Vorteil ist [232].

Beim Trainieren der Random Forest Regression zeichnen sich schlechte Prognosemethoden dadurch aus, dass sich ihre Ergebnisse unkorreliert zu den anderen Ergebnissen verhalten [232].

Neuronale Netze

Eine weitere Möglichkeit ein Prognosemodell zu erstellen ist die Verwendung von neuronalen Netzen (engl. ANN - Artificial Neural Networks). Das Konzept von künstlichen neuronalen Netzen stammt aus der Neurobiologie und ist eine mathematische Darstellung eines Nervensystems. Dieses neuronale Netz besteht aus beliebig vielen Knoten (Neuronen), die mit gerichteten und gewichteten Verbindungen verbunden sind. Durch das Trainieren des neuronalen Netzes werden die Gewichtungen manipuliert, sodass ein Zusammenhang zwischen den Eingangsinformationen und den zu prognostizierenden Zielwert hergestellt wird. Ein neuronale Netz ist fertig trainiert, wenn der Fehler der Prognosen auf bekannten Werten den Anforderungen genügt [177].

In der Zusammenstellung [107] bisheriger Forschung und Verwendung verschiedener Prognosemethoden für die Windenergie-Vorhersage werden sehr unterschiedliche Ergebnisse bei der Nutzung von neuronalen Netzen aufgeführt. Denn je nach Datengrundlage und des verwendeten Vorhersagehorizonts war die Prognosegüte teilweise nur geringfügig besser als die Verwendung klassischer statistischer Modelle wie zum Beispiel der *Autoregressive Integrated Moving Average* [107].

Andererseits konnten neuronale Netze zeigen, dass gewisse Korrelationen wie Luftdruck und Temperatur zur resultierenden Windgeschwindigkeit weitaus mehr Relevanz haben als Windrichtung, Luftfeuchtigkeit, Sonneneinstrahlung oder Regen [107].

3.4.2.4 Ensemble Modelle

Eine sehr häufig angewendete Methode um Prognoseergebnisse zu verbessern ist Ensemble Modelle aufzustellen. Dies bedeutet, verschiedene Prognosemodelle werden kombiniert. Durch die Kombination von verschiedenen Modellen wird in der Regel der Prognosefehler minimiert, denn häufig sind verschiedene Vorhersagemodelle je nach Situation unterschiedlich gut [132].

Grundsätzlich wird diese Idee schon bei der Random Forest Regression verwendet (Abschnitt 3.4.2.3), allerdings bezieht sich nun das Kombinieren auf komplette Vorhersagemodelle und nur auf verschiedene Regressionsmodelle innerhalb eines Entscheidungsbaums.

Eine Kombination bringt vor allem dann Vorteile, wenn die verschiedenen Prognosemodelle auf unterschiedlichen Eingangsinformationen basieren. Eine beliebte Variante ein Ensemble Modell zu erstellen ist das *Bagging* Verfahren. Hierbei wird der Mittelwert der Ergebnisse von den unterschiedlichen Modellen berechnet und verwendet. Diese simple Methode minimiert den Prognosefehler vor allem bei unkorrelierten Ergebnissen und lässt sich ohne großen Rechenaufwand realisieren [132].

3.4.2.5 Fazit

Das Thema Windenergievorhersage hat durch das Wachsen von Smart Grid immer mehr an Bedeutung gewonnen. Zur effizienten Steuerung von Stromnetzen benötigt das Smart Grid System möglichst

viele Informationen von den Stromverbrauchern und den Stromerzeugern. Mit dem Anstieg der Anzahl von Offshore-Windparks tragen die Windenergieerzeuger immer mehr zum Gesamtanteil der Energieerzeugung bei. Somit wird es immer wichtiger gute Windenergieprognosen realisieren zu können.

Verschiedene Studien haben gezeigt, dass versucht wird mit verschiedenen Prognosemodellen ein optimales Windenergieprognosemodell zu finden. Da bisher nicht alle Paper ihre Prognosemodelle auf Grundlage der selben Methodik erzeugen, lässt darauf schließen, dass noch kein dominierendes Verfahren entdeckt worden ist.

Allerdings zeigt sich im Paper [107], dass neben dem eigentlichen Prognosemodell, die verschiedenen Eingangsinformationen sehr großen Einfluss auf die Prognosegüte haben. Zum einen haben die offensichtlichen Informationsquellen wie Wetter eine große Bedeutung, jedoch zeigten auch Kurzzeitprognosen allein auf Basis von historischen Energieerzeugungsdaten gute Prognoseergebnisse. Außerdem hat die Umgebung der Windkraftanlage einen sehr großen Einfluss auf die Prognose, da vergleichbare Modelle bei flachen Ebenen und Offshore Anlagen bessere Ergebnisse erzielten, als Windparks in einer hügeliger Landschaft.

Somit ist die Windenergie-Vorhersage ohne Zugriff auf Wetterdaten für Kurzzeitprognosen machbar. Diese Vorhersagemodelle sind damit nicht nur weniger komplex, sondern sparen damit die Kosten für einen kostenpflichtigen Wetterprognosedienst.

3.4.3 Vorhersage von Energieverbräuchen von Haushalten

Erneuerbare Energien sind ein wichtiger Bestandteil der Energieversorgung in Deutschland. Aufgrund des rasanten Ausbaus der Photovoltaik- und Windenergie in Deutschland müssen schon jetzt in Spitzenzeiten Stromerzeuger gedrosselt werden. Daher ist es wichtig die Einspeisung von Photovoltaikanlagen und Windenergieanlagen zu kennen, aber auch den Verbrauch der Abnehmer. Denn durch eine intelligente Regelung kann der Eigenverbrauchanteil des erzeugten Stroms erhöht werden. Ein weiterer Punkt weshalb es wichtig ist einen möglichst hohen Eigenverbrauchanteil anzustreben, ist die Tatsache, dass es mit der aktuellen Technologie nicht möglich ist, Energie effizient zu speichern.

3.4.3.1 Elektrische Last

Die elektrische Last ist die Umsetzung von Verbrauchern über einen Zeitabschnitt. Dieses bezeichnet man auch als Stromverbrauch. Das Lastprofil beschreibt die abgenommene elektrische Leistung über einen gewissen Zeitraum. In Tabelle 3.3 wurden die verschiedenen Verbraucher in zwölf Bereiche eingeteilt. Diesen Bereichen ist der prozentuale Anteil des Gesamtverbrauchs des Jahresdurchschnitts zugeordnet. Es lässt sich erkennen, dass mit zunehmender Haushaltsgröße der Gesamtbedarf steigt.

Bemerkenswert ist, dass in einem Haushalt ab fünf Personen, der mittlere Pro-Kopf-Verbrauch in etwa der Hälfte des Einpersonenhaushalts entspricht.

Bereich	Anteil	Anteile in den verschiedenen Haushaltsgrößen (%)					
		1-Prs.	2-Prs.	3-Prs.	4-Prs.	5-Prs.	6-Prs.
Beleuchtung	11,05	10,07	12,13	11,72	11,44	11,13	9,82
Umwälzpumpe	5,57	4,36	5,18	5,74	6,16	6,23	5,72
Warmwasser	11,50	14,96	12,13	10,99	10,22	9,89	10,79
Büro	12,18	14,61	11,80	11,68	11,45	11,98	11,54
TV/Audio	11,14	13,61	11,41	11,20	10,46	10,05	10,10
Kühlen	10,34	17,83	12,07	9,85	8,38	7,06	6,87
Trocknen	10,07	3,30	7,08	10,22	12,05	13,85	13,93
Kochen	8,38	7,60	9,24	8,55	8,81	8,11	7,95
Gefrieren	5,42	3,83	5,95	5,75	5,76	5,57	5,67
Spülen	5,37	2,33	4,59	5,60	6,37	6,88	6,42
Waschen	5,10	3,70	4,35	5,10	5,51	5,95	5,97
Diverses	3,90	3,79	4,08	3,62	3,39	3,31	5,22
Anzahl Datensätze	28.242	3.720	10.562	5.717	6.001	1.744	498
Jahresstromverbrauch	kWh	2.000	3.100	3.908	4.503	5.257	5.764

Tabelle 3.3: *Prozentuale Anteile der 12 Stromverbrauchsbereiche in den verschiedenen Haushaltsgrößen vgl. [220]*

Einflussfaktoren auf den Lastverlauf

Der Strombedarf über den Tag unterscheidet sich zwischen privaten Haushalten und industriellen und gewerblichen Kunden. Doch bei allen Kundengruppen besteht eine Korrelation zwischen der Last und den Werktagen, Wochenenden bzw. Feiertagen.

Neben dem Wochentyp beeinflusst auch die Jahreszeit den Lastverlauf. Dies lässt sich zum einen mit den unterschiedlichen Freizeitaktivitäten zu den verschiedenen Jahreszeiten erklären oder z. B. mit einem erhöhten Beleuchtungsbedarf im Winter. Industrieanlagen sind von saisonalen Einflüssen weniger betroffen, da beispielsweise im Sommer Klimalagen betrieben werden und im Winter Heizsysteme. Dies führt direkt zu einem weiteren wichtigen Einflussfaktor der Temperatur. Denn die Temperatur ist ausschlaggebend, ob geheizt werden muss oder klimatisiert wird. Aufgrund der Wärmeisolation von Gebäuden führen Temperaturänderungen nicht unmittelbar zu einer Laständerung, sondern sind leicht verzögert [136].

Neben den bereits genannten Einflussfaktoren gibt es viele weitere, welche in Abbildung 3.38 dargestellt sind.

- **Globalstrahlung**: Unter Globalstrahlung versteht man die auf die Erdoberfläche auftreffende Sonnenstrahlung. Bei einer niedrigen Strahlung (z. B. durch Nebel) steigt die Lichtlast [176].
- **Niederschlag**: Über den Zusammenhang von zwischen Regen und Last kann keine genaue Aussage gemacht werden. Sicher ist, dass Schneemengen, welcher den öffentlichen Verkehr zum Erliegen bringen, einen dämpfenden Effekt auf die Last haben [176].
- **Windchill**: Windchill beschreibt den Unterschied zwischen der gemessenen Lufttemperatur und der gefühlten Temperatur. Diese Differenz entsteht durch die hautnahe Luftschicht, welche den Körper umgibt. Durch diese Schicht empfindet man nicht die tatsächliche Temperatur. Wenn durch

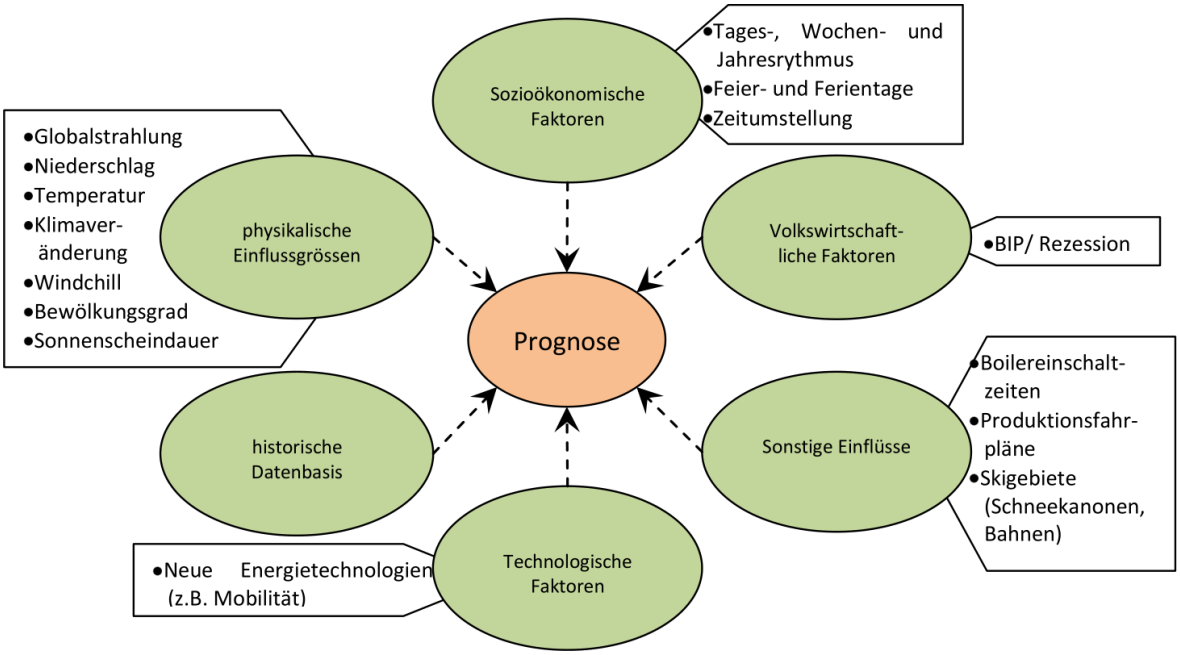


Abbildung 3.38: *Übersicht der Einflussgrössen [176]*

Wind diese „Isolationsschicht“ weggetragen wird, wird die vorherrschende Temperatur als kälter wahrgenommen. Windchill ist erst ab Temperaturen unterhalb von 7 spürbar. Der genaue Effekt von Windchill auf den Stromverbrauch ist nicht bekannt [176].

- **Sonnenscheindauer**: Die Sonnenscheindauer hat einen großen Einfluss auf die Elektrizitätsnachfrage. In Abbildung 3.39 ist erkennbar, dass die Lastspitze vom Zeitpunkt des Sonnenuntergangs abhängig ist. Denn je später der Sonnenuntergang stattfindet, desto niedriger ist die Last am Abend. Zudem hat die Sonnenstrahlung einen heizenden Effekt, welche sich ebenfalls dämpfend auf die Stromlast auswirkt [176].

- **Wochentag**: Der Abbildung 3.40 ist eindeutig zu entnehmen, dass die Tagestypen einen starken Einfluss auf den Lastverlauf haben. Die niedrigere Last am Wochenende in der Abbildung 3.40 ist darin begründet, dass Unternehmen am Wochenende größtenteils geschlossen haben [176].

- **Feier- und Ferientage**: Analog zu den Wochenenden ist an Feier- und Ferientagen der Lastverlauf niedriger [176].

Darüber hinaus nehmen Produktionsfahrpläne von großen Unternehmen einen Einfluss auf den Lastverlauf. Auch Faktoren wie die Bevölkerungsentwicklung, das Bruttoinlandprodukt (BIB), Energiepreise oder der Klimawandel beeinflussen den Lastverlauf [176]. Diese sind aber im Rahmen dieser Projektgruppe zu vernachlässigen. Die für die Projektgruppe relevanten Einflüsse sind:

- Temperatur
- Globalstrahlung

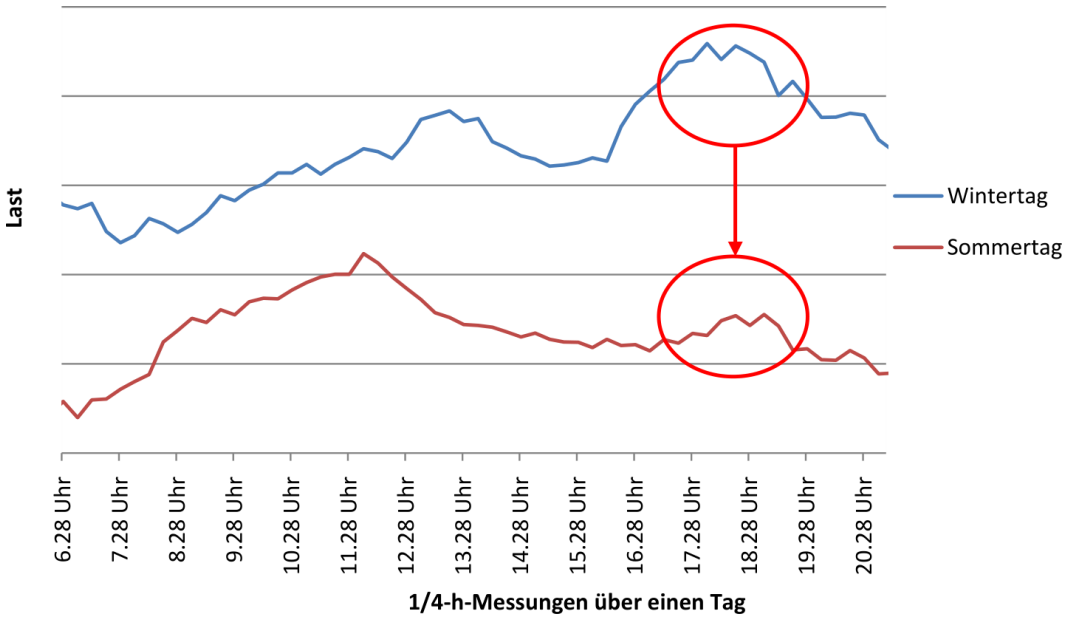


Abbildung 3.39: *Leistungsspitzen bei unterschiedlicher Sonnenscheindauer [176]*

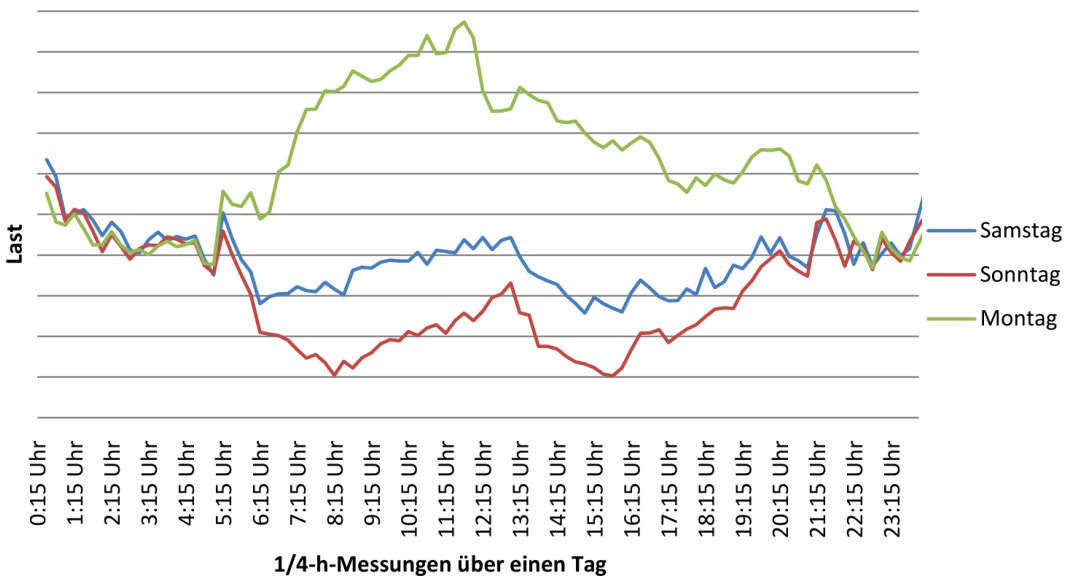


Abbildung 3.40: *Lastverläufe verschiedener Tagestypen [176]*

- Wochentag
- Jahreszeit

Da diese Daten einfach zu erhalten sind und einen großen Einfluss auf den Lastverlauf nehmen.

3.4.3.2 Verschiedene Typen von Prognosezeiträumen

Es werden drei verschiedene Typen von Prognosen auf Grundlage des Prognosezeitraumes unterschieden:

- long term load forecasting (LTLF)
- medium term load forecasting (MTLF)
- short term load forecasting (STLF)

Die LTLF beschäftigt sich mit Prognosezeiträumen von einem Jahr bis hin zu 20 Jahren [154]. Diese ist essenziell für die langfristige Entwicklung des Stromversorgungssystems. Denn LTLF bildet die Grundlage für die Bauplanung zukünftiger Erzeugungsanlagen, dass Erweitern von Generatoren und der Entwicklung des Stromnetzes. Die Genauigkeit der LTLF hat einen sehr großen Einfluss auf das Stromnetz, denn ein zu hoch ermittelter Bedarf führt zu unnötigen Investitionen, wohingegen ein zu niedrig prognostizierter Bedarf den Zusammenbruch des Stromnetzes bedeutet. Aufgrund des sehr langen Betrachtungszeitraums ist es sehr schwer, genaue Prognosen zu ermitteln. Denn die Prognose unterliegt einer Vielzahl von Einflussfaktoren. Daraus ergibt sich, dass Langzeitprognosen eine sehr geringe Genauigkeit aufweisen. Zudem spielt die LTLF für unsere Projektgruppe keine Rolle [124].

Einen Prognosezeitraum von einer Woche bis hin zu mehreren Wochen betrachtet die MTLF. Beeinflusst wird die MTLF von Faktoren wie beispielsweise der Jahreszeit, Fortschritten in der eingesetzten Technologie oder der Wartung von größeren Industrieanlagen [154]. Zudem wird die MTLF zur Bestimmung der Belastungsspitze in einer Zeitspanne genutzt. Mit den gewonnenen Informationen können Entscheidungen, wie beispielsweise nötige Wartungsarbeiten, entschieden werden. Die Änderungssensibilität ist zwischen der LTLF und STLF einzuordnen. Im Gegensatz zur LTLF bietet die MTLF eine höhere Prognosegenauigkeit [154].

Der für die Projektgruppe interessanteste Prognosezeitraum ist die STLF. Dabei wird ein Zeitraum von wenigen Minuten bis hin zu einer Woche betrachtet. STLF bildet die Grundlage für die Planung und den Betrieb der Stromversorgung. Zusätzlich bildet die STLF die Basis für eine wirtschaftlich rentablen Führung der Energieversorgung. Im Gegensatz zu der LTLF und der MTLF sind Lastprognosen in der STLF relativ genau und die Änderungssensibilität ist sehr hoch. Das bedeutet, Faktoren welche die STLF beeinflussen z. B. das Klima oder die aktuelle Preisgestaltung nehmen direkt Einfluss auf die Prognose [154].

3.4.3.3 Methoden der Lastprognose

Hauptziel der Lastprognose ist die Laststeuerung. Das bedeutet die Nachfrage nach Strom anhand der Verfügbarkeit zu steuern. Dazu wird im folgenden Kapitel eine Übersicht der Methoden zur Bestimmung der Last gegeben.

Regressionsanalyse

Bei dieser Methode werden nur die historischen Daten der elektronischen Last betrachtet. Diese werden geplottet und es wird versucht eine zugehörige mathematische Gleichung zu ermitteln. Dabei unterscheidet man zwischen der linearen Regression und der nicht linearen Regression [154].

In der Praxis sind die linearen Regressionsmodelle breit vertreten. Die allgemeine Form einer Regressionsgleichung ist in Gleichung 3.11 gegeben.

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p + a \tag{3.11}$$

Die unbekannten Parameter $\beta_0, \beta_1, \beta_2, \ldots \beta_p$ werden als Regressionskoeffizienten bezeichnet und können mit der Methode der kleinsten Quadrate bestimmt werden [136]. Das Restglied a wird häufig auch als Fehler bezeichnet. Dieser ist vorhanden, da in der Realität nie eine exakte Bestimmung der Regressionsfunktion erfolgen kann.

Dadurch, dass nur historische Daten betrachtet werden und alle anderen Faktoren, welche den Strombedarf beeinflussen, ausgeblendet werden, ist dies ein ungenaues Verfahren. Vorteil dieser Methode ist die Einfachheit. Denn schon mit wenigen Daten lassen sich Prognosen erstellen.

Lastprognose mithilfe neuronaler Netze

Neuronale Netze imitieren das Verhalten des menschlichen Gehirns. Künstliche neuronale Netze werden zur Lastprognose eingesetzt, indem der Zusammenhang zwischen der Last und deren Einflussgrößen modelliert wird [139]. Dadurch das künstliche neuronale Netze lernfähig sind, passen sich die berechneten Modelle automatisch an. Erworbenes Wissen wird über die Gewichtung der Kanten zwischen den Neuronen gespeichert. Dadurch entsteht ein präziseres Modell als bei der Regressionsanalyse, insbesondere bei mehrdimensionalen Eingangsgrößen [154] [139].

Neuronale Netze, die ein Anwendungsproblem lösen sollen, greifen typischerweise auf Backpropagation zurück. Als Aktivierungsfunktion dient häufig die Sigmoidfunktion:

$$y = \frac{1}{1 + e^{(-x)}} \tag{3.12}$$

Backpropagation besteht aus drei unterschiedlichen Neuronen. Dem Input-Layer, Hidden-Layer und den Output-Layer, vgl. Abbildung 3.41. Exogene Daten wie beispielsweise die aktuelle Temperatur oder der Wochentag werden über die Input-Neuronen empfangen. Dabei kann es sinnvoll sein Eingangsdaten in einem Bereich von $[-1,1]$ zu normieren. Die Hidden-Neuronen bilden das Modell und über den Output-Neuronen wird der prognostizierte Lastwert ausgegeben.

Die in Abbildung 3.41 eingezeichneten Kanten spielen eine wichtige Rolle in der Informationsverarbeitung innerhalb der Neuronen. Als Netzeingang (net_j) eines informationsverarbeitenden Neurons j lässt sich definieren:

$$net_j = \sum_{i=i}^{z} o_i \cdot w_{ij} \tag{3.13}$$

wobei o_i die Summe der vorherigen Einheiten ist und w_{ij} das Kantengewicht von Neuron i zu Neuron j. Anschließend wird der Aktivierungszustand des Neurons mittels des Netzeingang net_j und einer Aktivierungsfunktion y bestimmt.:

$$a_j = y(net_j) \tag{3.14}$$

Neben der bereits erwähnten Sigmoidfunktion (Gleichung 3.12) wird häufig auch die Identitätsfunktion verwendet. Hierbei entspricht der Netzeingabe dem Aktivierungszustand. Zuletzt wird der Aktvierungszustand a_j an die folgenden Neuronen weiter propagiert [136].

Die bereits erwähnten Gewichte w_{ij} können mit zufälligen Startwerten initialisiert werden. In der Trainingsphase kann anschließend der prognostizierte Lastgang mit dem tatsächlichem verglichen werden. Die mittlere quadratische Abweichung kann als Fehlerfunktion genutzt werden:

$$E_p = \frac{1}{2}\sum_j (t_{pj} - o_{pj})^2 \tag{3.15}$$

Hierbei stellt E_p den Fehler dar, $t_p j$ den gewünschten Sollwert und $o_p j$ der tatsächliche am Neuron anliegende Wert. Der Fehler wird von der Ausgabeschicht zurück an die Eingangsschicht propagiert werden und durch anpassen der Gewichte verringert sich der Fehler [136].

Neben den genannten Eingabedaten, kann das künstliche neuronale Netz mit weiteren komplexen Daten angelernt werden. Diese Daten können vergangene Lasten, das Wetter oder die Uhrzeit sein. Gut trainierte neuronale Netze besitzen eine hohe Genauigkeit. Ein weiterer Vorteil ist die kurze Antwortzeit.

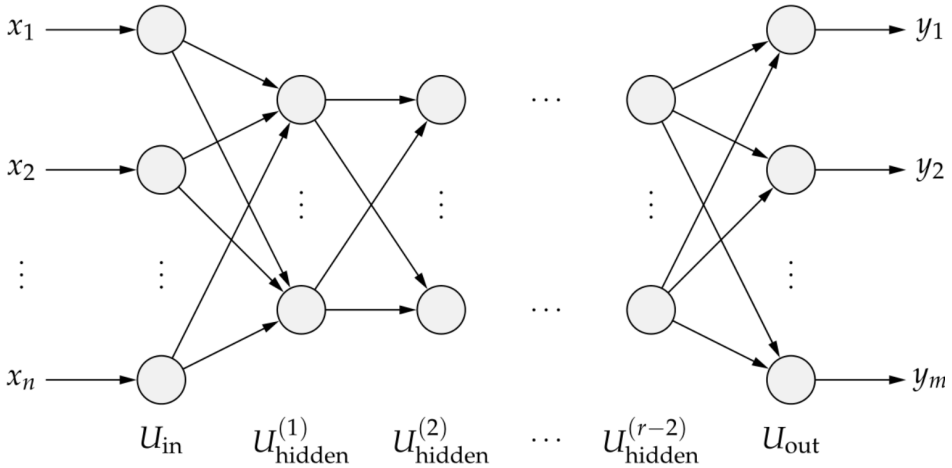


Abbildung 3.41: *Grundstruktur eines mehrschichtigen Vorwärtsnetzes mit verdeckten Schichten [178]*

Lastprognose mithilfe von Fuzzylogik

Mithilfe von Fuzzylogik ist es möglich linguistische Regeln auf ein Computerprogramm abzubilden. Dieses Prinzip wird erfolgreich in der Regelungs- und Steuerungstechnik eingesetzt. Zudem eignen sich Fuzzymethoden zur Mustererkennung von großen Datenmengen [136].

Um allgemeingültige Aussagen zu gewinnen, werden die am Eingang liegenden Daten *verrauscht* vgl. Abbildung 3.42.

Mittels der Zugehörigkeitsfunktion μ_A wird jedem Element am Eingang ein Zahlenwert zugeordnet, welcher den Zugehörigkeitsgrad des Elementes zur unscharfen Menge angibt. Dieses wird dem *inference system* übergeben [154].

Da die Qualität der Prognose sehr stark vom Regelwerk abhängig ist, ist es notwendig, dass Wissen über Korrelationen innerhalb der Daten vorhanden ist [238].

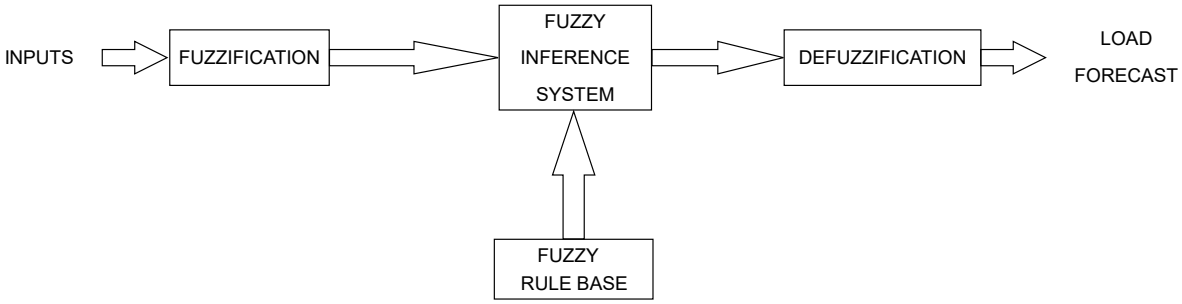


Abbildung 3.42: *Fuzzysystem zur Lastprognose [154]*

Lastprognose mithilfe von Evolutionären Algorithmen

EAs sind Optimierungsalgorithmen und gehören zur Klasse der Heuristiken. Diese Algorithmen orientieren sich an der Biologie, indem sie die Evolution als Optimierungsprozess betrachten. Hierbei kann die DNS als Lösung betrachtet werden. In Listing 3.10 ist der schematische Ablauf eines evolutionären Algorithmus dargestellt. Im ersten Schritt wird eine Menge an Lösungen benötigt, mithilfe derer der EA startet. Anschließend findet eine Rekombination der Lösungen statt. Anschließend werden die rekombinierten Lösungen mutiert. Dies geschieht, um dem Prozess eine zufällige Komponente hinzuzufügen. Zuletzt findet die Bewertung der neu gewonnenen Lösung statt und die Selektion der Lösungen für eine erneute Iteration. Terminiert werden kann der EA beispielsweise nach einer festgelegten Anzahl an Iterationen oder wenn eine vorher definierte Genauigkeit erreicht ist [174].

```
1  initialize population
2  repeat
3    repeat
4      crossover
5      mutation
6      phenotype mapping
7      fitness computation
8    until population complete
9    selection of parental population
10 until termination condition
```

Listing 3.10: *Grundlegender Evolutionärer Algorithmus vgl. [174]*

Dieses Prinzip lässt sich auf viele Probleme anwenden. Überträgt man dieses Vorgehen auf die Lastprognose, werden im ersten Schritt Prognosen mithilfe von historischen Daten erstellt. Diese werden dann mittels der Operatoren Crossover, Mutation manipuliert. Die entstandenen Prognosen werden bewertet. Gute Prognosen werden behalten und für die kommenden Iteration genutzt. Schlechte Prognosen werden verworfen. Nach Terminierung wird die Prognose mit der höchsten Fitness als Lösung betrachtet. Problem hierbei ist aber, dass nicht sichergestellt werden kann, ob ein globales Optimum gefunden wurde. Vorteil bei diesem Verfahren ist allerdings, dass kein Training benötigt wird und die Prognose relativ schnell erstellt ist [154].

3.4.3.4 Fazit

Zur genauen Prognose von Lasten werden eine Reihe von Einflussfaktoren benötigt. Je mehr berücksichtigt werden, desto genauer werden die Prognosen. Zudem spielt der betrachtete Prognosehorizont eine wichtige Rolle für Lastprognosen. Denn je größer der Prognosehorizont gewählt wird, umso ungenauer werden die erstellten Prognosen. Die vorgestellten komplexen Prognosemethoden liefern alle brauchbare Prognosen für kurze Prognosezeiträume.

4 Projektmanagement

4.1 Teammanagement

Um die Arbeit in einem Softwareprojekt zu strukturieren, sollte auf bewährte Projektmanagementmethoden zurückgegriffen werden. Diese unterstützen bei korrekter Anwendung eine effiziente und zielführende Durchführung eines Projektes. In diesem Abschnitt werden die Grundlagen der Projektgruppenarbeit beschrieben. Hierzu werden zunächst die Grundlagen des Projektvorgehensmodells *Scrum* erläutert. Anschließend werden die Rollen beschrieben, welche von den Projektgruppenmitgliedern zu besetzen sind. Danach werden der Ablauf des Projektes skizziert und einige Aspekte des Taskmanagements beschrieben. Zudem werden ausgewählte Softwarewerkzeuge zur Unterstützung der Prozesse genannt.

4.1.1 Projektvorgehensmodell Scrum

Scrum ist ein iteratives Vorgehensmodell, welches heutzutage (Stand Mai 2017) insbesondere in der Softwareentwicklung zunehmend vorzufinden ist. Das Projektvorgehensmodell Scrum ist für die Projektgruppenarbeit gut geeignet, da die meisten Projektgruppenmitglieder bereits Erfahrung in der Arbeit nach diesem Vorgehensmodell besitzen. Zudem bietet Scrum die Möglichkeit, das zu entwickelnde Produkt in Iterationen (auch Sprints genannt) zu entwickeln, so dass im Projektverlauf auf sich verändernde Anforderungen reagiert werden kann (vgl. [114], S. 102). Dies ist auch vor dem Hintergrund sinnvoll, dass zum Zeitpunkt der Anfertigung dieses Kapitels die Produktanforderungen noch sehr vage formuliert sind.

4.1.1.1 Artefakte

In diesem Abschnitt werden die Hauptelemente (üblicherweise als Artefakte bezeichnet) des Scrum-Vorgehensmodells beschrieben.

Der Product-Backlog enthält alle bekannten Produktanforderungen. Hierzu gehören sowohl funktionale und nicht-funktionale Anforderungen, als auch zu behebende Fehler (vgl. [188], S. 2). Anforderungen aus dem Product-Backlog, die innerhalb des jeweiligen Sprints umgesetzt werden sollen, befinden sich im Sprint-Backlog. Der Bearbeitungsfortschritt der Anforderungen aus dem Sprint-Backlog sollte gut nachverfolgbar sein. Daher sollten die Aufgaben einen gewissen Mindestumfang haben, aber auch nicht zu viel Aufwand erfordern (vgl. [128], S.75).

Das *Inkrement* steht am Ende der Wertschöpfungskette innerhalb des Scrum-Vorgehensmodells. Mit dem Begriff Inkrement wird der zum Ende jeder Iteration geschaffene Produktfortschritt bezeichnet.

Mit dem Sprint-Burndown-Chart kann der Sprintfortschritt auf einfache Weise visualisiert werden. Dies ermöglicht es, jederzeit einen Überblick darüber zu gewinnen, ob das Entwicklungsteam in der aktuellen Iteration innerhalb des Zeitplans arbeitet. Der verbleibende Arbeitsaufwand wird auf der Y-Achse eines Diagramms entlang der verbleibenden Sprintdauer auf der X-Achse abgetragen. Dabei sollte der verbleibende Aufwand möglichst gleichmäßig abgearbeitet werden. Alternativ kann auch die Anzahl der verbleibenden Aufgaben abgetragen werden, wobei die verbleibende Stundenanzahl i.d.R. eine größere Genauigkeit bieten kann. Abbildung 4.1 zeigt eine beispielhafte Burndown-Chart.

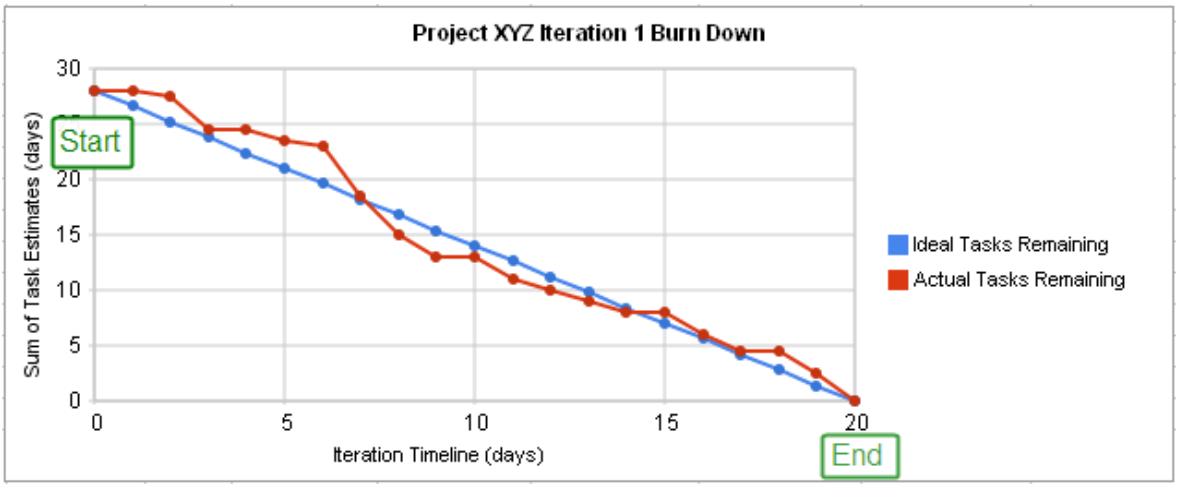


Abbildung 4.1: *Beispiel für ein Sprint-Burndown-Chart (Autor: I8abug, wikipedia.org, Lizenz: https://creativecommons.org/licenses/by-sa/3.0/deed.en)*

Am Ende eines Sprints kann die Anzahl der produktiv geleisteten Arbeitsstunden als *Velocity* in ein Diagramm übertragen werden. Somit ist es möglich, die Produktivität des Teams im Projektverlauf zu messen. Dies bietet die Möglichkeit, etwaige Zusatzaufwände, wie z.B. für Abstimmungstermine zu identifizieren und bei der Sprintplanung zu berücksichtigen. Auch kann so die Effektivität von Prozessverbesserungsmaßnahmen beurteilt werden.

4.1.1.2 Rollen

Im Rahmen von Scrum werden verschiedene Rollen definiert. Die Rollen sind klar abgegrenzt. Somit werden die Zuständigkeiten klar definiert. Dies sorgt für transparente Kommunikationswege innerhalb der Organisation bzw. des Teams.

Der Product Owner ist dafür verantwortlich, dass das Produkt möglichst viele der Kundenanforderungen erfüllt. Daher verwaltet er die Priorisierung der Anforderungen innerhalb des Product-Backlogs und gibt somit auch die Priorität vor, mit der die Anforderungen im Rahmen der Sprints bearbeitet werden. Die Rolle des Product Owners wird, wie vorgegeben, durch den Projektgruppenbetreuer wahrgenommen.

Die Aufgaben des Scrum-Masters bestehen darin, das Entwicklungsteam bei der Arbeit zu unterstützen. Die Unterstützung besteht dabei in erster Linie darin, den Ablauf der regelmäßigen Meetings im Rahmen des Scrum-Zyklus vorzubereiten und zu steuern. Zudem ist der Scrum-Master dafür verantwortlich, etwaige Hindernisse (auch Impediments genannt, wie z.B. fehlende Softwarelizenzen), die dem Sprintziel entgegenstehen, schnellstmöglich zu beseitigen. Darüber hinaus besitzt er gegenüber dem Product Owner auch eine Kontrollfunktion, in der er dafür Sorge zu tragen hat, dass die Sprintziele nicht auf unzulässige Weise während des Sprints geändert werden. (vgl. [247], S. 14 f.)

Das Entwicklungsteam steht in der Verantwortung, die Anforderungen aus dem Sprint-Backlog zu erfüllen. Dabei sollten die Teammitglieder möglichst abwechselnd Tätigkeiten aus den folgen-

den Aufgabenbereichen wahrnehmen, um Kopfmonopole[1] zu vermeiden: Anforderungsspezifikation, Implementierung, Test, Inbetriebnahme und Wartung. Innerhalb des Entwicklungsteams wurden im Rahmen des zweiten Projektgruppentreffens, zusätzlich zu den klassischen Rollen in Scrum, weitere Rollen definiert, wie die Rolle *Release-Engineer*, welche die Erstellung eines lauffähigen Exemplars der Software während des Sprints, sowie die Automatisierung und Betreuung des Continuous Integration-Systems verantwortet.

Aus der praktischen Erfahrung des Autors besteht bei vielen der Tätigkeiten, welche die Teammitglieder in den zuvor beschriebenen Rollen wahrzunehmen haben, die Möglichkeit der Werkzeugunterstützung und deren Automatisierung. In Unterunterabschnitt 4.1.2.1 werden beispielhaft Werkzeuge vorgestellt, welche einen hohen Automatisierungsgrad unterstützen. Da die Werkzeuge sich je nach eingesetztem Technologiestack unterscheiden können, kann an dieser Stelle keine konkrete Empfehlung ausgesprochen werden. Die Auswahl der Werkzeuge wird daher im Rahmen eines technischen Durchstichs im Projektverlauf erfolgen.

4.1.1.3 Ablauf und Taskmanagement

In diesem Abschnitt werden die Regeltermine einer Scrum-Iteration beschrieben. Abbildung 4.2 zeigt eine Übersicht des Ablaufs eines typischen Scrum-Prozesses: die Anforderungen aus dem Product-Backlog werden im Rahmen von Sprints inkrementell zu lauffähiger Software implementiert.

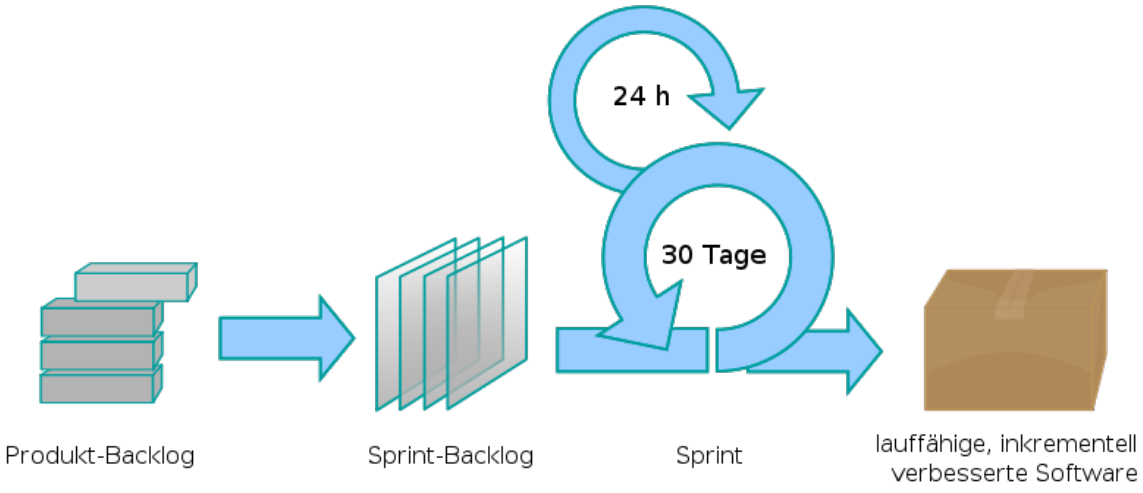


Abbildung 4.2: *Beispiel für einen Scrum-Prozess (Autor: Lakeworks, bearbeitet von Stefan Wallroth, wikipedia.org, Lizenz: `https://creativecommons.org/licenses/by-sa/3.0/deed.en`)*

[1] Mit Kopfmonopolen wird an dieser Stelle ein Phänomen beschrieben, bei dem einzelne Teammitglieder das alleinige Wissen zu bestimmten fachlichen oder technischen Themen besitzen. Dies führt insbesondere bei Abwesenheiten der entsprechenden Personen erfahrungsgemäß dazu, dass auf bestimmte Wissensfragmente nicht rechtzeitig zugegriffen werden kann. Dies kann die Erreichung des Sprint-Zieles gefährden.

Der Prozess des Taskmanagements sollte so gestaltet sein, dass das Erstellen neuer Aufgaben möglichst wenig Kommunikationsaufwand beinhaltet. Daher soll es für jedes Teammitglied möglich sein, Aufgaben zu erstellen. Nach der Erstellung einer Aufgabe sollten jeweils relevante Stakeholder von dem Ersteller der Aufgabe direkt informiert werden.

Sobald das Ziel des Projektes bekannt ist, kann eine Roadmap erstellt werden, die einen groben Zeitplan dafür enthalten soll, zu welchem Zeitpunkt welche Features des Systems umgesetzt sein sollte. Dabei sind auch die Meilensteine *vertikaler Prototyp*, *Halbzeitpräsentation* und *Endpräsentation* aus dem generellen Ablaufplan der Projektgruppe zu berücksichtigen. Der erstellten Aufgaben und deren Priorisierung sollten möglichst dieser Roadmap entsprechen, daher sollte diese Roadmap in Abstimmung mit dem Product Owner erarbeitet werden.

Im gesamten Projektverlauf soll stets darauf geachtet werden, dass die Meetings zielführend durchgeführt werden. Dazu sollten die Ziele jedes Meetings *SMART* (spezifisch, messbar, akzeptiert, realistisch und terminiert) formuliert werden.

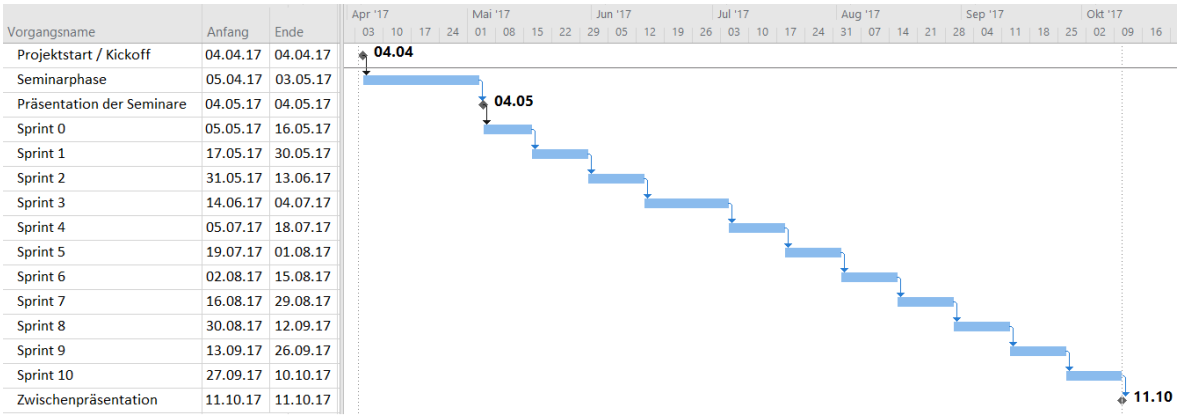
4.1.1.4 Sprintplanung

Vor der Sprintplanung sollte eine Kapazitätsplanung erfolgen. Hierbei werden die Verfügbarkeiten der Mitglieder des Entwicklungsteams für den Zeitraum des Sprints kumuliert. Dabei sind Urlaube und andere Abwesenheiten zu berücksichtigen, welche daher, soweit vorhersehbar, rechtzeitig vor der Kapazitätsplanung bekannt gegeben werden sollten.

Im Rahmen der Sprintplanung werden die im folgenden Sprint zu bearbeitenden Aufgaben ausgewählt. Dabei ist es sinnvoll, Aufgaben in Teilaufgaben zu zerlegen, wenn der Gesamtaufwand der Aufgabe als zu hoch eingeschätzt wird. Das Sprint-Backlog ist dabei nicht als statisches Konstrukt anzusehen. Es wird gemeinsam vom Product Owner und Entwicklungsteam gestaltet und kann während des Sprints verändert werden (vgl. [191], S. 299). In der Projektgruppe sollte eine Sprintdauer von 2-3 Wochen angestrebt werden. Eine kürzere Sprintdauer würde dazu führen, dass verhältnismäßig viel Zeit für Regeltermine wie das Estimation Meeting, Sprint-Review und die Sprint-Retrospektive, sowie die Releaseerstellung aufgewendet werden müsste, da diese innerhalb der Projektlaufzeit bei kürzeren Sprints entsprechend häufiger stattfinden müssten. Eine längere Sprintdauer hat den Nachteil, dass die Flexibilität im Bezug auf sich verändernde Anforderungen sinkt.
Eine besondere Herausforderung besteht darin, innerhalb des ersten Sprints die Anforderungen zu spezifizieren, eine Technologieauswahl zu treffen, die technische Infrastruktur in Betrieb zu nehmen, sowie einen technischen Durchstich zu erzielen.

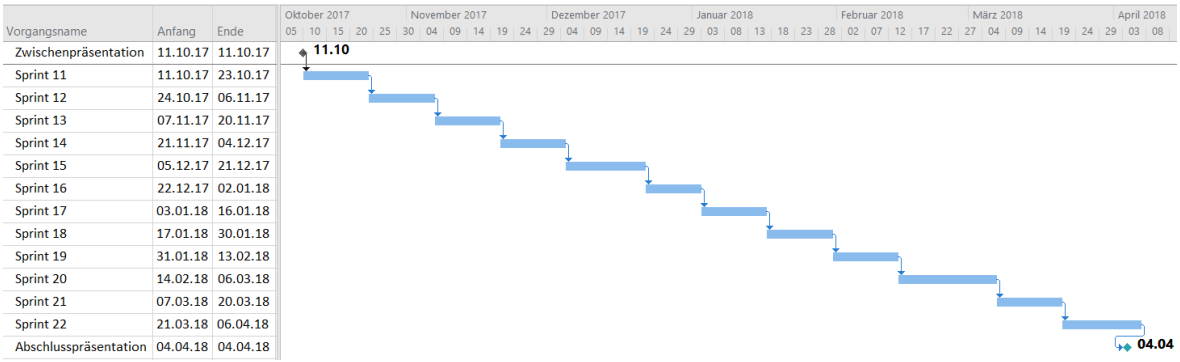
Im Laufe des Projekts wurde ein feingliedriger Sprintplan mit jedem einzelnen Sprint aufgestellt. Der Sprintplan in Abbildung 4.3 wurde bis zur Zwischenpräsentation am 11.10.2017 der Projektgruppe aufgestellt, welche als Meilenstein definiert ist und grob die Hälfte der gesamten Projektzeit abdeckt. Neben den einzelnen, durchnummerierten Sprints zeigt dieser Plan außerdem die zeitliche Zuteilung der Seminarphase. Die Seminarphase schließt mit den Präsentationen der verschiedenen Seminare am 04.05.2017 ab.

Für die zweite Projekthälfte wurde auch ein feingliedriger Sprintplan angelegt. Der in Abbildung 4.4 gezeigte Sprintplan startet mit der Zwischenpräsentation am 11.10.2017 und endet mit der Abschlusspräsentation am 04.04.2018. Die genauen Inhalte der jeweiligen Sprints stehen im Abschnitt 7.3.



Vorgangsname	Anfang	Ende
Projektstart / Kickoff	04.04.17	04.04.17
Seminarphase	05.04.17	03.05.17
Präsentation der Seminare	04.05.17	04.05.17
Sprint 0	05.05.17	16.05.17
Sprint 1	17.05.17	30.05.17
Sprint 2	31.05.17	13.06.17
Sprint 3	14.06.17	04.07.17
Sprint 4	05.07.17	18.07.17
Sprint 5	19.07.17	01.08.17
Sprint 6	02.08.17	15.08.17
Sprint 7	16.08.17	29.08.17
Sprint 8	30.08.17	12.09.17
Sprint 9	13.09.17	26.09.17
Sprint 10	27.09.17	10.10.17
Zwischenpräsentation	11.10.17	11.10.17

Abbildung 4.3: *Sprintplan Teil 1*



Vorgangsname	Anfang	Ende
Zwischenpräsentation	11.10.17	11.10.17
Sprint 11	11.10.17	23.10.17
Sprint 12	24.10.17	06.11.17
Sprint 13	07.11.17	20.11.17
Sprint 14	21.11.17	04.12.17
Sprint 15	05.12.17	21.12.17
Sprint 16	22.12.17	02.01.18
Sprint 17	03.01.18	16.01.18
Sprint 18	17.01.18	30.01.18
Sprint 19	31.01.18	13.02.18
Sprint 20	14.02.18	06.03.18
Sprint 21	07.03.18	20.03.18
Sprint 22	21.03.18	06.04.18
Abschlusspräsentation	04.04.18	04.04.18

Abbildung 4.4: *Sprintplan Teil 2*

4.1.1.5 Estimation Meeting

Das Estimation Meeting dient dazu, den Arbeitsaufwand jener Aufgaben abzuschätzen, die sich noch ungeschätzt im Product-Backlog befinden. Dabei können verschiedene Maßeinheiten zur Aufwandsschätzung eingesetzt werden, wie z.B. Story Points. Story Points stellen eine abstrakte Maßeinheit für den Arbeitsaufwand zur Erledigung einer Aufgabe dar. Eine weitere, beliebte Art, die Aufwände einer Aufgabe bzw. eines Sprints zu schätzen bzw. zu protokollieren, sind die benötigten Arbeitsstunden (vgl. [62], S 43).

Da für das Projekt ein wöchentlicher Workload von 14 Stunden pro Person vorgegeben wurde, bietet es sich an, die Aufwandsschätzungen ebenfalls in Stunden vorzunehmen. Somit ist ein einfacher Abgleich mit der ermittelten Verfügbarkeit aus der Kapazitätsplanung möglich. Auch in dem bereits ausgewählten Issue-Tracking-System *JIRA* werden die Projektaufwände in Stunden protokolliert. Daher bietet es sich an, die Zeitaufwände einheitlich in Stunden zu bemessen.

4.1.1.6 Daily Scrum

Während des Daily Scrum-Meetings berichtet jedes Mitglied des Entwicklungsteams der Reihe nach seinen Fortschritt innerhalb des Sprints seit dem letzten Daily Scrum Meeting. Dabei beantwortet die jeweilige Person idealerweise die folgenden Fragen:

- Was habe ich gestern (bzw. seit dem letzten Daily Scrum) gemacht?
- Was mache ich heute (bzw. bis zum nächsten Daily Scrum)
- Welche Probleme hindern mich ggf. an meinem Fortschritt?

Dieses Meeting dient dazu Kommunikationsprobleme zu identifizieren und zu lösen. Zudem erhält jedes Mitglied des Entwicklungsteam so regelmäßig Informationen darüber, wer an welchen Aufgabenpaketen arbeitet. Das Daily Scrum-Meeting sollte an einen engen Zeitplan geknüpft sein. Ausschweifende Diskussionen sollten vom Scrum-Master ggf. unterbunden und auf zielgerichtete Meetings mit den erforderlichen Personen verschoben werden. Im Rahmen der Projektgruppe ist aufgrund der verschiedenen Vorlesungszeiten der Studierenden, sowie der Wochenarbeitszeit von 14 Stunden pro Person innerhalb des Projektes ein Weekly Scrum-Meeting sinnvoll, da sich voraussichtlich nicht jeden Tag ein gemeinsamer Termin mit allen Projektgruppenmitgliedern vereinbaren lässt.

4.1.1.7 Definition of Done

Die Definition of Done beschreibt, welche Kriterien erfüllt sein müssen, damit eine Aufgabe des Sprints als abgeschlossen bezeichnet werden kann. Diese Definition of Done wird vom Entwicklungsteam in Abstimmung mit dem Product Owner erarbeitet.

Eine Aufgabe sollte ggf. nur dann als abgeschlossen betrachtet werden, wenn das Ergebnis gut nachvollziehbar dokumentiert und von einer anderen Person als der zuvor bearbeitenden Person gesichtet wurde. Im Rahmen der Softwareentwicklung sollten insbesondere folgende Aspekte bedacht werden: Erfüllung nicht-funktionaler Anforderungen, Quellcode sollte in das Versionskontrollsystem persistiert worden sein, das Betriebshandbuch sollte angepasst worden sein, ggf. Code-Review, erfolgreich durchgelaufene Unit-Tests.[2] Dabei sollte allerdings beachtet werden, dass die Erfüllung der Definition mit verhältnismäßigem Aufwand zu erreichen ist, um die Wirtschaftlichkeit des Projektes zu wahren.

Auch sollte die Definition of Done nicht als statisches Artefakt betrachtet werden, sondern bei Bedarf angepasst werden. Für unterschiedliche Aufgaben (z. B. Feature-Entwicklung oder Fehlerbehebung) könnten prinzipiell auch unterschiedliche Definitionen geschaffen werden. Ein weiterer wählbarer Ansatz wäre die Definition von Reifegraden. Die Einhaltung der Definition of Done kann durch den Product Owner überprüft werden. Für die Mitglieder des Entwicklungsteams kann eine Checkliste bereitgestellt werden, anhand derer diese das Einhalten der Definition of Done für die jeweilige Task prüfen können. Ist ein Teammitglied der Meinung, dass ein Task als *Done* markiert werden kann, so sollte es den Ticketstatus auf *Resolved* setzen und den Product Owner um eine Abnahme bitten. Ist die Abnahme erfolgreich, kann der Product Owner den Ticketstatus von *Resolved* auf *Closed* setzen. Wird die Abnahme verweigert, kann das Ticket vom Product Owner wieder geöffnet werden.

[2] Schlagen Unit-Tests sporadisch fehl, kann dies gleichzeitig ein Indiz für nicht erfüllte nicht-funktionale Anforderungen sein.

4.1.1.8 Ablauf von Sprints

Während der Sprints sollten die Aufgaben vom Zeitaufwand gesehen gleichmäßig von allen Projektgruppenmitgliedern abgearbeitet werden. Zudem sollten die implementierten funktionalen und nicht-funktionalen Anforderungen sprintbegleitend getestet werden. Diese Empfehlung lässt sich daraus ableiten, dass der Aufwand für Korrekturen in frühen Phasen der Entwicklung wesentlich weniger Aufwand bedeutet, als in späteren Phasen (vgl. [137], S. 21 f.). Dabei sollen möglichst agile Entwicklungsmethoden wie Test-Driven-Development (vgl. [62], S. 99 f.) auf Basis von automatisierten Unit- und Integrationstests eingesetzt werden. Auftretende Hindernisse, welche die Erreichung der Sprintziele verhindern, sollten von den Mitgliedern des Entwicklungsteam zeitnah an den Scrum-Master kommuniziert werden, damit dieser an der Beseitigung dieser Hindernisse arbeiten kann.

4.1.1.9 Sprint-Review und Sprint-Retrospektive

Das Sprint-Review-Meeting , welches am Ende eines Sprints stattfindet, dient dazu, dem Product Owner und ggf. anderen Stakeholdern das Ergebnis (Produktinkrement) des jeweiligen Sprints zu präsentieren. Die Präsentation sollte live am System erfolgen, um das Verhalten des Systems nach der Auslieferung bei dem Kunden möglichst vergleichbar darstellen zu können. Dabei bietet es sich an, dass, falls anwesend, der Kunde die Bedienung des Systems übernimmt, um etwaige verbleibende Usability-Probleme identifizieren zu können. Möglicherweise werden hier Aufgaben für zukünftige Sprints identifiziert.

In dem Meeting *Sprint-Retrospektive* wird der Verlauf des vergangenen Sprints aus methodischer Sicht reflektiert. Es besteht hier die Möglichkeit, z.B. die Qualität der Aufwandsschätzungen und die Arbeitseffizienz des Teams zu diskutieren (vgl. [76], S. 236). Dies dient vor allem dazu, Potenziale für Prozessverbesserungen zu identifizieren. Da dieses Meeting nach jedem Sprint stattfindet, wird hierüber eine kontinuierliche Verbesserung der Prozesse erzielt. Dafür erforderliche Maßnahmen sollten in diesem Meeting mit allen Teammitgliedern erarbeitet werden. Es sollte jedoch auch im Verlauf des Sprints möglich sein, bei erkannten Problemen schnell auf diese zu reagieren. Daher sollten die Teammitglieder etwaige Probleme direkt ansprechen und nicht bis zum Ende des Sprints damit warten.

4.1.2 Werkzeuge zur Prozessunterstützung

In diesem Abschnitt werden Werkzeuge vorgestellt, welche die Durchführung des Projektes und die Steuerung des Scrum-Prozesses unterstützen können. Dazu gehören sowohl Softwarewerkzeuge, als auch analoge Werkzeuge, u.a. zur Visualisierung von zu bearbeitenden Inhalten oder zur Aufwandsschätzung.

4.1.2.1 Softwarewerkzeuge

In diesem Abschnitt werden die Softwarewerkzeuge (JIRA, Confluence, Bitbucket, etc.) beschrieben, welche von der Projektgruppe genutzt werden können, um den Entwicklungsprozess zu unterstützen. JIRA ist ein verbreitetes Issue-Tracking-System (vgl. [62], S. 47). Es kann genutzt werden, um das Product-Backlog und das Sprint-Backlog in Form von User Stories, Epics, Aufgaben, Unteraufgaben und Fehlertickets zu pflegen. Zudem bietet es die Möglichkeit, benutzerdefinierte Dashboards

anzulegen, auf denen sich ein Nutzer die ihm zugewiesenen, offenen Aufgaben anzeigen lassen und nach Priorität sortieren lassen kann. Auch grafische Darstellungen, wie z.B. Sprint-Burndown-Charts können zu einem solchen Dashboard hinzugefügt werden. JIRA bietet zudem die Möglichkeit der Installation von Addons. So gibt es beispielsweise Addons, mit denen sich Commits aus Versionskontrollsystemen, welche den Schlüssel von Aufgaben in der Commitnachricht beinhalten, in den entsprechenden Aufgaben auflisten lassen. Somit lässt sich leicht nachvollziehen, aufgrund welcher Anforderung eine Änderung am Quellcode vorgenommen wurde.

Wichtige Informationen und Dokumentationen werden in einem Wiki abgelegt. Als Softwarelösung wurde bereits *Confluence* des Herstellers Atlassian ausgewählt. Dieses Wiki lässt sich gut mit JIRA integrieren. Beispielsweise lassen sich darin Aufgaben aus JIRA verlinken, so dass deren Status neben der Verlinkung angezeigt wird.

Als Versionsverwaltung hat die Projektgruppe bereits das System *Bitbucket* von Atlassian ausgewählt. Bitbucket ist eine webbasierte Applikation, welche das Versionskontrollsystem Git unterstützt. Die signifikantesten Unterschiede von Git gegenüber Subversion, einem anderen weit verbreiteten Versionskontrollsystem sind u.a. die Möglichkeit, Commits im Offlinemodus auszuführen, Fokus auf die Arbeit mit Branches und die fehlende Möglichkeit, einzelne Dateien auszuchecken. Neben Bitbucket existieren weitere, freie Webapplikationen, welche bereits ein Issue-Tracking integrieren, wie z.B. GitLab und GitHub.

Zur Kommunikation hat sich die Projektgruppe für den Messenger *Slack* entschieden. Dieser ist sowohl für mobile Endgeräte, als auch für den PC verfügbar. Für Nachrichten, die über längere Zeit persistiert und zuverlässig an alle Teammitglieder weitergeleitet werden sollen, wurde ein E-Mail-Verteiler eingerichtet. Dieser bildet die primäre Kontaktmöglichkeit mit dem Team für die anderen Projektgruppen. Details zu einzelnen Teilaufgaben können in Form von Kommentaren an den Aufgaben im JIRA kommuniziert werden.

Scrum verlangt von dem Entwicklungsteam, dass am Ende des Sprints ein fertiges Produktinkrement auslieferbar sein soll, wobei der Begriff *fertig* impliziert, dass die Auslieferung mit wenig oder gar keinem Aufwand erfolgen soll (vgl. [190], S. 177). Um den benötigten manuellen Aufwand für die Bereitstellung eines lauffähigen Produktes zu reduzieren bietet es sich an, nach dem DevOps-Prinzip zu verfahren.

DevOps ist ein Ansatz, um die Entwicklung und den Betrieb von Softwarelösungen zu verbessern. Durch die Harmonisierung der Prozesse von Entwicklung, Betrieb und Qualitätssicherung soll dabei die Auslieferungsgeschwindigkeit einer Organisation verbessert werden. Dabei spielt auch Automatisierung eine wichtige Rolle, da viele stabile Releases ermöglicht werden sollen (vgl. [130], S. 5). Hierzu gibt es verschiedene, weit verbreitete Softwarelösungen, die den Aufbau einer sogenannten Continuous Deployment Pipeline ermöglichen. Eine solche Pipeline umfasst üblicherweise mehrere Einzelschritte. Der Auslöser für den Ablauf der Continuous Deployment Pipeline ist oft das Einchecken von Quellcode in das Versionskontrollsystem. Danach wird der Source-Code automatisiert auf einem Build-Server ausgecheckt und kompiliert. Anschließend werden automatisierte Unit- und - soweit vorhanden, Integrationstests ausgeführt. Wurden diese erfolgreich beendet, können die kompilierten Softwarepakete in ein Binär-Repository (z.B. Artifactory von JFrog Ltd. oder Nexus von Sonatype Inc.) publiziert werden. Nach einer Freigabe können diese automatisiert auf ein Konsolidierungssystem installiert, dort getestet und nach Abnahme auf dem Produktivsystem installiert und konfiguriert werden. Populäre Softwaretools zur Automatisierung bzw. Unterstützung der Prozesse sind u.a. Jenkins, Travis

CI bzw. Crucible für Codereviews, JaCoCo für Code-Coverage-Analysen und Sonar für statische Codeanalyse.

4.1.2.2 Analoge Werkzeuge

An dieser Stelle sei noch erwähnt, dass die Prozesse auch durch analoge Werkzeuge unterstützt werden können. So können Aufgaben auch in Form von Ausdrucken oder handgeschriebenen Papierkarten an einer Wand in Form von Swimlanes angeordnet werden. Die Nachteile dabei sind jedoch, dass ein Teammitglied zur Bearbeitung der Aufgaben vor Ort sein muss und dass die gleichzeitige Bearbeitung durch mehrere Personen sich schwierig gestalten kann. Zudem ist die Nachverfolgung des Aufgabenstatus über lange Zeit oft nicht möglich, da diese Informationen zumeist nicht protokolliert werden. Für Meetings bietet es sich an, Pinnwände, Flip-Charts und einen Moderationskoffer (Bestückt mit verschiedenfarbigen Karten, Stiften, Pinnwandnadeln, etc.) zur verwenden, um Inhalte zu visualisieren.

Für die Aufwandsschätzung im Rahmen des Estimation Meetings kann die spielerische Methode *Planning Poker* angewandt werden. Ziel ist es, einen Konsens über den schätzungsweise benötigten Aufwand zur Abarbeitung einer Aufgabe zu erzielen. Dabei wird jedem an der Schätzung beteiligten Projektmitarbeiter ein Kartensatz mit Karten ausgeteilt, welche mit Zahlen nach der Fibonacci-Reihe beschriftet sind. Üblich sind hier auch die Kartenwerte 0, $\frac{1}{2}$, 1, 2, 3, 5, 8, 13, 20, 40, 100 und *?*. Jeder an der Schätzung Beteiligte wählt verdeckt eine Karte, deren Wert seiner Einschätzung nach den benötigten Story Points zur Bearbeitung der Aufgabe entspricht. Anschließend decken alle gleichzeitig die Karten auf. Gibt es Abweichungen zwischen den Schätzungen, wird über die Gründe für die jeweilige Einschätzungen diskutiert, bis ein Konsens erzielt wurde.

4.1.3 Fazit

Im Rahmen der Projektgruppe soll Scrum als Vorgehensmodell angewendet werden, um der entsprechenden Vorgabe seitens des Projektgruppenbetreuers gerecht zu werden. Dabei sollten die Prozesse jedoch pragmatisch gestaltet werden, um flexibel auf Veränderungen innerhalb des Projektes reagieren zu können. Hierzu ist es besonders wichtig, die Meetings zielführend zu gestalten, damit die dafür aufgewendete Zeit effizient genutzt werden kann.

4.2 Umsetzung in der Projektgruppe

4.2.1 Rollenverteilung

Als wichtiger Punkt um Verantwortlichkeiten und Ansprechpartner festzulegen, wurde zu Beginn der Projektgruppe die Rollenverteilung vorgenommen. Außerdem wurde festgelegt, was in den Aufgabenbereich der jeweiligen Rollen fällt.

Dokumentations- und Confluencebeauftragter

Diese Rolle wird von Friederike Bruns eingenommen und beinhaltet als Aufgaben das Festlegen der Latex- Dokumentenstruktur, die Unterstützung bei Latexproblemen und die Dokumentüberwachung (welche Texte fehlen noch oder müssen überarbeitet werden?). Außerdem zählen zu den Aufgaben die Confluencepflege, also das Hinzufügen fehlender Seiten, das Eintragen von Terminen und allgemeinen

Informationen und darüber hinaus die Überwachung der Einhaltung von Struktur. Als wichtige Skills werden dabei Latex- Kenntnisse und die Confluencehandhabung erachtet.

Git-Master

Die Aufgabe des Git-Masters wird von Tim Silhan übernommen, der als Ansprechpartner für Git und GitFlow Fragen dient. Außerdem ist er der Standardprüfer bei Pull-Requests, um die Sauberkeit des Git-Verlaufs sicherzustellen. Als Unterstützung übernimmt auch Timo Wolters teilweise diese Aufgabe.

GUI-Beauftragter

Viktor Spadi nimmt die Aufgabe des GUI-Beauftragten ein. Hierbei übernimmt er die Verantwortlichkeit für das Reviewen, beziehungsweise Erstellen der GUI-Mockups und das Festlegen des Farbschemas.

Kommunikationsbeauftragter

Die Kommunikation mit der Projektgruppe *DAvE* wird von Edward Ostertag übernommen. Insbesondere ist damit die Absprache von Terminen und das Verfassen, beziehungsweise Beantworten von E-Mails gemeint.

Odysseusbeauftragter

Da die Aufgabe des Odysseusbeauftragten sehr umfangreich ist und in verschiedene Bereiche eingeteilt werden kann, wird diese von Marcel Hamacher, Jens Plümer und Viktor Spadi erfüllt.
Der Odysseus-Beauftragte besitzt grundlegende Kenntnisse über das Odysseus-System und kann anfallende Implementierungsaufgaben im Kontext des Systems einordnen. Er ist Ansprechpartner und steht für alle Odysseues-Fragen zur Verfügung. Zu den grundlegenden Kompetenzen gehört das beherrschen der Erstellung von Bundles, Features und Produkten im OSGI/Equinox Kontext und des Einrichtens einer Entwicklungsumgebung für das Odysseus-System.
Der Odysseus-Beauftragte beherrscht die Service-Component-Architecture via OSGI, kennt die Hauptkomponenten des Odysseus-Systems und die wesentlichen Implementierungsschnittstellen (Operator-Schnittstelle, Access-Framework, Parser-Schnittstelle). Er kennt die Test-Strategie mit der Odysseus-Komponenten automatisiert getestet werden können und beherrscht die grundlegenden Anfragesprachen (PQL, CQL und OdysseusScript). Außerdem beherrscht er die Bedienung von Odysseus Studio. Als weiterführende Kompetenzen lässt sich Erfahrung mit dem Data Mining Feature, dem Evaluation Feature, dem Debugging von OSGI-Kompenten und der Implementierung von neuen Operatoren nennen. Darüber hinaus ist er Experte für Neuronale Netze im Odysseus-System und für den CQL2 Parser.

Projektplaner

Julian Droste-Rehling ist der Projektplaner der Projektgruppe E-Stream und erstellt die langfristigen Projektziele, so wie den Meilensteinplan. Dabei ist es wichtig, das Hauptziel des Produktes im Fokus zu behalten, also gegebenenfalls auf die Anpassung der Sprintziele und die Aufgabenpriorisierung zu achten.

Qualitätsbeauftragter

Im Zusammenhang des gewissenhaften Reviewens von Aufgaben und der zusätzlichen Überwachung durch den Git-Beauftragten wurde diese Aufgabe in die Verantwortung von jedem Projektgruppenmitglied gelegt.

Release-Engineer

Timo Wolters und Edward Ostertag sind die Release-Engineers und sind zuständig für die Administration von Jenkins, für das Anlegen und Warten von Jobs und für die Unterstützung bei der Reparatur von defekten Integrationstests. Zu den Aufgaben gehört nicht das Beheben von Build Fehlern wegen Syntaxfehlern oder fehlerhaften Unit-Tests. Zu den benötigten Skills zählen zum Beispiel grundlegende Aufgaben in Jenkins mit Groovy zu lösen (Stages, Steps, ...).

Scrum-Master und Projektleiter

Die Aufgaben des Scrum-Masters und Projektleiters werden von Florian Decker und Eugen Frikel übernommen. Hierbei soll darauf geachtet werden, dass die Scrum Prozesse umgesetzt werden. Sie übernehmen die Moderation bei scrumbezogenen Besprechungen (Starfish, ...) und beseitigen Hindernisse. Sie achten außerdem zusätzlich darauf, dass das Sprintziel erreicht werden kann. Das Team wird durch sie vor unberechtigten Eingriffen während des Sprints geschützt. Zu den benötigten Skills zählen Kenntnisse über Scrum.

Testmanager

Christian Peters ist der Testmanager und behält einen Überblick über die Teststrategien der Systeme und über die Einhaltung der Teststrategie in der Gruppe. Außerdem recherchiert und stellt er Testmethoden für das Projekt vor. Er besitzt erste Erfahrungen mit Softwaretests.

Verpflegungsbeauftragter

Die Rolle des Verpflegungsbeauftragten wurde mit Julian Droste-Rehling besetzt und beinhaltet die Beschaffung von Nahrung. Unabdingliche Skills sind dabei ernährungsphysiologische Kenntnisse.

4.2.2 Teamsitzungen und Protokolle

Für die Teamsitzungen wird vorab von dem Moderator der Woche eine Liste von Tagesordnungspunkten (TOPs) erstellt. Ein Beispiel hierfür lässt sich in Abbildung 4.5 finden. Zu Beginn jedes Treffens findet das Weekly Scrum statt, wo jeder der Gruppe über seine Tätigkeiten und Probleme der letzten Woche berichtet. Anschließend werden verschiedene wichtige Punkte (in diesem Beispiel: Treffen mit Viktor, Schnittstelle PG-DAvE und PG-DAvE Workshop) besprochen. Findet an diesem Tag ein Sprintabschluss beziehungsweise Sprintstart statt, so gibt es als TOPs außerdem ein Sprint Review, die Sprint Retrospektive und die Sprintplanung für den nächsten Sprint. Zuletzt werden unter dem TOP *Sonstiges* zusätzliche kleinere Dinge besprochen. In diesem Fall die Planung des Hackathons. In den letzten Wochen vor Ende der Projektgruppe wurde außerdem am Ende jeder Sitzung ein zweites

TOPs

Weekly Scrum		A	Alle	~ 10 Min
Treffen mit Viktor		A	@Michael Brand	~ 10 Min
Schnittstelle PG-DAVE	• PG-DAVE war heute überraschenderweise hier und hat ihre neue Schnittstelle vorgestellt.	A	@Eugen Frikel	~ 15 Min
PG-DAVE (Workshop)	• Wollen ein gemeinsamen Workshop veranstalten, um die Schnittstelle ein für allemal festzulegen.	I	Alle	~ 15 Min
Sprint Review		A	Alle	~ 30 Min
Sprint Retrospektive		A	Alle	~ 30 Min
Sprint Planung		A	Alle	~ 30 Min
Sonstiges	• Hackathon	I	Alle	

[1] B = Beschluss, I = Information, A = Aufgabe, F = Frage

Abbildung 4.5: *Beispiel für Liste von Tagesordnungspunkten (TOPs)*

Weekly Scrum eingeführt, bei welchem jedes Gruppenmitglied sein Vorhaben für die nächste Woche erläutert.

In Abbildung 4.6 ist das zugehörige Protokoll dieser Sitzung zu finden. Neben den Metadaten zu der Sitzung (Datum, Beginn, Ende, Moderator und Protokollant), beinhaltet es eine Abwesenheitsliste und Notizen zu den jeweiligen TOPs. Im Sprintreview werden die noch nicht vollendeten Tasks festgehalten und wie mit diesen umgegangen wird, beziehungsweise was dabei die Probleme waren.

4.3 Kooperation mit PG-DAvE

Ein Ziel der Projektgruppe, war die Kooperation mit der Projektgruppe DAvE (Datenarchivierung von Energiedaten) aus der Abteilung Wirtschaftsinformatik/VLBA. Aufgabe ist es eine gemeinsame Schnittstelle zu entwickeln, über welche zum Beispiel die von Smart Metern erhaltenen Daten in dem Langzeitdatenarchiv gespeichert werden können. Ebenso sollen hierüber Daten aus dem Langzeitdatenarchiv abgefragt werden können.
Erste Absprachen ergaben, dass das erste Treffen nach der Seminarphase und dem Entwickeln eigener Anwendungsfälle Ende August stattfinden soll. In diesem initialen Treffen sollen die Meilensteinpläne der Projektgruppen, die allgemeine Kommunikation, die gemeinsamen Anwendungsfälle und die von PG E-Stream bereits entwickelte Swagger Schnittstelle besprochen werden. Es wurde beschlossen, dass eine Verbindung zwischen den Systemen der Projektgruppen bestehen soll und im Folgenden die Anwendungsfälle ausgearbeitet und Minimalanforderungen umgesetzt werden sollen. Optionale Ziele wären der Austausch von Wetterdaten und eine gemeinsame Abschlusspräsentation beziehungsweise eine gemeinsame Demonstration. Für die bessere Kommunikation wurde sich auf einen E-Mail Verteiler und einen gemeinsamen Bereich in Confluence geeinigt. Es wurden die folgenden gemeinsamen Anwendungsfälle festgelegt:

- Abfrage von Rohdaten (bidirektional, minimal)
- Speicherung von Rohdaten (unidirektional, E-Stream DAvE, minimal)
- Modellbau auf Anfrage (optional)

2018-01-17 Protokoll

Angelegt von Julian Droste-Rehling, zuletzt geändert am Jan 24, 2018

Datum	Beginn	Ende:	Moderator	Protokollant
2018-01-17	14:15 Uhr	17:00 Uhr	Jens Plümer	Julian Droste-Rehling

Abwesenheit

- Edward Ostertag und Tim Silhan

Weekly Scrum (nur Hindernisse/Probleme)

- Viktor/Florian → Task war umfangreicher als geplant

Treffen von Michael mit Viktor von PG DAvE

Inhalte von PG DAvE wurde von Viktor gesichtet und er schaut sich morgen (Do. 18.01.18) unsere Architektur an.
Danach wird Viktor mit Michael die weitere Kooperation der PGs besprechen und Ziele festlegen.

Später (im Februar) hat Michael noch ein weiteres Treffen Herrn Gomez (Abteilung VLBA Leitung).

Treffen mit PG DAvE

Es wurde die "neue" Schnittstelle der PG DAvE vorgestellt.

Datenaustausch rein über Kafka Topic: Jeder hat einen eigenen Kafka Topic und dann Zugriff auf den jeweiligen anderen Topic.

Problem: Nur per SSH-Tunnel Zugriff zur PG DAvE API (Vorschlag: Port forwarding über einen unserer Server).

Vorschlag von PG DAvE: Evtl. kleiner Workshop mit beiden PGs für eine Schnittstellen und Use-Case Festlegung (und Überarbeitung).

Hinweis:

Treffen mit Viktor (PG DAvE) U61 am 18.01.18 um 14:00.

Sprintreview

nicht fertige Tasks / Tasks noch in Ausführung:

~~PGESTREAM 335~~ - [Implementierung] Topologie für die Anzeige der Netzdaten DONE

Fake Service lieferte veraltete Schemata

~~PGESTREAM 380~~ - Jenkins build E-Stream - e2e Test fixen DONE

Jenkins e2e Tests nicht fixbar, da Server fast immer down war (Offis-seitig)

~~PGESTREAM 309~~ - XML-Converter in die Simulation einbinden DONE

XML Converter wird nun weiter fortgesetzt (Task war zurückgestellt)

~~PGESTREAM 394~~ - [Konzept] Umsetzen des Data-Minings DONE

Task ist noch in Bearbeitung und sollte eigentlich zu diesem Sprint fertig sein, wird nun in kleiner Gruppe möglichst schnell bearbeitet.

Quasi fertig:

~~PGESTREAM 315~~ - System Usability Scale Fragebogen + Umsetzungskonzept DONE

Vorstellung Usability Test Fragebogen inkl. Auswertematrix und die Einverständniserklärung für Probanten.

(Außerdem wurde ein Doku-Text anlegt und Platzhaltern für den Endgültigen Test)

~~PGESTREAM 393~~ - [Implementierung] Querymanager GUI DONE

Vorstellung der QueryManager Implementierung

Code Review

Vorstellung Implementierung des QueryManagers durch Viktor

Nun existiert eine *Running Queries Table* und eine *Query Template Table* im Playground-Tab inkl. enthaltener Dialoge.

Retrospektive

siehe 2018-01-17 Retrospektive

Sprintplanung

Der vorgeplante Sprint wurde durch Planning Poker per Hand mit Storypoints geschätzt.

In der Gruppe wurden ein paar Taskbeschreibungen angepasst, sodass die Aufgaben konkreter formuliert sind.

Hinweis für diesen Sprint: Es sind sehr viele Gruppentasks vorhanden, somit sollen diese größeren Tasks noch innerhalb der Gruppe aufgeteilt werden.

Abbildung 4.6: *Beispiel für ein Sitzungsprotokoll*

- Einbinden von Wetterdaten (optional)

Zum nächsten Treffen einen Monat später wurden die Anwendungsfälle weiter ausgearbeitet und definiert, dass die Abfrage von historischen Smart Meter Daten, historischen Wetterdaten (DWD) und von Mining Modellen (PMML) möglich sein soll. Außerdem ist das Erstellen von Mining Model Aufträgen und das Senden von Rohdaten zur Speicherung als Ziel gesetzt worden.
Da PG-DAvE die Kooperation als optional missverstanden hat und in der Entwicklung ihres Systems noch nicht so weit fortgeschritten war, wurde die Zusammenarbeit erst im Januar wieder aufgenommen. Dadurch wichen die Schnittstellendefinitionen voneinander ab.
In einem kurz darauf folgenden Meeting wurde das weitere Vorgehen mit den jeweiligen Betreuern besprochen. In diesem Sinne wurde zunächst der von PG E-Stream entwickelte PG-DAvE Mock vorgestellt und festgelegt, inwiefern eine Zusammenarbeit noch möglich ist. Es wurde sich daraufhin geeinigt, innerhalb eines gemeinsamen Workshops die Schnittstelle fertig zu stellen und ein Minimalszenario umzusetzen. Demnach sollten Anfragen möglich sein, auf welchem Topic welche PMML bereitgestellt werden können und welche Felder diese PMML enthält. Daraufhin soll PG DAvE eine Anfrage stellen können, auf welchem Topic sich die Rohdaten befinden und diese auslesen. Anschließend kann von PG E-Stream ein PMML Schema mit codierter Anfrage geschickt werden. Dann kann das von PG DAvE verarbeitete PMML auf das Topic zurückgeschickt werden.
Innerhalb des Workshops wurde daraufhin erreicht, dass PG DAvE von PG E-Stream die Rohdaten gesendet bekommt (das Schema entspricht dem Abfragen von historischen Smart Meter Daten), dass PG-DAvE Modelle und historische Smart Meter Daten liefert und außerdem, dass PG E-Stream PG DAvE einen Ausschnitt der Daten der Mosaik Simulation (ein Beispieldatensatz) liefert. Unter Anforderungsanalyse Unterabschnitt 5.1.7 lassen sich die ausgeführten Anwendungsfälle mit Diagrammen finden.

4.4 Meilensteinplan

Ein wichtiger Bestandteil des Projektmanagements ist ein Meilensteinplan mit zeitlichen und ereignisorientierten Zielen. Der wesentliche Bestandteil eines Meilensteinplans ist die Terminplanung eines Projektes. Ein Meilenstein definiert ein klares Ziel, welches keinerlei zeitliche Ausprägung besitzt und kein Aktion beinhaltet.

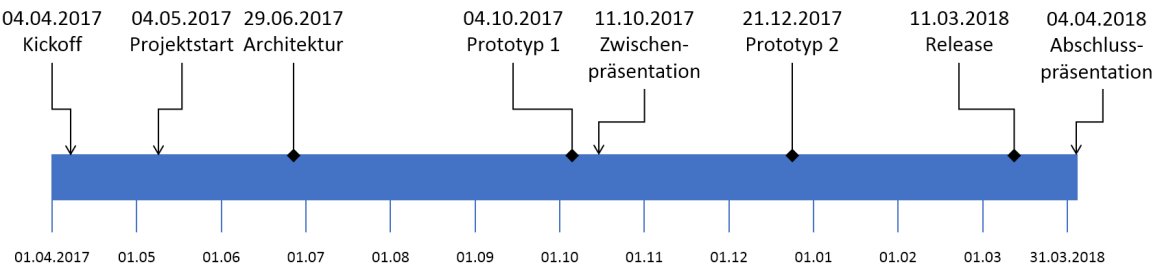


Abbildung 4.7: *Meilensteinplan*

Der Meilensteinplan in Abbildung 4.7 zeigt in der Zeitachse ein vollständiges Jahr vom 01.04.2017 bis zum 31.03.2018 über das sich die Projektgruppe erstreckt. Als zeitliche Absteckung der Projektgruppe wurden Markierungspfeile gewählt, welche mit dem Kickoff-Termin am 04.04.2017 startet und der Abschlusspräsentation am 04.04.2018 endet.

Der erste Termin des Projekts ist der *Projektstart* am 04.05.2017, welcher zum einen das Abschließen der Seminarphase beinhaltet und zum anderen den Kickoff zur Produkterstellung darstellt.
Zu Beginn der Entwicklungsphase wurde als Meilenstein die Fertigstellung der *Architektur* am 29.06.2017 definiert. Unter Architektur wird die geplante Konstellation der verschiedenen Teilsysteme verstanden. Konkret beinhaltete dieses Ziel, dass die Kommunikation zwischen Odysseus, mosaik, Apache Kafka, dem Spring Backend, einer Datenbank und dem Frontend definiert wird. Des Weiteren wurde bestimmt, dass eine CI-Umgebung inklusive eines Jenkins auf den PG internen VMs funktionsfähig den Projektmitgliedern zur Verfügung steht. Mithilfe dieses Meilensteins wird gewährleistet, dass ab diesem Termin die Entwicklung des Produkts starten kann.

Als nächster Meilenstein wurde die Fertigstellung eines ersten Prototyps (*Prototyp 1*) am 04.10.2017 definiert, welcher als Randbedingung hatte, dass dieser vor der Zwischenpräsentation am 11.10.2017 erreicht wird. Bei dem ersten vertikalen Prototypen des Produkts, sollen alle Teilsysteme mit simplen Funktionen interagieren sowie erste unsupervised Datamining Algorithmen implementiert sein.
Der folgende Termin ist kein konkreter Projektmeilenstein, denn die *Zwischenpräsentation* des Projekts am 11.10.2017 ist zwar ein zeitlich terminiertes Ziel, allerdings beinhaltet dieser Termin eine Aktion, die zudem nicht das Endprodukt selbst betrifft. Daher werden Präsentationen des Projekts wie andere wichtige Punkte mit einem einfachen Pfeil markiert.

Darauf folgt der Meilenstein, die Fertigstellung des *Prototyps 2*, welcher terminlich auf den letzten Arbeitstag des Jahres auf den 21.12.2017 gelegt wurde. Bei diesem Prototypen sollen neben den Implementationen von unsupervised auch supervised Datamining Algorithmen integriert werden.

Nach der erfolgreichen Erstellung von zwei inkrementellen Prototypen, folgt mit der Fertigstellung des Endprodukts der Schlussmeilenstein (*Release*). Bis zu diesem Zeitpunkt können noch neue Funktionen implementiert werden, danach wird das System evaluiert, sodass keine Änderungen am System mehr vorgenommen werden dürfen. Mit der Fertigstellung der Evaluation des Produkts folgt die *Abschlusspräsentation* am 04.04.2018 und damit endet die Projektgruppe.

4.5 Scrum

4.5.1 Magic Estimation

Magic Estimation ist eine Schätzmethode, mit der viele User Stories in kurzer Zeit geschätzt werden können. Es kann also der gesamte Product Backlog in wenigen Minuten geschätzt werden. Als Schätzeinheit kann, wie beim Planning Poker, zum Beispiel die Fibonacci Reihe oder die Cohn Skala verwendet werden. Die Cohn Skala besteht aus der Zahlenreihe 0, 1, 2, 3, 5, 8, 13, 20, 40, 100. Als Hilfe für das Schätzen kann eine Referenz Story bestimmt werden. Diese sollte möglichst einfach zu schätzen und für alle verständlich sein. Dieser Story wird ein Wert zugewiesen, zum Beispiel 3. Die anderen Stories werden relativ zu der Referenz Story geschätzt. Wenn also die Referenz Story den Wert 3 hat und eine andere Story den Wert 5, bedeutet es, dass die Story fast doppelt so aufwendig ist, wie die Referenz Story. Und wenn die andere Story den Wert 2 hat, sollte diese mit weniger Aufwand, als die Referenz Story erledigt werden können.

Bei dieser Schätzmethode werden zuerst die Planning Poker Karten auf den Boden oder auf den Tisch gelegt. Planning Poker Karten sind Karten, auf denen zum Beispiel die Fibonacci Zahlen stehen. Anschließend werden die Backlogeinträge vorgestellt. Diese werden dann gleichmäßig unter den Mitgliedern des Entwicklerteams verteilt. Ab diesem Zeitpunkt sollte keine verbale oder nonverbale Kommunikation zwischen den Teammitgliedern stattfinden. So sollen Absprachen vermieden werden.

Jedes Teammitglied schätzt die ihm zugewiesenen Stories und legt diese zu der jeweiligen Karte. Wenn also der Schätzer denkt, dass eine Story die Komplexität 2 hat, wird diese zu der Karte mit dem Wert 2 gelegt. Eine Platzierung zwischen zwei Karten ist nicht erlaubt. Wenn ein Schätzer also denkt, seine Story hat eine Komplexität von 4, darf er sie nicht zwischen die Karten mit den Werten 3 und 5 legen, sondern muss sich für eine Karte entscheiden. Anschließend schauen sich die Teammitglieder die von den anderen geschätzten Stories an. Wenn sie der Meinung sind, dass die Story eine andere Komplexität hat, wird diese Story zu der für sie passenden Karte gelegt. Der Moderator merkt sich die Stories, die hin und her bewegt werden. Bei diesen besteht noch Klärungsbedarf, da die Teammitglieder sich nicht einigen konnten. Diese Stories können auch aus dem Spiel genommen werden. Zusätzlich kann es eine Karte mit einem *?* geben. Zu dieser können Stories gelegt werden, bei denen nicht klar ist, was dort gemacht werden muss oder die zu komplex sind.

Nachdem alle Stories geschätzt wurden, finden Diskussionen über die oft verschobenen Stories statt. Dabei wird vor allem über die Stories geredet, bei denen der Wert stark geschwankt hat. Wenn eine Story also nur von 2 auf 3 verschoben wurde, kann dies unter Umständen auch ignoriert werden. Bei diesen Diskussionen können offene Fragen geklärt werden. Wenn man sich auf einen Wert einigen konnte, wird die jeweilige Story auch nachträglich noch zu der passenden Karte gelegt.

Nach der Schätzung hat das Team eine ungefähre Vorstellung davon wie lange es dauern könnte, bis der komplette Product Backlog abgearbeitet wurde. Stories, die mit einem hohen Wert geschätzt wurden, sollten nochmal überarbeitet werden. Diese können von dem Entwicklerteam aufgeteilt werden. Stories, die auf die Fragezeichen-Karte gelegt wurden, sollten überdacht werden. Stories, die eine hohe Priorität hatten, sollten allerdings möglichst schnell aufgeteilt werden, um schnell erledigt zu werden.

Der Vorteil dieser Schätzmethode ist, dass viele Backlogeinträge in kurzer Zeit geschätzt werden können, da bei dieser Methode keine unnötigen Diskussionen stattfinden. Dennoch stellt sich die Frage, ob diese schnell gemachten Schätzungen auch verlässlich sind, da unter Umständen auch wichtige Diskussionen nicht stattfinden. Es finden aber immer noch einige wichtige Diskussionen statt, diese verschieben sich jedoch auf später. Bei dieser Schätzmethode geht es also nicht so sehr um die Präzision, sondern darum, eine erste aussagekräftige Schätzung zu haben. Für genaue Schätzungen ist Planning Poker besser geeignet, dauert aber auch länger.

Wir verwenden Magic Estimation um unser Product Backlog zu schätzen. Allerdings befinden sich im Product Backlog keine User Stories, sondern Aufgaben. Für das Schätzen werden keine Story Points verwendet, sondern Zeitstunden. Der Ablauf ist aber ähnlich. Die Planning Poker Karten werden auf den Tisch gelegt. Die Backlog Items werden mithilfe von JIRA ausgedruckt und unter den Teammitgliedern aufgeteilt. Jedes Teammitglied schätzt die ihm zugewiesenen Aufgaben und legt diese zu der passenden Karte. Ist eine Aufgabe unklar, wird diese zu der Fragezeichenkarte gelegt. Anschließend überprüft jeder ob er die Aufgaben der anderen genauso schätzen würde. Wenn nicht, legt er die Aufgabe zu der für ihn passenderen Karte und sagt die Nummer der Aufgabe, damit diese notiert werden kann. Zum Schluss wird über die Aufgaben, die umgelegt wurden, diskutiert. Aufgaben, die zu der Fragezeichenkarte gelegt wurden, werden überdacht. Die Schätzungen werden in das JIRA Board eingetragen.

4.5.2 Starfish

Um die Retrospektive von Sprints produktiver zu gestalten, wurde in diesem Projekt nach der Starfish Methode vorgegangen.

Hierbei wird das Feedback der Gruppe in fünf Kategorien eingeteilt:

- Start: „Welche Verhaltensweisen sollten wir etablieren?“
- Keep: „Welche Verhaltensweisen sollten wir beibehalten?“
- More: „Welche Verhaltensweisen sollten wir noch besser ausprägen/fördern?“
- Less: „Was sollten wir weniger tun?“
- Stop: „Womit sollten wir unbedingt aufhören?“

Das Feedback der Teammitglieder wird auf Klebezettel geschrieben, die dann jeweils einer der fünf Kategorien zugeordnet und ggf. nach Themen gruppiert werden. Anschließend wird über die Anmerkungen und mögliche Lösungen diskutiert, welche schließlich umgesetzt werden sollen.

Um unseren Anforderungen an eine schnellere Retrospektive zu genügen, wurde diese Methode an unsere Bedürfnisse angepasst. So konnten die die Feedbackpunkte durch einen Slack-Bot bereits vor der Sitzung anonym gesammelt und zusammengetragen werden. Hierdurch wurde sich eine regere Beteiligung erhofft, denn durch den Slack-Bot konnte dennoch überwacht werden, dass jedes Gruppenmitglied sich an der Retrospektive beteiligt. Aus den besprochenen Punkten werden neue Actions abgeleitet und überprüft, ob die alten Actions erfüllt wurden. Ein Beispiel für eine durchgeführte Retrospektive ist in Abbildung 4.8 dargestellt.

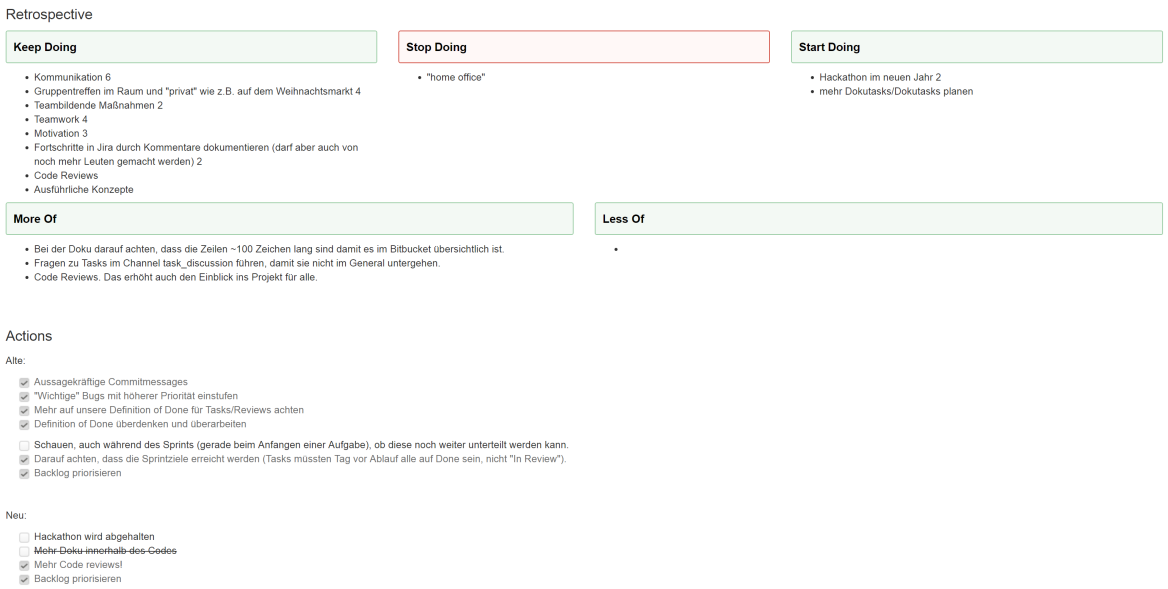


Abbildung 4.8: *Beispiel für Sprint-Retrospektive*

5 Anforderungsanalyse

5.1 Einleitung

Dieses Kapitel bildet die Spezifikation, welche als Artefakt aus der Anforderungsanalyse hervorgegangen ist und alle Anforderungen umfasst, die das hier entwickelte Softwaresystem beschreiben. Zunächst wird in Unterabschnitt 5.1.1 die Notwendigkeit für eine Anforderungsanalyse motiviert, indem Probleme genannt werden, die in einem Softwareentwicklungsprozess auftreten können. Unterabschnitt 5.1.2 definiert das Ziel der Projektgruppe und formuliert die Problemstellung, die sich für die Entwicklung eines Softwaresystems ergibt. Außerdem werden relevante Begriffe, die Systemgrenzen und abstrakte Funktionen des Softwaresystems beschrieben. Unterabschnitt 5.1.3 geht auf die Struktur des Dokumentes ein, beschreibt die definierten Anforderungstypen und gibt an, wie die Spezifikation generell verwendet werden kann.

5.1.1 Zielsetzung der Anforderungsanalyse

Die Anforderungsanalyse ist wesentlicher Bestandteil eines modernen Softwareentwicklungsprozesses und beschreibt die systematische Entwicklung einer Spezifikation, in der beschrieben wird was ein softwaregestütztes System tun soll. Die Spezifikation stellt eine vollständige, konsistente sowie eindeutige Beschreibung des Systems dar, sodass sie zwischen allen Beteiligten als Grundlage für Vereinbarungen verwendet werden kann [33]. Verschiedene Studien haben gezeigt, dass viele Softwareprojekte scheitern oder unzureichende Softwarelösungen entstehen, wenn eine Anforderungsanalyse nicht angemessen ausgeführt wird. Beispielsweise sind fehlerhaft erhobene Anforderungen verantwortlich für 50% aller Softwarefehler in einem Softwareprojekt [217]. Die Korrektur dieser Softwarefehler wird mit zunehmenden Projektfortschritt aufwendiger [33], sodass Fehlentwicklungen frühzeitig erkannt werden müssen, um möglichst kosteneffizient und ressourceneffektiv entwickeln zu können. Die Softwarequalität des Produkts und der Projekterfolg sind also unweigerlich mit einer systematisch durchgeführten Anforderungsanalyse verknüpft.

Die Anforderungsanalyse dient zur Kommunikation zwischen dem Systemanalytiker, Designer, Programmierer und dem Projektmanager. Der Systemanalytiker muss weitere Projektbeteiligte auf der Entwickler- und Nutzerseite identifizieren und mit in das Projekt einbeziehen. Er muss eine gemeinsame, konsistente Projektterminologie einführen und in einem Projektglossar festschreiben, sowie das Bewusstsein dafür schaffen, dass Anforderungen permanent überprüft und angepasst werden müssen. Diese Aspekte werden durch die hier entwickelte Anforderungsanalyse aufgenommen. Der Designer und Programmierer erhält durch die Spezifikation eine einheitliche Beschreibung für Arbeitsaufgaben. Der Projektmanager erhält die Möglichkeit die Randbedingungen der Systemherstellung zu überwachen und zu quantifizieren. Darüber hinaus kann das System nach Beendigung des Projekts vollständig durch die Recherche dieses Dokuments verstanden werden, sodass z.B. projektfremde Entwickler Erweiterungen und Wartungsarbeiten relativ unproblematisch planen und ausführen können.

Es ergeben sich für den Entwicklungsprozess verschiedene Problembereiche, die für fehlerhafte Anforderungen verantwortlich gemacht werden können und durch eine geeignete Anforderungsanalyse überwunden werden sollen:

- **Mangelnde Systematik** drückt sich beispielsweise durch stark unterschiedliche Beschreibungsstile der Anforderungen aus. Zu detaillierte und konkrete Implementierungsbeschreibungen führen dazu,

dass lösungsorientiert anstelle von problemorientiert beschrieben wird. Dadurch werden bereits frühzeitig Design- und Technologieentscheidungen getroffen, die möglicherweise erst zu einem späteren Zeitpunkt folgerichtig entschieden werden können. Weitere Probleme ergeben sich durch:

  - fehlende Strukturierung der Anforderungen
  - fehlende Anforderungen
  - diffuse Anforderungen
  - doppelte Anforderungen
  - unvollständige Anforderungen
  - mangelnde Qualität der Anforderungen:
    1. Anforderungen sind unvollständig
    2. Anforderungen sind mehrdeutig
    3. Anforderungen sind widersprüchlich
    4. Anforderungen sind redundant
    5. Anforderungen sind unnötig

- **Hohe Systemkomplexität** kann dazu führen, dass die Domäne des Systems nicht weitreichend genug erfasst ist und dadurch relevante Aspekte des Systems verborgen bleiben. Zudem ist es problematisch, dass die Systemgrenzen nicht klar abgegrenzt werden können und so nicht alle Schnittstellen des Systems erkannt werden. Daher ist es sinnvoll die Belange der verschiedenen Anforderungen zu identifizieren und klar voneinander zu trennen (Trennung der Belange).
- **Unzureichende Kommunikation** zwischen den Projektbeteiligten entsteht dadurch, dass die Beteiligten unterschiedliche Sprachen sowie Fachjargons sprechen und nicht jeder den gleichen Wissenstand mit in das Projekt bringt. Dies kann dazu führen, dass das Projektziel und die Vorstellung über das System unterschiedlich von den einzelnen Akteuren aufgefasst wird. Generell gilt, dass frühzeitig alle Projektbeteiligten identifizieren werden müssen, also all diejenigen die an der Entwicklung, Vertrieb und Benutzung involviert sind.
- **Verändernde Anforderungen** treten auf, wenn sich die Rahmenbedingungen eines Projektes ändern. Beispielsweisen kann sich im Verlauf des Projektes herausstellen, dass aufgrund neuer Erkenntnisse (Erweiterung des Domänenwissens) die Umsetzung einer Systemfunktion nicht wie geplant realisiert werden kann. Dementsprechend müssen Veränderungen als Teil des Projekts angenommen und behandelt werden. Außerdem ist es sinnvoll die Änderungen einer Anforderungen gegen das Projektziel zu validieren.

Zur Durchführung einer Anforderungsanalyse ergeben sich zwei Varianten. Die Analyse wird durch ein mehrstufiges Prozessmodell (vgl. Wasserfallmodell [135]) betrieben werden oder durch ein iteratives Prozessmodell, welches die Anpassung von Anforderungen über den gesamten Entwicklungsprozess erlaubt. Ein iteratives Prozessmodell behandelt die Problematik der sich verändernden Anforderungen. Es unterteilt den Entwicklungsprozess in zwei Phasen, welche als Anforderungserhebung und als Anforderungsmanagement bezeichnet werden [24].

- **Anforderungserhebung** umfasst alle Tätigkeiten, die sich mit dem Erfassen, Beschreiben, Analysieren, Validieren, Verhandeln und Verabschieden von neuen Anforderungen befassen:

  - Erfassen bezeichnet, dass Erkennen der Notwendigkeit für einen Systemaspekt
  - Beschreiben bezeichnet, das Formulieren eines Anforderungstextes und charakterisiert eine Anforderung
  - Analysieren bezeichnet, das kritische Auseinandersetzen mit einer Anforderungen
  - Validieren bezeichnet, dass eine Anforderungen zum Projektziel beiträgt
  - Verhandeln bezeichnet, dass alle Beteiligten ihr Einverständnis zu einer Anforderung geben
  - Verabschieden bezeichnet, dass eine Anforderung dokumentiert und registriert wird
- **Anforderungsmanagement** umfasst das Verändern einer bereits erhobenen Anforderung, d.h. die Anforderung durchläuft einige oder alle Tätigkeiten, wie sie in der Anforderungserhebung beschrieben sind. Notwendig ist es, dass die Änderungen einer Anforderungen von alle Beteiligten verhandelt und verabschiedet werden.

5.1.2 Vision

In diesem Abschnitt wird die Motivation für die Durchführung des Projekts vorgestellt und es wird auf die Problemstellung eingegangen, welche von der Projektgruppe gelöst werden soll. Hierfür wird eine Problembeschreibung sowie die Auswirkungen des Problems geschildert. Anschließend wird der Systemkontext des Systems charakterisiert, sodass die Rahmenbedingungen des Projektzieles eingegrenzt werden können. Des Weiteren wird eine Lösungsskizze präsentiert, welche einen groben Systementwurf darstellt und einen initialen Ausgangspunkt für weitere Überlegungen darstellt. Zu der beschriebenen Lösungsskizze werden außerdem erste Bewertungskriterien definiert, mit denen die Qualität des Systems bewertet werden kann.

5.1.2.1 Motivation

Die Europäische Union hat in einem Beschluss aus dem Jahr 1996 festgeschrieben, dass der Energiesektor liberalisiert werden soll. Dieser Beschluss ermöglichte erstmals den freien Wettbewerb zwischen den unterschiedlichen Akteuren im Energiesektor [134], was zu einer verbraucherorientierten Energieerzeugung geführt hat. Die Entwicklung einer nachhaltigen und ressourcenschonenden Energieerzeugung wurde dadurch vorangetrieben. Dieser Trend ist seit dem beschlossenen Atomausstieg in Deutschland stärker denn je, sodass der Bedarf dezentraler Energieerzeuger, wie Photovoltaik- und Windkraftanlagen, laufend zunimmt. Die Energieinfrastruktur im deutschen Energiesektor befindet sich in einem Restrukturierungsprozess, der den Ausbau erneuerbarer Energien unterstützt und eine effiziente Energieerzeugung sowie -verteilung sicherstellen soll. Heutzutage integrieren Verteilnetze bereits zentralen und dezentralen Energieerzeuger, welche durch verschiedene softwaregestützte Systeme gesteuert und organisiert werden.

Zukünftig sollen sogenannte intelligenten Energienetzwerke entstehen, die die Energieerzeuger so steuern sollen, dass immer nur so viel Energie produziert, wie benötigt wird. Hierfür werden verschiedene Netzinformationen benötigt. Die Entwicklung eines Softwaresystems für die Verarbeitung und Bereitstellung dieser Informationen ist deshalb ein wichtiger Beitrag für die Energiewende. Die Projektgruppe E-Stream bearbeitet diese aktuelle Forschungsfrage und entwickelt ein solches System für die Verbrauchsmessungen von Haushalten.

5.1.2.2 Problembeschreibung

Das intelligente Energienetzwerk, welches auch als Smart Grid bezeichnet wird, muss das Verbrauchs- und Erzeugungsverhalten aller Netzteilnehmer kennen, um eine Lastverteilung im Verteilnetz automatisiert durchführen zu können. Es kann nur dann eine effiziente Verteilung erreicht werden, wenn detailliert und zeitnah bekannt ist, wie viel Energie an welcher Stelle im Netz zu welchem Zeitpunkt erzeugt und verbraucht wird. Entsprechende Sensordaten werden bereits heutzutage im Bereich der Industrie von Großkunden sowie Kraftwerken aufgezeichnet und ermöglichen so eine registrierte Lastgangmessung. Ein völlig neuer Bereich umfasst die Energieverbräuche von Haushaltskunden, welche bislang nicht erfasst werden können, da eine flächendeckende Verteilung von Messgeräten mit der benötigten Auflösung nicht vorhanden ist. Dies soll sich in Zukunft mit der deutschlandweiten Einführung von sogenannten Smart Metern ändern [1]. Hierbei besteht das Problem, dass keine Aussage darüber getroffen werden kann, wie viele Daten verarbeitet werden müssen, was die Entwicklung eines geeigneten Softwaresystems erschwert.

Der derzeitige Stand ist, dass die existierenden Systeme der Verteilnetzbetreiber einerseits nicht auf die große Datenlast ausgelegt sind und andererseits die Systeme keine Datenanalyse für Smart Meter Daten unterstützen. Es ist daher abzusehen, dass mit der landesweiten Einführung von Smart Metern die Nachfrage an geeigneten Systeme stark ansteigen wird. Zukünftig sollen die Systeme Prognosen zum Lastgang im Verteilnetz durchführen können, um den Verteilnetzbetreiber rechtzeitig auf mögliche Schwankungen im Netz aufmerksam zu machen. Bislang existieren keine etablierten und getesteten Verfahren für eine solche Art von Prognose. Eine frühzeitige Forschung ist deshalb notwendig, um den Bedarf an stabilen und zuverlässigen Systemen bedienen zu können.

Eine weitere Problematik bei der Netzdatenverarbeitung ist, dass die Datenmenge durch eine konventionellen Datenverarbeitung nicht praktikabel verarbeitet werden kann, wenn das System zeitnah Ergebnisse produzieren muss. Die anfallende Datenmenge variiert je nach Anzahl an Smart Metern, die im Verteilnetz integriert sind, und kann mehrere Hunderttausend oder Millionen Datenpunkte beinhalten, die zu einem einzelnen Messzeitpunkt aufgenommen werden. Ein Pilotprojekt in Los Angeles, USA hat gezeigt, das bei einem Verteilnetz mit 1,4 Millionen Haushalten durchschnittlich 2 TB an Daten pro Tag produziert, wenn die Messfrequenz 1 Minute beträgt und die übertragenen Daten pro Übertragung 1 KB betragen [244].

- **Energieversorgungsunternehmen** wird es ermöglicht, die aktuell erzeugte Energie der Erzeugungsanlagen, zu betrachten. Dazu werden die eintreffenden Daten visuell aufbereitet. Ebenso wird es den Operatoren der Energieversorgungsunternehmen ermöglicht, den Verbrauch aller Stromabnehmer zu erfassen.
- Den **Kunden** wird es ebenfalls ermöglicht den eigenen aktuellen Energieverbrauch zu erfassen. Kunden, die ebenfalls Strom ins Netz einspeisen, können den Verlauf der Einspeiseleistung ihrer Anlage ebenso einsehen.
- Sollte es aufgrund unerwarteter Störungen zum Ausfall einer Energieerzeugungsanlage kommen. So wird dieses dem **Wartungspersonal** der Energieversorgungsunternehmen über die Weboberfläche unmittelbar mitgeteilt, sodass dieses mit den dargestellten Informationen die Störung direkt lokalisieren kann und ggf. weitere Schritte für die betreffende Anlage eingeleitet werden können.

[1] Gesetz über den Messstellenbetrieb und die Datenkommunikation in intelligenten Energienetzen. Online `https://www.gesetze-im-internet.de/messbg/index.html`, letzter Zugriff: 03.04.2018

- Durch die Zusammenarbeit mit der **Projektgruppe DAvE** soll es möglich sein, neue Modelle in das System einzuspielen. Zudem ist es Ihnen Möglich, aus den Datenströmen Langzeitdaten zu gewinnen, welche weiter genutzt werden können.
- Schließlich kann autorisiertes Personal der Energieversorgungsunternehmen neue Nutzer in das System eintragen, welche dann Zugriff auf die zuvor beschriebenen Daten haben. Zudem können Sie bestehende Nutzer mit neuen Rechten ausstatten, sowie diese wieder entziehen. Zuletzt können Nutzer auch gelöscht werden.

5.1.2.3 Einordnung in den Systemkontext

Das zu entwickelnde System wird in den Smart Grid Systemkontext eingebettet. Daraus ergeben sich verschiedene Rahmenbedingungen: Die Kommunikation zwischen Systemen im Smart Grid wird durch moderne Internetkommunikationsverfahren realisiert. Ein System muss zuverlässig und verfügbar sein, auch wenn kritische Lastzustände erreicht werden. Werden personenspezifische Daten verarbeitet, müssen diese gegebenenfalls anonymisiert und unbedingt durch moderne Verschlüsselungsverfahren verschlüsselt werden, sodass der Zugriff der Daten ausschließlich nur für Befugte zugänglich ist. Zum Beispiel der zuständige Operator des Verteilnetzes. Das System wird primär für die Verteilnetzbetreiber entwickelt.

Die Kernfunktionen des Systems sind die Verarbeitung von Smart Meter Daten und die Darstellung dieser Daten sowie Prognoseergebnisse. Die Netzdaten können als kontinuierliche und potenziell unendliche Datenströme aufgefasst werden, welche durch die Methoden der Datenstromverarbeitung verarbeitet werden können. Die Datenstromverarbeitung beinhaltet die Verarbeitung der Smart Meter Daten in Echtzeit, sodass keine persistente Datenspeicherung erfolgt, sondern die Daten solange im Hauptspeicher des Systems gehalten werden bis diese verarbeitet worden sind. Hierfür werden effiziente Algorithmen und eine verteilte Systemarchitektur benötigt, um eine hohe Datenrate und niedrige Latenz bei der Datenverarbeitung zu erreichen. Die Schwierigkeit besteht darin, dass neue Analysemethoden für Netzdatenströme entwickelt werden müssen. Gleichermaßen muss die Darstellung performant umgesetzt werden, wenn Echtzeitdaten visualisiert werden sollen. Das System sollte durch internationale Standards mit anderen System interoperabel kommunizieren können, um beispielsweise Analyseergebnisse austauschen zu können.

Darüber hinaus soll das Systeme keine Funktionalitäten im Sinne der Verteilnetzautomatisierung realisieren. Systeme für Verteilnetzautomatisierung sind für die Steuerung von Kraftwerken und Anlagen im Verteilnetz zuständig, die Strom in das Energienetz einspeisen. Auch soll die Überwachung und Steuerung von Haushalten (Smart Home) nicht durch das System realisiert werden. Dennoch kann das System Erweiterungsschnittstellen bereitstellen, um die genannten Funktionalitäten in weiteren Forschungsprojekten erweitern zu können.

5.1.2.4 Auswirkungen des Problems

Die Energiewende ist verantwortlich dafür, dass vor allem in Deutschland konventionellen Energieerzeuger, wie Kohle-, Gas- und Atomkraftwerke, stetig an Bedeutung verlieren und regenerative, erneuerbare Energiequellen zunehmend als Energielieferanten ins Gesamtnetz integriert werden. Die Integration finden sowohl lokal als auch nicht lokal statt, sodass Haushalte bereits private Photovoltaikanlagen und Blockheizkraftwerke installiert haben und als eigenständige Energieerzeuger Strom in

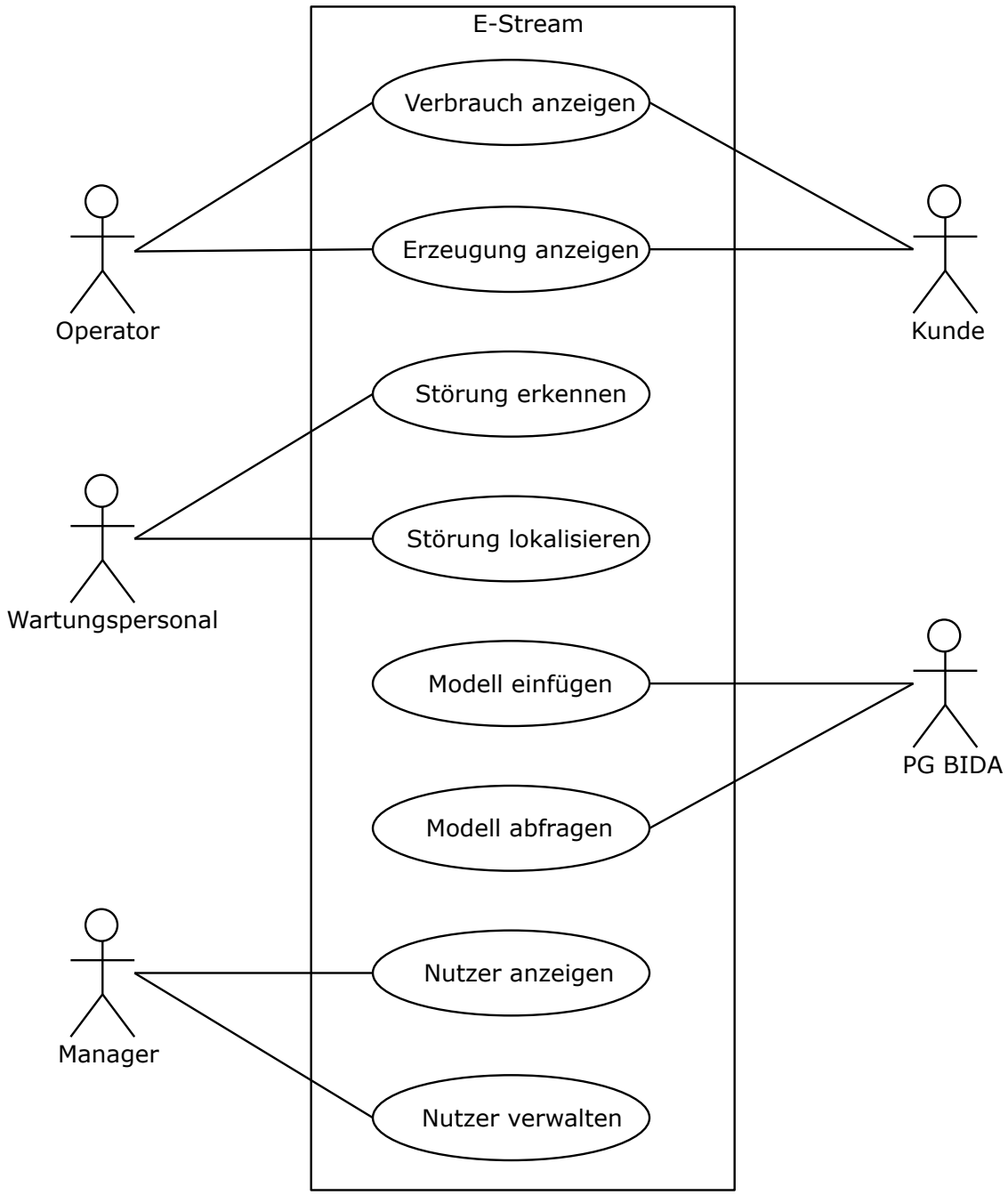


Abbildung 5.1: *initiale Anwendungsfälle*

das Netz einspeisen. Die nicht lokale Ebene beschreibt beispielsweise die Installation von Windparks und Biogasanlagen, die bereits einen nennenswerten Beitrag bei der Energieerzeugung leisten. Das Zusammenspiel dieser zentralen und dezentralen Energiequellen muss koordiniert werden, sodass die Netzstabilität gewährleistet werden kann und Störungen zeitgerecht behoben werden können. Die Lastverteilung spielt hierbei eine besondere Rolle, da erzeugte Energie, ohne eine effektive Speicherung

nicht lange an einem Ort verweilen kann, sodass die Energie möglichst dorthin transportiert werden muss, wo sie verbraucht werden kann. Das Energienetz soll also möglichst geringe Verluste erzielen.

Das zukünftige Smart Grid soll die beschriebene Problematik lösen, sodass die Kopplung zentraler und dezentraler Energielieferanten effizient gelingen kann, was vor allem in Bezug auf die steigende Nachfrage regenerativer Energie von Bedeutung ist. Die benötigte Smart Meter Datenverarbeitung ist wesentlicher Bestandteil dafür, dass die Integration und Restrukturierung langfristig gelingen kann. Auf lange Sicht sollen neue Dienste im Endkundenbereich durch die Entwicklung eines einheitlichen Smart Grids entstehen. Beispielsweise könnte der Endkunde Informationen erhalten, die Aufschluss darüber geben, ob seine private Photovoltaikanlage einwandfrei funktioniert oder nicht oder er kann kontrollieren ob die gemessenen Verbräuche tatsächlich korrekt sind.

5.1.2.5 Skizze einer Lösungsidee

Die beschriebene Problemstellung soll durch ein Softwaresystem gelöst werden, welches verschiedene Komponenten und Teilsystem zu einem Gesamtsystem integriert. Die Komponenten und Teilsysteme müssen unterschiedliche Funktionalitäten umsetzen und den Anforderungen der Problemstellung gerecht werden. Eine Kernfunktionalität soll die Verarbeitung und Analyse von Smart Meter Daten sein, was durch ein bereits bestehendes Datenstrommanagementsystem (DSMS) umgesetzt werden kann. Neben der Verarbeitung sollen Smart Meter Daten und Analyseergebnisse für den Benutzer visualisiert werden. Hierfür ist eine Web-Frontend Komponente denkbar, da hierdurch leicht eine Entkopplung zur Geschäftslogik ermöglicht wird. Des Weiteren muss eine Möglichkeit zur persistenten Datensicherung existieren, sodass beispielsweise eine Benutzerverwaltung realisiert werden kann. Gleichzeitig soll das System mit einem Big Data Archive kommunizieren und mit diesem Daten austauschen können. Das Zusammenspiel dieser Teilsysteme und -komponenten soll durch einen Backend Server koordiniert werden. Zur Entwicklung des Systems wird unter anderem eine Netzsimulation für Smart Meter Daten benötigt, sodass das System und entsprechende Szenarien getestet werden können. Dadurch kann die Problematik nicht vorhandener Smart Meter Daten gelöst werden und es kann das dynamische Verhalten des Systems in Echtzeit getestet werden.

In Abbildung 5.2 ist ein Architektur-Mockup zu sehen, welches die Beziehungen zwischen den einzelnen Teilsystemen und -komponenten zeigt. Die Pfeile symbolisieren die Kommunikationsrichtung und den Datenaustausch zwischen den Komponenten und Systemen, welche selbst durch Rechtecke dargestellt sind.

Die Integration einer bereits existierenden Softwarelösung eines DSMS in das Gesamtsystem bietet Vorteile gegenüber einer eigenen Implementierung. Erstens ist die Entwicklung eines DSMS zeitaufwendig und liegt nicht innerhalb der zeitlichen Rahmenbedingungen des Projekts. Zweitens kann davon ausgegangen werden, dass ein jahrelang entwickeltes und betriebenes DSMS, in verschiedenen Anwendungsszenarien ausreichend getestet ist. Dadurch ist weitestgehend sichergestellt, dass das DSMS zuverlässig, performant und wartbar ist. Eine weitere wichtige Eigenschaft für das DSMS ist die Erweiterbarkeit um zusätzliche Funktionen, um beispielsweise neue Prognosemodelle für Smart Meter Daten entwickeln und integrieren zu können.

Das Web-Frontend muss verschiedene Darstellungsmöglichkeiten zur Verfügung stellen, um Smart Meter Daten und Analysedaten entsprechend für den Benutzer zu visualisieren. Beispielsweise sollen Darstellungen von Graphen, Histogrammen und Karten für die Visualisierung von GPS-Daten

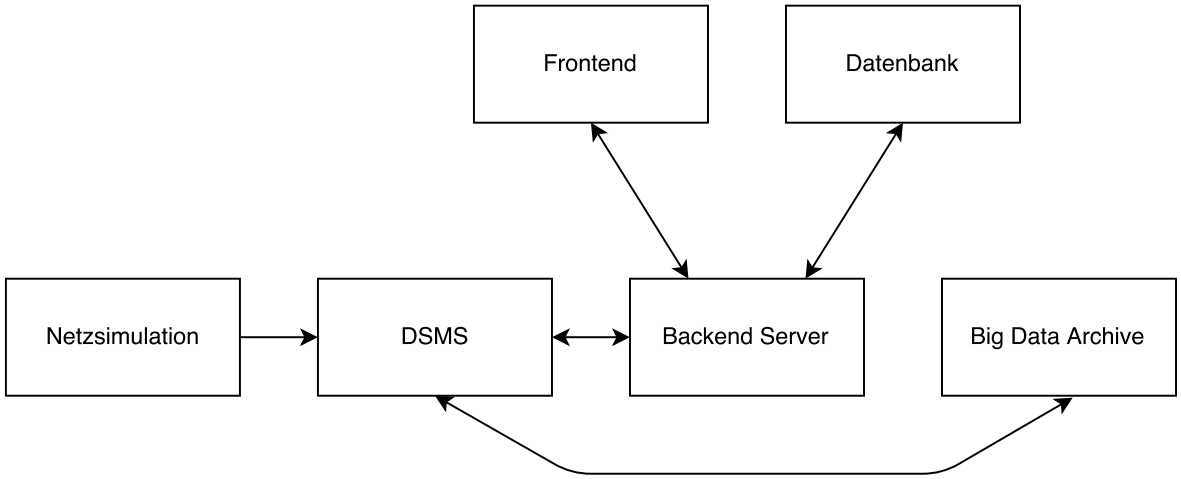


Abbildung 5.2: *Mockup der Systemarchichtektur*

auswählbar sein. Die Benutzeroberfläche könnte dem Benutzer folgende Funktionen zur Verfügung stellen:

- Smart Meter auf einer Karte anzeigen
- Smart Meter Topologien anzeigen
- aktuellen Verbrauchstand eines Smart Meters anzeigen
- durchschnittlichen Verbrauchsstand eines Smart Meters in den letzten drei Wochen anzeigen
- maximale Energieeinspeisung eines Wohnblocks anzeigen
- Prognose über Verbrauchsverhalten anzeigen
- Prognose über Einspeiseverhalten anzeigen
- Benachrichtigung über Fehlerzustände anzeigen lassen
- Benutzer anlegen und verwalten
- Darstellungsoptionen der Website konfigurieren

Die Netzsimulation muss durch ein geeignetes Framework umgesetzt werden, sodass verschiedene Szenarien simuliert werden können und vor allem das System bezüglich seiner Performanz evaluiert werden kann. Eine effiziente Datenverarbeitung zeichnet sich durch eine geringe Latenz und hohe Datenrate aus. Die Entwicklung einer eigenen Simulationssoftware ist nicht im zeitlichen Rahmen des Projekts vorgesehen.

Es soll eine standardisierte Schnittstelle zu einem Big Data Archive entwickelt werden, sodass das Archive verschiedene Machine Learning Modelle an das DSMS schicken kann. Die Analyseergebnisse sollten dann direkt vom DSMS in das Big Data Archive geschrieben werden, welche persistent gespeichert und für weitere Analysemethoden verwendet werden können. Hier wird eine Kooperation zwischen der Projektgruppe *E-Stream – Big Data Analytics im Smart Grid* und der Projektgruppe

Big Data Archive Data Science zur Energiewende angestrebt, die für die Entwicklung des Big Data Archive verantwortlich ist.

Die Kommunikation zwischen den einzelnen Teilsytemen und -komponenten soll durch standardisierte Austauschformate und Kommunikationsprotokolle realisiert werden, welche im Smart Grid Kontext relevant sind. Hier ergeben sich beispielsweise folgende Möglichkeiten:

- Smart Meter Daten können durch die Spezifikation IEC 61850 beschrieben und übertragen werden
- Smart Meter Daten können durch DLMS/COSEM beschrieben und übertragen werden
- Smart Meter Daten können durch CIM beschrieben werden
- Machine Learning Modelle und Analyseergebnisse können durch PMML beschrieben werden
- Zur Steuerung des DSMS wird eine entsprechende Anfragesprache zurückgegriffen
- Die Kommunikation mit der Datenbank kann je nach Datenbank durch SQL, HQL oder NoSQL-Anfragesprachen realisiert werden
- Die Kommunikation mit dem Big Data Archive kann durch PMML umgesetzt werden

5.1.2.6 Bewertungskriterien der Lösungsidee

Um die Qualität des Softwaresystems evaluieren zu können, müssen verschiedene Bewertungskriterien in Hinblick auf die Systemaspekte berücksichtigt werden:

- Die **Performanz** des Systems ist bzgl. der Echtzeitdatenanalyse und -visualisierung essentiell, da nur durch eine effiziente Verarbeitung geringe Latenzen und eine hohe Datenrate erreicht werden können. Hierfür sollte vor allem das DSMS skalierbar sein und eine verteilte Anfrageverarbeitung unterstützen, welche beispielsweise über ein verteiltes Rechensystem erfolgen kann. Das Frontend muss ebenfalls eine effiziente Implementierung aller Darstellungskomponenten gewährleisten, damit der Benutzer bei der Bedienung nicht eingeschränkt wird. Zur Überprüfung der Performanz können verschiedene Simulationsszenarien entworfen werden für die das System in extremen Situationen getestet wird.
- Im Bezug auf **Interoperabilität** sollte das System standardisierte Schnittstellen zwischen seinen systeminternen Komponenten verwenden, sodass bei Bedarf einzelne Komponenten und Teilsysteme leicht ausgetauscht werden können. Das gleiche gilt für Schnittstellen, die an den Systemgrenzen liegen und die Kommunikation zwischen anderen Smart Grid Anwendungen beschreiben. Dadurch ist sichergestellt, dass das System leicht in verschiedenen Smart Grid Kontexten eingebettet werden kann.
- Die **Erweiterbarkeit** des Systems bezieht sich auf die Komponenten und Teilsysteme, welche durch ein entsprechendes Architektur-Framework abstrahiert werden sollten, sodass diese leicht um neue Funktionalitäten erweitert werden können. Beispielsweise soll es möglich sein weitere Machine Learning Algorithmen und Prognosemodelle zum DSMS hinzuzufügen. Des Weiteren ist es wünschenswert, wenn das System um völlig neue Funktionalitäten erweitert werden kann. So könnte das System beispielsweise Daten aus dem Bereich der Verteilnetzautomatisierung integrieren oder im Bereich des Smart Home eingesetzt werden.
- Die **Wartbarkeit** des Systems charakterisiert sich durch zwei Bereiche: Es sollten wenige Wartungen durchgeführt werden müssen, wobei im Idealfall nach dem einmaligen Einrichten des Systems

gar keine Wartungen notwendig sein sollten. Zweitens sollten Wartungen leicht durchführbar sein und das verantwortliche Personal durch eventuelle Schulungen auf die Wartungsarbeiten vorbereitet werden.

- Die **Benutzbarkeit** des Systems zeichnet sich dadurch aus, dass die Bedienung der Visualisierungskomponenten für den jeweiligen Benutzer klar verständlich ist und keine Missverständnisse auftreten. Dies lässt sich möglicherweise durch eine Usability Studie überprüfen, sodass bereits in der Entwicklungsphase an den Benutzer angepasste Bedienoberflächen evaluiert werden können.
- Die **Sicherheit** des Systems sollte durch moderne kryptografische Verfahren gewährleistet werden, sodass personenspezifische Daten vertrauchlich und glaubwürdig verarbeitet werden können.
- Im Bezug auf **Datenrecht** soll der Zugriff auf Daten insofern möglich sein, wenn der jeweilige Benutzer die entsprechenden Zugangsrechte besitzt. Das System sieht keine Speicherung von personenspezifischen Daten vor und sollte nur für eine geschützte Datenübertragung verantwortlich sein.

5.1.3 Überblick

Die Spezifikation gliedert sich in Unterabschnitt 5.1.4, Abschnitt 5.2 und dem Kapitel 13. Der Unterabschnitt 5.1.4 umfasst einen Überblick über das System, welches durch ein Domänenmodell beschrieben wird. Hinzu kommt die Beschreibung des Systems durch ein Anwendungsfalldiagramm und ein Kontextdiagramm sowie allgemeine Systemfaktoren. Weiter wird eine Machbarkeitsstudie präsentiert, also eine Argumentation, ob das hier zu entwickelnde System realisierbar ist.

Der Abschnitt 5.2 liefert eine Auflistung aller erhobenen Anforderungen, welche jeweils durch einen Indentifier identifiziert werden können. Die Anforderungen werden durch einen Beschreibungstext beschrieben. Es ergeben sich insgesamt sieben verschiedene Arten von Kategorien für Anforderungen:

- Funktionale Anforderungen
- Leistungsanforderungen
- Schnittstellenanforderungen
- Datenbank-Anforderungen
- Entwurfsanforderungen
- Qualitätsanforderungen
- weitere Anforderungen

5.1.4 Allgemeine Rahmenbedingungen

Das System versucht die Richtlinien der deutschen Gesetzgebung zu erfüllen. Dies bedeutet, dass die Übertragung von Smart Meter Daten durch DLMS/COSEM stattfinden und daher durch das System unterstützt werden muss. Für den Austausch von Daten mit anderen Systemen soll ein standardisiertes Datenformat durch PMML benutzt werden. Dadurch lassen sich sowohl Daten für das Data Mining austauschen, als auch Modelle und Informationen zum Modell sowie das benutzte Verfahren angeben. Im Fall von Datenströmen kann durch eine Erweiterung auch die Fenstergröße hinzugefügt werden.

Für den Server alleine wird durch das Spring-Framework ein System empfohlen, welches etwa 1GB Hauptspeicher zur Verfügung stellen kann. Darüber hinaus können dem System aus verschieden Quellen Daten zur Verarbeitung hinzugefügt werden, indem diese Daten in ein Kafka-Cluster geschrieben werden. Darüber hinaus lassen sich Ergebnisse aus der Verarbeitung auch wieder aus dem Kafka-Cluster auslesen.

Das Austauschformat zwischen dem Client und Server ist JSON, da dieses ohne weiteren Aufwand von beiden übersetzt werden kann. Das Verbindungsprotokoll zwischen Client und Server ist HTTP für einmalige Anfragen und das Websocket-Protokoll für eine dauerhafte Verbindung, die zur automatischen Aktualisierung des Clients genutzt werden kann.

5.1.4.1 Grenzen des Systems

Das System ist ein Prototyp zur Analyse und Visualisierung von Smart Grid Daten und dient daher zur Verarbeitung selbiger. Darüber hinaus kann es Hinweise zu Störungen, zum Beispiel Lastspitzen oder Unterversorgung, geben und dem Verteilnetzbetreiber unterstützen, diese Störungsquellen einzugrenzen. Des Weiteren soll es möglich sein Unregelmäßigkeiten einzelner Haushalte zu erkennen. Dazu zählt die Einspeisung von Netz-Strom als Solarstrom oder das Übertragen fehlerhafter Zählerständen. Abschließend kann das System Hinweise auf Ausfälle von Teilnetzen oder Quellen im Niedrigstrombereich geben.

Neben Hinweisen kann das System außerdem auch Prognosen über den Verbrauch und die Erzeugung für das Netzleitpersonal geben, welche auf Basis dieser Vorhersagen entsprechende Steuerungsmechanismen anstoßen können.

Das System selber wird aber keine Steuerung des Verteilnetzes vornehmen oder Steuerungs- und Messeinheiten ersetzen. Außerdem ist es als Hilfe für das Verteilnetzbetreiberpersonal konzipiert und nicht zur vollständigen Automatisierung gedacht. Darüber hinaus können Konzepte dieses Prototyps für weitere Systeme aufgegriffen, angepasst und übernommen werden.

5.1.5 Annahmen und Abhängigkeiten

Innerhalb der Projektgruppe wird mit dem Projekt NetzDatenStrom kooperiert, um den Prototypen zu entwickeln. Dabei ist es möglich Anforderungen an die vom Projekt NetzDatenStrom entwickelte Simulation zu stellen, welche innerhalb der Projektgruppe für Netzdaten verwendet wird und Ergebnisse aus der Simulation visualisiert.

Zur Verarbeitung wird mit dem Odysseus-Team gearbeitet, da die Verarbeitung nach dem Datenstrommodell geschieht. Dabei erweitert die Projektgruppe Odysseus, um Data Mining Fähigkeiten und Konfigurierbarkeit von Data Mining Operatoren und wird dabei durch das Odysseus Team unterstützt.

Außerdem arbeitet diese Projektgruppe mit der Projektgruppe DAvE zusammen, welche Data Mining Modelle auf Basis historischer Daten zur Verfügung stellt und die Daten dieser Projektgruppe nutzen, um den Bestand ihres Archivs zu erweitern.

Für die IT-Infrastruktur der Projektgruppe wird mit dem Softwarelabor der Abteilung Informationssysteme zusammengearbeitet, sodass die Projektgruppe Zugriff auf einen Atlassian-Stack aus JIRA, Bitbucket und Confluence sowie eine Menge von virtuellen Maschinen hat, die nach eigenem Bedarf mit Software und einem Demo-System ausgestattet werden können.

5.1.5.1 Annahmen über die Systemumgebung

Das System benutzt Kafka als Message Broker zwischen verschiedenen Teilsystemen. Das System ist in der Lage die Konfiguration des Clusters selber vorzunehmen.

Durch die Entwicklung mit Java ist eine aktuelle Version von Java 8 zum Betrieb nötig. Alle Abhängigkeiten von Bibliotheken und Frameworks werden in einer ausführbaren Datei gebündelt und müssen nicht weiter beachtet werden. Für den Betrieb des Servers wird eine Portfreigabe benötigt, durch die auf das Webinterface zugegriffen werden kann.

Zur Verwaltung von Nutzern und zur Persistierung von Ereignissen und Einstellungen wird eine Datenbank benötigt. Durch die Verwendung von *Java-Database-Connectivity* (JDBC) und *Hibernate* ist es möglich eine beliebige relationale Datenbank zu verwenden.

Für die Datenverarbeitung nach dem Datenstrommodell wird Odysseus benutzt. Dafür ist es erforderlich, dass eine Instanz oder bei steigender Last OdysseusNet vorhanden ist, welche mit den Features für die Verarbeitung der Projektgruppe ausgestattet ist.

Da das Smart Grid sich noch im Aufbau befindet und die Nutzung von Daten von Verteilnetzbetreibern durch das Datenschutzgesetz nicht erlaubt ist, wird auf die im Projekt NetzDatenStrom entwickelte Simulation für Smart Grid Daten zurückgegriffen, welche über das Simulationsframework *mosaik* betrieben wird. Daher soll diese auf unbestimmte Zeit laufen.

5.1.5.2 Risikoanalyse

Der Änderungsaufwand, das Gesamtsystem an Änderungen der *mosaik*-Simulation anzupassen, ist relativ groß, da Anfragen, der `DataHandler` in Odysseus und mögliche Ergebnisse maßgeblich von der Simulation beeinflusst werden. Eine Änderung des Datenformats der Simulation hat zum Beispiel zur Folge, dass der `DataHandler` neu implementiert werden muss. Eine Erweiterung oder Änderung der Simulation durch zusätzliche oder entfernte Attribute ist gefolgt von einer Anpassung des Servers, des Anfragesystems, um entsprechende Anfragen in Odysseus zu installieren, und einer Erweiterung der REST-API für zusätzliche Ergebnisse.

Odysseus implementiert Datenstromverarbeitung durch den Intervallansatz. Als Forschungsprojekt sind alle Aspekte des Systems Gegenstand für Änderungen. Daher kann eine Änderung des Anfragesystems oder der Verlust von Operatoren zu einem erheblichen Mehraufwand führen, da fehlende Funktionalität neu implementiert oder wiederhergestellt werden muss.

Kafka befindet sich momentan noch in der Beta-Phase. Daher ändert sich die API Kafkas mit jedem Release. Für eine Nutzung von Kafka muss daher auf eine saubere Kapselung der API geachtet werden, sodass eine Aktualisierung Kafkas einen möglichst geringen Teil des Systems betrifft.

Zum Schluss hängt das System auch von dem Produkt der Projektgruppe DAvE ab. Ein Wegfall des Systems der Projektgruppe DAvE lässt sich nicht direkt im System kompensieren. Allerdings lässt sich, mit einem vergleichsweise geringen Aufwand, ein Server aufsetzen, der die Aufgabe des Archivs im Rahmen dieser Projektgruppe übernimmt. Der Server kann über statische Modelle verfügen, welche in einem Intervall an das Gesamtsystem geschickt wird.

5.1.6 Anwendungsfälle

Im Folgenden werden die Anwendungsfälle, die sich aus den Anforderungen ergeben, beschrieben. Zusammengehörige Anwendungsfälle werden in einigen Fällen in einem größeren Anwendungsfall zusammengefasst, um die Übersichtlichkeit zu verbessern.

5.1.6.1 UC-PGESTREAM-01 Topologie bereitstellen

In diesem Anwendungsfall bezieht die Projektgruppe E-Stream die unterliegende Topologie der Smart Meter von der NetzDatenStrom Simulation. Die Topologie soll beim Start des E-Stream Systems bezogen werden. Hierfür muss sichergestellt sein, dass die Topologie von der NetzDatenStrom Simulation bereitgestellt wird und die Simulation aktiv ist. Wenn diese Vorbedingungen erfüllt sind, kann die Projektgruppe E-Stream Topologiedaten abfragen. Optional können diese Daten transformiert und so effektiver im System verarbeitet werden. Die Abfrage der Topologiedaten sollte sofort geschehen, also kein langwieriger Prozess sein. Das Anwendungsfalldiagramm ist in Abbildung 5.3 und die tabellarische Darstellung in Tabelle 14.1 zu sehen.

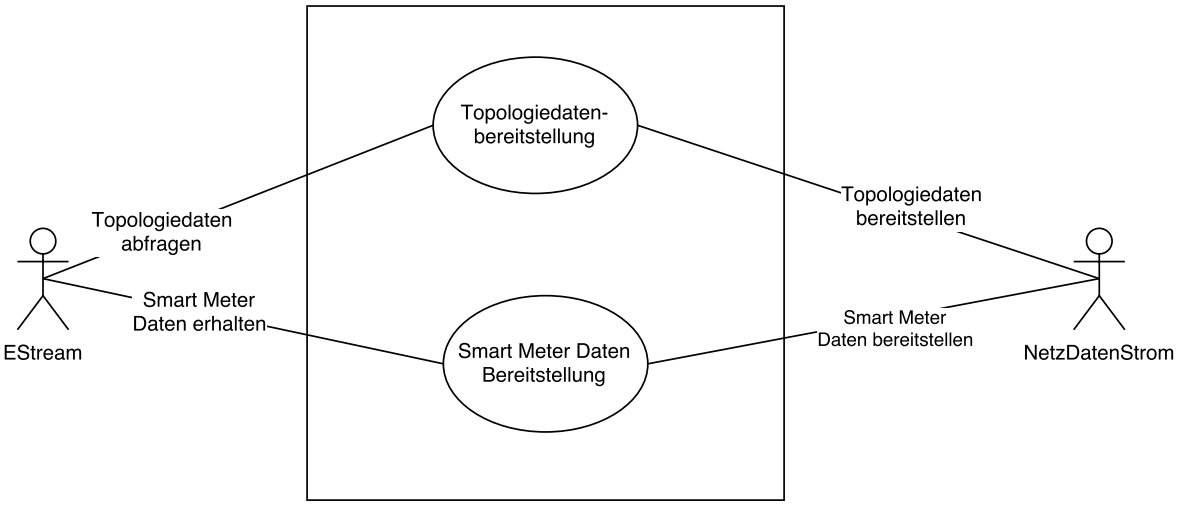


Abbildung 5.3: *UC-PGESTREAM-01 und UC-PGESTREAM-02*

5.1.6.2 UC-PGESTREAM-02 Smart Meter Daten erhalten

Um die Smart Meter Daten zu erhalten, ist ein Zusammenspiel des E-Stream Systems und der NetzDatenStrom Simulation notwendig. Beide Systeme müssen hierfür aktiv sein. Nach dem Start des E-Stream Systems sollen Smart Meter Daten kontinuierlich empfangen werden. Dies setzt voraus, dass die Simulation diese Daten produziert und über eine bekannte Schnittstelle anbietet. Die Daten werden dann vom E-Stream System standardmäßig alle 15 Minuten erhalten, sodass das letzte Zeitfenster immer verfügbar ist. Das 15 Minuten Intervall ist ein Standard, der aus datenschutzrechtlichen Gründen so vorgegeben wird. Theoretisch erfolgt die Übertragung kontinuierlich, also immer wenn neue Daten verfügbar sind. Dies entspricht jedoch in der Praxis einem 15 Minuten Intervall. Dieser Anwendungsfall ist in Abbildung 5.3 visualisiert und in Tabelle 14.2 tabellarisch dargestellt.

5.1.6.3 UC-PGESTREAM-03 Verbrauch und Erzeugung prognostizieren

Der Verbrauch und die Erzeugung der Smart Meter müssen prognostiziert werden können. Ein Nutzer des E-Stream Systems startet eine Prognoseanfrage. Je nachdem welche Art von Prognose angefragt wurde, werden die benötigten Daten von verschiedenen Datenquellen bezogen. Bei Prognosen auf einem kurzem Zeitraum beziehungsweise dem letzten Datenfenster werden lediglich die Daten aus dem DSMS, in diesem Fall Odysseus, benötigt. Wenn ein längerer Zeitraum zur Prognose herangezogen werden soll, müssen zusätzlich historische Daten aus dem Langzeitdatenarchiv der PG DAvE bezogen werden. Auf diesen Daten wird danach eine Regression ausgeführt, um die Entwicklung des Verbrauchs beziehungsweise der Erzeugung zu prognostizieren. Die Vorbedingung hierzu ist, dass historische Smart Meter- und Wetterdaten verfügbar sind. Zusätzlich können die Abweichungen der Kurz- und Langzeitprognoseverglichen werden. Nachdem alle Verarbeitungsschritte erfolgt sind, wird die Prognose dem Nutzer des Systems angezeigt. Die Regression wird sofort nach dem Stellen der Anfrage erstellt. Dieser Anwendungsfall ist zusammen mit Unterunterabschnitt 5.1.6.4 in Abbildung 5.4 visualisiert. In Tabelle 14.3 ist er tabellarisch dargestellt.

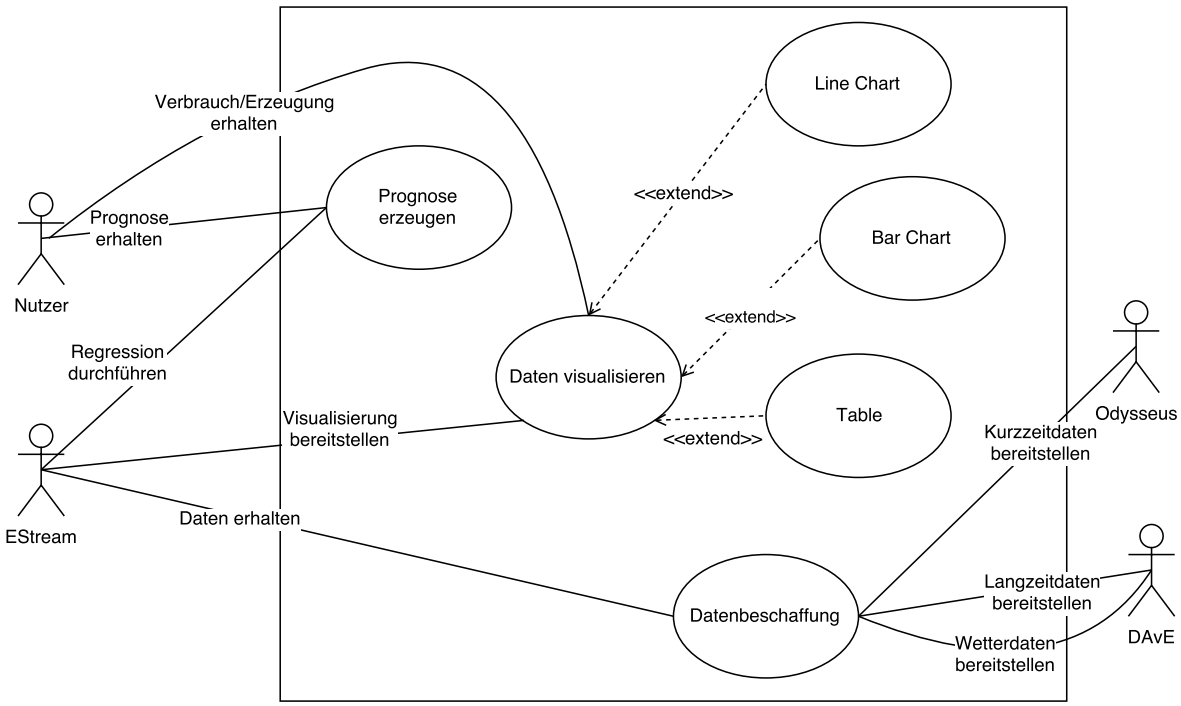


Abbildung 5.4: *UC-PGESTREAM-03 und UC-PGESTREAM-04*

5.1.6.4 UC-PGESTREAM-04 Verbrauch und Erzeugung anzeigen

In diesem Anwendungsfall geht es darum, den Verbrauch und die Erzeugung, eventuell auch mit den Ergebnissen einer Prognose, anzuzeigen. Es wird vorausgesetzt, dass alle anzuzeigenden Daten vorhanden sind und der Nutzer die Berechtigung hat, diese Daten einzusehen.

Aufgrund der Anzahl an Smart Metern, ist es normalerweise nicht sinnvoll den Verbrauch oder die Erzeugung eines einzelnen Smart Meters anzuzeigen. Deswegen werden die Verbrauchs- und Erzeugungsdaten akkumuliert bevor sie visualisiert werden. Mögliche Verfahren sind unter anderem Durchschnitt, Maximum, Minimum, Median, Standardabweichung oder Varianz. Diese Werte werden für jedes Zeitfenster der Daten berechnet und fassen dieses Fenster damit in einer oder wenigen Kenngrößen zusammen. Für die ausgewählten Kenngrößen werden anschließend passende Darstellungsarten gewählt. Mögliche Optionen beinhalten hier generische Tabellen, Linien- und Balkendiagramme oder topologische Visualisierungen, wie zum Beispiel ein Netz. Nachdem die Darstellungsart festgelegt wurde, werden die visualisierten Ergebnisse dem Nutzer angezeigt. Falls die Daten für die Berechnung der Kenngrößen nicht bezogen werden können, sollen die letzten gespeicherten Daten verwendet und eine entsprechende Fehlermeldung ausgegeben werden.

Die Visualisierungskomponente bietet dem Nutzer die Möglichkeit, eine Auswahl der anzuzeigenden Daten zu treffen. Der Nutzer kann zwischen Verbrauch, Erzeugung, Wetterdaten oder einer beliebigen Kombination dieser Optionen wählen und sie anzeigen lassen. Des Weiteren kann das Zeitfenster spezifiziert werden. Etwa kann ein Nutzer die Daten des letzten Tages oder der letzten Stunde etc. anzeigen lassen. Beim Verändern dieser Parameter muss eventuell eine neue Anfrage oder Prognose gestartet werden. Die Auswahl des Nutzers soll mindestens bis zum Sitzungsende gespeichert bleiben. Ein Diagramm zum hier dargestellten Anwendungsfall ist in Abbildung 5.4 zu sehen und in Tabelle 14.4 tabellarisch dargestellt.

5.1.6.5 UC-PGESTREAM-05 Topologie visualisieren

Ein Nutzer des E-Stream Systems soll die Möglichkeit haben, die Topologie des Netzes zu visualisieren. Es wird vorausgesetzt, dass der Nutzer hierzu die benötigten Rechte hat und Topologiedaten vorhanden sind. Für die Visualisierung der Topologie sind drei verschiedene Optionen vorhanden. Erstens die abstrakte Topologie, welche das Netz als Graph visualisiert. Hier geht es vor allem um die Verbindungen der Smart Meter beziehungsweise Smart Meter Gateways und nicht um eine korrekte Darstellung der geografischen Lage. Die zweite Visualisierungsart ist eine geografische Karte, die die Topologie auf ein Satellitenbild projiziert und die genaue geografische Lage der Netzbestandteile visualisiert. Die letzte Option visualisiert die Topologie in einer Baumstruktur. Diese Visualisierungsmöglichkeit soll vor allem dazu genutzt werden, in den anderen Darstellungen zu navigieren. Wird ein Knotenpunkt in der Baumstruktur markiert, führt dies zu einer Hervorhebung des gleichen Knotenpunkts in den anderen Visualisierungsarten oder die Sicht wird auf diesen Knotenpunkt und seine Kindelemente eingeschränkt. Die Voraussetzung für die Visualisierung mittels Baumstruktur ist also, dass eine der anderen Visualisierungsarten bereits erfolgreich geladen wurde.

Zusätzlich soll es möglich sein, auszuwählen, welche Daten der Smart Meter in der Topologie sichtbar sind und auf welches Zeitfenster sie sich beziehen. Diese Parameter sollen mindestens bis zum Sitzungsende gespeichert werden, damit der Nutzer sie nicht bei jeder Benutzung einstellen muss. Falls keine Daten für das ausgewählte Zeitfenster vorhanden sind, soll eine entsprechende Fehlermeldung ausgegeben werden. Dieser Anwendungsfall wird in Tabelle 14.8, Tabelle 14.9 und Tabelle 14.10 weiter beschrieben und im Anwendungsfalldiagramm Abbildung 5.5 visualisiert.

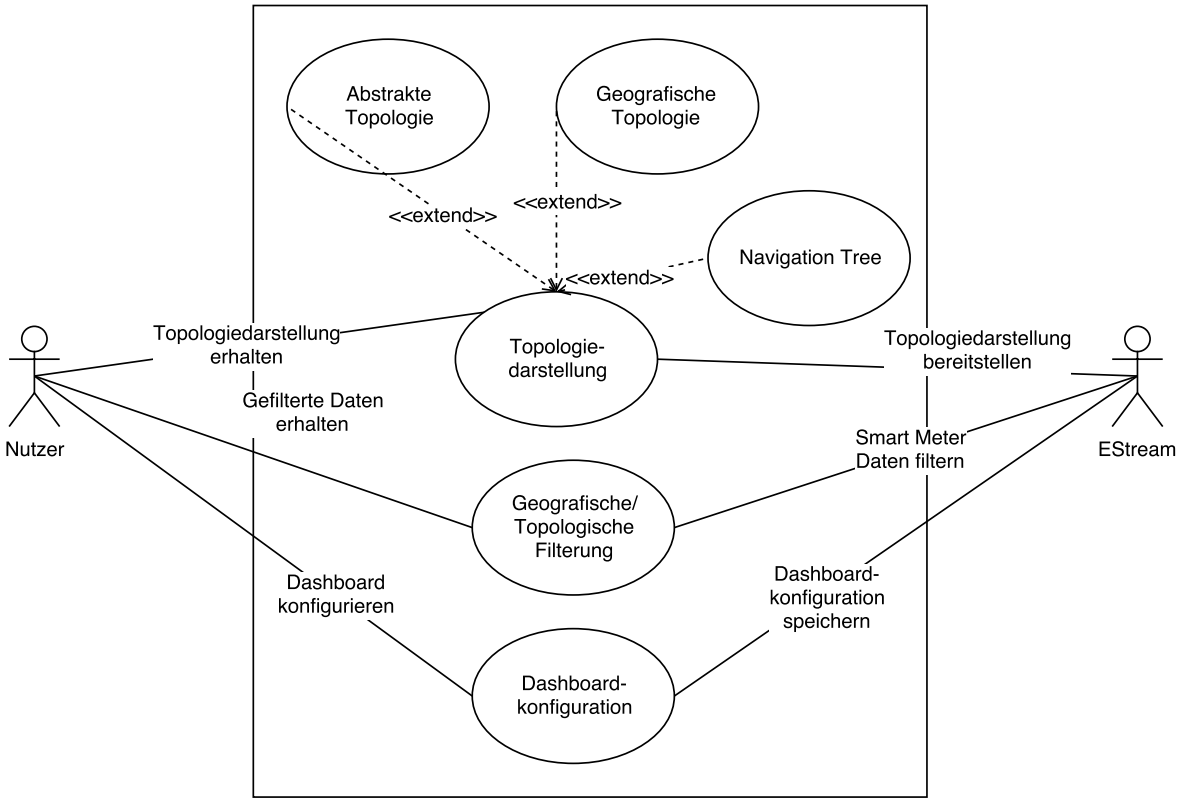


Abbildung 5.5: *UC-PGESTREAM-05, UC-PGESTREAM-07 und UC-PGESTREAM-08*

5.1.6.6 UC-PGESTREAM-06 Plausibilitätsprüfung

Das E-Stream System soll dazu in der Lage sein, die Plausibilität der Smart Meter Daten zu analysieren. Hierbei werden Störungen und Unregelmäßigkeiten in den Daten erkannt und visualisiert. Störungen sind Ausfälle im Netz. Dies kann sich auf ein ganzes Netz, ein Teilnetz oder einzelne Smart Meter beziehungsweise Smart Meter Gateways beziehen. Zur Erkennung dieser Störungen können Schwellwertverfahren oder Klassifizierungen der Daten benutzt werden. Die Klassifizierung benutzt dabei einen von einem Big Data Archive bereitgestellten Klassifikator. Unregelmäßigkeiten bedeuten in diesem Kontext, dass der betrachtete Erzeuger oder Verbraucher nicht ausgefallen ist, aber ein von der Norm abweichendes Verhalten zeigt. Beispiele im Bezug auf eine Solarzelle beinhalten die Einspeisung von Strom bei Nacht, eine zu hohe Einspeisung für die aktuellen Wetterbedingungen und stark abweichende Erzeugung im Vergleich zu benachbarten Erzeugern. Für die Erkennung dieser Unregelmäßigkeiten können ebenfalls Schwellwertverfahren eingesetzt werden oder eine Ausreißererkennung direkt auf dem Datenstrom, beispielsweise mittels K-Nearest-Neighbors, durchgeführt werden.

Nachdem die Störungen und Unregelmäßigkeiten erkannt wurden, müssen sie lokalisiert werden. Das bedeutet, dass die Topologie so gefiltert werden muss, dass sie das erkannte Fehlverhalten eingrenzt. Der Nutzer wird über alle erkannten Fehler informiert und die gefilterte Topologie wird angezeigt. Voraussetzungen für diesen Anwendungsfall sind, dass der Nutzer die Berechtigung hat diese Daten einzusehen und er angemeldet ist, Smart Meter Daten erhaltenen werden, ein Klassifikator für den die

Fehlererkennung vorhanden ist und dass eine Störung oder Unregelmäßigkeit aufgetreten ist. Falls keine Smart Meter Daten erhalten werden, soll mit dem letzten verfügbaren Daten gearbeitet werden und eine entsprechende Fehlermeldung ausgegeben werden. Der Anwendungsfall wird in Tabelle 14.11 beschrieben und in Abbildung 5.6 visualisiert.

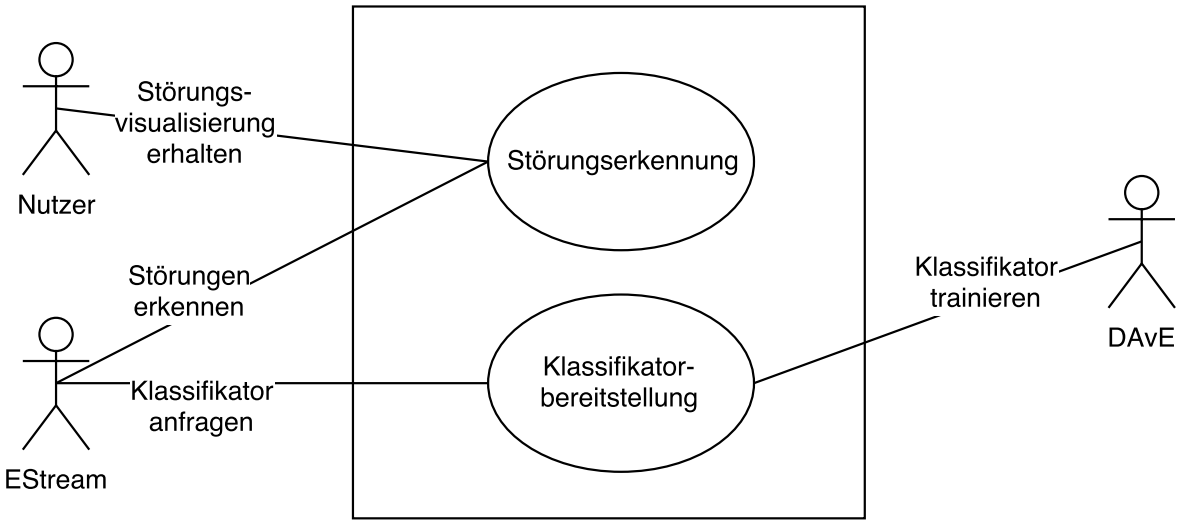


Abbildung 5.6: *UC-PGESTREAM-06*

5.1.6.7 UC-PGESTREAM-07 Geografische/topologische Filterung von Smart Meter Daten

Nachdem die Smart Meter Daten und die Topologie abgefragt und erhalten wurden, soll eine geografische beziehungsweise topologische Filterung durchgeführt werden können. Das Ergebnis der Filterung wird dann mit Hilfe einer der Visualisierungsarten für Topologien angezeigt. Die Voraussetzungen für die Filterung sind, dass der Nutzer die Berechtigung hat, die Daten einzusehen und dass die Topologie und Smart Meter Daten verfügbar sind. Der Nutzer hat für die Filterung mehrere Möglichkeiten. Zum einen kann die Topologie auf eine bestimmte Region eingegrenzt werden. Zum anderen kann eine Auswahl von Transformatoren (bevorzugt Netzbetreiber), Smart Meter Gateways oder Smart Metern getroffen werden. Wenn die Auswahl getroffen wurde, werden die gefilterten Smart Meter Daten visualisiert. Falls keine Smart Meter Daten vorhanden sind, sollen die letzten verfügbaren Daten benutzt werden und eine entsprechende Fehlermeldung ausgegeben werden. Als Referenz zu diesem Anwendungsfall dient Tabelle 14.15 und er wird in Abbildung 5.5 visualisiert.

5.1.6.8 UC-PGESTREAM-08 Dashboard konfigurieren

Ein Nutzer des E-Stream Systems soll die Möglichkeit haben, das Dashboard zu konfigurieren. Dieser Prozess wird durch das Öffnen des “Dashboard konfigurieren” Menüs gestartet. Hier kann der Benutzer die Daten und Darstellungsformen auswählen, die angezeigt werden sollen, sowie deren Position im Dashboard. Zu den Darstellungsarten gehören die Topologievisualisierungen, zum Beispiel die topologische und geografische Karte, sowie die Diagrammarten, zum Beispiel Linien- und Balkendiagramme. Für die ausgewählte Darstellungsart muss danach spezifiziert werden, welche

Daten angezeigt werden sollen. Als letztes muss die Position innerhalb des Dashboards ausgewählt werden, wo die Visualisierungskomponente platziert werden soll. Die Größe dieser Komponente kann nachträglich verändert werden und auch die Darstellungsform kann geändert werden, wenn diese kompatibel ist.

Das Dashboard wird sofort nach dem Konfigurieren angezeigt und die Konfiguration wird gespeichert, sodass diese nach dem nächsten Anmelden wieder geladen werden kann und der Benutzer das Dashboard nicht erneut konfigurieren muss. Der Anwendungsfall ist in Abbildung 5.5 visualisiert und in Tabelle 14.16 tabellarisch dargestellt.

5.1.6.9 UC-PGESTREAM-09 Anfragen der Datenverarbeitung verwalten

Als Administrator des E-Stream Systems soll man die Möglichkeit haben, Datenverarbeitungsanfragen zu erstellen und zu verändern. Dies geschieht über das “Anfragen verwalten”-Fenster. In diesem Fenster kann eine neue Anfrage aus einer vorhandenen Vorlage oder eine neue Vorlage erstellt werden. Die Datenart und Verarbeitungsschritte sind in der Vorlage spezifiziert. Falls eine Vorlage ausgewählt wird, wird diese geladen und kann anschließend angepasst werden. Die so entstandene Anfrage kann gespeichert werden. Falls eine neue Vorlage erstellt werden soll, müssen alle Bestandteile beschrieben werden. Dazu gehören die Datenart und Verarbeitungsschritte. Dieser Anwendungsfall ist in Tabelle 14.17 beschrieben und in Abbildung 5.6 visualisiert.

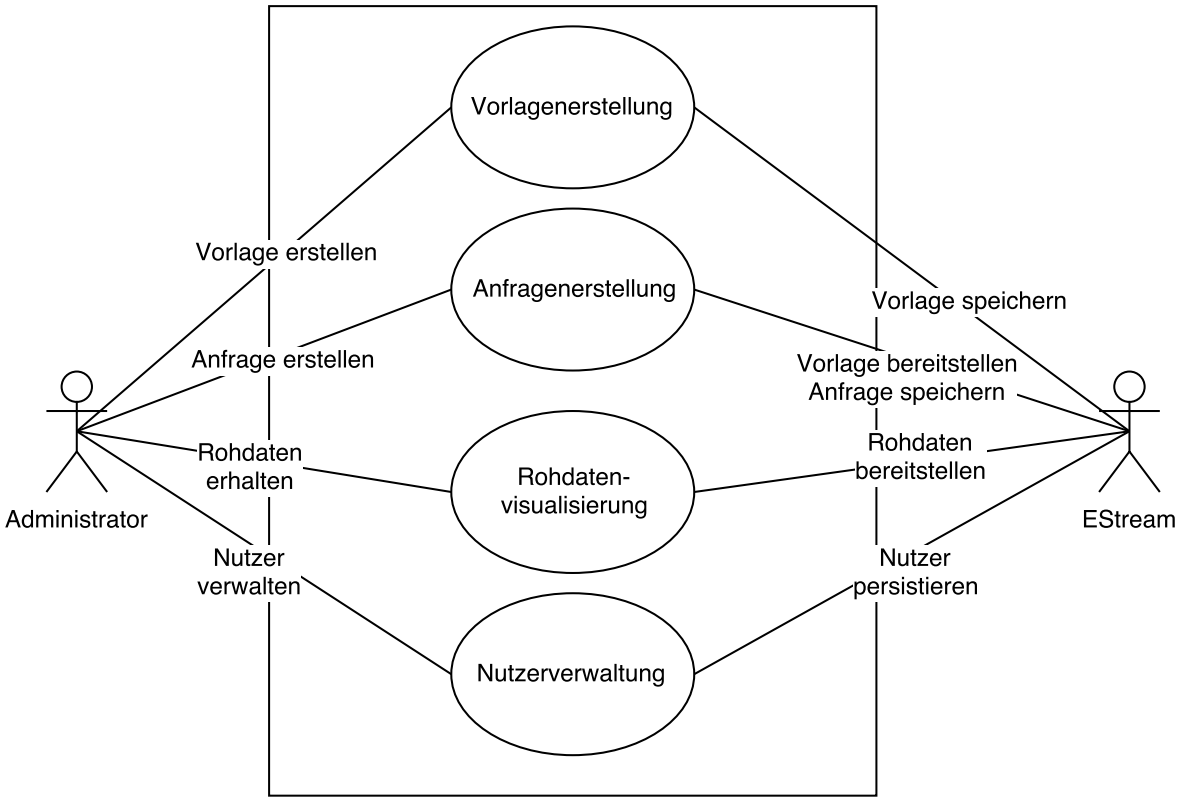


Abbildung 5.7: *UC-PGESTREAM-09-10-11*

5.1.6.10 UC-PGESTREAM-10 Rohdaten in Tabelle anzeigen

Ein Administrator des E-Stream Systems soll die Rohdaten direkt einsehen können. Die Darstellung erfolgt in tabellarischer Form, kann in bestimmten Fällen jedoch auch mit anderen Komponenten visualisiert werden. Wenn der Administrator die Rohdatenkomponente öffnet, kann er die Art der Rohdaten auswählen, zum Beispiel Smart Meter Daten oder Wetterdaten und außerdem die Rohdaten filtern. Es soll möglich sein, die Rohdaten zum Beispiel nach einem bestimmten Smart Meter Gateway zu filtern. Diese Einstellungen sollen mindestens bis zum Sitzungsende gespeichert werden, sodass der Nutzer die Konfiguration nicht erneut vornehmen muss. Die Komponente soll in diesem Fall mit der gespeicherten Konfiguration geöffnet werden. Dieser Anwendungsfall ist im Anwendungsfalldiagramm Abbildung 5.7 visualisiert und in Tabelle 14.18 tabellarisch dargestellt.

5.1.6.11 UC-PGESTREAM-11 Anfragen der Datenverarbeitung verwalten

Im “Nutzer verwalten”-Fenster soll ein Administrator die Nutzer des Systems verwalten können. Dies beinhaltet neue Nutzer anzulegen und bestehende Nutzer zu bearbeiten. Einem bestehenden Nutzer sollen Rechte gegeben oder entzogen werden können. Nutzer sollen auch vollständig gelöscht werden können. Wenn ein neuer Nutzer angelegt wird, müssen die Minimalinformationen zu diesem Nutzer eingetragen werden, also mindestens der Nutzername und das Passwort. Des Weiteren sollen ihm verschiedene Rechte zugewiesen werden können beziehungsweise soll er in mehreren Nutzergruppen sein können. Die Änderungen werden direkt persistiert. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.7 zu finden und die tabellarische Referenz in Tabelle 14.19.

5.1.7 Anwendungsfälle in Kooperation mit PG DAvE

In diesem Abschnitt werden die in Kooperation mit PG DAvE entwickelten Anwendungsfälle vorgestellt.

5.1.7.1 Abfrage von historischen Smart Meter Daten

Die Verbindlichkeit dieses Anwendungsfalls ist obligatorisch. Rohdaten sollen im DLMS/COSEM Format sein. Von dem System E-Stream wird eineAnfrage mit Parametern über eine HTTP-REST-Schnittstelle gestellt. Zu den Parametern zählt eine ID (zum Beispiel ein Hash-Wert), das Datenformat (JSON oder XML) und es wird eine fertige CQL-Anfrage übergeben. Daraufhin überprüft das System DAvE die Anfrage und gibt Rückmeldung. Bei valider Anfrage werden die Daten von dem System DAvE für die Übergabe verarbeitet. Anschließend erstellt das System DAvE ein Kafka-Topic mit der übergebenen ID (dem Hash-Wert) als Topicbezeichnung. Die Daten werden über das Kafka Topic bereitgestellt. Das System E-Stream ruft dann die Daten über Kafka Connect ab. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.8 zu finden und die tabellarische Referenz in Tabelle 14.20.

5.1.7.2 Abfrage von historischen Wetterdaten

Die Verbindlichkeit dieses Anwendungsfalls ist obligatorisch. Von dem System E-Stream wird eine Anfrage mit Parametern über eine HTTP-REST-Schnittstelle gestellt. Zu den Parametern zählt eine ID (zum Beispiel ein Hash-Wert), das Datenformat (JSON oder XML) und es wird eine fertige PQL-

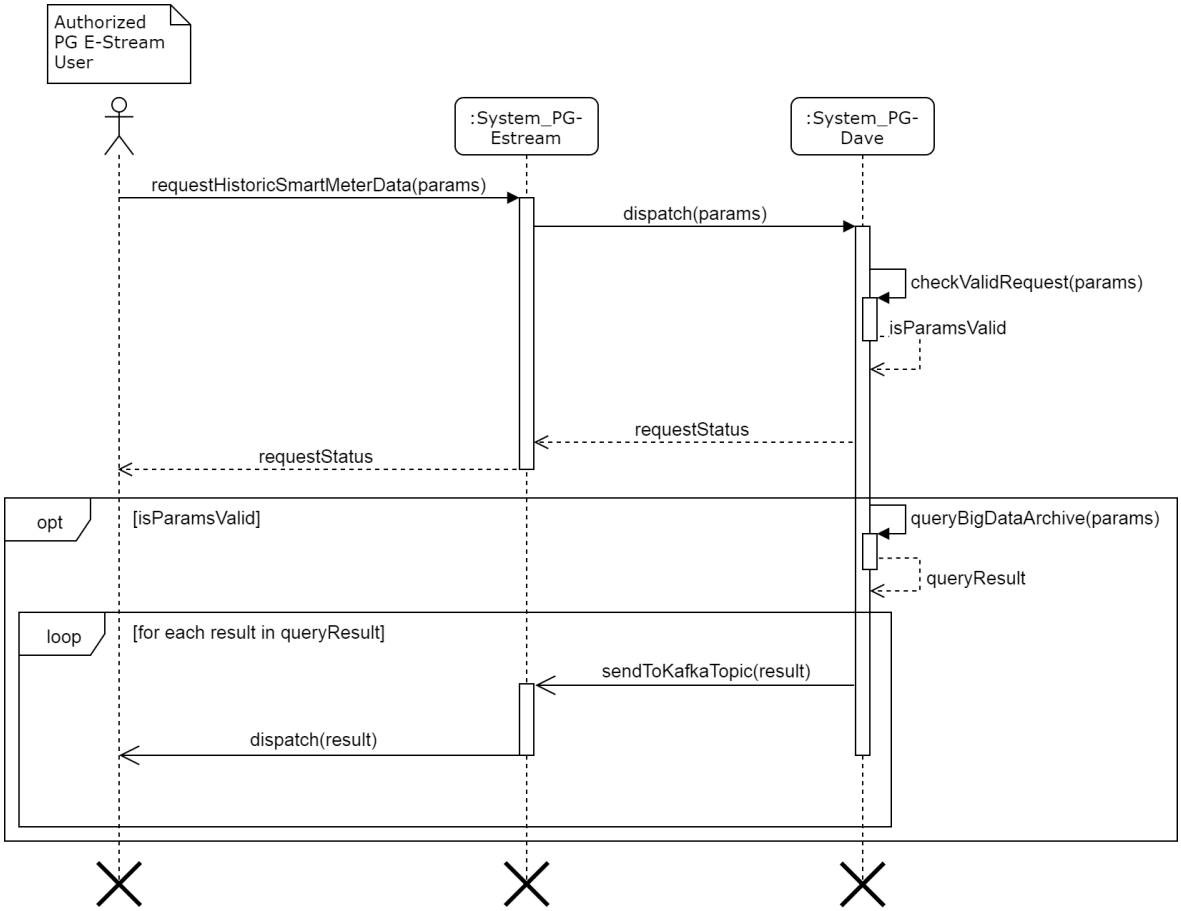


Abbildung 5.8: *Abfrage historischer Smart Meter Daten*

Anfrage übergeben. Daraufhin überprüft das System DAvE die Anfrage und gibt Rückmeldung über eine HTTP-Response. Bei valider Anfrage werden die Daten von dem System DAvE für die Übergabe verarbeitet. Anschließend erstellt das System DAvE ein Kafka-Topic mit der übergebenen ID (dem Hash-Wert) als Topicbezeichnung. Die Daten werden über das Kafka Topic bereitgestellt. Das System E-Stream ruft dann die Daten über Kafka Connect ab. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.9 zu finden und die tabellarische Referenz in Tabelle 14.21.

5.1.7.3 Abfrage von Mining Modellen (PMML)

Bereits bestehende PMML Modelle können über eine HTTP-Anfrage abgefragt werden. Die HTTP-Antwort enthält bei erfolgreicher Anfrage und dem Vorhandensein mindestens einer passenden PMML Spezifikation im Big Data Archive, das zeitlich letzte passende PMML Modell. Falls noch kein Model mit dieser Spezifikation erstellt wurde, wird diese Information als HTTP-Antwort verschickt. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.10 zu finden und die tabellarische

Referenz in Tabelle 14.22.

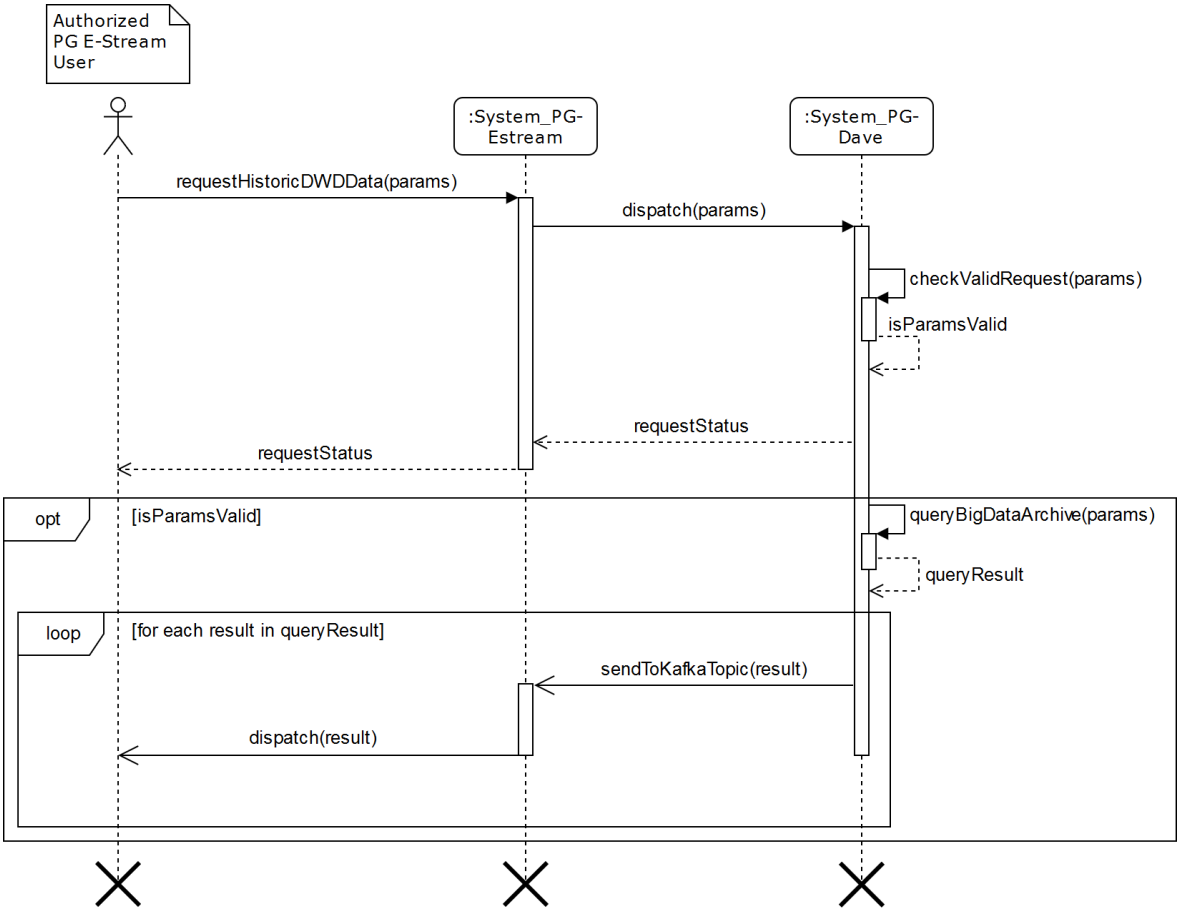


Abbildung 5.9: *Abfrage historischer Wetterdaten*

5.1.7.4 Erstellen von Mining Model Aufträgen

Dieser Anwendungsfall ist optional. Das System E-Stream stellt einen Miningmodellauftrag über eine HTTP-REST-Schnittstelle. Dieser Auftrag muss folgende Parameter beinhalten:

- Use Case: Power Forecast oder Load Forecast
- Dimensionseingrenzung: Eingrenzung der einzubeziehenden Daten
- gewünschtes Model (z. B. Regeln, Cluster, Klassifikationen o.ä.)
- Hash zum bestehenden Modell mit übergeben

Das System DAvE prüft daraufhin den Auftrag und gibt eine Rückmeldung als HTTP-Response. Bei valider Anfrage wird das Modell erstellt und bei erfolgreicher Bearbeitung wird dies auf den dafür vorgesehen Topic des Systems DAvE gesandt. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.11 zu finden und die tabellarische Referenz in Tabelle 14.23.

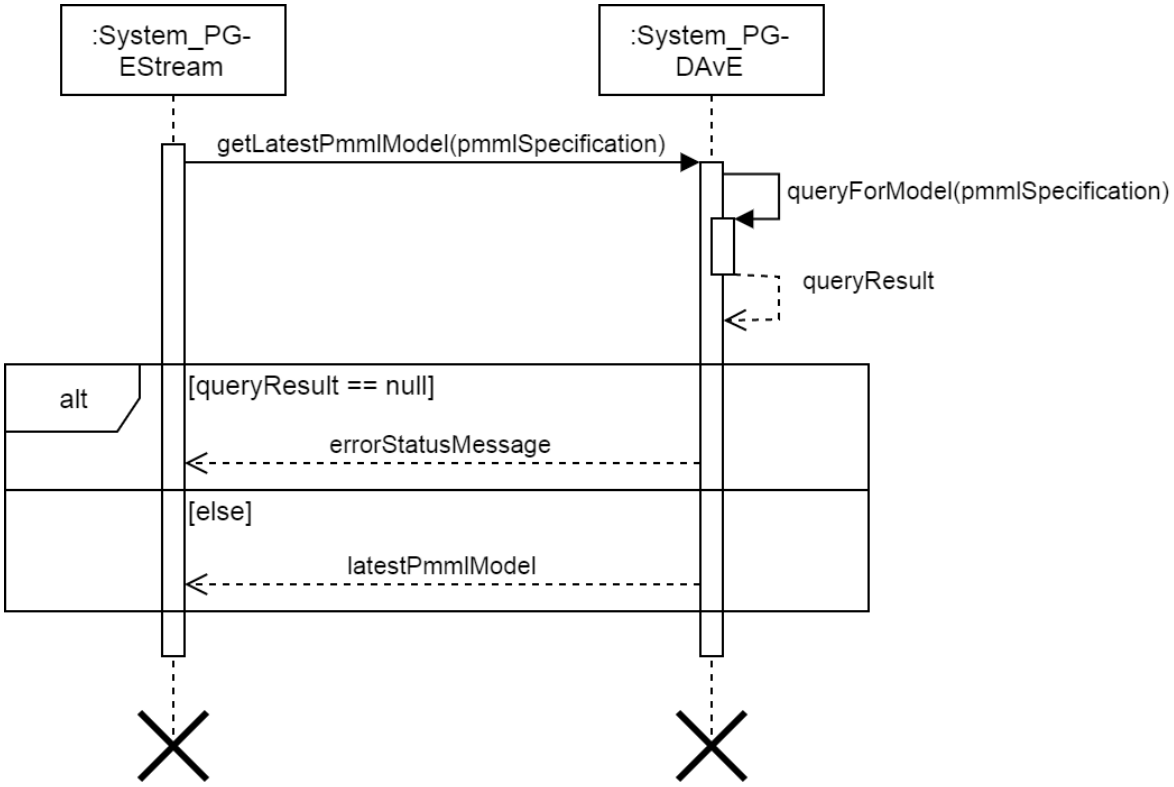


Abbildung 5.10: *Abfrage von Mining Modellen (PMML)*

5.1.7.5 Senden von Rohdaten zur Speicherung

Dieser Anwendungsfall ist optional und bezieht sich auf Smart Meter Daten in DLMS/COSEM Format. Die neu erhaltenen Daten des Netzdatensystems werden auf dem passenden Topic einer Kafka Instanz des E-Stream Systems gesandt und können so durch das System DAvE empfangen, aufbereitet und archiviert werden. Ein Diagramm zu diesem Anwendungsfall ist in Abbildung 5.12 zu finden und die tabellarische Referenz in Tabelle 14.24.

5.2 Spezifische Anforderungen

Im folgenden Abschnitt werden alle Funktionale und Leistungsanforderungen aufgelistet, die die diese Projektgruppe in ihrer Arbeit und in ihrer Zielsetzung definieren.

5.2.1 Funktionale Anforderungen

[f001] Das System muss den aktuellen Stromverbrauch im Smart Grid erfassen.

[f002] Das System muss die aus PV-Anlagen eingespeiste Energie in das Smart Grid erfassen.

[f003] Das System muss Störungen erkennen.

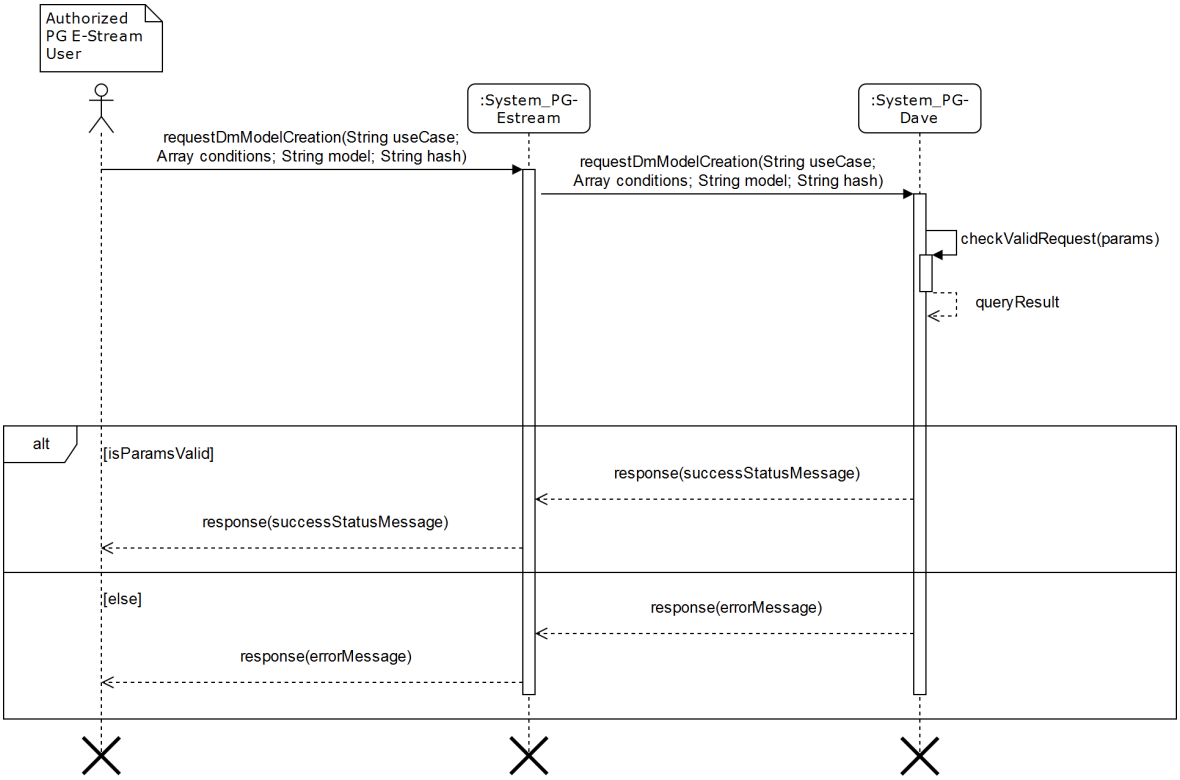


Abbildung 5.11: *Erstellen von Mining Model Aufträgen*

[f004] Das System soll den Ausfall einzelner oder mehrerer Netzknoten als Störung erkennen.

[f005] Das System soll unerwartete Stromeinspeisung als Störung erkennen.

[f006] Das System soll fehlerhafte Daten als Störung erkennen. ([f014])

[f007] Das System kann eine drohende Netzüberlastung erkennen.

[f008] Das System soll Störungen lokalisieren.

[f009] Das System muss Prognosen zum Netzstatus berechnen.

[f010] Die Prognose muss den Stromverbrauch für spezifische Zeiträume umfassen.

[f011] Die Prognose muss die Menge des eingespeisten Stroms für spezifische Zeitpunkte umfassen.

[f012] Die Prognosen können für unterschiedliche Prognosezeiträume unterschiedlich genau sein.

[f013] Die Prognose kann die Stromerzeugung einzelner PV-Anlagen umfassen.

[f014] Das System kann Daten einer Plausibilitätsprüfung unterziehen.

[f015] Das System muss über eine Nutzerverwaltung verfügen.

[f016] Das System muss Daten persistieren können.

[f017] Ein Netzbetreiber muss die Daten einsehen können.

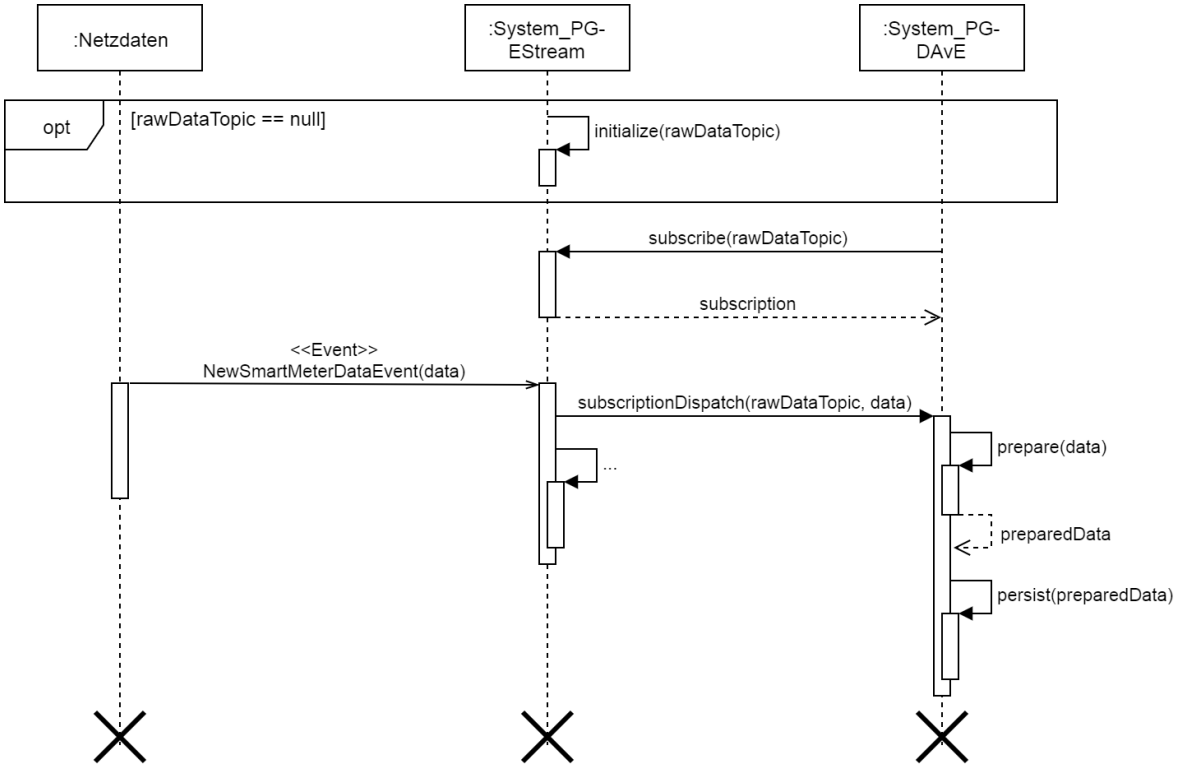


Abbildung 5.12: *Senden von Rohdaten zur Speicherung*

[f018] Das System muss die verarbeiteten Daten über eine Web-Schnittstelle darstellen. [i005]

[f019] Das System muss durch Personal beim Verteilnetzbetreiber bedienbar sein.

[f020] Die Daten sollen anonymisiert verarbeitet werden.

[f021] Das System muss Data Mining Verfahren anwenden.

[f022] Das System soll Prognosen anhand von Data Mining Verfahren erstellen können.

[f023] Das System soll Data Mining Ergebnisse aus anderen Systemen importieren können. ([i053])

[f024] Die Data Mining Verfahren sollen konfigurierbar sein.

5.2.2 Leistungsanforderungen

[p001] Die Durchführung des Projekts soll durch einen Scrum-Prozess geschehen.

[p002] Dokumente für das Projekt müssen in Confluence erstellt werden.

[p003] Der Scrum-Prozess muss in JIRA dokumentiert werden.

[p004] Die Zeiterfassung für das Projekt muss mit JIRA-Tempo durchgeführt werden.

[p005] Die Zeiten für Meetings müssen auf PGESTREAM1-Orga geloggt werden.

[p006] Die Simulation von Niederspannungsnetzwerken muss mit mosaik durchgeführt werden.

[p007] Die Verarbeitung der Datenmengen muss mit Odysseus realisiert sein.

[p008] Die Versionskontrolle soll nach Git Flow geschehen.

[p009] Eingabedaten sollen über mosaik simuliert werden.

[p010] Bitbucket muss als Git Remote Repository verwendet werden.

[p011] Die Dokumentation des Sprints wird zu Beginn des Sprints an eine Person verteilt.

[p012] Ein abgeschlossenener Task muss die Definition of Done erfüllen.

- [p013] Der (Feature-)Branch ist auf den aktuellen Develop-Branch gerebased.
- [p014] **Bei der Dokumentation**: Die Dokumentation lässt sich per Build-Skript kompilieren (ohne Error oder Bibliograhpy Warnungsmeldungen).
- [p015] **Beim E-Stream -Produkt**: Der Server lässt sich per *Gradle fullBuild* im Developmentprofile mit Vagrant-Docker und im Productionprofile kompilieren und der Client per *ng serve* mit dem *Ahead-of-Time* Parameter starten.
- [p016] Tests laufen lokal erfolgreich durch (wenn Tests im Jenkins fehlschlagen, muss ein Bugfix Task angelegt werden)
- [p017] **Nur beim E-Stream -Produkt**: Die Client Seite lässt sich aufrufen, man kann sich einloggen und das Dashboard / die Seite funktioniert normal (falls es nicht schon beim Review getestet wurde).
- [p018] Der Pull-Request wurde gestellt und akzeptiert.
- [p019] Task wurde für die Review freigegeben (in "ToReview"geschoben) und das Review wurde angenommen.
- [p020] Der JIRA Task wurde auf "Fertig (accepted)"gezogen.
- [p021] Der Branch wurde auf den Develop-Branch gemerged und der Branch gelöscht.

5.3 Schnittstellenanforderungen

5.3.1 Externe Schnittstellen

Die Beschreibung der externen Schnittstellen kann in vier Kategorien eingeteilt werden: Die Benutzungsschnittstelle, die Hardwareschnittstellen, die Softwareschnittstellen und die Kommunikationsschnittstellen. Für das System dieser Projektgruppe sind in erster Linie die Schnittstellen zum Benutzer und die Softwareschnittstellen relevant. Für Kommunikationsschnittstellen genauso wie für Hardwareschnittstellen wurden keine Anforderungen gefunden.

Die Anforderungen an die Benutzungsschnittstellen sind dadurch gegeben, dass die Benutzungsoberfläche des Systems eine Weboberfläche sein soll. Um diese Weboberfläche für möglichst viele Geräte nutzbar zu machen, soll diese sich an die Maße und Auflösung des Anzeigegeräts anpassen. Dafür soll das Paradigma des Responsive Webdesigns in der Gestaltung der Weboberfläche benutzt werden. Um dem Benutzer effektiv aktuelle Netzzustände und Ergebnisse aus Vorhersagemodellen

anzuzeigen, soll die Benutzungsschnittstelle die Daten grafisch anzeigen können. Da dem System durch eine bereitgestellte Simulation topologische Daten für Smart Grids zur Verfügung stehen, sollen dem Benutzer diese auf einer interaktiven geografischen Karte dargestellt werden. Weiterhin soll sich die Benutzeroberfläche bei Veränderungen im Datenbereich selbstständig aktualisieren, um so dem Nutzer unter anderem Warnungen oder Hinweise anzeigen zu können.

Da mehrere Benutzer mit dem System arbeiten können sollen, muss das System Oberflächen anbieten, in denen sich die Benutzer an- und abmelden können, bevor sie das Dashboard des Systems erreichen. Weiterhin soll das System die Möglichkeit bieten in einer Oberfläche Einstellungen an dem Dashboard vorzunehmen.

Bei den Softwareschnittstellen gibt es Anforderungen an zwei Hauptschnittstellen. Das sind zunächst die Netzdaten, die das System verarbeiten kann und Data-Mining Modelle, die in das System eingehen, um auf den eingehenden Daten Data-Mining Verfahren anzuwenden. Ersteres soll mit Hilfe der Standards Common Information Model (CIM) und DLMS/COSEM gelöst werden. Mit dem CIM existieren Standardreihen, die Entitäten, Assoziationen und Prozesse der Energiewirtschaft modellieren und mit COSEM existiert ein Standard für eine einheitliche Modellierung der Messung und Übertragung im Bereich von Smart Meter Daten. Damit das System eine einheitliche Semantik mit anderen externen Systemen teilt und somit die Integration zwischen diesem System und Externen gelingt, soll eine Schnittstelle für eingehende Netzdaten existieren, die auf diesen Standards aufbaut.

Des Weiteren muss die Interoperabilität mit Systemen gewährleistet werden, die das System für Data-Mining benutzen sollen. Für eingehende Modelle, mit denen die Netzdaten ausgewertet werden sollen, soll der Predictive Model Markup Language Standard (PMML) verwendet. Somit kann ebenfalls bei dieser Schnittstelle eine einheitliche Semantik bei Vorhersagemodellierung zwischen diesem und externen Systemen geschaffen werden.

5.3.2 Benutzungsschnittstelle

[i001] Die Benutzungsschnittstelle kann nach dem IMIS Styleguide gestaltet werden.

- [i002] Das Design der Benutzungsschnittstelle soll responsive sein.
  - [i003] Die Benutzungsschnittstelle soll Teile der Oberfläche bei Änderungen der Daten selbst aktualisieren.
  - [i004] Die Benutzungsschnittstelle kann sich nach Änderungen in der Konfiguration selbst aktualisieren.
- [i005] Daten müssen visualisiert werden.
  - [i006] Zur Visualisierung können Graphen benutzt werden.
  - [i007] Zur Visualisierung können Topologien benutzt werden.
  - [i008] Zur Visualisierung können geographische Daten verwendet werden.
  - [i009] Zur Visualisierung können Tabellen benutzt werden.
  - [i010] Zur Visualisierung können Listen verwendet werden.

[i011] Das System kann eine geographische Karte darstellen.

[i012] Das System kann auf einer geographischen Karte Smart Meter Gateways anzeigen.

[i013] Das System kann auf einer geographischen Karte Smart Meter anzeigen.

[i014] Das System kann auf einer geographischen Karte PV-Anlagen anzeigen.

[i015] Das System muss die Menge des in das Smart Grid eingespeisten Stroms anzeigen.

[i016] Das System kann aktuelle Energieumwandlung von PV-Anlagen, in einem ausgewählten Bereich, aggregiert anzeigen.

[i017] Das System kann aktuelle Energieumwandlung einzelner PV-Anlagen anzeigen.

[i018] Das System soll den aktuellen Energieverbrauch im Smart Grid anzeigen.

[i019] Das System kann den aktuellen Verbrauch aller Smart Meter Gateways aggregiert anzeigen.

[i020] Das System kann den aktuellen Verbrauch von aus Smart Meter Gateways, in einem ausgewählten Bereich, aggregiert anzeigen.

[i021] Das System kann den aktuellen Verbrauch einzelner PV-Anlagen anzeigen aggregiert anzeigen.

[i022] Das System kann Warnungen anhand spezifizierter Grenzwerte visualisieren.

[i023] Das System muss Prognosen zum Netzstatus anzeigen.

[i024] Das System soll die prognostizierte Menge des eingespeisten Stroms darstellen.

[i025] Das System soll den prognostizierten Stromverbrauch darstellen.

[i026] Das System soll einen Vergleich zwischen prognostiziertem Stromverbrauch und prognostizierter Stromerzeugung darstellen.

[i027] Das System soll einen Vergleich zwischen Stromverbrauch und -erzeugung darstellen.

[i028] Das System soll einen Vergleich zwischen aktueller und prognostizierter Stromerzeugung anzeigen.

[i029] Das System kann bei Abweichungen zwischen aktueller und prognostizierter Stromerzeugung Warnungen anzeigen.

[i030] Das System soll einen Vergleich zwischen aktuellem und prognostiziertem Stromverbrauch anzeigen.

[i031] Das System kann bei Abweichungen zwischen aktuellem und prognostiziertem Stromverbrauch Warnungen anzeigen.

[i032] Das System muss Störungen visualisieren.

[i033] Das System kann Störungen auf einer schematischen Karte darstellen.

[i034] Das System kann Störungen auf einer topologischen Karte darstellen.

[i035] Das System kann Störungen als Warnungen einblenden.

[i036] Das System soll dem Benutzer verschiedene Oberflächen anzeigen.

[i037] Das System soll dem Benutzer eine Landingpage anzeigen.

[i038] Das System soll dem Benutzer eine Anmeldeseite anzeigen.

[i039] Das System soll dem Benutzer eine Optionenseite anzeigen.

[i040] Das System soll dem Benutzer ein Dashboard anzeigen.

[i041] Das Frontend soll internationalisiert sein.

[i042] Das Frontend soll für Deutschland lokalisiert sein.

[i043] Das Frontend kann für weitere Sprachen lokalisiert sein.

5.3.3 Softwareschnittstellen

[i044] Das System muss Daten nach dem COSEM-Protokoll verarbeiten können.

[i045] Daten konform zum COSEM-Protokoll müssen im XML-Format verarbeitet werden können.

[i046] Das System soll Daten von Smart Meter Gateways verarbeiten können.

[i047] Das System soll Daten von Smart Metern verarbeiten können.

[i048] Das System soll Daten von PV-Anlagen verarbeiten können.

[i049] Das System soll nach dem DLMS-Standard kommunizieren können.

[i050] Das System soll COSEM-konforme Daten über TCP/UDP empfangen.

[i051] Das System soll Verschlüsselungsmechanismen unterstützen.

[i052] Das System soll Authentifizierungsmechanismen unterstützen.

[i053] Das System muss PMML-konforme Dokumente verarbeiten können.

[i054] Das System soll PMML-konforme Modelle verarbeiten können.

[i055] Das System soll PMML-konforme Ausgaben eines Modells erzeugen können.

[i056] Das System kann eine Schnittstelle zum Empfang von Smart Meter Gateway Daten bereitstellen.

[i057] Das System soll Daten konform zu einem Common Information Model Profil (CIM) verarbeiten können.

[i058] Das System muss über eine Schnittstelle zu einem Langzeitdatenarchiv verfügen.

[i059] Die Schnittstelle zu einem Langzeitdatenarchiv kann über Apache Kafka realisiert werden.

[i060] Das System soll eine Schnittstelle zur Abfrage der Rohdaten zur Verfügung stellen.

5.3.4 Datenbank-Anforderungen

An eine Datenbank werden im Rahmen der Projektgruppe diverse Anforderungen gestellt. Dabei ist insbesondere die sichere Speicherung von Benutzerdaten zur Zugangskontrolle, aber ggf. auch an die Speicherung von Daten zur Prognose des Netzstatus zu denken. Wesentlich ist hierbei die persistente Sicherung der Daten sowie die Erfüllung von Transaktionssicherheit, aber auch die Sicherheit der Datenbank selbst. Darüber hinaus ist die Administration der Datenbank zu Konfigurations- und Verwaltungszwecken erforderlich und eine Datenbank sollte möglichst von vielen Betriebssystemen unterstützt werden. Um maximale Flexibilität zu ermöglichen, kann eine Datenbank zudem verschie-

dene Datenformate unterstützen und bei Bedarf durch die Unterstützung verschiedener Formen von Skalierbarkeit an sich ändernde Performance-Anforderungen anpassbar sein.

[d001] Die Datenbank muss Daten persistent speichern können.

[d002] Die Persistierung der Benutzerdaten und der Systemkonfiguration soll durch ein etabliertes Datenbankmanagementsystem geschehen.

5.3.5 Entwurfsanforderungen

[de001] Das System muss durch eine Client/Server-Architektur realisiert sein.

[de002] Die Kommunikation zwischen Server und Client kann über HTTP erfolgen.

[de003] Das System muss über eine bidirektionale Kommunikation verfügen.

5.3.6 Qualitätsanforderungen

[q001] Das Frontend muss die Performance-Anforderungen erfüllen

[q002] Das System kann das Frontend in Echtzeit aktualisieren.

5.3.7 Weitere Anforderungen

[o001] Die Projektgruppe muss mit der Projektgruppe DAvE zusammenarbeiten.

6 Tools

6.1 Evaluation

In diesem Abschnitt werden die verschiedenen benötigten Tools für das Viewframework, die Visualisierung, Buildtools, Datenbanksysteme und Backend Web Frameworks evaluiert. Hierfür wird eine Nutzwertanalyse durchgeführt, welche im folgenden Abschnitt erläutert wird.

6.1.1 Nutzwertanalyse

Die Nutzwertanalyse (NWA) ist eine Methodik zur Evaluierung der Alternativen in komplexen Entscheidungsproblemen. Sie stellt ein rationales Hilfsmittel bei der Entscheidungsfindung dar. Das Ergebnis der Nutzwertanalyse ist eine kompakte Kennzahl pro Alternative, die den Nutzen widerspiegelt. Sie kann als eine Variante einer Pro- und Kontra-Liste gesehen werden, bei der die einzelnen Punkte gewichtet werden [283].

Mathematisch betrachtet, wird für jede Alternative ein gewichtetes Mittel aller Attributbewertungen gebildet. Es gilt:

$$v(a) = \sum_{r=1}^{m} w_r v_r(a_r)$$

wobei $v(a)$ das Ergebnis der NWA für Alternative a ist, w_r die Gewichtung des Attributes r und $v_r(a_r)$ die Bewertung des Attributes r für die Alternative a. Des Weiteren muss die Summe der Gewichtungen w_r 1 ergeben.

$$\sum_{r=1}^{m} w_r = 1$$

Die Entscheidungsfindung mit der NWA kann in sechs Schritte unterteilt werden [279].

1. Festlegung der Alternativen
2. Definition der Bewertungskriterien
3. Gewichtung der Bewertungskriterien
4. Festlegung des Bewertungsmaßstabes
5. Bewertung der Alternativen
6. Summierung und Auswahl

In diesem Projekt werden für den Bewertungsmaßstab die Werte 1 bis 10 verwendet, wobei 1 die schlechteste und 10 die beste Bewertung darstellt. Gleiche Bewertungen sind erlaubt. Die NWA wird durch eine Tabelle dargestellt, in der die Bewertungskriterien gegeneinander abgewogen werden, um eine Gewichtung zu erhalten. Hierbei bedeutet 0, dass das Kriterium weniger wichtig als das Vergleichskriterium ist. 1 bedeutet gleich wichtig und 2 bedeutet wichtiger. Die Gewichtung ergibt sich dann als relativer Anteil der Summe eines Attributes an der Gesamtsumme aller Attribute.

$$w_r = \frac{score(r)}{\sum_{r=1}^{m} score(r)}$$

Die Alternative, die den höchsten Nutzwert $v(a)$ erhält erfüllt die Kriterien am besten und ist der favorisierte Kandidat für die letztendliche Entscheidung. Eine beispielhafte vollständige NWA ist in Abbildung 6.1 zu sehen.

Bewertungskriterien	Kriterium 1	Kriterium 2	Kriterium 3	Kriterium 4	Kriterium 5	Kriterium 6	Kriterium 7	Kriterium 8	Kriterium 9	Kriterium 10	Kriterium 11
Kriterium 1		1	2	2	2	2	2	1	1	0	2
Kriterium 2	1		1	1	2	2	2	1	1	0	1
Kriterium 3	0	1		1	1	1	1	0	0	0	1
Kriterium 4	0	1	1		0	1	1	0	0	1	2
Kriterium 5	0	0	1	2		1	0	0	1	2	1
Kriterium 6	0	0	1	1	1		1	0	0	0	1
Kriterium 7	0	0	1	1	2	1		0	1	0	1
Kriterium 8	1	1	2	2	2	2	2		2	2	2
Kriterium 9	1	1	2	2	1	2	1	0		1	1
Kriterium 10	2	2	2	1	0	2	2	0	1		2
Kriterium 11	0	1	1	0	1	1	1	0	1	0	
Summe	5	8	14	13	12	15	13	2	8	6	14
Gewichtungsfaktor	0,05	0,07	0,13	0,12	0,11	0,14	0,12	0,02	0,07	0,05	0,13

												Summe
Alternative 1	Bewertung	9	10	9	10	10	10	10	10	8	7	9	
	Teilnutzwert	0,41	0,73	1,15	1,18	1,09	1,36	1,18	0,18	0,58	0,38	1,15	**9,39**
Alternative 2	Bewertung	8	5	7	10	10	5	6	7	8	7	9	
	Teilnutzwert	0,36	0,36	0,89	1,18	1,09	0,68	0,71	0,13	0,58	0,38	1,15	**7,52**
Alternative 3	Bewertung	9	8	6	8	10	9	1	10	8	9	10	
	Teilnutzwert	0,41	0,58	0,76	0,95	1,09	1,23	0,12	0,18	0,58	0,49	1,27	**7,66**
Alternative 4	Bewertung	3	5	7	1	10	8	8	7	8	1	1	
	Teilnutzwert	0,14	0,36	0,89	0,12	1,09	1,09	0,95	0,13	0,58	0,05	0,13	**5,53**
Alternative 5	Bewertung	3	10	1	6	10	8	1	7	10	10	1	
	Teilnutzwert	0,14	0,73	0,13	0,71	1,09	1,09	0,12	0,13	0,73	0,55	0,13	**5,53**

Abbildung 6.1: *Beispiel einer Nutzwertanalyse*

6.1.2 Viewframework

Da der Client unseres Projekts eine aufwendige Benutzeroberfläche beinhalten wird, die Graphen, Karten und Benutzersteuerelemente integriert, wird in der Entwicklung aus verschiedenen Gründen auf vorhandene Frameworks zurückgegriffen. In der Regel gibt es für viele Anwendungszwecke, die wir verfolgen, große Open-Source Projekte, die sich mit verschiedensten Problemstellungen befassen. Beispielsweise muss der DOM der Webseite durch Javascript manipuliert werden, um eine dynamische Webseite erzeugen zu können. Für einige Anwendungsfälle ist die vorhandene DOM-API zwar verwendbar, jedoch wird der Quellcode in kürzester Zeit unübersichtlich für komplexere Funktionalitäten. Gleichzeitig existieren bereits Paradigmen, die viele strukturelle Fehler vermeiden

und viel Funktionalität von Anfang an bieten. Aus diesen Gründen wird auf verschiedene Frameworks zurückgegriffen, die in der Evaluation gut zusammen kooperieren, um Arbeitsaufwand zu reduzieren.

6.1.2.1 Web Frameworks

Angular

Bei *Angular* handelt es sich um eine Reihe von JavaScript-Frameworks, die seit 2009 von Google entwickelt wird [143]. Da *Angular* unter der MIT-Lizenz steht und auf Github.com gehostet ist, wird es (Stand Mai 2017, alle Versionen kombiniert) von knapp 2000 Programmierern kontinuierlich verbessert. Unter den JavaScript-Frameworks, die im Front-End in Browsern laufen, ist *Angular* laut den vergebenen Sternen auf Github unter den meist benutzten Frameworks für Single-Page-Webanwendung (SPA) [170]. *Angular* wurde ursprünglich 2009 als reines Javascript-Framework zum Einbinden via Script-Tag verwendet, jedoch gleichzeitig im September 2016 bei Release von Angular2, welches komplett auf TypeScript basiert, abgelöst. Mit der Verwendung von TypeScript als Programmiersprache, geht auch der größte Unterschied einher, denn *Angular2* ist dadurch nicht mehr ohne Compiler zu verwenden.

Bei TypeScript handelt es sich um ein von Microsoft entworfenes Superset von JavaScript, welches auf dem ECMAScript-6-Standard basiert und zusätzlich weitere Konzepte aus bekannten Programmiersprachen, wie Java und C# realisiert [83]. Durch den Compiler wird der TypeScript-Quellcode (je nach Einstellung des Compilers) auf einen gängigen JavaScript-Standard reduziert (üblicher Weise ES2015), um die Browserkompatibilität zu maximieren [195]. Der Programmierer braucht sich also nicht mehr zusätzlich damit befassen, verschiedene Implementierungen für Browser zu schreiben (auch als polyfill Problem bekannt), da die gängigen Zusätze beim Kompilieren implizit im fertigen JavaScript-Code eingepflegt werden.

Bei der Evaluation wird der Fokus auf *Angular2* gelegt, da *AngularJS* (*Angular1*) mittlerweile nicht mehr weiterentwickelt wird. Seit Anfang 2017 gibt es zusätzlich *Angular4*, welches semantisch komplett abwärtskompatibel zu *Angular2* ist, jedoch intern viele Verbesserungen (Ladezeitverkürzung, Reduktion der gepackten Größe) mit sich bringt. Im folgenden wird nur noch von *Angular* gesprochen (gemeint ist *Angular4*), um Missverständnisse zu vermeiden. *Angular* erlaubt dem Programmierer seine Anwendung nach dem Model-View-ViewModel-Pattern zu gestalten [109]. Beim MVVM-Pattern geht es um die Trennung der View und des Modells durch Datenbindung (Data Binding). Angular ist komponentenorientiert und trennt damit das View-Model von der restlichen Implementierung. Sogenannte Services werden in der Regel durch die vom Framework gesteuerte Dependency-Injection in benutzte Komponenten injiziert und sorgen dafür, dass auf Events oder Datenänderungen reagiert werden kann [144]. Solche Services werden verwendet, um zum Beispiel Inhalte über eine Webschnittstelle (z. B. einen RESTFull Service) auszutauschen. Über die beschriebene Grundstruktur hinaus können zusätzlich viele weitere Pakete von *Angular* eingebunden werden, die z. B. das Routing ermöglichen. Zusätzlich gibt es eine große Anzahl an Component-Frameworks, die bereits fertige UI-Komponenten wie Slider, Buttons, Switches oder Date-Picker realisieren, um die Entwicklung zu vereinfachen. *Angular* abstrahiert gleichzeitig das DOM eines Browsers, sodass ggf. durch eine Javascript-Engine (z. B. NodeJS) die Seite auf dem Server gerendert werden kann. Das ist wichtig, um Crawlern von Suchmaschinen normalerweise dynamisch generierte Inhalte übergeben zu können, damit sie kein Skript ausführen müssen und der Ranking-Score nicht beeinträchtigt wird [156].

React

React ist ein Open-Source Javascript Framework, das 2003 von Facebook veröffentlicht wurde und seit dem von Facebook und einer großen Community auf GitHub weiterentwickelt wird [87]. Es ist dazu gedacht User Interfaces zu erstellen. Das Framework ist sehr weit verbreitet und wird von diversen Unternehmen seit längerer Zeit erfolgreich eingesetzt. Dazu gehören unter anderem Netflix, Imgur und Paypal [183].

Der Grundgedanke ist, interaktive Views zu erstellen, die jeweils einen Zustand der Applikation repräsentieren. Dabei werden die Views an die Modelldaten gebunden und *React* kümmert sich um das Rendern und Updaten. Die Einzelteile werden in Komponenten aufgeteilt, die sich selbst verwalten. Diese einfachen Komponenten können zusammengesetzt werden, um komplexe User Interfaces zu erstellen.

React ist unabhängig vom restlichen Stack, kann jedoch auch serverseitig gerendert werden, wenn ein Node Server benutzt wird. Die Designprinzipien von *React* setzen stark auf Modularität. Einzelne Komponenten sollten einfach miteinander kombinierbar sein, auch wenn sie von dritten geschrieben wurden. Es wird außerdem versucht, das Framework möglichst kompakt zu halten und keine zu speziellen Features von Haus aus zu liefern. Allgemein fokussiert sich die Entwicklung von *React* auf Anforderungen, die bei den Produkten von Facebook anfallen, jedoch wird die Community bei Änderungen immer einbezogen [88].

React ist ein weit verbreitetes Framework, mit einer großen Communty, das gute Basisfunktionalität bietet. Es werden jedoch andere Frameworks benötigt, um die fehlenden Funktionen zu realisieren. Des Weiteren bietet *React* kein Two-Way-Binding an, was die Interaktivität der User Interfaces einschränkt.

Vue

Vue.js ist ein JavaScript-Webframework, das 2013 von Evan You initiiert wurde. Aktuell liegt es in Version 2.2.6 vor und wird von einer großen Online-Community weiterentwickelt [281]. Ziel von Vue.js ist das Erstellen von reaktiven Webanwendungen; somit konkurriert Vue.js unter anderem direkt mit Frameworks wie *React* von Facebook oder *Angular* von Google [267].

Ähnlich wie bei *React* sind SPA, die mittels Vue.js geschrieben sind, schnell. Dieser Umstand wird dadurch erreicht, dass sowohl *React* als auch *Vue* ein virtuelles Document Object Model (DOM) nutzen [225]. Vue.js ist in der Lage, die minimale Anzahl an Komponenten zu bestimmen, um die minimale Anzahl an DOM-Manipulationen bei einer Änderung des Zustands der Anwendung zu benötigen.

Ein besonderes Feature von Vue.js ist die hohe Reaktivität. Die Modelle sind einfache JavaScript-Objekte. Sobald diese sich ändern, wird die View ebenfalls aktualisiert [269].

Ebenfalls unterstützt Vue.js Komponenten. Besonders bei sehr großen Anwendungen kann es notwendig sein, die Anwendung in kleine Komponenten aufzuteilen, sodass diese wiederverwendet werden können [268].

Weiter legt Vue.js einen großen Fokus auf Animationen. So sind in Vue.js eine sehr große Anzahl an Möglichkeiten enthalten, Übergangseffekte anzuwenden, sobald der DOM manipuliert wird. Dazu gehören unter anderem CSS-Klassen für Übergänge, mittels JavaScript das DOM manipulieren und beispielsweise integrierte Drittanbieter-JavaScript-Animationsbibliotheken [270].

Vue.js erfreut sich aufgrund seiner Einfachheit bei Entwicklern großer Beliebtheit. Häufig wird hervorgehoben, dass der Einstieg in Vue.js sehr problemlos, der Quelltext einfach zu lesen und zur Einarbeitung die Dokumentation ausreichend ist [237].

Ember

Ember.js ist neben den bereits genannten Webframeworks ein weiteres JavaScript-Webframework zum Erstellen von SPAs. Erschienen ist *Ember.js* im Jahr 2011. Begründet wurde das Projekt unter anderem vom Entwickler Yehuda Katz [79]. Es basiert auf dem Model-View-Viewmodel-Muster und wird vom *Ember* Core Team weiterentwickelt. *Ember.js* ist sehr weit verbreitet und wird von vielen namhaften Seiten aktiv eingesetzt, unter anderem LinkedIn, Twitch.tv und Yahoo [182] [81]. Neben dem erstellen von Webanwendungen ist es ebenfalls möglich, Desktopanwendungen mit *Ember.js* zu realisieren [64].

Ember.js besteht aus fünf Schlüsselkonzepten. Das erste ist *Routes*. Routes sind verantwortlich für den Zustand der Anwendung und somit dafür, welche Komponenten aktuell sichtbar sind. Durch Aktionen des Benutzers wird der Router und somit die Manipulation der URLs ausgelöst. Zusätzlich ist der Router zuständig für das Rendering der Templates, das Bereitstellen der Modelle und das Weiterleiten und Überleiten von einer Route zur nächsten [127]. Das nächste Schlüsselkonzept sind Modelle. Modelle sind Objekte, um die zugrundeliegenden Daten dem Benutzer in der Anwendungsschicht zu präsentieren. Wichtig hierbei ist, dass Modelle persistent sein müssen. Das bedeutet, ein Benutzer, der das Modell manipuliert, möchte, dass Änderungen nach Schließen des Browserfensters weiterhin gültig sind. In *Ember.js* ist es möglich, mithilfe von Bibliotheken auf einfache Weise Daten vom Server im JavaScript Object Notation (JSON)-Format zu bekommen und diese nach erfolgreicher Manipulation wieder zurückzusenden. Außerdem lassen sich auch neue Modelle im Browser erzeugen [80]. Ein weiteres Schlüsselkonzept stellen die Templates dar. Dazu nutzt *Ember.js* die Handlebars Templating Bibliothek. Die Templates beinhalten statischen HTML-Code sowie dynamische Inhalte, die durch zwei geschwungene Klammern gekennzeichnet werden [82]. Weiterhin gehören Komponenten zu den wichtigen Konzepten von *Ember.js*. Komponenten sind eigene HTML-Tags, dessen Verhalten durch JavaScript-Code beschrieben wird [78]. Das letzte wichtige Konzept sind Services. Die Services-Objekte von *Ember.js* entsprechen dem Singleton-Entwurfsmuster. Diese Objekte können Daten beinhalten, die für die gesamte Dauer der Session relevant sind.

Knockout

Bei *Knockout.js* handelt es sich ebenfalls um ein JavaScript-Webframework zur Erstellung von Webclients. Es wurde 2010 von Steve Sanderson, einem Microsoft-Mitarbeiter initiiert. Damit ist es eines der ersten MVVM-Portierungen nach JavaScript. Wie die anderen Frameworks ist es mittels *Knockout* möglich, Teile des User-Interfaces zu aktualisieren, wenn sich das zugrundeliegende Datenmodell ändert. Durch einen Datenbindungsmechanismus ist es möglich, HTML-Teile an das Datenmodell zu binden. Ein weiteres Merkmal von *Knockout* ist die Erweiterbarkeit von bereits vorhandenen Webanwendungen. Es ist somit möglich, *Knockout* auf ein bereits vorhandenes Projekt aufzusetzen, ohne besonders große architektonische Änderungen vorzunehmen. Zudem ist *Knockout.js* mit gerade mal 13kb sehr klein [165]. Besonders hervorzuheben ist, dass *Knockout* keinen Router, keinen AJAX-Support, internen Nachrichtenbus oder Modul Loader mitbringt. Stattdessen legt *Knockout* den Fokus auf die Zwei-Wege-Datenbindung zwischen den JavaScript-Objekten und dem DOM [8].

6.1.2.2 Evaluation

Für die Evaluation wurden fünf Alternativen ausgewählt, die in Unterunterabschnitt 6.1.2.1 beschrieben wurden. Die Bewertung erfolgt mit Hilfe einer Nutzwertanalyse mit folgenden Kriterien:

1. Internationalisierung
2. Validierung (Formvalidierung, ...)
3. Routing
4. Modularisierung (Push Render Pre-Cache Lazy-load)
5. Templating
6. Data-Binding
7. Services (Dependency Injection, ...)
8. Filter
9. Browser Support
10. Compiled Size
11. Geschwindigkeit

Hierbei sind die Kriterien Data-Binding, Routing und Geschwindigkeit besonders wichtig, da eine nachträgliche Implementierung ihrer Funktion sehr aufwändig wäre. Auslagerung dieser Kernfunktionen in andere Frameworks sollte ebenfalls vermieden werden. Weitere wichtige Kriterien, die vor Allem bei der Strukturierung helfen sind Templating und Modularisierung. Die restlichen Kriterien beschreiben Features, die hilfreich beim Erstellen eines Web Frontends sind, die jedoch mit mäßigem Aufwand ausgelagert oder selbst implementiert werden könnten. Das Ergebnis der Gewichtung der Kriterien kann in Abbildung 6.2 eingesehen werden.

Bei den Kernfunktionen zeigte vor Allem *Knockout* Schwächen, da es kein Routing ermöglicht. *React* hingegen schnitt beim Data-Binding am schlechtesten ab, da es lediglich unidirektionale Bindings erlaubt. Die Bewertung der Geschwindigkeit erfolgte über eine Abbildung, die dem langsamsten Framework eine Bewertung von 1 und dem schnellsten eine Bewertung von 10 zuwies. Dazwischen wurde linear interpoliert. Dadurch entsteht eine erzwungene Diskrepanz in der Bewertung der Frameworks. Dies bedeutet jedoch nicht, dass eine 1 als Bewertung der Geschwindigkeit zwangsweise zu langsam ist, sondern nur dass es die langsamste Alternative ist. Die gleiche Abbildung wurde auch für die Bewertung der Compiled-Size verwendet und ist dementsprechend zu interpretieren.

Die Nutzwertanalyse hat ergeben, dass *Angular* mit einem Nutzwert von $9,39$ am besten zu den Kriterien passt. Der optimale Nutzwert ist 10. *React* und *Vue* sind ca. gleichauf auf dem zweiten Platz. *Ember* und *Knockout* teilen sich den dritten Platz. Das Ergebnis lässt sich dadurch erklären, dass *Angular* im Vergleich zu den anderen Frameworks viele Funktionen beinhaltet, die für wichtig befunden wurden. *React* z. B. ist darauf ausgelegt, nur grundlegende Features zu bieten und baut darauf auf, dass Nutzer fehlende Features selber implementieren oder andere Frameworks dafür benutzen. Es ist absichtlich ein lightweight Framework. Die Evaluation war jedoch darauf ausgelegt, ein Framework zu finden, das möglichst komplett ist. *Angular* hat lediglich beim Browser Support, der Compiled-Size und der Geschwindigkeit nicht die beste Bewertung. Doch selbst in diesen Kriterien steht es relativ

gut da. *Vue* ist vom Konzept her am ähnlichsten zu *Angular*, schneidet jedoch in den Kriterien meist schlechter ab. Vor allem die fehlenden Services, z. B. zur Datenbereitstellung, fallen hier ins Gewicht.

In der Gesamtbewertung hat *Angular* 1,74 Punkte Vorsprung vor den beiden Zweitplatzierten und hat damit klar den höchsten Nutzen, gemessen an den Kriterien. Es stellt demnach den besten Kandidaten aus den fünf Alternativen dar, die für die Realisierung der Struktur des Web Frontends ausgewählt wurden.

6.1.2.3 Bewertungskriterien

Zur Bewertung der Frameworks werden folgende Kriterien betrachtet:

- Internationalisierung
- Validierung (Formvalidierung, ...)
- Routing
- Modularisierung (Push Render Pre-Cache Lazy-load)
- Templating
- Data-Binding
- Services (Dependency Injection, ...)
- Filter
- Browser Support
- Compiled Size
- Geschwindigkeit

Die Bewertung erfolgt mit Hilfe einer Produktentscheidungsmatrix.

Bewertungskriterien	Internationalisierung	Validierung	Routing	Modularisierung	Templating	Data-Binding	Services	Filter	Browser-Support	Compiled-Size	Geschwindigkeit
Internationalisierung		1	2	2	2	2	2	1	1	0	2
Validierung	1		1	1	2	2	2	1	1	0	1
Routing	0	1		1	1	1	1	0	0	0	1
Modularisierung	0	1	1		0	1	1	0	0	1	2
Templating	0	0	1	2		1	0	0	1	2	1
Data-Binding	0	0	1	1	1		1	0	0	0	1
Services	0	0	1	1	2	1		0	1	0	1
Filter	1	1	2	2	2	2	2		2	2	2
Browser-Support	1	1	2	2	1	2	1	0		1	1
Compiled-Size	2	2	2	1	0	2	2	0	1		2
Geschwindigkeit	0	1	1	0	1	1	1	0	1	0	
Summe	5	8	14	13	12	15	13	2	8	6	14
Gewichtungsfaktor	0,05	0,07	0,13	0,12	0,11	0,14	0,12	0,02	0,07	0,05	0,13

												Summe
Angular	Bewertung	9	10	9	10	10	10	10	10	8	7	9	
	Teilnutzwert	0,41	0,73	1,15	1,18	1,09	1,36	1,18	0,18	0,58	0,38	1,15	**9,39**
React	Bewertung	8	5	7	10	10	5	6	7	8	7	10	
	Teilnutzwert	0,36	0,36	0,89	1,18	1,09	0,68	0,71	0,13	0,58	0,38	1,27	**7,65**
Vue	Bewertung	9	8	6	8	10	9	1	10	8	9	9	
	Teilnutzwert	0,41	0,58	0,76	0,95	1,09	1,23	0,12	0,18	0,58	0,49	1,15	**7,54**
Ember	Bewertung	3	5	7	1	10	8	8	7	8	1	1	
	Teilnutzwert	0,14	0,36	0,89	0,12	1,09	1,09	0,95	0,13	0,58	0,05	0,13	**5,53**
Knockout	Bewertung	3	10	1	6	10	8	1	7	10	10	1	
	Teilnutzwert	0,14	0,73	0,13	0,71	1,09	1,09	0,12	0,13	0,73	0,55	0,13	**5,53**

Abbildung 6.2: *Nutzwertanalyse verschiedener Frontend Web Frameworks*

6.1.3 Visualisierung

In diesem Abschnitt wird auf die Evaluation der Visualisierungstools eingegangen, die für die Projektgruppe genutzt werden könnten. Dabei werden folgende Bibliotheken miteinander verglichen:

- GoogleCharts
- Chart.js
- D3
- Leaflet
- SmoothieCharts
- OpenLayers
- ngx-charts

6.1.3.1 Bewertungskriterien

Zur Bewertung der Bibliotheken werden die im folgenden erläuterten Kriterien betrachtet. Hierbei wird bereits auf die Vor- und Nachteile der einzelnen Bibliotheken eingegangen und an welcher Stelle sie in der Bewertungsreihenfolge stehen.

Die letztendliche Bewertung erfolgt mit Hilfe einer Produktentscheidungsmatrix.

Performance

Die Performance für die Visualisierungstools wurde aufgrund der Updatefunktion, beziehungsweise Schnelligkeit bei der Visualisierung, wie das Rendering bei neuen Daten stattfindet und ob es bei bestimmten Funktionen zu einer Verlangsamung kommt. Daher steht bezüglich der Performance das Visualisierungstool *D3* sehr weit vorne, denn hier ist kein komplettes Rendering bei neuen Daten nötig und es gibt kein general update Pattern, was bedeutet, dass ein sehr guter Umgang mit einer großen Menge an Daten herrscht [224]. Da bei dem Visualisierungstool *Smoothie Charts* Echtzeitdaten als Streams dargestellt werden, findet hier ebenfalls eine sehr schnelle und gute Visualisierung der erhaltenen Daten statt [272]. Bei *Leaflet* und *OpenLayers* ist die Performance abhängig davon, ob bestimmte Funktionen ausgeführt werden. *OpenLayers* wird langsamer, wenn zu viele Features oder der Browser Firefox verwendet werden. Werden jedoch die neuen Style Instanzen nicht bei jedem Aufruf der Style Funktion neu erzeugt, kann die Performance wieder verbessert werden [40] [48]. Bei *Leaflet* beispielsweise, wird durch zoomen und zu vielen sichtbaren Markern die Performance deutlich schlechter [49] [20]

Bei *Google Charts* findet ein Rendering bei neuen Daten statt, was entsprechend an der Performance bemerkbar wird, wenn schnell viele Daten erhalten und verarbeitet werden. Auch bei *Chart.js* wird durch die Update Funktion, also die Animation zwischen alten und neuen Daten, die Performance verlangsamt, wenn viele Daten erhalten werden. Das passiert insbesondere, wenn mehrere Charts auf einer Seite erstellt werden, so dass die Animationsframes für jedes Diagramm gleichzeitig aufgerufen werden [252] [265]. Da bei *ngx-charts* bei neuen Daten immer alles neu gerendert wird, macht es den Bildaufbau langsam. Da zudem jede Aktualisierung durch ein Flackern auffällt, steht diese Tool auf dem letzten Platz [192].

Datenstromkompatibilität

Die Datenstromkompatibilität wurde danach bewertet, ob es *LineCharts* gibt und wie die Performance des Tools ist. Da *D3* viele verschiedene individualisierbare Möglichkeiten zur Darstellung und eine sehr gute Performance bietet, wird dies als Bestes bewertet. Darauf folgt *Chart.js*, denn es sind zwar gute Graphen verfügbar und individualisierbar, doch die Performance ist schlechter. Ebenso sieht es für *ngx-charts* aus, mit einer etwas schlechteren Performance. Da bei *Google Charts* die Graphen nicht individualisierbar sind, aber die Performance schlechter ist, schließt es sich an. Aufgrund der sehr guten Performance, aber der mangelnden Graphen folgt *Smoothie Charts* [272]. Es folgen *OpenLayers*, da nur Maps dargestellt werden können aber die Performance besser ist als bei *Leaflet*. Somit steht *Leaflet* auf dem letzten Platz.

Geodatenkompatibilität

Zur Darstellung von geografischen Informationen müssen Bibliotheken eine Möglichkeit anbieten, verschiedene Arten von geografischen Daten interpretieren zu können. Dies ist eine zwingende Anforderung an die Bibliothek. Die *Google Charts* API bietet die Möglichkeit geografische Daten darzustellen, verwendet dazu allerdings keine offenen Datenformate [39]. *D3* bietet hingegen die Unterstützung des offenen Geodatenformats GeoJSON. Obwohl *Google Charts* und *D3* die Darstellung grundsätzlich unterstützen, sind die Bibliotheken *Leaflet* und *OpenLayers* genau für diese Anwendung optimiert. Die beiden Bibliotheken unterstützen zudem mehrere verschiedene Geodatentypen.

Offlinefähigkeit

Eine weitere Anforderung an die Bibliothek ist die Offlinefähigkeit, sodass das System autark funktionieren kann. Da fast alle Visualisierungstools als eigene herunterladbare Laufzeitbibliotheken konzipiert wurden und nur die *Google Chart* API, ein angebotener Dienst, der nur über das Internet funktioniert, erhalten alle Bibliotheken außer *Google Charts* die volle Punktzahl.

Grapheninteraktivität

Die Grapheninteraktivität wird danach bewertet, ob und wie leicht sie möglich ist. Bis auf bei *Smoothie Charts* ist bei allen Visualisierungstools die Grapheninteraktivität gegeben [277] [213]. Da die Interaktivität jedoch bei *Google Charts* nur im online-Betrieb verwendet werden kann, fällt dieses Tool in der Bewertung ebenfalls zurück [50].

Browserkompatibilität

Grundsätzlich sind alle betrachteten Bibliotheken mit allen modernen Browsern kompatibel, da alle als Grundvoraussetzung HTML5 haben. Zum Rendern der Diagramme verwenden alle Bibliotheken HTML5 Canvas oder CSS3. Nur *OpenLayers* [212] bietet als einzige Bibliothek einen zweiten Renderer (WebGL) an.

Lizenzbedingungen

Bis auf die *Google Chart* API verwenden alle betrachteten Bibliotheken OpenSource Lizenzen. Daher wurden diese auch mit den Höchstpunkten 8-10 bewertet und die *Google Chart* API erhält durch das reine Anbieten der Schnittstelle 0 Punkte. Die *Google Chart* API [145] ist zwar frei verwendbar, da allerdings der Dienst auf externen Servern läuft und sich die Lizenzierungsbedingungen jederzeit ändern können, ist eine dauerhafte Verwendung nicht garantiert. Außerdem ist es problematisch, dass zur Diagrammerstellung alle Daten an Google übermittelt werden und deren weitere Verwendung dann nicht mehr kontrollierbar ist. Die Abstufungen der Bewertungen auf 8 Punkten beruht auf der Verwendung von unterschiedlichen OpenSource Lizenzen. *Chart.js* [69], *Smoothie Charts* [273] und *ngx-charts* [22] verwenden die MIT-Lizenz, welche die Verwendung, Veröffentlichung (mit Veränderungen), Verkauf und Lizenzierung ausdrücklich erlaubt und erhalten daher 10 Punkte. *Leaflet* [1] und *OpenLayers* [59] verwenden die 2-Klausel BSD-Lizenz, welche mit der MIT-Lizenz kompatibel ist, allerdings nur die Verwendung und die Veröffentlichung (mit Veränderungen) explizit erlaubt. *D3* [38] verwendet die 3-Klausel BSD-Lizenz, welche zudem die Bewerbung des Produktes mit dem Namen

der Autoren verbietet. Mit jeder weiteren Einschränkung hat jede Bibliothek einen Punkt abgezogen bekommen.

Frameworkunterstützung

Die Frameworkunterstützung wurde auf Basis der angebotenen Charts, der Dokumentation, der Größe der Community und die dadurch angebotenen Beispiele bewertet. Da *OpenLayers* sehr gute und viele Beispiele mit optimaler Dokumentation besitzt und zusätzlich vielen Tutorials, steht dies auf dem ersten Platz bezüglich Frameworkunterstützung. *D3* schließt sich auf Grund der sehr großen und aktiven Community, der Vielzahl an Beispielen und verfügbaren vorgefertigten Graphen an. Denn vollständige eigene Charts erstellen ist eine sehr anspruchsvolle Aufgabe. Auch *Leaflet* punktet mit Anwendungsbeispielen und einer sehr guten Dokumentation. In *Chart.js* werden bereits viele Charts angeboten und es können diese mit Hilfe von Tutorials leicht umgesetzt werden. Durch viele angebotene Charts, welche allerdings nur manuell und ohne Hilfsklassen erstellt werden müssen, folgt *Google Charts*. Da *ngx-charts* Informationen nur über Git anbietet und es schwieriger ist Tutorials zu finden, können hier nicht so viele Punkte vergeben werden [205]. *Smoothie Charts* bietet eine intuitive Nutzung, aber wenig Community und Dokumentation [272].

Ästhetik

Die Ästhetik der Graphen, die mit Hilfe der Visualisierungstools erstellt werden können, wurden auf Grundlage der möglichen Designs, verfügbaren Styles und Individualisierbarkeit bewertet. So steht das Tool *D3* durch die Möglichkeit eigene Graphen zu erstellen und die Unterstützung der großen Community durch zur Verfügungstellung vorgefertigter Designs sehr weit vorn. Auch *Leaflet* bietet sehr schöne und durch das simple Design intuitive Graphen/Maps an [32]. Ebenso schön können diese mit OpenLayers dargestellt werden. *ngx-charts* sind simpel aber dennoch schön und übersichtlich. *Google Charts* und *Chart.js* teilen sich einen Platz, da die Darstellung der Graphen mit *Chart.js* ähnlich zu der von *Google Charts* ist. Beide sehen zwar modern aus, sind jedoch simpel und beinhalten hauptsächlich die typischen Google Farben. *Smoothie Charts* hingegen bietet nur ein Design an, wobei dieses sehr schön Datenströme darstellen kann. Außerdem heben sich wegen dem schwarzen Hintergrund dunklere Farben nicht stark ab.

Templateverfügbarkeit

Ein weiteres Kriterium zur Entscheidung einer Visualisierungsbibliothek ist die Verfügbarkeit von Vorlagen, da die reine Verwendung einer Bibliothek im Vordergrund steht. Je mehr Vorlagen vorhanden sind, desto höher ist die Wahrscheinlichkeit, dass eine Vorlage existiert, die den Anforderungen größtenteils entspricht. Mithilfe dieser einfachen Quantifizierung kann eine Entscheidung getroffen werden, ohne zu wissen, welche Anforderungen die Bibliothek erfüllen muss. Die *D3* Bibliothek hat mit Abstand am meisten Vorlagen und erhält damit die Höchstpunktzahl. Darauf folgt *OpenLayers*, welches zwar nicht konkrete Kompositionen von mehreren Eigenschaften präsentiert, aber es bietet für jede Einstellung und Option ein vollständiges Beispiel mit Beschreibung. Alle anderen Bibliotheken besitzen gegenüber den 150 Einstellungsbeispiele von *OpenLayers* nur eine geringe Menge an Vorlagen. Die *Google Charts* API bietet 29 Beispieldiagramme und *ngx-charts* hat eine Informationsseite mit 29 verschiedenen Diagrammen. *Chart.js* bietet mit 33 Anwendungsbeispielen nur geringfügig mehr. *Leaflet* besitzt wie *OpenLayers* keine konkreten Beispiele die mehrere Einstellungen kombinieren, bietet allerdings eine Informationsseite auf der die Einstellungsmöglichkeiten an 14 Beispielen erklärt

werden. Das Schlusslicht mit nur drei Beispielen ist *Smoothie Charts*. Allerdings bietet *Smoothie Charts* als eine minimalistische Bibliothek mit nur einer Diagrammart nicht viele Optionen an, sodass anhand dieser drei Beispiele alles erklärt sein sollte.

Community

Durch die Unterstützung der Community können viele Fehler schon entdeckt worden sein, sodass Kompatibilitätsprobleme mit Standardsystemen nicht mehr existieren sollten. Des Weiteren können durch eine aktive Community auch sporadisch auftretende Fehler schonmal vorgekommen sein und bieten dafür häufig eine Lösung an. Falls dennoch Probleme auftreten, ist eine aktive Community eine weitere Möglichkeit sich Hilfe zu holen. Um hier eine Aussage treffen zu können ohne sich zeitaufwendig mit den Inhalten der Foren der unterschiedlichen Bibliotheken zu beschäftigen, werden einfach die Anzahl der geschriebenen Beiträge in den Foren betrachtet. Auch bei dieser Auswertung ist die *D3* [53] Bibliothek mit knapp 7000 Beiträgen das aktivste Forum. Darauf folgt das *Chart.js* [52] Forum mit etwas mehr als 3000 Beiträgen und *Leaflet* [55] mit etwas weniger als 3000 Beiträgen. Nicht signifikant weniger haben die Bibliotheken *OpenLayers* [57] und die *Google Charts* API [54] mit um die 2500 Beiträgen. Mit großem Abstand folgen die *ngx-charts* [56] Bibliothek mit ca. 300 Beiträgen und *Smoothie Charts* [58] mit ca. 100 Beiträgen. Trotz der unterschiedlichen Größe bzw. Aktivität der Communitys waren keine Beiträge älter als 14 Tage (Stand 02.06.2017), sodass daraus geschlossen werden kann, dass keine dieser Bibliotheken nur einen großen Bestand an alten Beiträgen hat, sondern wirkliche eine noch aktive Community dahintersteht.

6.1.3.2 Nutzwertanalyse

Die Nutzwertanalyse ergab mit den vorgestellten Bewertungskriterien einen Vergleich zwischen der Visualisierungstools *Google Charts*, *Chart.js*, *D3*, *Leaflet*, *OpenLayers* und *ngx-charts*. Das Ergebnis liefert dabei eine Gesamtbewertung der Tools mit folgender Reihenfolge: *D3* mit 8.91, *OpenLayers* mit 8.36, *Leaflet* mit 6.95, *Chart.js* mit 6.68, *ngx-charts* mit 6.11, *Smoothie Charts* mit 5.39 und *Google Charts* mit 3.82. Es folgt daraus, dass *D3* zusammen mit *OpenLayers* genutzt wird, da diese sich durch die Möglichkeit der Erstellung von Maps mit *OpenLayers* gegenseitig ergänzen. Es können mit Unterstützung einer großen Community und einer Vielzahl an Templates schöne, individualisierbare und interaktive Graphen erzeugt werden, wobei zusätzlich die Vorteile der Offlinefähigkeit, Datenstromkompatibilität, Browserkompatibilität genutzt werden und dennoch eine hohe Performance erreicht werden kann.

6.1.4 BuildTools

In einem agilen Projekt wird am Ende eines jeden Sprints ein Produktinkrement ausgeliefert [26]. Das Kompilieren aller Quelldateien mit Ressourcen und Abhängigkeiten, sowie das anschließende Packen ist zeitaufwendig und muss für jedes neue Inkrement wiederholt werden. Gleichzeitig ergibt sich die Problematik, dass sich die Abhängigkeiten im Verlauf des Projekts und sogar innerhalb eines Sprints verändern können. Die Abhängigkeiten müssen auch ggf. aktualisiert werden, um sicher zu gehen, dass immer die aktuellste Version verwendet wird. Zusätzlich muss jeder Softwareentwickler im Projekt dafür sorgen, dass zu jedem Zeitpunkt die korrekten Abhängigkeiten auf seinem Arbeitssystem vorhanden sind. Nachdem das Produkt letztendlich erzeugt worden ist, wäre es sinnvoll das Produkt durch

Bewertungskriterien	Performance	Datenstromkompatibel	Geodatenkompatibel	Offlinefähigkeit	Graphenineraktivität	Browserkompatibilität	Lizenzbedingung	Frameworkunterstützung	Ästhetik	Templateverfügbarkeit	Community
Performance		1	2	1	1	0	1	1	1	2	0
Datenstromkompatibel	1		0	1	0	0	1	0	0	1	0
Geodatenkompatibel	0	2		2	0	0	1	0	0	1	0
Offlinefähigkeit	1	1	0		0	0	1	0	0	0	0
Graphenineraktivität	1	2	2	2		1	1	2	1	2	0
Browserkompatibilität	2	2	2	2	1		2	2	1	2	2
Lizenzbedingung	1	1	1	1	1	0		0	0	0	0
Frameworkunterstützung	1	2	2	2	0	0	2		0	1	1
Ästhetik	1	2	2	2	1	1	2	2		2	2
Templateverfügbarkeit	0	1	1	2	0	0	2	1	0		0
Community	2	2	2	2	2	0	2	1	0	2	
Summe	10	16	14	17	6	2	15	9	3	13	5
Gewichtungsfaktor	0,09	0,15	0,13	0,15	0,05	0,02	0,14	0,08	0,03	0,12	0,05

												Summe
Google Charts	Bewertung	6	6	5	0	5	9	0	7	8	3	4	
	Teilnutzwert	0,55	0,87	0,64	0,00	0,27	0,16	0,00	0,57	0,22	0,35	0,18	**3,82**
ChartJS	Bewertung	6	7	0	10	10	9	10	8	8	3	6	
	Teilnutzwert	0,55	1,02	0,00	1,55	0,55	0,16	1,36	0,65	0,22	0,35	0,27	**6,68**
D3	Bewertung	10	10	5	10	10	9	8	8	10	10	10	
	Teilnutzwert	0,91	1,45	0,64	1,55	0,55	0,16	1,09	0,65	0,27	1,18	0,45	**8,91**
Leaflet	Bewertung	7	2	10	10	10	9	9	8	10	1	5	
	Teilnutzwert	0,64	0,29	1,27	1,55	0,55	0,16	1,23	0,65	0,27	0,12	0,23	**6,95**
Smoothie Charts	Bewertung	10	5	0	10	0	9	10	5	4	1	1	
	Teilnutzwert	0,91	0,73	0,00	1,55	0,00	0,16	1,36	0,41	0,11	0,12	0,05	**5,39**
OpenLayers	Bewertung	9	3	10	10	10	10	9	10	10	9	4	
	Teilnutzwert	0,82	0,44	1,27	1,55	0,55	0,18	1,23	0,82	0,27	1,06	0,18	**8,36**
NGX-Charts	Bewertung	5	6	0	10	10	9	10	6	10	3	1	
	Teilnutzwert	0,45	0,87	0,00	1,55	0,55	0,16	1,36	0,49	0,27	0,35	0,05	**6,11**

Abbildung 6.3: *Nutzwertanalyse Visualisierung*

entsprechende Testfälle zu testen. Diese anfallenden Wartungsarbeiten können durch die Verwendung entsprechender Softwarelösungen, welche als Build Tools bezeichnet werden, automatisiert werden.

Ein Build Tool erzeugt automatisch ein Produkt mit all seinen aktuellen Abhängigkeiten und Ressourcen. Damit ein Build Tool die sogenannte Dependency Resolution durchführen kann, erhält es üblicherweise eine Referenz auf ein Repository, welches eine Sammlung von verschiedenen Bibliotheken darstellt. Durch den Zugriff auf das Repository kann ein Build Tool dann alle benötigten Abhängigkeiten herunterladen bzw. aktualisieren, wenn der nächste Build Prozess angestoßen wird [147, 51].

Für ein Projekt mit Client-Server-Architektur gibt es kein monolitisches Build Tool, das alle Anforderungen, die an diesen Entwicklungsprozess gestellt werden, erfüllen kann. Daher wird eine Kombination verschiedener Tools benötigt, die sich gegenseitig ergänzen und sich zusammen zu einem Build Tool System verknüpfen lassen. Dieses System wird als Build Tool Stack bezeichnet. Typisch für

ein Webprojekt ist ein Frontend Build Tool mit einem einzelnen Repository und einem Backend Build Tool, welches als Controller für das Frontend Build Tool fungiert. Dabei werden die produzierten Artefakte des Frontend Build Tools vom Backend Build Tool in das Produkt eingebettet [162].

Daher werden im Folgenden zunächst bekannte Frontend Build Tools betrachtet, um danach auf Build Tools für ein Java Backend einzugehen. Abschließend werden die verschiedenen Tools anhand einer Nutzwertanalyse evaluiert, um festzustellen, welcher Build Tool Stack für den Entwicklungsprozess geeignet ist.

6.1.4.1 Frontend Tools

Das Frontend Build Tool muss neben der Abhängigkeitsverwaltung im Build Prozess auch die Bandbreitennutzung beachten. Diese kann minimiert werden, indem überflüssige Zeichen wie Zeilenumbrüche und Leerzeichen zur Formatierung und Lesbarkeit im Build entfernt werden, was als Minification bezeichnet wird [65]. Es ergeben sich durchaus noch weitere Optimierungsmöglichkeiten, die im Bezug auf die einzelnen Tools kurz angesprochen werden.

Zunächst wird der *node-package-manager (npm)* vorgestellt, welcher zur Auflösung von Abhängigkeiten im Bereich der JavaScript Entwicklung verwendet werden kann. *npm* wird für viele verschiedene Tools benötigt, beispielsweise für das Dependency-Management-Tool *bower*, den Taskrunner *Grunt* und den Module Bundler *webpack*.

npm

npm ist der zentrale Package-Manager für JavaScript-Anwendungen und wird standardmäßig von NodeJS verwendet. Abhängigkeiten werden über eine JSON-Datei wie in Listing 6.1 in den Zeilen 21-26 spezifiziert. Der Rest des Listings wird von *npm* benutzt um das Modul zu verwalten und Informationen zum Modul anzuzeigen [208].

```
1  {
2    "name": "my_package",
3    "description": "",
4    "version": "1.0.0",
5    "description": "",
6    "main": "index.js",
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1"
9    },
10   "repository": {
11     "type": "git",
12     "url": "https://github.com/###/my_package.git"
13   },
14   "keywords": [],
15   "author": "",
16   "license": "ISC",
17   "bugs": {
18     "url": "https://github.com/###/my_package/issues"
19   },
20   "homepage": "https://github.com/###/my_package",
21   "dependencies": {
22     "my_dep": "^1.0.0"
23   },
24   "devDependencies" : {
```

```
25      "my_test_framework": "^3.1.0"
26    }
27  }
```

Listing 6.1: *Auto-generierte* `package.json` *nach [207]*

bower

bower erweitert *npm* um die Möglichkeiten Bibliotheken außerhalb des *npm*-Repositories in das Projekt einzubinden. Dafür erstellt *bower* eine eigene JSON-Datei, die über ein interaktives Script generiert wird. Ein Beispiel für eine *bower.json*-Datei findet sich in Listing 6.2. *bower* kann über Konsolenbefehle oder direkt in JavaScript benutzt werden [41].

```
1  {
2    "name": "your-name-here",
3    "main": [
4      "js/motion.js",
5      "sass/motion.scss"
6    ],
7    "dependencies": {
8      "get-size": "~1.2.2",
9      "eventEmitter": "~4.2.11"
10   },
11   "devDependencies": {
12     "qunit": "~1.16.0"
13   },
14   "moduleType": [
15     "amd",
16     "globals",
17     "node"
18   ],
19   "authors": [
20     "### <###@###>"
21   ],
22   "license": "MIT",
23   "ignore": [
24     "**/.*",
25     "node_modules",
26     "bower_components",
27     "test",
28     "tests"
29   ],
30   "private": true
31 }
```

Listing 6.2: *Auto-generierte* `bower.json` *nach [187] (gekürzt)*

Grunt

Grunt ist ein Open Source Task Runner für JavaScript Projekte, der seit 2012 entwickelt wird und auf *Node.js* basiert. Es ist möglich eigene Tasks zu definieren oder vordefinierte Tasks über Plugins zu laden. Durch die Kommandozeile kann ein lokaler *Grunt*-Prozess gestartet werden, der automatisch die Gruntfile lädt und ausführt. Hierdurch können mehrere *Grunt*-Prozesse gleichzeitig gestartet werden.

Die Gruntfile beschreibt, welche Tasks definiert sind und welche Plugins geladen werden sollen. Ein Task wird deklarativ durch JavaScript und JSON beschrieben, sodass sich Tasks sehr flexibel gestalten lassen. Durch das umfangreiche Angebot an Plugins, werden bereits Tasks, wie Konkatenieren von Dateien, Validierung, Minification und die Ausführung von Tests, unterstützt [7].

webpack

webpack ist ein konfigurierbarer Open Source Module Bundler für JavaScript, welcher seit 2012 entwickelt und gepflegt wird. Es wird *Node.js* für die Installation benötigt und wird über die Kommandozeile bedient. *webpack* ermöglicht den modularen Aufbau von JavaScript Anwendungen sowohl für Frontend als auch Backend. Hierfür generiert *webpack* einen Abhängigkeitsgraphen, der sich rekursiv aus den Abhängigkeiten der einzelnen Module ergibt. Die Module werden dann in kleineren Bundles zusammengefasst, sodass diese beispielsweise vom Browser dynamisch geladen werden können. Durch das Bundling können verschiedene Optimierungen vorgenommen werden, wie Code Splitting, Minification oder Hot Loading [169, 168].

Es werden neben Modulen, die in purem JavaScript implementiert sind, auch andere unterstützt. Beispielsweise kann über die ES2015 `import` Anweisung, die CommonJS `require()` Anweisung oder über eine AMD `define` Anweisung ein Modul eingebunden werden. Generell kann jedes Modul unabhängig von seiner Programmiersprache oder JavaScript-Dialektes (CoffeeScript, TypeScript, ESNext) von *webpack* verarbeitet werden, wenn ein entsprechender Loader existiert. Loader ermöglichen direkt in den Build Prozess einzugreifen, um beispielsweise ein CoffeeScript-Modul in ein entsprechendes JavaScript-Modul zu übersetzten. Es können auch Komponenten aus Frameworks, wie *React* oder AngularJS integriert werden, die wiederum durch das Hot Loading performant verarbeitet werden können. Neben zahlreichen vordefinierten Loadern, kann *webpack* durch benutzerspezifische Loader erweitert werden [169].

Eine weitere Möglichkeit zusätzliche Features einzubinden, sind Plugins. Plugins ermöglichen die Konfiguration des Build Prozesses auf dem Low-Level, d.h. es können interne Mechanismen, wie die Kompilierung, anpasst werden. Auch das Pluginsystem ist umfangreich aufgestellt [168].

6.1.4.2 Backend Tools

Das Backend Build Tool muss insofern konfigurierbar und erweiterbar sein, dass ein geeignetes Frontend Build Tool mit angemessenem Aufwand eingebettet werden kann. Hierfür ist vor allem entscheidend, ob bereits über entsprechende Plugins die Integration ermöglicht wird oder ob hierfür neue Schnittstellen konfiguriert werden müssen.

Zunächst wird das Build Tool *Ant* und der Dependency Manager vorgestellt, welche von der Apache Software Foundation entwickelt werden. Darauf folgen die Tools *Gradle* und Maven, welche beide in Java implementiert sind.

Ant + Ivy

Ant ist ein Open Source Build Tool, welches seit 2000 entwickelt wird, unter Apache Lizenz steht und vielfach verwendet wird. Durch seine Implementierung in Java ist es plattformunabhängig und kann durch zahlreiche Bibliotheken erweitert werden. *Ant* orientiert sich an dem weit verbreiteten Build Tool make, versucht aber die Probleme, welche make verursacht, zu umgehen. Zur Konfiguration und

Beschreibung des Build Prozesses wird eine Extensible Markup Language (XML)-Datei (Build-File) verwendet. Hier existieren keine formalen Konventionen, das heißt es muss *Ant* genau gesagt werden, wo sich die Quelltextdateien befinden und wo die Artefakte ausgegeben werden sollen. Außerdem ist *Ant* prozedural aufgebaut, das bedeutet, dass der Build Prozess komplett selbst definiert werden muss und vordefinierte Tasks entsprechend an das eigene Projekt angepasst werden müssen. Im Vergleich zu anderen Build Tools soll *Ant* leichter zu erlernen sein, wobei auch komplexere Schritte im Build Prozess benutzerspezifisch umgesetzt werden können [98].

Das Dependency Management wird in *Ant* durch den Dependency-Manager *Ivy* umgesetzt. *Ivy* wird seit 2014 unter der Apache Lizenz entwickelt und kann unter anderem zusammen mit *Gradle*, Grails und Jenkins verwendet werden. Durch eine externe XML-Datei können Abhängigkeiten zu Modulen beschrieben werden. Die Abhängigkeiten werden dann von *Ivy* transitiv aufgelöst und die entsprechenden Ressourcen aus einem Artefakt-Repository geladen. Das Repository kann entweder ein privates selbstverwaltetes sein oder ein öffentliches, wie beispielsweise das *Maven*-Repository. Zu jedem Modul kann ein eigener Conflict Manager eingesetzt werden, der versucht die auftretenden Konflikte zu beheben. Die Erweiterung über Plugins ist ebenfalls möglich, sodass *Ivy* um verschiedene Dependency Resolver und Parser erweitert werden kann [97].

Gradle

Gradle ist ein Build-Tool, das verschiedene Sprachen wie Java, C++ und Python unterstützt. Dabei nutzt *Gradle* bestehende *Maven*- und *Ivy* Repositories, um seine Abhängigkeiten aufzulösen. Anders als *Maven* unterstützt *Gradle* kontinuierliche Builds sowohl über Continuous Integration als auch in der Entwicklung, damit ein manuelles Kompilieren nicht nötig ist [149].

Ein Gradle-Modul wird über Groovy definiert, dabei besitzt *Gradle* in etwa den gleichen Funktionsumfang wie *Maven* und kann auch durch Plugins erweitert werden. Zusätzlich ist es möglich unabhängige Gradle-Projekte miteinander zu verbinden und *Ant*- bzw. *Maven*-Projekte zu integrieren [148].

Listing 6.3 zeigt ein Beispiel für eine `build.gradle`-Datei, die zentrale Datei für ein Gradle-Modul.

```
1  apply plugin: 'java-library'
2
3  repositories {
4      jcenter()
5  }
6
7  dependencies {
8      api 'org.apache.commons:commons-math3:3.6.1'
9
10     implementation 'com.google.guava:guava:21.0'
11
12     testImplementation 'junit:junit:4.12'
13 }
```

Listing 6.3: *Beispiel für ein `build.gradle` nach [146]*

Maven

Maven ist ein XML-basiertes Build Tool für Java-Projekte. Es schreibt eine feste Projektstruktur vor, die sich aus den Best Practices in der Java-Community ergeben hat. Dabei verfügt *Maven* über ein eigenes Repository-Format mit einem zentralen Repository und der Möglichkeit eigene Repositories zu definieren. Dadurch besitzt *Maven* eine sehr gute Unterstützung zur Auflösung von Abhängigkeiten und die Möglichkeit fremde Bibliotheken einzubinden [94].

Der Build-Prozess von *Maven* wird durch *goals* definiert. Dabei gibt es vordefinierte *goals* zum Kompilieren, Testen und Installieren des Projekts in ein *Maven*-Repository. Dieser Prozess kann durch eigene *goals* erweitert werden. Zusätzlich ist der Prozess durch Plugins erweiterbar, die ein Build auf Basis bestimmter Frameworks ermöglichen [95].

In Listing 6.4 ist ein Beispiel für ein *Maven*-Project-Object-Model (*pom*) für ein einzelnes Java-Projekt, das ein eigenes Profil definiert.

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.example</groupId>
  <artifactId>example-project</artifactId>
        <version>1.0.0-SNAPSHOT</version>
  <build>
   <pluginManagement>
       <plugins>
         <plugin>
           <artifactId>maven-antrun-plugin</artifactId>
           <version>1.3</version>
         </plugin>
       </plugins>
     </pluginManagement>
  </build>
  <profiles>
    <profile>
      <id>release-profile</id>
      <build>
        <plugins>
          <plugin>
            <inherited>true</inherited>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-source-plugin</artifactId>
            <executions>
              <execution>
                <id>attach-sources</id>
                <goals>
                  <goal>jar</goal>
                </goals>
              </execution>
            </executions>
          </plugin>
        </plugins>
      </build>
    </profile>
  </profiles>
</project>
```

Listing 6.4: *Beispiel* `pom.xml` *nach [96] (bearbeitet und gekürzt)*

In *Maven* ist es möglich eine Projekthierarchie aufzubauen, wodurch ein Projekt in viele kleine Projekte aufgeteilt werden kann, die über ein übergeordnetes Projekt zusammen gebaut werden können [93].

Durch das Pluginsystem von *Maven* ist es möglich Ant-Tasks direkt über *Maven* auszuführen und Dokumentation in Form von Webseiten oder durch ein PDF automatisch zu generieren. Dazu kann *Maven* mit einem Versionskontrollsystem integriert werden und ist in allen bekannten Java-IDEs integrierbar [92].

6.1.4.3 Evaluation

Bewertungskriterien	Interoperabilität	Dependency Resolution	Multi-Build	Einrichtungkomplexität	Wartungsaufwand	Multi-Projekt-Unterstützung	Automatisierte Test	Deployment	Jenkins-Kompatibilität	IDE-Integration	Community & Dokumentation	Performanz	Funktionalität
Interoperabilität		0	0	0	0	0	0	0	0	0	0	0	2
Dependency Resolution	2		1	0	2	0	1	0	0	0	1	0	2
Multi-Build	2	1		0	1	0	1	0	1	0	1	1	2
Einrichtungkomplexität	2	2	2		2	2	2	0	1	0	2	2	2
Wartungsaufwand	2	0	1	0		1	2	0	1	0	2	2	2
Multi-Projekt-Unterstützung	2	2	2	0	1		2	0	0	1	2	2	2
Automatisierte Test	2	1	1	0	0	0		0	0	0	1	0	2
Deployment	2	2	2	2	2	2	2		2	2	2	1	2
Jenkins-Kompatibilität	2	2	1	1	1	2	2	0		0	2	0	2
IDE-Integration	2	2	2	2	2	1	2	0	2		2	2	2
Community & Dokumentation	2	1	1	0	0	0	1	0	0	0		1	2
Performanz	2	2	1	0	0	0	2	1	2	0	1		2
Funktionalität	0	0	0	0	0	0	0	0	0	0	0	0	
Summe	22	15	14	5	11	8	17	1	9	3	16	11	24
Gewichtungsfaktor	0,14	0,10	0,09	0,03	0,07	0,05	0,11	0,01	0,06	0,02	0,10	0,07	0,15

														Summe
Maven grunt	Bewertung	8	8	10	4	9	5	10	8	10	10	5	5	6	
	Teilnutzwert	1,13	0,77	0,90	0,13	0,63	0,26	1,09	0,05	0,58	0,19	0,51	0,35	0,92	**7,51**
Maven webpack	Bewertung	8	8	10	4	8	7	10	8	10	10	4	5	10	
	Teilnutzwert	1,13	0,77	0,90	0,13	0,56	0,36	1,09	0,05	0,58	0,19	0,41	0,35	1,54	**8,06**
Gradle grunt	Bewertung	7	10	10	4	9	8	10	8	5	10	7	10	6	
	Teilnutzwert	0,99	0,96	0,90	0,13	0,63	0,41	1,09	0,05	0,29	0,19	0,72	0,71	0,92	**7,99**
Gradle webpack	Bewertung	8	10	10	4	8	10	10	8	5	10	6	10	10	
	Teilnutzwert	1,13	0,96	0,90	0,13	0,56	0,51	1,09	0,05	0,29	0,19	0,62	0,71	1,54	**8,67**
Ant grunt	Bewertung	6	8	10	4	9	5	10	8	10	10	10	8	6	
	Teilnutzwert	0,85	0,77	0,90	0,13	0,63	0,26	1,09	0,05	0,58	0,19	1,03	0,56	0,92	**7,96**
Ant webpack	Bewertung	7	8	10	4	8	7	10	8	10	10	9	8	10	
	Teilnutzwert	0,99	0,77	0,90	0,13	0,56	0,36	1,09	0,05	0,58	0,19	0,92	0,56	1,54	**8,64**

Abbildung 6.4: *Nutzwertanalyse Build Tools*

Die Evaluation besteht aus der Nutzwertanalyse der sechs unterschiedlichen Build Tool Stacks, die sich aus einem Backend Build Tool ergeben, die wiederum mit *Grunt* bzw. *Grunt* und *webpack* gepaart werden. Die Nutzwertanalyse wird durch die Tabelle in Abbildung 6.4 beschrieben. Sie setzt sich aus insgesamt 13 Bewertungskriterien zusammen:

- **Interoperabilität** → Plattformabhängigkeit und Kompatibilität zwischen den Tools
- **Dependency Resolution** → Unterstützung für automatisches Management von Abhängigkeiten
- **Multi-Build** → Unterschiedlich konfigurierte Builds können erzeugt werden
- **Einrichtungskomplexität** → Einarbeitung in die einzelnen Tools und initialer Konfigurationsaufwand
- **Wartungsaufwand** → Frequenz und Umfang von Wartungsarbeiten der Tools
- **Multi-Projektunterstützung** → Es können mehrere Projekte verwaltet werden
- **Automatisierte Tests** → Unterstützung für automatisierte Tests nach dem Build
- **Deployment** → Nach dem Build kann ein automatisches Deployment erfolgen
- **Jenkins-Kompatibilität** → Integration in das bestehende Jenkins-System ist möglich
- **IDE-Integration** → Tools bieten eine Integration in gängige IDEs
- **Community & Dokumentation** → Größe der Community und Qualität der Dokumentation
- **Performanz** → Tools erfüllen Performanzanforderungen
- **Funktionalität** → Funktionsumfang und Qualität der Funktionen

Die Nutzwertanalyse hat ergeben, dass die beiden Stacks mit *Gradle* die höchsten Bewertungen erreicht haben. Dies liegt vor allem daran, dass *Gradle* in verschiedenen Bereichen unabhängig von der Paarung mit *Grunt* und *webpack* ausschlaggebend die Bewertung beeinflusst. Besonders im Bereich Dependency Resolution funktioniert *Gradle* besser im Vergleich zu *Maven* und *Ant*. Auch im Bereich Performanz und Multi-Projektunterstützung liegt *Gradle* vorne. Hingegen erfordert die Jenkins-Intergration von *Gradle* einen deutlichen Mehraufwand als es mit *Maven* oder *Ant* der Fall wäre. Die Bewertungskriterien IDE-Integration, automatisierte Tests sowie Multi-Build ergeben für alle drei Java Build Tools die gleichen Werte.

Werde die beiden Stacks GW und AW betrachtet, fällt auf, dass AW eine geringe Verbesserung erzielt als GW. Zum Teil liegt dies daran, dass mit jedem zusätzlichen Framework die Einrichtungskomplexität und der Wartungsaufwand steigt. Außerdem ergeben sich Abzüge für die Bewertung für die Kriterien Community und Dokumentation sowie Performanz. Ersteres ist dadurch zu erklären, dass die Dokumentation von *webpack* nicht ausführlich genug ist. Zweitens ermöglicht *webpack* Verarbeitungsschritte im Build Prozess des Frontends, welche weder durch *Grunt* noch *Gradle* bedient werden können. Deshalb wird das Bewertungskriterium Funktionalität in Verbindung mit *webpack* grundsätzlich höher bewertet.

6.1.5 Datenbanksysteme

Zur Realisierung des Projekts ist auch die Speicherung von Daten notwendig. Hierzu können neben Daten für eine Benutzerverwaltung auch Daten gehören, die mit Odysseus verarbeitet wurden. Auch die Speicherung sehr komplexer Datenstrukturen kann sich im Laufe der Projektgruppe als notwendig erweisen. An die Datenbank werden daher diverse Anforderungen gestellt, während es gleichzeitig eine sehr große Auswahl möglicher Datenbankmanagementsysteme (DBMS) gibt. Daher wurde innerhalb

der Projektgruppe zunächst eine Vorauswahl an verschiedenen Datenbanksystemen getroffen, welche im Folgenden vorgestellt und anschließend mittels einer Nutzwertanalyse bewertet werden sollen.

6.1.5.1 Datenbanken

Bei der Vorauswahl möglicher Datenbanksysteme wurden Vertreter verschiedener Datenbankprinzipien gewählt. So wurden mit *MongoDB*, *CouchDB* und *Redis* drei NoSQL-Datenbanken [1] und mit *MySQL*, *MariaDB* und *Oracle Express* drei relationale Datenbanken ausgewählt.

MongoDB ist eine seit 2007 entwickelte und 2009 erstmals veröffentlichte Datenbank, welche zu den dokumentenorientierten NoSQL-Datenbanken gezählt wird. Der Name ist vom englischen *hu**mongo**us* („gigantisch“/„riesig“) abgeleitet und gleichzeitig namensgebend für das entwickelnde Unternehmen „*MongoDB, Inc.*“. Wesentliche Ziele bei der Entwicklung von *MongoDB* sind insbesondere gute Performance bei der Verwaltung großer Datenmengen sowie eine einfache Skalierbarkeit der Datenbank. *MongoDB* wird unter anderem von der New York Times, Foursquare und SourceForge eingesetzt und gilt aktuell als eine der meistverwendeten dokumentenorientierten Datenbanken, welche unter Open-Source-Lizenz verfügbar ist [199].

Ein weiteres dokumentenorientiertes DBMS ist CouchDB , welches unter der Apache-Lizenz in der Version 2.0 vertrieben wird. *CouchDB* ist dabei als Datenbank mit einem hohen Grad an Skalierbarkeit und Leistung entwickelt worden, welche insbesondere auf verteilten Systemen laufen soll. Hieran orientiert sich auch der Name, welcher für ***C**luster **of** **u**nreliable **c**ommodity **h**ardware **D**ata **B**ase* („Datenbank auf einem Cluster aus unzuverlässiger Standard-Hardware“) steht. *CouchDB* wird unter anderem von der Credit Suisse Bank und der BBC eingesetzt [101].

Das dritte Datenbanksystem, welches dem NoSQL-Prinzip folgt, ist *Redis* . *Redis* unterscheidet sich dabei jedoch von *MongoDB* und *CouchDB* indem es nicht dem dokumentenorientierten Ansatz folgt, sondern auf einer Schlüssel-Wert-Datenstruktur basiert. Zudem ist *Redis* eine In-Memory-Datenbank. Das heißt, die Daten werden im Arbeitsspeicher vorgehalten und nur bei Bedarf auf Festplatten gespeichert. *Redis* erschien erstmals 2009 und wurde von Salvatore Sanfilippo insbesondere mit dem Ziel großer Leistungsfähigkeit beim Lesen und Schreiben von Daten entwickelt und unter der BSD-Lizenz veröffentlicht [230].

Als Gegenpart zu den NoSQL-Datenbanken können die relationalen Datenbanken gesehen werden. *MySQL* ist einer der bekanntesten Vertreter der relationalen DBMS. Erstmals wurde *MySQL* 1994 vom Unternehmen *MySQL* AB entwickelt, welches 2008 von Sun Microsystems übernommen wurde. Da Sun Microsystems 2010 von Oracle übernommen wurde, gehört auch das *MySQL*-Datenbanksystem heute zu Oracle und wird unter der GPL sowie einer kommerziellen Lizenz vertrieben. Letztere beinhaltet insbesondere Support durch Oracle selbst. *MySQL* kann dabei als Quasi-Standard gesehen werden und wird unter anderem von YouTube, Twitter, Facebook und Google verwendet [214].

Seit 2009 gibt es mit *MariaDB* einen von Michael Widenius initiierten Fork von *MySQL*. Widenius gilt als Hauptautor von *MySQL* und begründete die Entwicklung von *MariaDB* insbesondere mit der Übernahme von *MySQL* durch Oracle. Inzwischen wird *MariaDB* von der *MariaDB* Corporation weiterentwickelt und unter der GPL vertrieben. Zudem wurde *MariaDB* inzwischen um viele Funktionen erweitert und die Entwicklung auf Performance und Flexibilität konzentriert. Auch bezüglich Sicherheit und Skalierbarkeit hat sich *MariaDB* inzwischen deutlich von *MySQL* abgesetzt. Die Entwicklung

[1] NoSQL = not only SQL

wird dabei unter anderem von Google sowie der EU finanziell unterstützt und neben Google, Mozilla und Slackware wird *MariaDB* von Wikipedia verwendet und ersetzt inzwischen *MySQL* in mehreren Linux-Distributionen [99].

Das dritte betrachtete relationale DBMS ist die Express Edition der *Oracle Datenbank* (kurz: *Oracle Express*). Während die Standard Edition als Marktführer unter den Datenbanksystemen gilt, unterliegt die Express Edition verschiedenen Einschränkungen. Dafür ist die *Oracle Express* kostenfrei und richtet sich insbesondere an Entwickler und Bildungseinrichtungen, welche die Datenbank zu Testzwecken verwenden können. Zudem kann *Oracle Express* ohne große Anpassungen auf eine kostenpflichtige Version umgestellt werden [215].

6.1.5.2 Bewertungskriterien

Die Nutzwertanalyse der zur Auswahl stehenden Datenbanken erfolgt anhand folgender Kriterien: Kosten, Betriebssystemunterstützung, Verbreitungsgrad, Importfunktionen, Skalierbarkeit, Sicherheit, PG-relevante Features, Reifegrad, Limitierungen, Performance und Transaktionssicherheit. Diese orientieren sich insbesondere auch an den Empfehlungen zur Auswahl einer Datenbank-Software des Bundesamtes für Sicherheit in der Informationstechnik (BSI), wobei für diese Analyse einige Punkte zusammengefasst wurden. Im Folgenden erfolgt daher zunächst eine Erläuterung der hier verwendeten Kriterien und der zugrundeliegenden Annahmen.

Im Rahmen der Projektgruppe stehen keine finanziellen Mittel für den Kauf von Softwarelizenzen zur Verfügung. Daher sollen kostenfreie, möglichst als Open Source-Software veröffentlichte Datenbanksysteme eingesetzt werden.

Damit das von der Projektgruppe entwickelte Softwaresystem einfach genutzt werden kann, sollen die zugrundeliegenden Datenbanksysteme auf möglichst vielen Betriebssystemen installiert und ausgeführt werden können. Zudem ist es bei großer Betriebssystemunterstützung für die Projektgruppenmitglieder möglich, die Datenbanksysteme für eine lokale Entwicklungsumgebung auf dem eigenen Computer zu installieren. Überlicherweise werden von den Projektgruppenmitgliedern die Betriebssysteme Windows 10, macOS X und Linux-Systeme eingesetzt. Dabei ist zu beachten, dass der Installationsaufwand möglichst gering bleibt.

Damit bei möglichen Problemen das Internet oder die Bibliothek als Quelle für deren Lösung herangezogen werden können, sollen die eingesetzten Datenbanksysteme eine große Nutzergemeinschaft aufweisen. Zudem erhöht sich hierdurch die Wahrscheinlichkeit zu diversen Problemen gute Tutorials, Dokumentationen und Beispielprogramme zu finden. Zudem sollten die Datenbanksysteme von großen, etablierten Softwareherstellern genutzt werden, die oftmals einen wichtigen Beitrag zur Weiterentwicklung der Datenbanksysteme leisten. Dies kann entweder durch aktive Mitwirkung an der Implementierung oder durch das Finden und Melden von Softwarefehlern und -schwachstellen erfolgen.

Um Daten auf einfache Art und Weise zwischen verschiedenen Systemen migrieren zu können, sollen die Datenbanksysteme Standardformate wie z. B. XML, Comma-separated values (CSV), Tab-separated values (TSV) oder JSON importieren bzw. exportieren können. Somit kann der Aufwand für die Implementierung eigener Importfunktionalitäten reduziert oder gänzlich vermieden werden. Auch die Unterstützung proprietärer Formate, wie z. B. Microsoft-Excel-Dokumente kann einen Vorteil bieten. Auch zusätzliche, frei verfügbare Programme können einen Datenimport oder -export erleichtern.

Daher wird das Vorhandensein solcher Programme als positiv für das Kriterium Importfunktionen bewertet.

Damit das Gesamtsystem zukünftigen Anforderungen genügen kann, sollte auch die darunterliegende Persistenzschicht mit den Anforderungen wachsen können. Für die Skalierbarkeit können Datenbankabfragen beispielsweise über mehrere Knoten verteilt werden. Dies sollte möglichst automatisch durch das Datenbanksystem selbst erfolgen. Auch die Möglichkeit des Hinzufügens zusätzlicher Datenbankknoten zum Zwecke der Datenbankpartitionierung ist bei der Bewertung der Skalierbarkeit zu berücksichtigen. Analog zu Datenimport- und Datenexport-Tools, wird die Verfügbarkeit von frei erhältlichen Hilfsprogrammen zur Skalierung als positiv bewertet. Ein Beispiel für ein solches Programm ist der Datenbankproxy MaxScale des Herstellers der Datenbank MariaDB.

Das zu entwickelnde System soll Energieverbrauchsdaten verarbeiten und die Verarbeitungsergebnisse speichern können. Diese Daten, insbesondere viertelstündige Smart Meter Daten, lassen prinzipiell Rückschlüsse auf die privaten Lebensumstände von Endverbrauchern zu. Diese Daten sind besonders vor der unberechtigten Einsichtnahme oder Veränderung, aber auch gegen Verlust zu sichern. Auch wenn im Rahmen der Projektgruppe zum jetzigen Zeitpunkt (Stand 20.05.2017) nur simulierte Daten verarbeitet werden sollen, ist es wichtig, dass die Datenbanksysteme die Sicherheit der in ihnen gespeicherten Daten gewährleisten können. Die Daten sind dabei sowohl bei der Übertragung (z. B. mittels TLS-Protokoll), aber auch während der dauerhaften Speicherung (z. B. durch symmetrische oder asymmetrische Verschlüsselung) zu schützen. Der Zugriff auf die Datenbanken, etwa zur Datenabfrage oder Konfiguration der Datenbank, sollte nur befugten Personen gestattet sein. Hierbei können passwort- oder zertifikatsbasierte Verfahren angewandt werden. Auch die Anbindung an andere Authentifizierungsprovider, wie z. B. Kerberos, LDAP oder RADIUS sind oft hilfreiche Sicherheitsfeatures.

Zu den für die Projektgruppe relevanten Features gehört insbesondere die Möglichkeit zur Anbindung an Odysseus. Auch der Reifegrad ist in gewissem Maße von Bedeutung, da hier unter Anderem Update-Zyklen und die Wahrscheinlichkeit von Bugs in der Datenbank-Software untersucht werden sollen.

Ebenfalls bedeutsam sind mögliche Limitierungen, von welchen es natürlich möglichst wenige geben soll. Im Umfeld von Datenbanken noch wichtiger ist jedoch die Performance: gerade im Zusammenspiel mit Odysseus muss ein DBMS in der Lage sein, auch in kurzer Zeit große Datenmengen zu speichern oder viele Datenabfragen zu beantworten.

Das in dieser Analyse am stärksten gewichtete Kriterium ist jedoch die Transaktionssicherheit. Da hier davon ausgegangen wird, dass ggf. auch (analysierte) Smart Meter Daten gespeichert werden sollen, ist ein Datenverlust in jedem Fall zu vermeiden. Auch das Auslesen veralteter Daten durch gleichzeitiges Lesen und Schreiben ist in jedem Fall zu verhindern. Mit der Transaktionssicherheit soll daher in dieser Nutzwertanalyse untersucht werden, ob die Datenbanken die ACID-Kriterien [2] einhalten oder welche anderen Mechanismen es zur Transaktionssicherheit gibt.

6.1.5.3 Evaluation

Die Bewertung der hier analysierten DBMS führt zur in Abbildung 6.5 dargestellten Nutzwertmatrix. In den folgenden Abschnitten werden die Bewertungen genauer erläutert, bevor in Unterunterabschnitt 6.1.5.4 schließlich ein Fazit der Analyse und somit eine Empfehlung gegeben wird.

[2] ACID für*atomicity, consistency, isolation, durability*

Bewertungskriterien	Kosten	OS Unterstützung	Verbreitungsgrad	Importfunktionen	Skalierbarkeit	Sicherheit	PG-relevante Features	Reifegrad	Limitierungen	Performance	Transaktionssicherheit
Kosten		0	0	0	0	0	1	1	0	1	2
OS Unterstützung	2		0	0	1	0	0	0	0	1	2
Verbreitungsgrad	2	2		0	2	2	2	1	2	2	2
Importfunktionen	2	2	2		1	2	2	1	2	2	2
Skalierbarkeit	2	1	0	1		1	1	0	0	1	2
Sicherheit	2	2	0	0	1		0	0	0	1	1
PG-relevante Features	1	2	0	0	1	2		0	0	1	1
Reifegrad	1	2	1	1	2	2	2		2	2	2
Limitierungen	2	2	0	0	2	2	2	0		1	2
Performance	1	1	0	0	1	1	1	0	1		1
Transaktionssicherheit	0	0	0	0	0	1	1	0	0	1	
Summe	15	14	3	2	11	13	12	3	7	13	17
Gewichtungsfaktor	0,14	0,13	0,03	0,02	0,10	0,12	0,11	0,03	0,06	0,12	0,15

												Summe
MongoDB	Bewertung	10	10	8	3	10	5	3	4	7	8	5	
	Teilnutzwert	1,36	1,27	0,22	0,05	1,00	0,59	0,33	0,11	0,45	0,95	0,77	**7,10**
Oracle Express	Bewertung	10	9	10	5	5	5	10	8	4	3	10	
	Teilnutzwert	1,36	1,15	0,27	0,09	0,50	0,59	1,09	0,22	0,25	0,35	1,55	**7,43**
MySQL	Bewertung	10	10	9	3	6	5	10	8	6	5	10	
	Teilnutzwert	1,36	1,27	0,25	0,05	0,60	0,59	1,09	0,22	0,38	0,59	1,55	**7,95**
MariaDB	Bewertung	10	10	6	3	8	6	8	8	6	7	10	
	Teilnutzwert	1,36	1,27	0,16	0,05	0,80	0,71	0,87	0,22	0,38	0,83	1,55	**8,21**
CouchDB	Bewertung	10	10	5	1	10	5	3	3	7	8	5	
	Teilnutzwert	1,36	1,27	0,14	0,02	1,00	0,59	0,33	0,08	0,45	0,95	0,77	**6,95**
Redis	Bewertung	10	10	7	1	10	1	2	2	3	10	4	
	Teilnutzwert	1,36	1,27	0,19	0,02	1,00	0,12	0,22	0,05	0,19	1,18	0,62	**6,23**

Abbildung 6.5: *Nutzwertanalyse der Datenbankmanagementsysteme*

Kosten

Da alle betrachteten Datenbanken für die Projektgruppe kostenfrei und teilweise sogar als Open Source-Software zur Verfügung stehen, wurde dieses Kriterium für alle betrachteten Datenbanksysteme mit zehn Punkten bewertet.

Betriebssystemunterstützung

MongoDB erhält eine Bewertung von zehn Punkten, da es alle derzeit (Stand 23.05.2017) erhältlichen Windows-Versionen ab Windows Vista sowie Linux, OS X ab 10.7, Solaris und FreeBSD unterstützt. Auch *CouchDB* unterstützt diese Betriebssysteme, daher wird auch *CouchDB* hier mit zehn Punkten bewertet. *Redis*, *MariaDB* und *MySQL* erhalten ebenfalls zehn Punkte für dieses Kriterium, da sie ebenfalls auf allen unter diesem Abschnitt genannten Betriebssystemen lauffähig sind. *Oracle Express* erhält nur neun Punkte da die Installation unter FreeBSD im Vergleich zu anderen Betriebssystemen wesentlich aufwändiger erscheint.

Verbreitungsgrad

Für die Bewertung des Verbreitungsgrades wurde die Bewertung von der Internetseite „www.db-engines.com“ verwendet. Laut Auskunft auf derselben Internetseite, wird das dort aufgeführte Ranking wie folgt ermittelt (vgl. [151]):

- Anzahl der Nennungen auf Websites über Messungen der Treffer in Suchmaschinen (Google, Bing, Yandex), Suchbegriffe: <Systemname> (z. B. Redis) + „Database“ also z. B. „*Redis* Database“
- Allgemeines Interesse am System gemessen über Häufigkeit der Suchen in Google Trends
- Häufigkeit von technischen Diskussionen über das System gemessen über Anzahl der Fragen in bekannten IT-relevanten Communities wie Stackoverflow oder DBA Stack Exchange
- Anzahl an Job-Angeboten, in denen das System genannt wird, über Job-Suchmaschinen „Indeed“ und „Simply Hired“.
- Anzahl an Profilen in sozialen Netzwerken, in denen das System aufgeführt wird (LinkedIn und Upwork)
- Relevanz in sozialen Netzwerken: Anzahl von Twitter Tweets, in denen das System genannt wird

Für das auf „www.db-ranking.com“ bestplatzierte Datenbanksystem (*Oracle*) werden zehn Punkt vergeben, das nächstbestplatzierte Datenbanksystem erhält neun Punkte (*MySQL*). Acht Punkte erhält *MongoDB*, gefolgt von *Redis* mit sieben Punkte. *MariaDB* erhält sechs Punkte und *CouchDB* lediglich fünf Punkte.

Importfunktionen

Für jedes unterstützte, nicht-proprietäre Importformat wird ein Punkt vergeben. Für jedes unterstützte proprietäre Format eines Konkurrenzunternehmens werden zwei Punkte vergeben, da Export und Import solcher Dateiformate auf andere Weise oft gar nicht oder nur mit hohem Aufwand möglich sind. Auch die alternative Möglichkeit des Datenimports über ein kostenfreies Drittanbietertool wird mit einem Punkt bewertet.

MongoDB unterstützt CSV, TSV und JSON und erhält daher drei Punkte. *Oracle* erhält für die Unterstützung von TSV, CSV und XML drei Punkte, sowie zwei Punkte für die Unterstützung von Excel-Spreadsheets (proprietäres Format des Konkurrenten Microsoft). *MySQL* und *MariaDB* können nativ CSV und JSON importieren, erhalten zusätzlich jeweils einen Punkt für die Importmöglichkeit über das Tool „PHPMyAdmin“, wodurch die Bewertungen jeweils mit drei Punkten erfolgen. *CouchDB* unterstützt nur den Import von JSON, während *Redis* nur den Import von Key-Value-Paaren unterstützt. Daher erhalten diese beiden Datenbanksysteme nur jeweils einen Punkt für die Importfunktionalität.

Skalierbarkeit

MongoDB, *CouchDB* und *Redis* sind für ihre einfache Skalierbarkeit bekannt. Es gibt viel Dokumentation zu diesem Aspekt, weshalb dieses Kriterium jeweils mit zehn Punkten bewertet wird. Die Skalierbarkeitsfunktionen von *Oracle* sind in der Express-Edition nur in sehr begrenztem Umfang verfügbar, weshalb hier lediglich fünf Punkte erreicht werden. Auch bei *MySQL* sind die Möglichkeiten der Skalierbarkeit nur in der kostenpflichtigen Enterprise-Edition verfügbar. Allerdings ist auch die kostenfreie Open Source Community-Edition über den Datenbankproxy „MaxScale“ skalierbar weshalb

diese hier sechs Punkte erhält. MaxScale wird von dem Hersteller des Datenbanksystems MariaDB entwickelt. Daher ist es sehr gut für die Skalierung von MariaDB-Instazen geeignet. *MariaDB* erhält somit eine Bewertung von acht Punkten.

Sicherheit

Die meisten der betrachteten Datenbanksysteme unterstützen gängige Sicherheitsfunktionen. *MongoDB*, *Oracle Express*, *MySQL* und *CouchDB* unterstützten die Authentifizierung durch Benutzernamen und Passwort, X.509 Zertifikate, rollenbasierte Zugriffskontrolle und Verschlüsselungsfunktionen. Für jeden dieser fünf Aspekte erhalten die Datenbanksysteme jeweils einen Punkt. Somit werden *Oracle Express*, *MongoDB*, *MySQL* und *CouchDB* mit fünf Punkten bewertet. *MariaDB* erhält sechs Punkte, da die Authentifizierung über einen Lightweight Directory Access Protocol (LDAP)-Server nativ unterstützt wird. *Redis* erhält lediglich einen Punkt, da die Authentifizierung nur über Benutzername und Passwort möglich ist und Verschlüsselungsmechanismen selbst implementiert werden müssen.

PG-relevante Features

Die Analyse der für die Projektgruppe relevanten Features bezieht sich insbesondere auf die Möglichkeiten der Verwendung der Datenbank im Zusammenhang mit Odysseus. Da für *MySQL* und die *Oracle* Datenbank bereits entsprechende Plugins in Odysseus zur Verfügung stehen, erhalten beide hier die volle Punktzahl. Auch *MariaDB* wird mit acht Punkten noch gut bewertet, da sie als relationale Datenbank mit verfügbarem JDBC-Treiber mit nur geringem Aufwand mit Odysseus verbunden werden kann.

MongoDB und *CouchDB* erhalten nur drei Punkte, obwohl es auch für sie Java-Treiber gibt, welche eine Integration möglich machen würden. Jedoch unterliegen sie als dokumentenorientierte Datenbanken einem grundsätzlich anderen Konzept, welches die Integration erschweren kann. Darüber hinaus müssen bei diesen Datenbanken Operationen wie das Suchen oder Einfügen von Daten selbst implementiert werden, da keine Abfragesprachen zur Verfügung stehen (siehe Abschnitt 6.1.5.3).

Mit nur zwei Punkten erhält *Redis* hier die wenigsten Punkte, da es auch für sie keine bestehende Lösung zur Odysseus-Integration gibt und die Key-Value Datenstruktur nicht ohne weiteres auf das Szenario der Projektgruppe übertragbar ist. Ein zusätzlicher Nachteil gegenüber *MongoDB* und *CouchDB* ist hier die Tatsache, dass es sich bei *Redis* um eine In-Memory-Lösung handelt, welche also wie auch Odysseus mit dem Arbeitsspeicher arbeitet, sodass entsprechend hohe Anforderungen an die Ausstattung mit Arbeitsspeicher bestünden.

Reifegrad

Auch bezüglich des Reifegrads liegen *MySQL* und *Oracle* mit jeweils acht Punkten vorn. *Oracle* profitiert hier insbesondere von der langen Entwicklung, die bis in die 1970er-Jahre zurück reicht. Zudem sprechen die professionelle Entwicklung und der weltweite Einsatz auf Produktivsystemen für *Oracle* (siehe Abschnitt 6.1.5.3). Allerdings erhält die hier bewertete Express Edition Support nur über das Oracle-Forum und aufgrund der Lizenz gibt es keine besonders große Community, welche zusätzlichen Support oder Weiterentwicklungen garantieren könnte. So wird das DBMS zwar von Oracle kontinuierlich weiterentwickelt, doch stammt die letzte Version aus dem November 2016 (Stand: Mai 2017).

Sehr viel aktueller ist *MySQL*, dessen neueste Version im April 2017 veröffentlicht wurde (Stand: Mai 2017) und welches ebenfalls bereits seit über 20 Jahren entwickelt wird. Daher sind hier ebenso wie bei der *Oracle* Datenbank nur wenige Bugs zu erwarten. Gegenüber *Oracle* kann *MySQL* jedoch mit einer sehr großen Community punkten. Nachdem *MySQL* von Oracle gekauft wurde, haben sich jedoch viele Entwickler von *MySQL* abgewandt, weshalb die Datenbanken bezüglich des Reifegrads an dieser Stelle gleich bewertet werden.

Von der Abwanderung der Entwickler profitiert *MariaDB*. Hier ist ein großer Zuwachs in der Community-Größe zu erwarten, was insbesondere durch die Integration von *MariaDB* in mehrere Linux-Distributionen zu begründen ist. Als ursprünglicher Fork von *MySQL* kann *MariaDB* zudem von der jahrelangen Entwicklung von *MySQL* profitieren, sodass *MariaDB* auf einer ausgereiften Code-Basis beruht. Da *MariaDB* zudem sehr regelmäßig weiterentwickelt wird und die aktuellste Version vom Mai 2017 stammt (Stand: Mai 2017), wird auch sie mit acht Punkten bewertet, obwohl die eigenständige Entwicklung erst 2009 begann.

Mit nur vier Punkten wird der Reifegrad von *MongoDB* bewertet. Auch diese Datenbank ist mit der aktuellsten Version vom April 2017 (Stand: Mai 2017) zwar ausgesprochen aktuell und kann auf eine inzwischen zehnjährige Entwicklung zurückblicken, jedoch gibt es seitens des Anbieters keinen Support für die kostenlose Version und es wird lediglich auf allgemeine Hilfeseiten wie *Stack Overflow* verwiesen. Da zudem das Prinzip der dokumentenorientierten Datenbank vergleichsweise neu ist, hat sich auch sonst bislang keine sehr große Community um *MongoDB* gebildet und die Anzahl an Installationen ist vergleichsweise gering.

Nur drei Punkte erhält CouchDB, dessen Version 2.0 im September 2016 veröffentlicht wurde und noch immer die aktuellste Version ist (Stand: Mai 2017). Wie *MongoDB* wird auch *CouchDB* inzwischen seit über zehn Jahren entwickelt, aber auch hier hat sich bislang keine Community entwickelt, welche mit der von *MySQL* oder *MariaDB* vergleichbar wäre. Zudem liegt *CouchDB* in Datenbank-Rankings in der Beliebtheit weit abgeschlagen gegenüber allen anderen hier analysierten DBMS, was sich an dieser Stelle negativ auf die Bewertung auswirkt [152].

Redis ist zwar nach Rankings beliebter, erhält für den Reifegrad dennoch nur zwei Punkte. Sie ist die jüngste Datenbank in diesem Vergleich und bedient mit ihrer Key-Value-Datenspeicherung und als In-Memory-Datenbank ein sehr spezielles Anwendungsgebiet. Zudem verzichtet auch *Redis* auf direkten Community-Support und verweist auf Seiten wie *Stack Overflow*. Da *Redis* zudem aktuell im wesentlichen ausschließlich vom ursprünglichen Entwickler weiterentwickelt wird, schneidet die Datenbank auch bezüglich des Reifegrads am schlechtesten in diesem Vergleich ab.

Limitierungen

Bezüglich der technischen und konzeptionellen Limitierungen kann keine der untersuchten Datenbanken die volle Punktzahl erreichen, da alle gewissen Beschränkungen unterliegen. *MongoDB* und *CouchDB* erreichen jedoch mit jeweils sieben Punkten die besten Werte. Dies liegt insbesondere daran, dass sie als dokumentenorientierte Datenbanken sehr flexibel bezüglich der Form der zu speichernden Daten sind. Hierdurch ist die Verwaltung auch komplexester Datenstrukturen möglich. Jedoch bieten beide Datenbanken keine Abfragesprachen, wie sie von relationalen Datenbanken bekannt sind. Einfüge-, Lese-, Update- und Lösch-Operationen werden stattdessen als objektspezifische Methoden realisiert und müssen bei Bedarf auf Seiten des Clients individuell programmiert oder zumindest angepasst werden. Insbesondere im Zusammenhang mit Odysseus kann dies einen erheblichen Mehraufwand bedeuten.

MySQL, *MariaDB* und *Oracle* bieten dagegen mit der *Structured Query Language (SQL)* eine entsprechende Abfragesprache, sodass sie diesbezüglich keiner Limitierung unterliegen. Jedoch muss ihre relationale Struktur als Einschränkung gesehen werden, da hier komplexe Datenstrukturen nur über diverse Tabellen verteilt abgebildet werden können. Dies kann beim Lesen und Schreiben der Daten zu Verlusten bei in der Geschwindigkeit führen und insbesondere das Zerlegen der Strukturen aufwändig gestalten. *MySQL* und *MariaDB* erhalten hier sechs Punkte, die *Oracle* Datenbank aufgrund weiterer relevanter Einschränkungen nur vier Punkte. Beispielsweise ist die *Oracle Express* Edition auf die Verwendung von maximal 1GB Arbeitsspeicher sowie eines CPU-Kerns beschränkt, was sich im Vergleich zu *MySQL* und *MariaDB* deutlich auf die Leistungsfähigkeit auswirken kann. Zudem wurde der Funktionsumfang auch in vielen weiteren Bereichen eingeschränkt. Hierzu gehören unter Anderem fehlende Caching-, Sicherheits- und Skalierungs-Mechanismen [63].

Mit nur drei Punkten erreicht *Redis* auch bei den Limitierungen nur den letzten Platz. Dies begründet sich insbesondere mit der Key-Value-Datenstruktur, die eine Speicherung komplexer Daten nur unter erheblichen Aufwand zulässt. Da jedoch in der Projektgruppe vornehmlich komplexe oder zumindest relationale Daten zu verarbeiten sind, ist das Key-Value-Prinzip an dieser Stelle als deutliche Limitierung zu sehen. Hinzu kommt, dass auch komplexe Abfragen nicht möglich sind, sondern jegliche Zusammenführung von Daten auf der ebene des Clients zu implementieren ist.

Performance

Bei der Bewertung der Performance liegt *Redis* als In-Memory-Datenbank mit zehn Punkten klar vorn. Auch wenn dies im Zusammenhang mit Odysseus als negativ bewertet wurde (siehe Abschnitt 6.1.5.3), wirkt sich die Datenverarbeitung im Arbeitsspeicher deutlich positiv auf die Geschwindigkeit aus. Auch die Verwendung einer vergleichsweise einfachen Datenstruktur lässt hier positive Auswirkungen bezüglich von Lese-, Such- oder Schreib-Operationen erwarten.

Bei den anderen hier verglichenen Datenbanksystemen lässt sich die Performance jedoch nicht ohne Weiteres vergleichen, sondern hängt stark vom jeweiligen Anwendungsfall ab. So würden *Oracle*, *MySQL* und *MariaDB* besser und schneller mit relationalen Daten arbeiten, während *MongoDB* und *CouchDB* einen Vorteil bezüglich komplexer Datenstrukturen haben. Da im Rahmen der Projektgruppe zumindest zum Teil auch komplexe Datenstrukturen verarbeitet werden müssen, werden *MongoDB* und *CouchDB* an dieser Stelle mit jeweils acht Punkten bewertet. Sie sind hier performanter, da teure Join-Operationen bei ihnen entfallen, da zusammenhängende Daten bei ihnen bereits innerhalb eines Dokuments gespeichert werden können.

Die relationalen Datenbanken belegen in diesem Vergleich also die hinteren Plätze und werden daher untereinander noch einmal genauer beurteilt. Durch die Einschränkung des verwendeten Arbeitsspeichers und nur eines CPU-Kerns (siehe Abschnitt 6.1.5.3, kann *Oracle Express* mit nur drei Punkten bewertet werden und erreicht den letzten Platz. Im direkten Vergleich zwischen *MySQL* und *MariaDB* erzielt *MariaDB* sieben und *MySQL* fünf Punkte. *MariaDB* kann hier durch optimierte Caching- und Optimierungsverfahren deutliche Vorteile gegenüber *MySQL* erzielen [248].

Transaktionssicherheit

Im am stärksten gewichteten Faktor in dieser Analyse können die drei relationalen Datenbanken überzeugen. Sowohl *Oracle Express*, als auch *MySQL* und *MariaDB* erhalten für die Transaktionssicherheit jeweils zehn Punkte, da sie die ACID-Kriterien erfüllen. Bei ihnen ist eine Transaktion also in jedem

Fall entweder ganz oder gar nicht ausgeführt, führt zu einem konsistenten Datenzustand, gegenseitige Beeinflussungen von Transaktionen werden verhindert und bei Abschluss einer Transaktion ist der Datenbestand garantiert gesichert.

MongoDB, *CouchDB* und *Redis* können die ACID-Eigenschaften nicht ohne Weiteres erfüllen, sondern verletzen sie zum Teil sogar bewusst um Performance-Gewinne zu ermöglichen. So kann *MongoDB* beispielsweise nicht jederzeit einen konsistenten Datenbestand garantieren, sondern setzt auf das Modell der *Eventual Consistency*. Damit wird ausgedrückt, dass ein Datensatz erst nach einer gewissen Zeit als konsistent gilt. Insbesondere auf verteilten Datenbanken kann es also zu Situationen kommen, in welchem Teile des Datenbestands nicht aktuell sind. Auch bezüglich der persistenten Speicherung gibt es Einschränkungen, da die Daten in der Standardkonfiguration nur alle 60 Sekunden dauerhaft gespeichert werden. Daher erhält *MongoDB* für die Transaktionssicherheit nur fünft Punkte.

Da auch *CouchDB* dem Prinzip der *Eventual Consistency* folgt, wird auch diese Datenbank mit fünf Punkten bewertet. Im Gegensatz zu *MongoDB* bietet *CouchDB* zwar eine persistente Speicherung, kann dafür jedoch keine Isolation garantieren. Es ist also möglich alte Datenbestände zu lesen, während diese durch andere Transaktionen verändert werden (*Optimistic Locking)*.

Auch *Redis* setzt auf *Optimistic Locking* und *Eventual Consistency*, garantiert also weder Konsistenz, noch Isolation oder Dauerhaftigkeit. Dabei muss die persistente Speicherung zunächst extra aktiviert und konfiguriert werden, dann gehen jedoch Performacevorteile verloren (siehe Abschnitt 6.1.5.3). Insbesondere aufgrund des hierdurch möglichen Datenverlusts erhält *Redis* daher für die Transaktionssicherheit nur vier Punkte.

6.1.5.4 Fazit

Wie die vorangegangene Analyse gezeigt hat, unterscheiden sich die untersuchten Datenbanksysteme in einigen Bereichen erheblich, weisen an anderer Stelle jedoch auch einige Gemeinsamkeiten auf. Grundsätzlich wäre jedoch die Verwendung jeder der vorgestellten Datenbanken möglich. Dennoch zeigen sich recht deutliche Unterschiede in der Auswertung.

So schneidet *Redis* mit einem Nutzwert von 6,23 insgesamt am schlechtesten ab. Mit einem Nutzwert von 6,95 erreicht *CouchDB* den vorletzten Platz. Mit 7,1 erreicht *MongoDB* den besten Wert unter den nicht-relationalen Datenbanken. Den dritten Platz dieser Analyse erreicht *Oracle* mit 7,43 und *MySQL* den zweiten Platz mit 7,95. Am besten schneidet *MariaDB* mit einem sehr hohen Nutzwert von 8,21 ab. Dabei ist jedoch zu bedenken, dass insbesondere *MySQL* und *Oracle Express* auch aufgrund des für Odysseus bereits vorhandenen Plug-Ins viele Punkte erzielen konnten (siehe Abschnitt 6.1.5.3 und es sich wie bei *MariaDB* um relationale Datenbanksysteme handelt. Sollte sich das relationale Datenmodell im Laufe der Projektgruppe als ungenügend herausstellen, genügt es nicht statt *MariaDB* die zweitplatzierte Datenbank, *MySQL*, zu wählen, sondern es sollte ggf. eine NoSQL-Datenbank in Betracht gezogen werden.

Insgesamt ergibt die Nutzwertanalyse daher, dass für die Projektgruppe *MariaDB* als DBMS verwendet werden sollte und bei Bedarf *MongoDB* als dokumentenorientierte Datenbank als Alternative verwendet werden kann.

6.1.6 Backend Web Frameworks

Im Software Backend Bereich existieren eine Vielzahl an verschiedensten Frameworks die ihre Daseinsberechtigung für spezielle Einsatzzwecke haben. Bei der Begrenzung der Frameworks für die Evaluation und die spätere Verwendung in der Projektgruppe, stand vor allem die benutzte Programmiersprache des Frameworks im Vordergrund, um die Einarbeitungszeit für alle Gruppenmitglieder möglichst gering zu halten. Außerdem spielt der Verbreitungsgrad, somit die Bekanntheit mit einhergehender Community und auch die Nutzung in Produktivsystemen eine wichtige Rolle. Mit diesen Kiterien als Einschränkung für die Auswahl, wurde sich für insgesamt fünf zu evaluierende Frameworks entschieden. Dabei wurden vier auf Java basierende Frameworks (*Vaadin*, *Java Platform, Enterprise Edition (Java EE)*, *Play 2* und *Spring Boot*) und ein JavaScript Framework (*Node.js*) ausgewählt. Im folgenden Abschnitt werden die einzelnen Frameworks vorgestellt und auf ihre Vor- und Nachteile eingegangen. Abschließend wird die Entscheidung für das gewählte Framework mit Hilfe einer Nutzwertanalyse in Form einer Entscheidungsmatrix erklärt.

6.1.6.1 GWT/Vaadin

GWT ist ein ursprünglich von Google entwickeltes Open-Source Framework. Mit *GWT* können Webanwendungen in Java entwickelt werden. Die Client-Server Kommunikation geschieht mithilfe von Remote Procedure Calls (RPCs), die in *GWT* auf Java-Servlets basieren. Mit *GWT* können die Entwickler serverseitig als auch clientseitig die Programmiersprache Java verwenden. Die aktuelle *GWT* Version ist *GWT* 2.8 und unterstützt auch Java 8 Features, wie zum Beispiel Lambdas. Außerdem bietet *GWT* Unterstützung bei der Umsetzung des MVP Entwurfsmusters.

GWT beinhaltet die Java API, einen Compiler und einen Development Server. Damit können die Entwickler in Java programmieren und *GWT* erstellt aus diesem Java Code den JavaScript Code. Zusätzlich wird dieser JavaScript Code von *GWT* optimiert. So entsteht von Java unabhängiger JavaScript Code, der in allen gängigen Browsern läuft. Mit *GWT* können viele Standard Java Klassen ganz normal verwendet werden. Auch das Debuggen von Java Klassen ist ganz normal mit einer Java-IDE wie *eclipse* oder *JetBrains IntelliJ IDEA* möglich. Aber auch alles was mit JavaScript möglich wäre, ist mit *GWT* möglich [121].

Vaadin

Vaadin ist ein Java Web-Framework, mit dem das Erstellen und Warten von webbasierten Anwendungen vereinfacht werden soll. Es unterstützt zwei verschiedene Programmiermodelle, das serverseitige und das clientseitige Modell. Das serverseitige Modell ist mächtiger, da so Anwendungen erstellt werden können ohne das Web zu beachten. Die Anwendung wird so programmiert, als wäre es eine normale Desktop Anwendung, die mit Swing, AWT oder SWT entwickelt wird. *Vaadin* kümmert sich um die Client-Server Kommunikation zwischen dem Browser und dem Server. Die Entwickler müssen sich zudem nicht mit Browser-Technologien wie HTML oder JavaScript auseinander setzen. Die Architektur des serverseitigen Modells besteht aus der clientseitigen Engine und dem serverseitigen Framework. Die Client-Engine wird im Browser als JavaScript Code ausgeführt. Da JavaScript in jeden Browser ausgeführt werden kann, werden keine zusätzlichen Plugins benötigt. Um aus dem Java Code JavaScript Code zu erzeugen wird *GWT* verwendet. So werden auch hier die Vorteile von *GWT* genutzt und es werden Webanwendungen entwickelt, die von allen gängigen Browsern ausgeführt werden können. Außerdem findet Client und Server Entwicklung bei beiden in der Programmiersprache Java

statt. Auch können neue Widgets in Java entwickelt werden und es gibt viele andere Bibliotheken für Vaadin. Das Aussehen kann zum Beispiel mithilfe von CSS definiert werden. Die UI-Logik der Anwendung wird auf dem Server mit Java-Servlets ausgeführt. Die Client-Server Kommunikation geschieht im Hintergrund mithilfe von AJAX (Asynchronous JavaScript and XML) [184].

Die Kern Elemente des *Vaadin* Frameworks können kostenlos verwenden werden. Dazu gehört auch das Add-on Verzeichnis, das Forum und die Dokumentation. *Vaadin* Pro kostet dagegen 160$ pro Monat bzw. 100$ pro Monat wenn ein Jahresvertrag abgeschlossen wird. Es beinhaltet zusätzlich TestBench, Designer, Charts, Board und Spreadsheet. TestBench ist ein Werkzeug zum Ausführen von automatisierten Tests mit Vaadin. Der Designer ist ein GUI-Builder, mit dem schnell und einfach grafische Benutzeroberflächen erstellt werden können. Charts ist eine sehr mächtige Bibliothek zum Erstellen von Diagrammen (Charts). Das *Vaadin* Board ist ein mächtiges Layout-Element zum Erstellen von Benutzeroberflächen, die „responsive“ sind, also an die Größe des Bildschirms anpassen. Mit dem Spreadsheet können Mircosoft Excel Tabellen in die Anwendungen integriert werden. Außerdem können diese Tools mit *Vaadin* Pro in unbegrenzt vielen Projekten verwendet werden. Es gibt auch noch *Vaadin* Prime, welches 1190$ pro Monat kostet. Mit *Vaadin* Prime kann Kontakt zu den *Vaadin* Entwicklern aufgenommen werden und hat mehr Einfluss auf diese [185].

Vor- und Nachteile von GWT/Vaadin

Ein für die Projektgruppe wichtiger Vorteil ist, dass wir mit GWT/*Vaadin* clientseitig und serverseitig die Programmiersprache Java verwenden können, denn Java wird an der Universität Oldenburg gelehrt und ist dadurch jedem aus der Projektgruppe bekannt. Außerdem wurde *GWT* an der Universität Oldenburg bereits im Softwareprojekt verwendet und ist dadurch vielen aus der Projektgruppe bereits bekannt, wodurch sich die Einarbeitungszeit verkürzen würde. Durch die bereits integrierten RPCs wäre eine Client-Server Kommunikation bereits vorhanden.

Mit *Vaadin* hätte die Projektgruppe zusätzliche Vorteile, da sich beispielsweise nicht mehr um die Client-Server Kommunikation gekümmert werden müsste. Außerdem sind mit *Vaadin* mehr Widgets bereits vorhanden und es gäbe mit *Vaadin* Pro zudem einen GUI-Designer. Die umfangreiche Charts Bibliothek wäre auch eine sehr große Hilfe für die Projektgruppe, in Betracht der späteren Datenvisualisierung.

Der Nachteil ist, dass *Vaadin* Pro nicht frei erhältlich ist und somit von der Projektgruppe nicht verwendet werden kann. Dadurch entfällt der Vorteil, den es mit der Charts Bibliothek gäbe. Allerdings sind auch die Kernelemente von *Vaadin* alleine nicht so mächtig, um die Anforderungen der Projektgruppe zu erfüllen. Außerdem ist *GWT* nicht mehr aktuell und wird kaum in neuen Projekten eingesetzt, wodurch die Community nicht mehr weiter wächst.

Ein weiterer Nachteil ist, dass clientseitig auch GWT/*Vaadin* verwendet werden muss. Dadurch müsste die Projektgruppe auf aktuellere clientseitige Webframeworks, wie Angular[3] verzichten.

6.1.6.2 Java EE

Java EE, früher J2EE, ist eine Platform, die heute mit Beiträgen von Branchenexperten, kommerziellen und Open Source Organisationen, Java User Groups und unzähligen anderen Personen entwickelt wird. Die aktuelle Version ist *Java EE 7*.

[3] `https://angular.io/`

Das Ziel von *Java EE* ist es den Entwicklern mächtige APIs zur Verfügung zu stellen, mit den die Entwicklungszeit verkürzt, die Anwendungskomplexität reduziert und die Anwendungsleistung verbessert wird. Die *Java EE-Plattform* wird über den *Java Community Process (JCP)* entwickelt, der bei der Entwicklung von allen Java-Technologien eingesetzt wird. Dabei erstellen Expertengruppen *Java Specification Requestss (JSRs)* um so die verschiedenen *Java EE* Technologien zu definieren. Die Konfiguration mit XML ist in *Java EE* optional. Stattdessen können die Entwickler Java-Annotationen direkt im Quellcode verwenden. Auch können die von den Komponenten benötigten Ressourcen über Dependency-Injektion übergeben werden. Dependency-Injektion kann auch mithilfe von Annotationen in *Enterprise JavaBeans (EJB)* Containern, Webcontainern und Anwendungsclients verwendet werden [61].

Java EE besteht heute aus vielen Spezifikationen, die für unterschiedliche Bereiche der Entwicklung mit Java zuständig sind. Die meisten zielen auf Anwendungen, die in einem Applikationsserver ausgeführt werden. Dabei wird die Anwendung in vier Bereiche unterteilt. Benutzeroberflächen werden mit *Java Server Faces (JFS)* programmiert. Diese laufen in Servlet-Containern ab. Die Geschäftslogik wird mithilfe von EJBs programmiert. Datenbankzugriffe werden mit der *Java Persistence API (JPA)* ausgeführt [274].

Leider wird die Weiterentwicklung von *Java EE* zur Zeit vernachlässigt. Oracle konzentriert sich mehr auf andere Bereiche, wie zum Beispiel den Cloud Bereich. Aus diesem Grund wurde das Veröffentlichungsdatum von der neuen *Java EE* Version 8 schon mehrmals verschoben und es passiert in dem Bereich auch wenig. Viele Ingenieure, die für die Weiterentwicklung von *Java EE* zuständig waren, wurden in andere Bereiche versetzt oder haben das Unternehmen verlassen. Es wird versucht von Oracle ein aussagekräftiges offizielles Statement zur Zukunft von Java EE zu erhalten, was Oracle jedoch verweigert. Selbst große Unternehmen, die im JCP involviert sind, bekommen keine Antwort [204].

Java EE 7

Die aktuelle *Java EE* Version ist *Java EE 7*. Es ist „leichtgewichtiger“ als die Vorgänger. Vorher hatte *Java EE* keinen guten Ruf bei Softwareentwicklern. Der Grund dafür war unter Anderem die große Menge an Boilerplate-Code, der beim Entwickeln von Anwendungen mit *Java EE* entstanden ist. Dies hat sich mit *Java EE 7* gebessert. Durch Annotationen werden XML-Konfigurationen fast komplett ersetzt, durch Standards werden unnötige Individualisierungen vermieden und durch einfache POJOs (Plain Old Java Objects) werden umfangreiche Klassenhierarchien abgelöst [186].

Vor- und Nachteile von Java EE 7

Ein großer Vorteil ist auch hier, dass Java an der Universität Oldenburg als erste Programmiersprache gelehrt wird. Durch die klar spezifizierten Komponenten wird die Einarbeitung in *Java EE 7* vereinfacht. Außerdem ist so ein großer Teil der serverseitigen Architektur vorgegeben, wodurch schneller eine Server Architektur entwickelt werden kann. Durch die bereits integrierte Dependency-Injektion wird das verwalten von Abhängigkeiten erleichtert. Mithilfe von JPA müssen SQL-Statement nicht von den Entwicklern selbst implementiert werden.

Ein Nachteil ist der viele Boilerplate-Code, der auch noch beim Entwickeln von Anwendungen mit *Java EE 7* entsteht. Ein weiterer Nachteil ist, dass die Weiterentwicklung von *Java EE* vernachlässigt wird. So kann es mit Konkurrenten, wie zum Beispiel Spring, nicht mithalten.

6.1.6.3 Node.js

Node.js (ab hier nur noch als *Node* bezeichnet) ist eine Plattform, die es möglich macht Serverapplikationen in JavaScript zu schreiben. Es zeichnet sich durch ein reichhaltiges Ökosystem und hohe Skalierbarkeit aus. *Node* ist in der Industrie weit verbreitet und wird vor allem eingesetzt um Websysteme zu entwickeln, die einer hohen Belastung durch zeitgleiche Zugriffe von vielen Nutzern standhalten müssen oder große Datenmengen an viele Nutzer gleichzeitig schicken müssen.

Für die Weiterentwicklung und Verbreitung von *Node* wurde 2016 die *Node.js* Foundation gegründet. Sie agiert nach dem *Open Government* Prinzip und hat diverse Mitglieder aus der Industrie wie Microsoft, IBM, Intel, Joyent, etc. Mit der Gründung der *Node.js* Foundation wurde auch der Releasezyklus von *Node* geändert, bei dem unter Anderem jetzt auch ein Long Term Support eingerichtet wurde.

Wird das Projekt *node* auf Github betrachtet, kann gesehen werden, wie stark es in der Open Source Community verbreitet ist. Zum Zeitpunkt des Schreibens haben über 1300 Personen an der Weiterentwicklung des Projekts mitgewirkt und es kommen Änderungen in Abständen von nur einigen Tagen zum Projekt, was die rege Weiterentwicklung des Projekts untermauert [100].

JavaScript auf dem Server

Durch Googles hochperformante V8 Engine ist es möglich JavaScript mit einer ähnlichen Geschwindigkeit wie statisch kompilierte Programmiersprachen auszuführen. *Node* baut ebenfalls auf V8 auf und erweitert V8 um eine Standardbibliothek, mit der es möglich ist Netzwerkanwendungen zu schreiben, die außerhalb des Browsers laufen [282]. Das bringt den Vorteil mit sich, dass Entwickler in einer einzigen Programmiersprache den kompletten Technologiestack entwickeln können. Das heißt, Entwickler, die schon mit JavaScript aus dem Frontend bekannt sind, können sich schnell in *Node* einarbeiten und müssen so keinen kontextuellen Tausch zwischen den einzelnen Abschnitten einer Applikation machen.

Ein weiterer Vorteil von *Node* ist es, dass *Node* fast ausschließlich auf ein asynchrones Programmierparadigma setzt. Bis auf einige I/O Operationen, die auch synchron ausgeführt werden können, wird bei jeder Operation in *Node* nach dem Aufruf die Kontrolle über diese Operation abgegeben und erst wenn die Operation ausgeführt wurde, verarbeitet *Node* das Ergebnis der Operation weiter [123]. In der Zwischenzeit kann die mit *Node* entwickelte Applikation weiter Operationen aufrufen oder sich im Ruhezustand befinden. Beispielsweise kann eine Applikation, die einen Datenbankzugriff machen soll, den Aufruf an die Datenbank starten und ist danach frei für weitere eingehende Aufrufe. Es wird angenommen, dass sich eine Datei, die die Applikation beobachten soll, verändert hat und archiviert werden muss. Das kann gemacht werden, da die Daten aus der Datenbank noch nicht bereit zur Weiterverarbeitung sind. Sobald sie es sind, wird *Node* von dem Betriebssystem benachrichtigt und kann diese weiterleiten. Was dabei wichtig zu verstehen ist, während die Applikation auf die Daten gewartet hat, hat *Node* nicht blockiert, sondern konnte eine weitere Operation auf der veränderten Datei durchführen.

Vor- und Nachteile von Node

Eine der Stärken von *Node* ist die Skalierbarkeit durch das asynchrone Programmierparadigma mithilfe von JavaScript Callbacks. Dadurch können z. B. mit einem *Node* Server viele I/O Aufrufe hintereinander verarbeitet werden, ohne dass der Server blockiert.

Außerdem hat *Node* den Vorteil, dass es nativ mit JSON arbeiten kann. Dadurch ist es bei der Serialisierung mit JSON sehr schnell und es besteht keine Abhängigkeit zu externen Bibliotheken. So kann auch mit Hilfe von Frameworks wie Express.js schnell und effizient eine JSON API verarbeitet und erstellt werden [123].

Bei den Nachteilen muss auf jeden Fall genannt werden, dass *Node* ohne jeglichen Erweiterungen zwar sehr schnell ist, allerdings hat das einen Preis: Die Standardbibliothek von *Node* umfasst nur das Nötigste um eine Netzwerkapplikation zu schreiben, so dass Entwickler entweder vieles selbst implementieren müssen oder auf die große Auswahl an externen Bibliotheken und Paketen in einem Repository wie dem npm. Änderungen an den Bibliotheken oder an dem Repository wo die Bibliotheken verwaltet werden, können allerdings zu ernsthaften Komplikationen führen, was ersichtlich war, als ein Entwickler seine Bibliotheken aus dem npm genommen hat [201].

Einer der großen Vorteile von *Node*, die einheitliche Programmiersprache zwischen Backend und Frontend, kann für die Projektgruppe allerdings als Nachteil angesehen werden. Zwar ist JavaScript in der Industrie eine weit verbreitete Programmiersprache, dennoch ist sie im universitären Umfeld nicht jedem so bekannt wie Java. Bei einer Entscheidung für *Node*, müssten sich mehr Teilnehmer in JavaScript einarbeiten, als wenn JavaScript nur im Frontend benutzt wird. Damit würde deutlich mehr Zeit für die Einarbeitung in JavaScript und *Node* eingehen, als wenn die Projektgruppe sich für eine auf Java basierte Technologie entscheidet. Da hauptsächlich Java an der Universität Oldenburg unterrichtet wird, haben alle Gruppenmitglieder Erfahrung im Umgang mit Java und so bleibt nur noch die Einarbeitungszeit in beispielsweise neue Bibliotheken oder Frameworks übrig.

6.1.6.4 Play 2

Das Play Framework (abgekürzt als *Play* oder *Play 2*, seit Version 2) ist ein Backend Framework das seit Version 2 in der Programmiersprache Scala[4] geschrieben ist. Durch die Interoperabilität zwischen Java und Scala Bibliotheken und eigens an Java angepasste APIs kann *Play* jedoch auch mit Java benutzt werden.

Im Vergleich zu anderen Java Frameworks unterscheidet sich *Play* darin, dass es nicht Java EE Spezifikationen implementiert, sondern sich stark an Webframeworks in anderen Programmiersprachen orientiert, wie etwa Ruby on Rails[5]. Der Grund dafür ist, dass der Fokus stärker auf Web-Entwicklung gelegt wurde und es wurde darauf geachtet, dass Entwickler direkt mit HTTP arbeiten können, möglichst wenig Boilerplate-Code schreiben müssen oder standardmäßig mit Hot Deployment ihre Applikationen entwickeln können [181].

Play ist Reactive

Bei dem *Play* Framework wurde das Ziel gesetzt, dass es *Reactive* ist, das heißt, es soll sehr kurze Antwortzeiten haben, im Falle eines Fehlers soll nicht das Gesamtsystem ausfallen, sondern weiterhin reagieren können, das System soll sich an eine Belastung von außen anpassen können und es soll über einen asynchronen Nachrichtenaustausch kommuniziert werden [35].

Somit ähnelt *Play* im Gegensatz zu traditionellen Backend Frameworks in Java eher *Node.js* als z. B. *Spring*. Das heißt wenn *Play* eine Anfrage verarbeitet, blockiert das Framework nicht so lange bis die

[4] `https://www.scala-lang.org/`
[5] `http://rubyonrails.org/`

Anfrage fertig verarbeitet ist und das Ergebnis zurückgesendet werden kann, sondern leitet die Anfrage an die Middleware Akka[6] weiter und ist so frei weitere eingehende Anfragen anzunehmen und diese an Akka weiterzuleiten [181]. Nachdem Akka eine Anfrage verarbeitet hat, wird das Ergebnis davon zurück an *Play* geschickt und *Play* kann das Ergebnis an einen Client ausgeben.

Vor- und Nachteile von Play

Ein Vorteil von *Play* sind die Annahmen über moderne Webentwicklung, die den Programmierer schnell und iterativ entwickeln lassen können. Das heißt *Play* folgt Best Practices und ist so konfiguriert, dass dem Entwickler möglichst viel Arbeit bei der Erstellung und Organisation von Projekt abgenommen werden soll und so die Produktivität des Entwicklers steigert.

Ein weiterer Vorteil von *Play* ist die asynchrone Verarbeitung von eingehenden Anfragen. Durch die Integration von Akka können z. B. Daten in *Chunks* an einen Client gesendet werden, sobald Teile der Daten verfügbar sind. Damit können Akka Datenstreams zeitnah verschickt werden, sobald ein Chunk bereit ist, bis die vollständige Antwort dem Client in Form aller Chunks vorliegt. Weiterhin ermöglicht *Play* durch die Implementation des Comet Modells und unter Benutzung von WebSockets Uni- und Bidirektionale Kommunikation zwischen Client und Server, die ebenfalls asynchron abläuft [181].

Ein Nachteil von *Play* ist die starke Anbindung an das Build Tool *sbt*[7]. Es gibt zwar Möglichkeiten auch andere Build Tools wie z. B. *Gradle* zu benutzen, allerdings wäre es in dem Fall notwendig eine sich noch in Entwicklung befindliche Erweiterung zu benutzen, bei der es eventuell noch zu Änderungen in der API kommen kann. Um den bestmöglichen Toolsupport zu erhalten, wäre die Arbeit mit sbt notwendig, was im Rahmen dieses Projekts nicht vorgesehen ist.

Ein weiterer Nachteil von *Play* ist die wenig starke Verbreitung im Vergleich zu anderen Frameworks wie z. B. *Spring*. Dadurch gibt es auch keine so große Community hinter dem Framework, die Hilfe anbietet oder Informationen über die Nutzung zur Verfügung stellt. Zwar gibt es eine ausführliche Dokumentation auf der Webseite des Frameworks, die allerdings auch die einzige aktuelle Referenz ist. Werden die Anzahl an Fragen auf Stack Overflow[8] verglichen, können zum Zeitpunk des Schreibens etwas über 14.000 Fragen zu *Play* erhalten werden. Zu *Spring* hingegen gibt es über 100.000 und zu *Spring* Model View Controller (MVC) alleine über 40.000 Fragen. Somit können Schwierigkeiten, die bei der Entwicklung mit *Play* auftreten und die nicht in der Dokumentation beschrieben werden, eventuell schwieriger behoben werden, da diese weniger dokumentiert sind.

Außerdem kommt dazu, dass *Play* nicht nur mit Java benutzt wird, sondern auch mit Scala, was ebenfalls bedeutet, dass sich bei der Recherche auch mit Fragen und Antworten in Scala befasst werden muss. Somit bleibt eine Einarbeitung in Scala nicht erspart, auch wenn die eigentliche Entwicklung mit Java geschehen soll. Das bedeutet dann auch, dass ein Teil der Zeit im Projekt dafür benutzt werden würde, sich in eine Technologie einzuarbeiten, die letztendlich nicht für das Endprodukt relevant ist.

[6] `http://akka.io/`
[7] `http://www.scala-sbt.org/`
[8] `https://stackoverflow.com/tags`

6.1.6.5 Spring Boot

Spring Boot[9] basiert auf dem *Spring Framework*[10]. Das *Spring Framework* ist ein OpenSource Anwendungsframework, welches es ermöglichen soll die Entwicklung von Enterprise-Java-Anwendungen zu vereinfachen und dabei auf bewährte Programmierparadigmen zurückzugreifen. Die erste Version wurde im Jahre 2002 von Rod Johnson zusammen mit seinem Buch „Expert One-on-One J2EE Design and Development“ publiziert. Aktuell gehört es zur Pivotal Software, Inc. und es wird in dem öffentlichen GitHub Repository[11] mit mehr als 14.000 Sternen, von mehr als 200 Beitragenden und mit einem Team bestehend aus 70 Personen[12] an der Weiterentwicklung des *Spring Frameworks* gearbeitet. Die letzten erschienenen Version sind der Release Kanditat 5.0.0-RC1 und die stabile Release Version 4.3.8. *Spring Boot* ist derzeit als Release in Version 1.5.3 verfügbar und arbeitet auf dem *Spring Framework* in Version 4.3.8. Die steigende Popularität von *Spring Boot* und dem *Spring Framework* wurde im Februar 2017 durch eine einmonatige Umfrage von JAXenter.de[13] mit knapp 1000 Teilnehmern weiter untermauert. Hierbei wurden diverse Kategorien aufgestellt und in jeder Kategorie, bei der es sich um Backend-Frameworks handelte, stand *Spring Boot* bzw. *Spring* MVC meist mit großem Abstand an erster Stelle. Außerdem befragte zeroturnaround.com im Oktober 2016[14] Java Programmierer, welches Java Web-Framework sie präferieren. Darüber hinaus analysierten sie im Februar 2017[15] und März 2017[16] die Popularität über StackOverflow, LinkedIn, GitHub und Google. Auch hier kam bei allen Befragungen bzw. Analysen, teilweise mit weitem Vorsprung, hervor, dass *Spring* (*Spring Boot*/*Spring Framework*) aktuell die präferierte Wahl eines Backend Frameworks ist.

6.1.6.6 Features

In diesem Abschnitt sollen Vorteile und Features des *Spring Frameworks* im Zusammenspiel mit *Spring Boot* vorgestellt werden, die in diesem Projekt Verwendung finden könnten.

Convention over configuration

Spring Boot erweitert das *Spring Framework* in der Weise, dass es versucht dem Entwickler ähnlich dem KISS-Prinzip[17] , alle unnötigen Konfigurationsarbeiten abzunehmen. Dabei ist es mittels *Spring Boot* möglich, komplett auf die XML Schema-basierende Konfiguration zu verzichten und dies über in *Spring Boot* eingeführte Annotationen zu automatisieren. Dazu bietet *Spring Boot* standardmäßig die Möglichkeit das Projekt nicht als .war Archiv zu deployen, sondern eine lauffähige .jar Datei zu erstellen, welche einen embedded Tomcat enthält und mit den benötigten Einstellungen startet (wahlweise auch Jetty). Für die initiale Bereitstellung der benötigten Dependencies bietet Pivotal den Spring Initializer[18] Web-Service an, welcher in modernen IDEs standardmäßig mit eingebunden ist. Diese Gründe lassen gerade Einsteiger in den Java Backend Bereich in kürzester Zeit funktionierende und produktionsfertige Applikationen entwickeln.

[9] `https://projects.spring.io/spring-boot/`
[10] `https://projects.spring.io/spring-framework/`
[11] `https://github.com/spring-projects/spring-framework`
[12] `https://spring.io/team`
[13] `https://jaxenter.de/framework-trends-2017-53153`
[14] `https://zeroturnaround.com/rebellabs/spring-vs-java-ee-survey-results/`
[15] `https://zeroturnaround.com/rebellabs/java-web-frameworks-index-by-rebellabs/`
[16] `https://zeroturnaround.com/rebellabs/java-web-frameworks-index-march-2017/`
[17] Das KISS-Prinzip (Keep it simple, stupid) versucht zu einem Problem die einfachste Lösung anzustreben.
[18] `https://start.spring.io/`

Aspect-oriented programming

Spring stellt ein eigenes Framework für die Aspektorientierte Programmierung (AOP) bereit, Spring AOP (aktuell in Version 2.0), das sich dabei jedoch der AspectJ pointcut language bedient. Mit der AOP ist es möglich, ein System nicht auf Basis von Klassen, wie es in der Objektorientierten Programmierung der Fall ist, sondern Aspekte zu nutzen. Somit lässt sich eine bessere Trennung der Belange ermöglichen als es mit der OOP der Fall ist. Ein klassisches Beispielszenario hierfür ist die Einbettung eines Loggers in alle Methoden aller Controller Klassen, um z. B. die Zeit von Beginn bis Ende zu messen. Dies wäre mit einem nur objektorientierten Ansatz sehr aufwändig und schlechter wartbar als mit einer aspektorientierten Lösung[19].

Dependency Injection

Der *Spring Core Container* stellt eine Reihe an Möglichkeiten bereit Bean Objekte zu erstellen, welche durch das von *Spring* entwickelte Dependency Injection Framework injiziert werden können. Dies dient zur Entkopplung einzelner Komponenten und Services. Die einfachste Variante ist es, die eigens erzeugten Beans in einer Klasse zu erstellen, welche per `@ComponentScan` annotiert ist und diese können dann an einer beliebigen Stelle über die Annotation @Autowired injiziert werden. Alle Klassen die per `@Component`, `@Service`, `@Repository`, `@Controller` und deren Subklassen annotiert werden, werden automatisch als Bean erstellt[20].

Data access

Das *Spring Data* Projekt versucht typische Schwierigkeiten bei der Anbindung und Arbeit mit Datenbanken zu adressieren. Außerdem wird durch die Bereitstellung generischer Repository Interfaces ein Abstraktionslevel geschaffen, welches es erlaubt, die Datenquellen über eine .properties Datei auszutauschen ohne auch nur eine Zeile an Code zu verändern. Außerdem existieren speziellere Subklassen des Repository Interfaces, welche bereits Standardoperationen wie z. B. die CRUD Operationen für alle Datenquellen implementieren. Zu den Schwierigkeiten, die das Data Projekt anspricht, gehört das Ressourcenmanagement, welches automatisch Datenbankressourcen erwirbt und freigibt. Außerdem werden datenbankspezifische Exceptions in eine allgemeingültige *Spring* Exception Hierarchy übersetzt[21]. Des Weiteren verfügt *Spring* über eine Transaktionssteuerung, welche über den EntityManager manuell oder mithilfe von Annotationen bewerkstelligt wird und vieles mehr. Darüber hinaus bietet *Spring* eine hervorragende Anbindung für Objektrelationales Mapping (ORM) z. B. per Hibernate. Dabei unterstützt es auch die JPA und stellt hierzu spezielle JpaRepositories[22] zur Verfügung. Falls Hibernate genutzt wird, können in den abgeleiteten Repository Klassen auf verschiedenste, jedoch einfachste Weise Queries erstellt werden. Die einfachste Variante ist es, Abfragen per Methodennamen zu deklarieren, dabei werden die Methodennamen automatisch durch *Spring Data* in Abfragen der dahinter liegenden Datenquelle übersetzt (Siehe Listing 6.5 Zeile 4). Für kompliziertere Abfragen könnte dies jedoch unübersichtlich werden und deshalb bietet *Spring Data* JPA die @Query Annotati-

[19] `https://docs.spring.io/spring/docs/current/spring-framework-reference/html/aop.html`

[20] vgl. `https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-spring-beans-and-dependency-injection.html`

[21] `https://docs.spring.io/spring/docs/current/spring-framework-reference/html/dao.html#dao-exceptions`

[22] `http://docs.spring.io/spring-boot/docs/current/reference/html/howto-data-access.html`

on, mit der eine Abfrage in der Hibernate Query Language (HQL) direkt an eine Methode gebunden werden kann (Für ein Beispiel Siehe Listing 6.5 Zeile 6). HQL wird dabei von den meisten moderneren IDEs um Syntax-Highlighting und teilweise Überprüfung unterstützt, somit ist eine schnellere und syntaktisch fehlerfreie Entwicklung möglich. Außerdem ist es möglich die *Spring Expression Language* (SpEL)[23] zu verwenden, womit Anfragen generalisiert werden können und vieles mehr.

```
1 @Repository
2 public interface UserRepository extends JpaRepository<UserEntity, Long> {
3   Optional<UserEntity> findOneByFirstname(String firstname);
4
5     @Query("SELECT COUNT(DISTINCT u.firstname)
6       FROM UserEntity u
7       WHERE LOWER(u.firstname) = LOWER(:firstname)")
8     Optional<UserEntity> countUsers($$@Param$$("firstname") String firstname);
9 }
```

Listing 6.5: *Spring Jpa Repository Beispiel*

Model–view–controller

Das *Spring* Web MVC Framework baut primär auf dem eigens erstellten DispatcherServlet auf, welches HTTP Anfragen an Handler weiterleitet. Die Handler können in *Spring Boot* z. B. per `@Controller` Annotation an einer Klasse erstellt werden und das passende URI Mapping für die Klasse als Präfix und jede sich in der Klasse befindenden Methode kann über die `@RequestMapping` Annotation festgelegt werden. Dabei können in die URI, unter anderem mit geschweiften Klammern und einem Namen, Wildcards eingefügt werden, welche innerhalb der Methoden per `@PathVariable` Annotation an den Parametern in die Methode übergeben werden können. Das RequestMapping enthält noch einen Parameter für den Typ der HTTP Anfrage. Außerdem kann durch die `@RestController` Annotation ein Handler erstellt werden, der weiß, dass es sich um eine REST API handelt und somit die Annotation `@ResponseBody` beinhaltet[24].

```
1 @RestController
2 @RequestMapping(value="/user")
3 public class UserResource {
4
5     @RequestMapping(value="/{name}/count", method= RequestMethod.GET)
6     int countUsersByName(@PathVariable String name) {
7         return -1; //Ersetze durch Datenbankabfrage o.  .
8     }
9 }
```

Listing 6.6: *Spring Rest Controller Beispiel*

[23] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/expressions.html

[24] https://docs.spring.io/spring/docs/current/spring-framework-reference/html/mvc.html

Security

Falls *Spring Boot* benutzt wird, kann *Spring Security* einfach über das org.springframework.boot.spring-boot-starter-security Artefakt[25] über ein beliebiges Build Tool (Siehe 6.1.4), oder händisch dem Klassenpfad hinzugefügt werden. Dadurch werden automatisch im Standardfall alle HTTP Endpunkte per Basisauthentifizierung nach RFC 2617[26] abgesichert. Der Username lautet hier standardmäßig user und das Passwort wird beim Starten der *Spring Applikation* zufällig generiert und in der Konsole angezeigt[27]. Diese Standardkonfigurationen sind natürlich einstellbar und nicht nur auf einen statischen Nutzer beschränkt. Über die Erstellung eines eigenen UserDetailsServices durch Ableiten desselbigen, können Rollenvergaben auf Basis von Datenquellen wie Datenbankanbindungen o. Ä. erstellt werden[28]. Durch die Erstellung diverser Nutzerrollen, können dadurch die HTTP Schnittstellenzugriffe nicht nur binär unterschieden und verweigert werden, sondern die HTTP Anfragen können durch die Annotation der passenden Controller Methoden mit `@Secured(ROLLEN_ ARRAY)` auf bestimmte Rollen beschränkt werden. Diese Annotationen lassen sich zusätzlich an alle Methoden von Klassen und Interfaces setzen. Falls noch eine feingranularere Spezifikation des Nutzers stattfinden muss, sollte die Annotation `@PreAuthorize` verwendet werden, welche, wie schon im Abschnitt Data Access erwähnt die SpEl unterstützt und somit sehr viel mächtiger ist[29]. Um bei komplexen API Schnittstellen nicht jede Methode annotieren zu müssen, ist es möglich über einen WebSecurityConfigurerAdapter Sicherheitseinstellungen auf Basis von URI Angaben zu tätigen (Siehe Listing 6.7).

```
1  @Configuration
2  static class WebSecurityConfiguration extends WebSecurityConfigurerAdapter {
3
4          @Override
5          protected void configure(HttpSecurity http) throws Exception {
6          http
7                  .authorizeRequests()
8                  .antMatchers("/api/admin/**").hasRole("ROLE_ADMIN")
9                  .anyRequest().authenticated();
10         }
11 }
```

Listing 6.7: *Spring Security WebSecurityConfigurerAdapter Beispiel*

Des Weiteren ist es möglich, das Verfahren für die HTTP-Authentifizierung zum Beispiel gegen Digest Access Authentication[30], OAuth (v1.0 oder v2.0)[31], OpenID[32] oder sogar LDAP[33] und weitere bereits implementierte Authentifizierungen austauschen. Darüber hinaus bietet *Spring* noch viele weitere Module für die Sicherheit, wie z. B. Web Application Security (unteranderem gegen XSS und CSRF, etc), WebSocket Security, Implementierungen Persistent Login (Remember-Me) und vieles mehr.

[25] http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/
[26] https://www.ietf.org/rfc/rfc2617.txt
[27] Das Passwort wird unter dem Logging-Level INFO geloggt. Beispielanzeige: Using default security password: 78fa095d-3f4c-48b1-ad50-e24c31d5cf35
[28] http://www.baeldung.com/role-and-privilege-for-spring-security-registration
[29] http://docs.spring.io/autorepo/docs/spring-security/current/reference/html/el-access.html#el-common-built-in
[30] http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#digest-processing-filter
[31] https://spring.io/guides/tutorials/spring-boot-oauth2/
[32] http://www.baeldung.com/spring-security-openid-connect
[33] https://spring.io/guides/gs/authenticating-ldap/

Testing

Das Entwicklerteam von *Spring* steht hinter dem Test-Driven-Development (TDD) und hat deshalb versucht eine elegante Lösung für das Integration Testing und dazu best practices für das Unit Testing zu finden. Falls der Entwickler sich an die Vorgabearchitektur, samt der verschiedenen Abstraktionsschichten hält, erhält er durch das mocking[34] von DAO bzw. Repository Interface Implementierungen. Darüber hinaus ist es mit *Spring* möglich Integration Tests durchzuführen, ohne die Applikation dafür zu deployen. Hierfür existieren zahlreiche Annotationen für die Konfiguration, dem Verwalten von Transaktionen (Commit, Rollback, etc), erweiterete Annotationen für JUnit, dem Unterscheiden zwischen verschiedenen Testprofilen, etc. Darüber hinaus ist es mit dem *Spring* MVC Test Framework möglich, *Spring* MVC Code unter Verwendung von JUnit oder TestNG zu testen. Damit können in deklarativer Form komplette HTTP Schnittstellen Tests samt Nutzerauthentifizierung, Sessionmanagement und Konversion der Data Transfer Objects (DTO) durchgeführt werden (Siehe Listing 6.8)

```
1  import static ... .servlet.request.MockMvcRequestBuilders.*;
2  import static ... .servlet.result.MockMvcResultMatchers.*;
3
4  @RunWith(SpringRunner.class)
5  @WebAppConfiguration
6  @ContextConfiguration
7  public class ExampleTests {
8
9      @Autowired
10     private WebApplicationContext wac;
11     private MockMvc mockMvc;
12
13     @Before
14     public void setup() {
15         this.mockMvc = MockMvcBuilders.webAppContextSetup(this.wac).build();
16     }
17
18     @Test
19     public void getAccount() throws Exception {
20         MockHttpServletRequestBuilder mockhsrb = get("/user/mustermann/count")
21                 .accept(MediaType.APPLICATION_JSON_UTF8_VALUE);
22
23         this.mockMvc.perform(mockhsrb)
24                 .andExpect(status().isOk())
25                 .andExpect(content().contentType(MediaType.APPLICATION_JSON))
26                 .andExpect(jsonPath("$.count").value("5"));
27     }
28 }
```

Listing 6.8: *Spring MVC Test Framework Beispiel*

Eine Gesamtübersicht für alle Möglichkeiten, die *Spring* für die Unterstützung beim Testen liefert, bietet die offizielle Dokumentation im Abschnitt Part IV. Testing[35] einen guten Überblick.

[34] In der Informatik verwendeter Vorgang, um etwas vorzutäuschen, in den meisten Fällen Daten (Mock-Objekte) für Modultests.

[35] https://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle/#testing

6.1.6.7 Bewertungskriterien

Nachdem die Auswahl an Backend Frameworks mit ihren Vor- und Nachteilen beschrieben wurde, befasst sich dieser Abschnitt mit der Evaluation der genannten Frameworks.

Die Evaluation geschah anhand der Nutzwertanalyse (Siehe Abbildung 6.6). Zur Bewertung wurden dabei folgende Kriterien betrachtet:

- Validierung
- REST Error Handling
- Services
- Performance
- Datenbankanbindung
- Templating
- Verbreitung
- Einarbeitungsaufwand
- Security
- Serialisierung (JSON & XML)
- Automatische API Dokumentation und Test

Bei der Validierung wird betrachtet, wie mit Hilfe des Frameworks Datenobjekte in den verschiedenen Schichten, wie beispielsweise die View oder in der Persistenzschicht, validiert werden können. Dabei wurde festgestellt, dass in Node, im Vergleich zu den anderen Frameworks, die Validierung nur über externe Bibliotheken möglich ist und dadurch mehr Boilerplate-Code entsteht. Die Java Frameworks haben Implementationen der JSR-303 Spezifikationen und so kann die Validierung über Annotationen gelöst werden. Dabei bietet *Spring* die umfangreichsten Möglichkeiten zur Validierung, wie z. B. die Validierung innerhalb von Controller[36] Methoden, was in *Java EE* so direkt nicht möglich ist, sondern über die Java interne Bean Validierung gehandelt werden muss. Im Vergleich zu *Play* geschieht dies jedoch mit weniger Boilerplate-Code. *GWT* implementiert zwar auch die JSR-303 Spezifikation, allerdings wird die Implementierung nicht mehr weiterentwickelt.

Bei dem REST Error Handling wurde betrachtet wie die Frameworks mit Exceptions und Fehlern bei der Abarbeitung von HTTP Requests umgehen. Dabei ist *Spring* der Favorit, da durch Controller Advices[37] und Exception Annotationen die Controller Klassen klein und gut leserlich bleiben, wohingegen es bei *Java EE* mit deutlich mehr Boilerplate-Code gelöst wird. Bei den anderen Frameworks muss die Fehler- und Exceptionbehandlung manuell geschrieben oder durch das manuelle Erstellen von Middleware gehandhabt werden.

Bei der Betrachtung, wie produktiv mit den Frameworks Services entwickeln kann, ergab sich, dass alle Frameworks Möglichkeiten bieten schnell und einfach Services zu entwickeln. Das Framework,

[36] Ein Controller beschreibt im Kontext von *Spring* eine Klasse, welche mit der @Controller Annotation oder einem davon abgeleiteten Interface wie z. B. der @RestController Annotation versehen ist.

[37] Eine mit dem @ControllerAdvice annotierte Klasse dient als Controller Advice und implementiert über die @ExceptionHandler Annotation diverse Methoden, welche beim Wurf einer zutreffenden Exception in einem Controller aufgerufen werden.

das heraussticht, ist *Spring*, da durch *Spring Boot* die Konfiguration der Services für den Standardfall vollkommen automatisiert erfolgt und spezielle Änderungswünsche oder Typisierungen können leicht über Annotationen der betroffenen Klassen eingestellt werden. Somit muss keine unleserliche .xml Konfigurationsdatei geführt werden, welches den Wartungsaufwand minimiert und dem Entwickler viel Arbeit abnimmt.

Bei der Performance wurde der Schwerpunkt auf die Anzahl an HTTP Requests/Responses mit JSON/XML Serialisierung pro Sekunde gelegt, da die Serialisierung von Daten zwischen den einzelnen Komponenten z. B. Datenaustausch mit COSEM oder CIM eine wichtige Rolle haben wird. Zur JSON Serialisierung hat techempower.com[38] Ergebnisse eines umfangreichen Benchmarks zur Verfügung gestellt. In dem Benchmark wurden die JSON Responses pro Sekunde gemessen und es ist deutlich zu sehen, dass *Node.js* (372,531 Responses) und *Play 2* (240,174 Responses) am Besten abgeschnitten hatten. Jedoch sollte erwähnt werden, dass *Node.js* die JSON Daten nicht wirklich serialisieren muss, sondern direkt auf den JSON Daten arbeiten kann. Das Problem stellt in dem Fall das XML Format von CIM und COSEM dar (Siehe Unterunterabschnitt 6.1.6.7).

Bei der Datenbankanbindung der Frameworks bieten *Spring* und *Play* die besten Möglichkeiten an. *Spring* bietet mit dem *Spring Data Projekt*[39] die Möglichkeit über das Ableiten von Repositories von der zu Grunde liegenden Datenbank zu abstrahieren und unterstützt dabei alle gängigen Datenbanken, egal ob relationale, NoSQL oder graphenorientierte Datenbanken. Das *Play* Framework bietet ebenfalls verschiedene Treiber für gängige Datenbanken an, die Konfiguration ist allerdings umständlicher als bei *Spring Boot*. Bei den anderen Frameworks muss viel manuell und mit mehr Aufwand konfiguriert werden.

Bei der Unterstützung von Templating Engines und View Frameworks können in *Spring*, *Java EE* und *Play* die gängigsten Template Engines wie z. B. Thymeleaf und andere Frameworks angebunden werden. Dadurch, dass *Node* eine JavaScript Plattform ist, ist die Verbindung zu Viewframeworks oder Templating Engines sehr gut implementiert. Bei *GWT*/*Vaadin* gibt es durch *Vaadin* viele View-Elemente, die benutzt werden können, allerdings besteht dann eine Abhängigkeit zu diesen und das Einbinden von Templates oder JavaScript Komponenten wird umständlich.

Bei der Verbreitung der Frameworks sind *Spring*, *Java EE* und *Node* die Spitzenkandidaten. *GWT* bzw. *Vaadin* wird zwar noch verwendet, jedoch werden so gut wie keine neue Projekte mit diesen Frameworks entwickelt, da z. B. die Weiterentwicklung von *GWT* stark abnimmt. Wie bereits weiter oben in dem *Play* Abschnitt erwähnt wurde, ist *Play* deutlich weniger verbreitet als die anderen Frameworks.

Bei dem Aufwand für die Einarbeitung schneiden die Frameworks ähnlich gut ab. Der Vorteil von *Spring* ist die sehr ausführliche Dokumentation der *Spring* Projekte mit denen der Start mit dem Framework stark erleichtert wird[40]. Der Nachteil von *Node* ist, dass die Entwicklung mit JavaScript geschieht und dafür müssten sich einige Mitglieder der Projektgruppe zunächst in JavaScript einarbeiten.

Bei dem Punkt Security wurden die Frameworks danach untersucht, wie sie Sicherheitsfeatures implementieren und benötigte Konzepte wie Sessions und Nutzervalidierung bereitstellen. In diesem Punkt ist erneut *Spring* der Favorit, da durch das *Spring Security* Projekt[41] eine umfangreiche Security

[38] Online: `https://www.techempower.com/benchmarks/`, letzter Zugriff: 25.05.2017.

[39] `http://projects.spring.io/spring-data/`

[40] Das „Hello World“ Beispiel für einen Rest Service in *Spring Boot* bedarf nach Angaben der Spring Entwickler, für Java-Programmierer, rund 15 Minuten. Online: https://spring.io/guides/gs/spring-boot/, letzter Aufruf: 25.05.2017

[41] `https://projects.spring.io/spring-security/`

Suite geliefert wird. Bei den anderen Frameworks müssen entweder Teile oder alles selbst implementiert werden oder die Konfiguration ist im Vergleich zu dem *Spring Security* Projekt schwieriger.

Der vorletzte Punkt, anhand dessen die Frameworks verglichen wurden, ist die Serialisierung von Objekten bei HTTP Requests. Hier ist durch die standardmässige Integration von Jackson[42] *Spring* wieder der Favorit. Somit können zu übertragende Objekte, per Java Klassentyp als Parameter in Controllermethoden übergeben und auch als Rückgabewert angegeben werden. Die Serialisierung/Deserialisierung zwischen Java und JSON oder Java und XML[43] übernimmt *Spring* automatisch. *Node* hat den Vorteil, dass nativ mit JSON gearbeitet werden kann. Allerdings muss für die Arbeit mit XML auf externe Bibliotheken zurückgegriffen werden. Bei den anderen Frameworks muss die Serialisierung explizit durchgeführt werden, wodurch viel Boilerplate-Code entsteht.

Der letzte Punkt ist die automatische Erstellung der Dokumentation für die REST API. Diese funktioniert durch die Einbindung von Swagger[44] bei allen Frameworks bis auf *GWT*/*Vaadin* erstklassig. Die Benutzung von Swagger mit *GWT*/*Vaadin* ist schwieriger, da durch die abnehmende Weiterentwicklung die Anbindung erschwert wird. Das Testen der REST API ist ebenfalls durch Swagger möglich, da mit Swagger-UI zu der automatisch generierten Dokumentation eine Oberfläche bereitgestellt wird, in der alle REST Calls händisch getestet werden können.

6.1.7 Fazit der Evaluation

Die Evaluation der Tools ergab, dass sich für das Viewframework für Angular entschieden wurde, für die Visualisierung wurde beschlossen D3 mit OpenLayers zu verwenden und als Buildtools werden Gradle und webpack genutzt. Außerdem wird als Datenbanksystem MariaDB und bei Bedarf MongoDB eingesetzt und als Backend Web Framework wird Spring herangezogen.

6.2 Continuous Integration Umgebung

Zur Durchführung des Projektes wurden zu Beginn drei Virtuelle Maschinen bereitgestellt. Die leistungsstärkste virtuelle Maschine dient im Projekt als sogenanntes Demosystem. Das bedeutet, dass dort der aktuellste Entwicklungsstand vom System deployed wird. Dieser ist dann dort zu Demonstrations- und Testzwecke erreichbar. Dem Demosystem sind 4 Kerne des Intel Xeon E5645 zugeordnet und 16 GB RAM.

Damit das System überhaupt lauffähig ist stellt die Demo-VM eine reihe von wichtigen Diensten bereit:

Apache HTTP Server Auf dem Demosystem ist Apache HTTP Server in der Version 4.4.18 installiert. In erster Linie wird Apache HTTP als Proxyserver benötigt.

Apache Kafka Ebenfalls auf dem Demosystem befindet sich Apache Kafka. Der Massagebroker stellt einen essenziellen Bestandteil unseres Systems dar. Auf dem Demosystem befindet sich Kafka

[42] `https://github.com/FasterXML/jackson-core`

[43] Um die Serialisierung/Deserialisierung von Java und XML nutzen zu können muss die jackson-dataformat-xml library von com.fasterxml.jackson.dataformat nachgeladen werden.

[44] `http://swagger.io/`

Bewertungskriterien	Validierung	REST Error Handling	Services	Performance	Datenbankanbindung	Templating	Verbreitung	Einarbeitungsaufwand	Security	Serialisierung (JSON & XML)	auto. API Doku. und Test
Validierung		1	0	0	0	2	2	1	0	0	1
REST Error Handling	1		0	1	0	2	2	1	0	0	1
Services	2	2		2	1	2	2	1	1	1	2
Performance	2	1	0		0	2	1	1	0	0	2
Datenbankanbindung	2	2	1	2		2	0	0	1	1	0
Templating	0	0	0	0	0		1	0	0	0	0
Verbreitung	0	0	0	1	2	1		1	0	0	0
Einarbeitungsaufwand	1	1	1	1	2	2	1		0	0	0
Security	2	2	1	2	1	2	2	2		1	2
Serialisierung (JSON & XML)	2	2	1	2	1	2	2	2	1		2
auto. API Doku. und Test	1	1	0	0	2	2	2	2	0	0	
Summe	13	12	4	11	9	19	15	11	3	3	10
Gewichtungsfaktor	0,12	0,11	0,04	0,10	0,08	0,17	0,14	0,10	0,03	0,03	0,09

												Summe
Spring	Bewertung	10	10	10	4	10	9	10	10	9	10	10	
	Teilnutzwert	1,18	1,09	0,36	0,40	0,82	1,55	1,36	1,00	0,25	0,27	0,91	**9,20**
JavaEE	Bewertung	9	9	8	6	5	9	10	9	7	5	10	
	Teilnutzwert	1,06	0,98	0,29	0,60	0,41	1,55	1,36	0,90	0,19	0,14	0,91	**8,40**
GWT/Vaadin	Bewertung	7	1	1	1	4	5	5	9	4	5	1	
	Teilnutzwert	0,83	0,11	0,04	0,10	0,33	0,86	0,68	0,90	0,11	0,14	0,09	**4,18**
Play	Bewertung	8	7	9	8	7	9	3	9	4	6	10	
	Teilnutzwert	0,95	0,76	0,33	0,80	0,57	1,55	0,41	0,90	0,11	0,16	0,91	**7,45**
Node.js	Bewertung	1	3	9	10	5	10	10	7	7	8	10	
	Teilnutzwert	0,12	0,33	0,33	1,00	0,41	1,73	1,36	0,70	0,19	0,22	0,91	**7,29**

Abbildung 6.6: *Nutzwertanalyse verschiedener Backend Frameworks*

innerhalb der Confluent Plattform. Die Confluent Plattform wurde ausgewählt, da diese eine Reihe von Features bereitstellt, welche Kafka von Haus aus nicht beherrscht. Darunter befindet sich zum einen die Möglichkeit Datensenken und Datenquellen für Kafka zu definieren.

Docker Docker ist auf der Demo-VM installiert und erlaubt es auf einfache Art und Weise, auf der virtuellen Maschine die selben Bedingungen zu schaffen, wie auf den Entwicklermaschinen. Somit ist es möglich das identische Produkt auf den Entwicklermaschinen und der Demo-VM auszuführen.

Jenkins Jenkins wurde auf zwei VMs installiert. Einmal auf der CA-VM und einmal auf der CI-VM, wobei die CI-VM als Slave konfiguriert wurde. Es dient in erster Linie zur Vereinfachung des Continuous Integration - Prozesses.

MariaDB Das Datenbankverwaltungssystem MariaDB befindet sich ebenfalls auf dem Demosystem und dient in erster Linie dem Persistieren der Benutzerdaten und der Anfrage-Templates für Odysseus. Ebenfalls wird das Datenbanksystem zum Speichern der Beitrage des wöchentlich stattfindende Starfishs, im Rahmen des Scrum-Prozesses, benutzt.

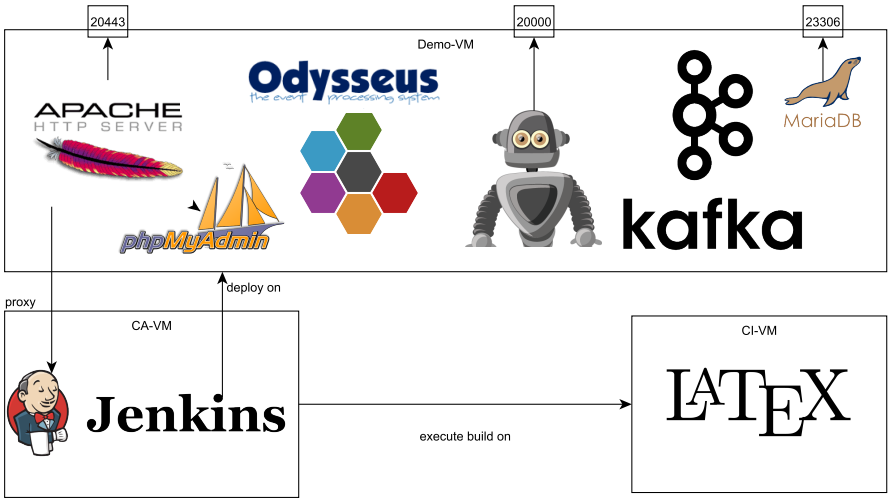


Abbildung 6.7: *Verteilung der Infrastruktur*

Odysseus Neben Apache Kafka ist Odysseus ein weiterer wichtiger Bestandteil unseres Produktes. Odysseus befindet sich ebenfalls auf dem Demosystem und bietet uns die Möglichkeit der Datenstromverarbeitung. Anpassungen an Odysseus, welche vorgenommen wurden, sind unter anderem die Installation folgender Features:

- REST
- PMML
- Kafka
- IEC 62056
- Web
- Mining

Scrumbot Der Scrumbot ist entstanden, um den Scrumprozess zu vereinfachen. Über Slack ist der Bot erreichbar und nimmt über ein Menü Punkte auf, welche im Starfish angesprochen werden. Dies geschieht anonym, lediglich die Einträger je Benutzer werden gezählt.

6.3 Toolbeschreibungen

6.3.1 Git

Um einen sauberen Projektablauf zu ermöglichen, wird ein Tool zur Versionskontrolle benötigt. Die Entscheidung, hierfür Git zu verwenden, basiert insbesondere darauf, dass Git ein verteiltes Versionskontrollsystem ist und Subversion (SVN) nicht. Andere Alternativen wurden nicht betrachtet, da sie nicht genug verbreitet sind und damit nicht in dem Umfang unterstützt werden, wie es bei Git und SVN der Fall ist. Des Weiteren bietet Git umfangreichere Funktionalitäten und erlaubt es den Entwicklern Änderungen am Projekt vorzunehmen, auch wenn sie im Moment keinen Zugriff auf das Hauptrepository haben, da jeder Entwickler auf einer eigenen, lokalen Kopie des Repositories arbeitet [66]. Zudem werden im Laufe der Projektgruppe voraussichtlich auch Änderungen am Projekt Odysseus vorgenommen werden müssen, bei welchem ebenfalls Git zur Versionskontrolle verwendet wird. Die Benutzung von Git für das Projekt E-Stream erhält die Einheitlichkeit und erspart den Gruppenmitgliedern das Einarbeiten in ein zweites Versionskontrollsystem.

6.3.1.1 GitFlow

Git bietet viele Möglichkeiten den Workflow eines Projektes zu gestalten. Ziel dieser Workflows ist die saubere Strukturierung der Arbeit, um die Entwicklung des Projektes einfacher nachvollziehen zu können. Des Weiteren kann die Struktur der Versionskontrolle die Iterationen der agilen Entwicklung widerspiegeln und damit den Entwicklungsprozess unterstützen. Ein Workflow, der diese Ziele unterstützt ist GitFlow. Er wurde im Jahr 2009 vom holländischen Entwickler Vincent Driessen ausgearbeitet und hat sich in verschiedenen Projekten bewährt. Driessen veröffentlichte seinen Workflow ein Jahr später auf seiner Website. Seitdem gewann der Workflow an Beliebtheit und wurde von zunehmend mehr Unternehmen und Personen verwendet. Abbildung 6.8 zeigt eine Übersicht des GitFlow Workflows [70].

GitFlow gibt ein Branching-Modell vor, das seinen Fokus auf die Sauberkeit von Releases setzt und ist damit gut für große Projekte geeignet. Des Weiteren werden Rollen an die Branches vergeben, die genau definieren wie diese mit anderen Branches interagieren dürfen und von wo sie abgezweigt werden[45]. Wie in den meisten Workflows wird ein zentrales Repository (*origin*) erstellt, das für alle Entwickler verfügbar persistiert wird. Jeder Entwickler arbeitet auf seiner lokalen Kopie dieses Repositories und veröffentlicht auf das zentrale Repository [17].

Die Branch-Struktur beinhaltet zwei permanente Branches, *master* und *develop*. Der *master-Branch* stellt immer den aktuellen Release dar und sollte zu jeder Zeit lauffähig sein. Der *develop-Branch* stellt den aktuellen Entwicklungsstand dar und beinhaltet alle fertigen Features, die im nächsten Release veröffentlicht werden sollen. Er dient als Integrations-Branch für Features und wird für nightly Builds benutzt [70]. Abbildung 6.9 zeigt die Grundstruktur eines GitFlow Repositories.

```
1 git checkout master
2 git branch develop
3 git checkout develop
4 git push -u origin develop
```

Listing 6.9: *Initialisierung von GitFlow*

[45] Abzweigen meint hier und im folgenden immer branchen (git branch)

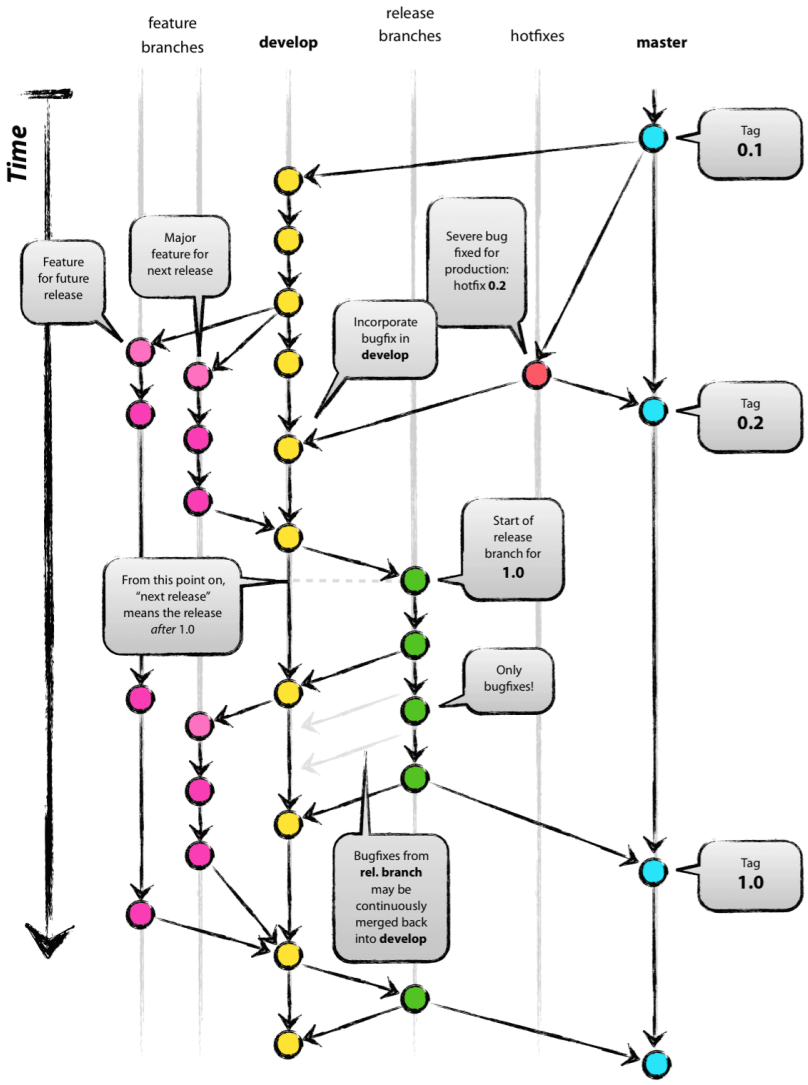


Abbildung 6.8: *Übersicht des GitFlow Workflows [70]*

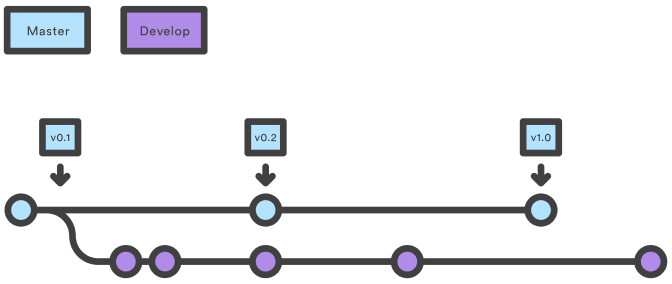


Abbildung 6.9: *Permanente Branches des GitFlow Workflows [17]*

Neben den beiden permanenten Branches gibt es drei Arten von Hilfs-Branches, *Feature-Branches*, *Release-Branches* und *Hotfix-Branches*. *Feature-Branches* werden immer vom *develop-Branch* abgezweigt und auch wieder in ihn integriert. Sie bekommen bei ihrer Erstellung ein Präfix, um sie als *Feature-Branch* zu kennzeichnen. Dieses Präfix ist nicht eindeutig festgelegt, im Weiteren Verlauf wird jedoch *feature/* als Präfix genutzt. Diese Art von Branch wird benutzt, um Features für einen zukünftigen Release zu implementieren. Dabei muss das Ziel nicht der nächste Release sein. Der *Feature-Branch* existiert so lange, bis er in den *develop-Branch* für den nächsten Release integriert wird oder als fehlgeschlagenes Experiment verworfen wird. Danach wird der Branch gelöscht. Typischerweise existieren die *Feature-Branches* nur in den lokalen Kopien des zentralen Repositories bei den Entwicklern und sollten nur veröffentlicht werden, wenn eine Kooperation mit anderen Entwicklern notwendig ist. Bei der Integration in den *develop-Branch* sollte beachtet werden, das *–no-ff* (No-Fast-Forward) Flag zu setzen, um einen Merge-Commit zu erzwingen. Dies hilft dabei die Commits zu identifizieren, die zu dem gleichen Feature gehören und ermöglicht so eine einfachere Übersicht im *develop-Branch* [70]. Abbildung 6.10 zeigt die Erweiterung der Grundstruktur um *Feature-Branches*.

```
1 git checkout -b feature/awesomeFeature develop
2 // Implement the feature
3 git checkout develop
4 git merge --no-ff feature/awesomeFeature
5 git branch -d feature/awesomeFeature
6 git push origin develop
```

Listing 6.10: *Feature-Branch workflow*

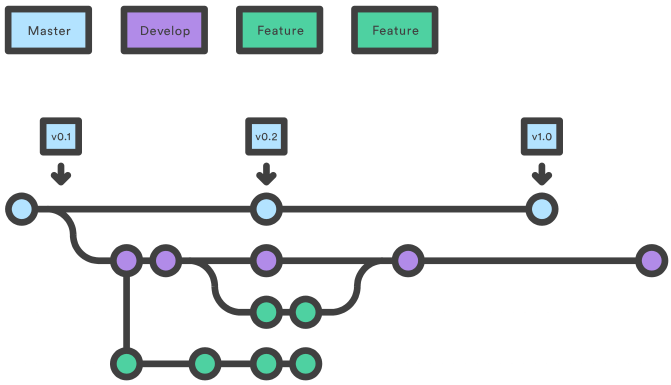


Abbildung 6.10: *Beispiel für Feature-Branches des GitFlow Workflows [17]*

Als nächstes werden die *Release-Branches* betrachtet. Sie werden immer von *develop* abgezweigt und sowohl in *master*, als auch in *develop* integriert, da mögliche Änderungen in den *Release-Branches* auch für *develop* relevant sind. Ihnen wird typischerweise das Präfix *release/* vorgestellt, um sie zu kennzeichnen. Ihr Ziel ist es, den Stand des aktuellen *develop* auf den nächsten Release vorzubereiten. Die Features bekommen einen Feinschliff bevor sie in den *master* integriert werden. Kleinere Bugs können behoben werden, Versionsnummern etc. können angepasst werden. Währenddessen kann auf dem *develop* weiter normal entwickelt werden. Ein *Release-Branch* sollte dann erstellt werden, wenn alle Features, die für den nächsten Release geplant sind, in den *develop* integriert wurden. Wenn

der *Release-Branch* produktionsreif ist, wird er in den *master* und den *develop* integriert, damit die Änderungen des Feinschliffs nicht verloren gehen, sondern auch im aktuellen Entwicklungsstand enthalten sind. Bei der Integration in den *master* wird der entsprechende Commit mit einer Versionsnummer versehen, die diesem Release zugeordnet wurde. Bei der Integration in den *develop* sollte beachtet werden, dass wahrscheinlich Konflikte auftreten, da z. B. die Versionsnummer nicht mehr übereinstimmt. Diese sollte auf dem *develop* übernommen werden. Nach der Integration wird der Branch gelöscht [70]. In Abbildung 6.11 wird der WorkFlow um einen *Release-Branch* erweitert.

```
1  git checkout -b release/1.2 develop
2  ./bumpVersion.sh 1.2
3  git commit -a -m "Bumped version to 1.2"
4  // Fix bugs, clean things up, ...
5  git checkout master
6  git merge --no-ff release/1.2
7  git tag -a 1.2
8  git checkout develop
9  git merge --no-ff release/1.2
10 git branch -d release/1.2
```

Listing 6.11: *Release-Branch workflow*

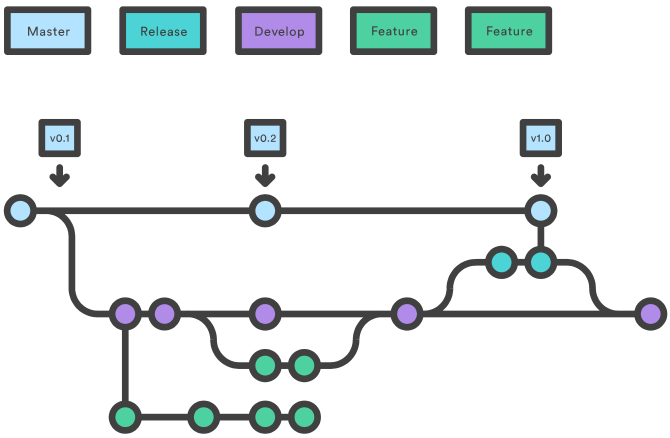


Abbildung 6.11: *Beispiel für einen Release-Branch des GitFlow Workflows [17]*

Die letzte Art von Hilfs-Branches sind die *Hotfix-Branches*. Sie werden vom *master* abgezweigt und sowohl in *master*, als auch in *develop* integriert. Ihre Aufgabe ist es, kritische Fehler in den Live-Versionen der Applikation zu beheben. Dazu werden sie vom Commit der entsprechenden Version abgezweigt und nach Beheben des Fehlers wie oben beschrieben wieder integriert. Hierbei muss beachtet werden, dass die Versionsnummer nach dem Abzweigen korrigiert wird. Falls es aktuell einen *Release-Branch* gibt, wird der Hotfix in den *Release-Branch* integriert anstatt direkt in *develop*. Da der *Release-Branch* nach seiner Fertigstellung in *develop* integriert wird, sind die Änderungen dort auch verfügbar. Falls der Hotfix für die Arbeit in *develop* essentiell ist, kann er auch direkt in den *Release-Branch* und *develop* integriert werden [70]. Die Erweiterung des GitFlow Workflows um *Hotfix-Branches* ist in Abbildung 6.12 zu sehen.

```
1  git checkout -b hotfix/1.2.1 master
2  ./bumpVersion.sh 1.2.1
3  git commit -a -m "Bumped version to 1.2.1"
4  // Fix bug
5  git checkout master
6  git merge --no-ff hotfix/1.2.1
7  git tag -a 1.2.1
8  git checkout develop
9  git merge --no-ff hotfix/1.2.1
10 git branch -d hotfix/1.2.1
```

Listing 6.12: *Hotfix-Branch workflow*

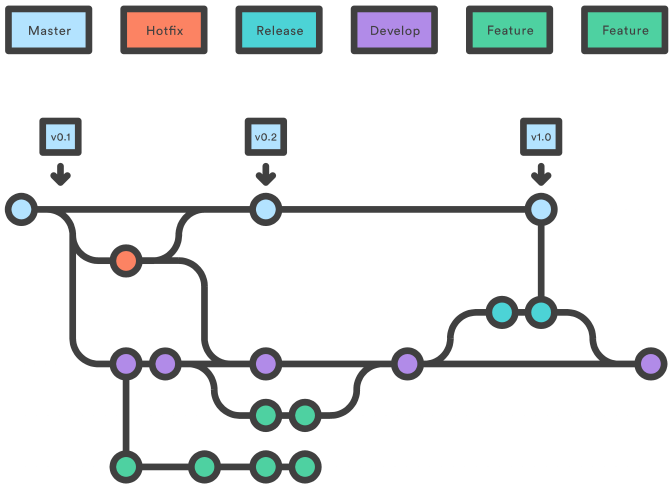


Abbildung 6.12: *Beispiel für einen Hotfix-Branch des GitFlow Workflows [17]*

Mit diesen Regeln formt GitFlow ein effektives Modell zur Verwaltung der Branches im Repository. Der Prozess vom einzelnen Feature über Bugfixes bis hin zu Releases ist strukturiert und für alle Teammitglieder transparent. Das Resultat ist ein sauberer Verlauf und eine gute Dokumentation des Arbeitsprozesses. Aufgrund dieser Vorteile wird das Modell in diesem Projekt angewandt.

Driessen beschreibt eine weitere Möglichkeit der Kooperation zwischen Entwicklern, in dem zusätzlich zum zentralen *origin-Repository* weitere *Remote-Repositories* angelegt werden, die jeweils auf ein lokales Repository eines anderen Entwicklers zeigen [70]. Diese Form der Kooperation wird hier jedoch nicht verwendet.

Da in diesem Projekt ein Review für jeden Task vorgesehen ist, muss das GitFlow Modell entsprechend angepasst werden. Die Erstellung der Branches bleibt unverändert und auch während der Bearbeitung wird nicht vom ursprünglichen Modell abgewichen. Nachdem der Task fertiggestellt wurde und bereit zum Review ist, muss der entsprechende Branch allerdings veröffentlicht werden, um dem Reviewer den Zugang zu ermöglichen. Der Reviewer merkt etwaige Fehler an oder korrigiert sie selbstständig, woraufhin die Korrektur mittels eines eigenen Commits eingepflegt wird. Dies

widerspricht dem Konzept dahingehend, dass die Branches normalerweise nur lokal beim Bearbeiter des Tasks existieren sollten und von dort aus direkt in den *develop*-Branch übernommen werden sollten. Eine Weitere Abweichung ist die Verwendung von *Pull-Requests*, um Änderungen in den Entwicklungsbranch zu übernehmen. Pull-Requests sind kein Git-Feature, sondern eine Erweiterung, die von vielen Git-Plattformen angeboten wird, um bessere Kontrolle der Branches zu ermöglichen. Dies hat ebenfalls zur Folge, dass die Branches veröffentlicht werden müssen, da sie dann auf der Git-Plattform verfügbar sind, um in das Projekt eingepflegt werden zu können. Eine Konsequenz hieraus ist, dass für die Branches gegebenenfalls ein Rebase ausgeführt werden muss, damit sich insgesamt eine sauberer, linearer Verlauf auf dem *develop* ergibt. Hierfür muss ein Force-Push verwendet werden, um den Remote-Branch nach dem Rebase zu überschreiben. Diese Aktion ist potentiell gefährlich, da durch sie, bei falscher Anwendung, Fortschritt auf dem Branch verloren gehen kann. Sie ist für die Durchführung des Arbeitsprozesses jedoch notwendig.

Einige Git-Clients, wie z. B. SourceTree von Atlassian unterstützen den GitFlow Workflow in ihrer Benutzeroberfläche [17]. Vincent Driessen selbst stellt ein CLI-Werkzeug bereit, das die GitFlow Funktionalitäten unterstützt [71]. Diese Werkzeuge erleichtern die Umsetzung des Workflows, sind aber nicht zwingend notwendig. Der GitFlow Workflow kann mit den Standard Git-Kommandos umgesetzt werden.

6.3.1.2 Interactive Rebase

Interactive Rebase ist eine Variation des Standard Rebase-Befehls. Der Zweck eines Interactive Rebase ist es, wie beim Standard Rebase Befehlt auch, den Verlauf der Versionskontrolle sauber zu halten und Konflikte zu vermeiden. Der Rebase-Befehl bietet eine Alternative zum Merge-Befehl. Beide dienen dazu, zwei Branches miteinander zu verbinden, realisieren dies jedoch unterschiedlich. In Abbildung 6.13 ist die Ausgangssituation für einen Merge bzw. Rebase zu sehen.

Ein Merge fasst alle Änderungen des einen Branches zusammen und erstellt einen Merge-Commit, um diese in den anderen Branch zu integrieren. Abbildung 6.14 zeigt den Verlauf nach einem Merge.

Ein Rebase hingegen wiederholt die Commits des einen Branches auf dem anderen und erstellt damit einen linearen Verlauf ohne einen überflüssigen Merge-Commit. Abbildung 6.15 zeigt den Verlauf nach einem Rebase.

Der Vorteil des Rebase-Befehls ist die Sauberkeit des Projektverlaufs, da keine unnötigen Merge-Commits generiert werden. Der Rebase-Befehl erstellt neue Commits und übertragt nicht einfach die alten Commits auf einen neuen Branch. Die goldene Regel des Rebasing lautet daher: „Never rebase while you're on a public branch“. Auf einem öffentlichen Branch, an dem mehrere Entwickler arbeiten sollte kein Rebase ausgeführt werden, da dies zu Konflikten bei allen anderen Entwicklern führt, da die alten Commits nicht mehr vorhanden sind [19].

Interactive Rebase ist ein nützliches Werkzeug, um den Verlauf der Versionskontrolle nachträglich zu ändern. Es bietet dem Entwickler unter anderem die Möglichkeit Commit-Nachrichten zu überarbeiten und Commits zusammenzufassen. Als Beispiel könnte davon ausgegangen werden, dass ein Entwickler auf seinem lokalen *Feature-Branch* ein neues Feature implementiert hat und bereit ist, dieses in *develop* zu integrieren. Kurz vor der Veröffentlichung bemerkt er jedoch, dass er vergessen hat den Code korrekt

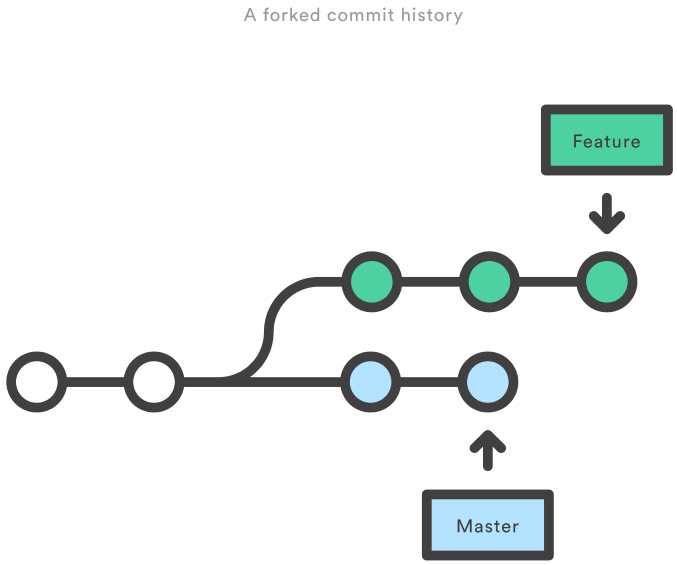


Abbildung 6.13: *Ausgangssituation für Merge bzw. Rebase [19]*

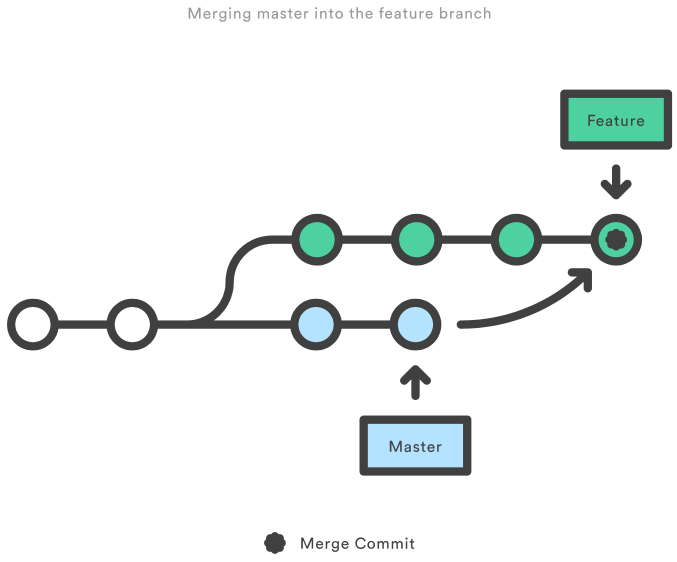


Abbildung 6.14: *Situation nach dem Merge [19]*

zu formatieren und außerdem würde er gerne die Commit-Nachricht eines Commits bearbeiten, damit klarer wird was dieser beinhaltet. Er erstellt einen neuen Commit, der die Formatierung des Codes enthält. Dieser Commit sagt im Verlauf des Projektes nichts über das Feature aus und macht den

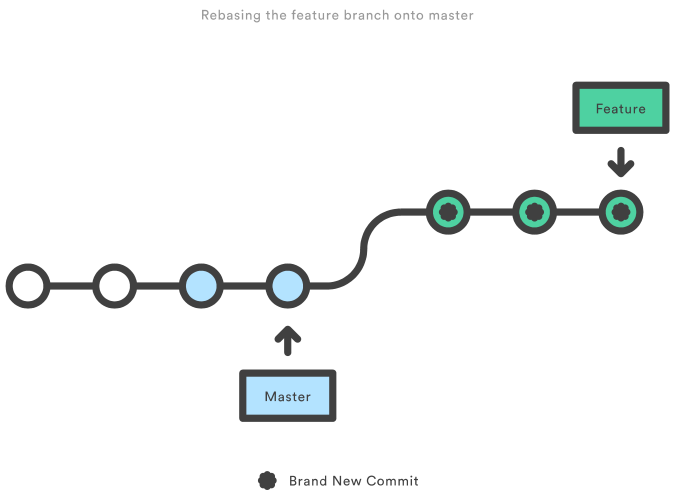


Abbildung 6.15: *Situation nach dem Rebase [19]*

Verlauf unübersichtlich. Mit dem Interactive Rebase Werkzeug kann er diesen Formatierungscommit in einen anderen Commit integrieren und die Commit-Nachrichten ändern. Der Verlauf spiegelt nun den Fortschritt des Features wieder [18].

Interactive Rebase bietet fünf Möglichkeiten zur Veränderung der Commits.

1. Zusammenfügen
2. Verändern des Inhalts
3. Verändern der Commit-Nachricht
4. Löschen
5. Neuanordnen

Zusammenfügen erlaubt dem Entwickler kleinere Commits, wie etwa das Formatieren des Programmcodes in andere Commits zu integrieren. Verändern des Commits erlaubt es unter Anderem Stashes auf den Commit anzuwenden. Das Löschen von Commits sollte nur in Ausnahmen geschehen, da es zu Problemen bei späteren Commits führen kann, die auf den Änderungen dieses Commits aufbauen. Das Neuanordnen vom Commits erlaubt es, die Reihenfolge der Commits zu ändern.

SourceTree unterstützt Interactive Reabse mit einer grafischen Benutzeroberfläche. Die einfachste Variante einen Interactive Rebase in SourceTree zu beginnen ist einen Anfangscommit zu wählen und *Rebase children of $SHA interactively* aus dem Kontextmenü auszuwählen [18].

6.3.2 Binary Repository

Wie im Unterabschnitt 6.3.1 beschrieben, eignet sich Git hervorragend für die Verwaltung und Versionierung von textuellen Dateien für den Entwicklungszyklus. Für die Verwaltung und das Veröf-

fentlichen von Binärdateien, wie zum Beispiel kompilierten Projektartefakten, zum Verwenden in Drittprojekten hingegen sollte ein Binary Repository verwendet werden. Die Verwendung eines Binary Repositories birgt einige Vorteile. So kann das Publizieren von Artefakten automatisiert werden. Alle publizierten Artefakt-Versionen sind zentralisiert zugreifbar und diese sind mit einer Vielzahl von bestehenden Build Tools (Siehe Unterabschnitt 6.1.4) inkrementell über die Versionsnummern in externe Projekte als Abhängigkeit einbindbar. Die meisten erhältlichen Binary Repositories werden zusammen mit einem Manager ausgeliefert, der die Verwaltung durch bereitgestellte Features erleichtert.

6.3.2.1 Technologien

Binary Repository Manager

Aktuelle Binary Repository Manager sind unter dem Begriff universelle Package Manager zu finden, darunter Apache Archiva, JFrog Artifactory, Inedo's ProGet und Sonatype Nexus. Sie unterscheiden sich dabei in einigen Eigenschaften, wie der Lauffähigkeit auf verschiedenen Betriebssystemen, der Speicherung der Artefakte, dem Publizieren von Plugins, uvm.[46]. Jedoch bieten sie alle die benötigten Eigenschaften eines Binary Repository, dem Publizieren von Artefakten im Bezug auf den Kontext der Arbeit. Darüber hinaus ist Archiva, Artifactory mit Artifactory OSS, ProGet mit ProGet Free, sowie Nexus mit dem Nexus Repository OSS in einer kostenlosen Variante verfügbar. Ein Nachteil der Binary Repository Manager in der kostenlosen Variante ist, dass diese selbst installiert, konfiguriert, gehostet und gewartet werden müssen. Somit fällt ein relativ großer Einarbeitungsaufwand an und eine hohe Verfügbarkeit kann nicht gewährleistet werden.

Package Repositories

Im Gegenzug zu den Binary Repository Managern existieren Drittanbieter Package Repositories, welche es durch die Angabe von öffentlichen, teilweise auch privaten Repositories ermöglichen automatisch Artefakte zu passenden Git Tags erstellen und veröffentlichen zu lassen. Diese können dann über die Angabe spezieller Repositories durch Build Tools in externe Projekte eingebunden werden. Ein bekannter Anbieter eines solchen Package Repositories ist JitPack[47], welches es ermöglicht JVM und Android Artefakte aus öffentlichen oder privat angegebenen GitHub bzw. Bitbucket Repositories zu kompilieren und versioniert zur Verfügung zu stellen.

6.3.2.2 Verwendung im Kontext der Projektgruppe

Da es sich bei der Entwicklung des Softwaresystems im Kontext der hier dokumentierten Arbeit zwar um ein projektgruppenübergreifendes System, jedoch mit abzählbaren Artefakten und Komponenten handelt, ist die Wahl der Technologie dem Aufwand der Einrichtung dem wirklichen Nutzen gegenüberzustellen, um so eine optimale Lösung zu finden.

Wie im Abschnitt 8.1 beschrieben und abgebildet, enthält das zu realisierende Softwaresystem Schnittstellen zu den externen Softwaresystemen der Projektgruppe DAvE (Big Data Archive) und Odysseus. Da es sich bei diesen Schnittstellen hauptsächlich um den Austausch von Daten und deren

[46] Für einen aktuellen Vergleich zwischen Binary Repository Managern und dessen Funktionen existiert die Binary Repository Manager Feature Matrix. Online unter: `https://binary-repositories-comparison.github.io/`. Letzter Zugriff: 17.09.2017

[47] Zu finden online unter: `https://jitpack.io/`. Letzter Zugriff: 17.09.2017

Struktur handelt, ist die Existenz eines einzigen Repositories für die auszutauschenden Daten und dessen inkrementelle Entwicklung ausreichend. Außerdem kann für die Anforderung an ein Binary Repository und dessen Einbindung in die externen Projekte festgestellt werden, dass Odysseus im aktuellen Stand kein Build Tool Support besitzt, weswegen dort Abhängigkeiten manuell installiert und aktualisiert werden müssen. Der Build Tool Support für Maven ist in Entwicklung und sollte daher bei der Technologiewahl unterstützt werden. Die Projektgruppe DAvE wird ihr System in der Programmiersprache Java entwickeln und als Build Tool Gradle bzw. Maven verwenden.

Zuletzt ist anzumerken, dass die hier beschriebene Arbeit unter keine lizenzrechtlichen Restriktionen fällt, somit entfällt für das Repository die Anforderung der Privatisierung, weshalb das Repository auf GitHub gehostet werden kann[48], um eine projekgruppenübergreifende Zusammenarbeit zu erleichtern.

Aus diesen Anforderungen und besonders im Bezug auf die Wichtigkeit der minimalen Einarbeitungszeit aller Projektbeteiligten ist JitPack für diesen Kontext die geeignetste Wahl. Somit kann der Vorteil von JitPack genutzt werden, dass das veröffentlichen eines neuen Artefakts direkt mit dem Build Tool Zugriff auf eine neue Release-Version über einen Git Tag oder den Commit-Hash automatisch geschieht. Außerdem wird die Einarbeitungs-/Konfigurationszeit aller beteiligten Entwickler minimiert, da durch die nicht vorhandene Authentifizierung keine lokale Konfiguration für den Zugang angelegt werden muss und eine Aktualisierung eines benötigten Artefakts in dem externen Projekt ist einfach durch das Inkrementieren der Versionsnummer bzw. Angabe des Commit-Hashs in der Build Tool Konfiguration möglich. Die publizierten Artefakte des öffentlichen Repositories inklusive Version und Integration in die Build Tools sind online einsehbar unter `https://jitpack.io/#PG-EStream/shared`.

6.3.3 Apache Kafka

Apache Kafka ist eine ursprünglich von LinkedIn entwickelte Software zur Verarbeitung von Datenströmen. Der ursprüngliche Zweck der Entwicklung lag dabei in der verteilten Verarbeitung extrem großer Mengen an Protokolldateien, welche gesammelt, ausgewertet und wieder verteilt werden sollten [157]. Hieraus ergibt sich auch ein Großteil des heutigen Funktionsumfangs von Apache Kafka, welcher insbesondere aus einer sehr guten Skalierbarkeit, der schnellen Verarbeitung von Daten ungeachtet des Datentyps sowie einem sehr hohen Durchsatz besteht.

Die von Kafka verwalteten Daten bestehen dabei aus einzelnen Nachrichten, welche von einem sogenannten *Producer* an Kafka gesendet und anschließend von einem *Consumer* verarbeitet werden. Die Kommunikation zwischen Producer und Consumer wird dabei über sogenannte *Topics* realisiert, welche einen thematischen Zusammenhang zwischen den Nachrichten herstellen. Abbildung 6.16 stellt die Kommunikation zwischen Producer und Consumer über einen Apache Kafka-Server schematisch dar. Dabei verwaltet Kafka die Topics und die Zuordnung der Daten selbst, sodass Producer und Consumer einzig die Bezeichung des Topics angeben müssen.

Einer der großen Vorteile von Kafka ist dabei, dass alle Nachrichten für eine gewisse Zeit persistent gespeichert werden. Dies wiederum ermöglicht eine asynchrone Ausführung von Producer und Consumer, was einer der Gründe für den Einsatz von Apache Kafka in der Projektgruppe ist. Darüber hinaus können mehrere Kafka-Server als Cluster betrieben werden, sodass das System auch bei steigenden

[48] `https://github.com/PG-EStream/shared`

Datenmengen keine Kapazitätsgrenzen hat, was zumindest ein theoretischer Anwendungsfall in der Projektgruppe ist.

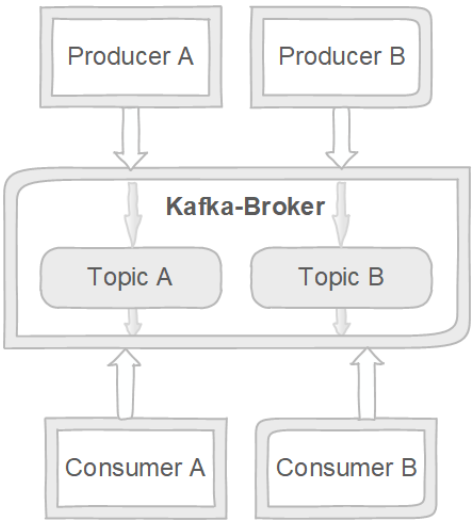


Abbildung 6.16: *Schematische Darstellung der Apache Kafka Kommunikation*

6.3.3.1 Zugriffsschutz in Apache Kafka

In der Standardkonfiguration sind in Apache Kafka keinerlei Authentifizierungsmechanismen aktiviert, da der ursprüngliche Anwendungszweck von Kafka keinen Zugriff aus dem Internet vorsieht. Jedoch bietet Kafka inzwischen mehrere Möglichkeiten einen Zugriffsschutz umzusetzen. Zur Authentifizierung von Benutzern ermöglicht Kafka seit Version 0.9 im Wesentlichen drei Mechanismen: TLS-Authentifizierung, die Verwendung von Kerberos sowie eine Authentifizierung über Benutzernamen und Passwort [160].

Die TLS-Authentifizierung, welche in der Kafka-Dokumentation sowie in den Konfigurationsdateien von Kafka noch nach dem inzwischen abgelösten SSL-Standard benannt ist, funktioniert im Wesentlichen über Zertifikate und Schlüssel zur Verschlüsselung der Daten. Es erhält also jeder Kafka Broker und jeder logische Client (z. B. Kafka Producer) ein Zertifikat sowie einen Schlüssel. Zudem werden sogenannte Truststores eingerichtet, in welchen hinterlegt wird, welche Zertifikate als vertrauenswürdig gelten. Beim Verbindungsaufbau können nun die vorhandenen und die im Truststore hinterlegten Zertifikate überprüft und die Verbindung erlaubt oder abgelehnt werden.

Kerberos und die Authentifizierung via Benutzername und Passwort werden in Apache Kafka über SASL (Simple Authentication and Security Layer) unterstützt. Innerhalb von SASL wird Kerberos über das Generic Security Service Application Program Interface (GSSAPI) unterstützt und ist selbst ein Protokoll eines verteilten Dienstes zur Authentifizierung, welches als eigener Server-Dienst angeboten werden muss. Dann dient der Kerberos-Dienst als Authentifizierungsstelle für den Client gegenüber dem Server sowie umgekehrt.

Die dritte von Apache Kafka unterstützte Möglichkeit zur Authentifizierung ist die Verwendung von Benutzernamen und Passwort, welche ebenfalls über SASL angeboten wird. Hier wird sie als PLAIN bezeichnet und Benutzername sowie Passwort im Klartext übertragen. Das bedeutet, für einen

wirklichen Schutz der Daten muss die Kommunikation über eine gesicherte/verschlüsselte Verbindung laufen. Daher ist PLAIN zwar zunächst unsicherer als die anderen Möglichkeiten, dafür jedoch auch deutlich einfacher in der Implementierung. Benutzernamen und die zugehörigen Passwörter müssen jedoch dem Kafka-Server über eine JAAS-Datei bekannt gemacht werden.

Der Aufwand der Einrichtung insbesondere für die Verwendung von Zertifikaten, aber auch Kerberos, ist sehr groß, da in beiden Fällen nicht nur der Kafka-Server selbst, sondern auch jeder Client entsprechend angepasst werden muss. Im Fall der Kerberos-Authentifizierung wäre zudem die Einrichtung eines zusätzlichen Server-Dienstes notwendig. Jedoch kann davon ausgegangen werden, dass keine der drei Maßnahmen für einen Produktivbetrieb im Kontext eines Netzbetreibers als ausreichend sicher angesehen werden kann und hier stattdessen vollständig auf eine Übertragung der Daten über das Internet (ohne die Verwendung zusätzlicher Protokolle) verzichtet würde.

Das Rechtemanagement kann in Apache Kafka über Access Control Lists (ACLs) erfolgen. Hierbei kann für jeden Nutzer festgelegt werden, welche Operationen dieser auf welchem Host und welchen Ressourcen (z. B. Topics) er ausführen kann. Auch entsprechende Zugriffs- und Verwaltungsrechte für verschiedene Broker in einem Cluster sowie ihrer Kommunikation untereinander lassen sich dabei konfigurieren.

ACLs lassen sich dabei auch verwenden um einzelne Systembereiche voneinander abzugrenzen. So wäre es möglich einem Consumer nur lesenden Zugriff auf ein Topic mit Daten zu geben, wodurch verhindert werden kann, dass die Datenintegrität dieses Topics durch eine Fehlkonfiguration in diesem Teil des Systems gefährdet wird. Es ist jedoch auch möglich Super-User anzulegen, welche sämtliche Operationen auf sämtlichen Ressourcen ausführen dürfen.

6.3.4 Vagrant

Moderne Softwareentwicklung wird durch die verschiedenen möglichen Konfigurationen von Entwicklungsumgebungen erschwert. Selbst wenn ein System auf einem bestimmten Produktionssystem läuft, geschieht die Entwicklung des Systems häufig mit einer anderen Entwicklungsumgebung. So läuft innerhalb der Projektgruppe das System unter Linux in der Produktionsumgebung, es wird jedoch unter Windows, Mac OS und Linux entwickelt. Damit möglichst wenig Fehler bedingt durch verschiedene Betriebssysteme oder Softwareversionen entstehen, ist es notwendig eine einheitliche Entwicklungsumgebung zu schaffen in der die Entwicklung stattfindet. Wenn darüber hinaus die lokale Entwicklungsumgebung identisch mit der des Produktionssystems ist, kann der Fehler vermieden werden, dass Unterschiede zwischen der lokaler Entwicklungsumgebung und dem Produktionssystem entstehen. Eine mögliche Lösung dafür ist es die für das System benötigte Infrastruktur in einer virtuellen Maschine bereitzustellen, die das gleiche Betriebssystem und die gleichen Softwarepakete wie das Produktionssystem hat. Um die Inbetriebnahme der virtuellen Maschine zu erleichtern und zu automatisieren, ohne dass man das gesamte Image der virtuellen Maschine zur Verfügung stellen muss, wurde mit Vagrant eine Lösung gefunden, die genau dies ermöglicht.

Vagrant ist ein Kommandozeilenprogramm, das die Verwaltung von virtuellen Maschinen erleichtert [129]. Vagrant benutzt als Hauptkomponente sogenannte Boxes, die eine virtuelle Maschine repräsentieren. Diese Boxes können individuell erstellt oder es kann eine von den vielen bereits zur Verfügung gestellten Boxen benutzt werden. Letzteres ist in der Regel einfacher, da diese Boxen meist schon vorkonfiguriert sind mit zum Beispiel bereits angelegten Nutzerprofilen und häufig verwendeter Software.

Die Konfiguration Vagrants findet in einer Vagrantfile statt. Dort kann angegeben werden, welche Box installiert und benutzt werden soll. Weiterhin ist es möglich Einstellungen wie Netzwerkkonfiguration für die virtuellen Maschine, virtuellen Speicher und Vorschriften (Provisions) in der Vagrantfile zu definieren. Provisions sind Automatisierungstechniken von Softwareinstallationen innerhalb der Box, die bei der Initialisierung einer Box ausgeführt werden. Dies reicht von Shell-Skripten bis hin zu Konfigurationen über Configuration Management Tools. Auf diesem Weg können benötigte Softwarepakete oder Programme installiert, falls notwendig neue Nutzer angelegt oder weitere Veränderungen an der virtuellen Maschine vorgenommen werden, die bei der erstmaligen Inbetriebnahme notwendig sind.

In der Projektgruppe wurde Vagrant zunächst dafür benutzt, eine einheitliche Entwicklungsumgebung für alle Gruppenteilnehmer zu schaffen. Dafür wurde eine Ubuntu 16.04 Box mit einem VirtualBox Provider eingerichtet. Beim Erstellen der Box wurden MariaDB, Odysseus und die Confluent Platform installiert und eingerichtet. Dieses brachte allerdings einige Nachteile mit sich. Bei Änderungen der Infrastruktur war es notwendig die eingerichtete Box zu entfernen und neu aufzusetzen. Die Dauer davon hängte stark von der Geschwindigkeit der Internetverbindung ab. Außerdem kam es zu Fehlern mit der Benutzung von Vagrant unter Mac OS und mit einigen Versionen von Vagrant in Kombination mit VirtualBox unter Windows. Ebenfalls war der große Arbeitsspeicherverbrauch ein großer Nachteil. Um die Infrastruktur für das System zu starten waren 8GB RAM notwendig.

Um die genanten Nachteile zu umgehen, fand für die Bereitstellung der Infrastruktur ein Wechsel auf Docker statt. Vagrant wird weiterhin benutzt, um Probleme unter Windows und MacOS mit Docker und der Confluent Platform zu vermeiden. Dafür wird eine Ubnutu 16.04 Linux Box benutzt, in der bei dem Provision Schritt Docker und Docker Compose installiert werden. Anschließend kann über ein Bash-Skript und Docker Compose die benötigte Infrastruktur für das gesamte System hochgefahren werden, was die Zeit für die Initialisierung der virtuellen Maschine deutlich reduziert.

6.3.5 Docker

Wie zuvor im Unterabschnitt 6.3.4 Vagrant beschrieben, wird im Idealfall bei der Softwareentwicklung die gleiche Umgebung für das Entwickeln, Testen und die Bereitstellung des Systems benutzt. Um dieses zu erreichen ist die Arbeit mit virtuellen Maschinen ein möglicher Ansatz, der, wie bereits erwähnt, Nachteile wie großen Speicherverbrauch und die Dauer für die Instanziierung einer Maschine mit sich bringt. Um diese Nachteile zu vermeiden und dennoch eine einheitliche Umgebung zu schaffen, sind Linux-Container eine Alternative zu virtuellen Maschinen.

Linux-Container sind Softwareprozesse, die in einem Betriebssystem isoliert vom Rest des Betriebssystems laufen [142]. Der Hauptunterschied zwischen virtuellen Maschinen und Linux-Containern ist, dass ein Container nicht ein vollständiges Gast-Betriebssystem benötigt. Stattdessen greifen Container auf den Kernel des Host-Betriebssystems zu und lassen anschließend benötigte Bibliotheken und die bereitgestellten Applikationen in dem Container isoliert laufen. Dieser Unterschied zwischen virtuellen Maschinen und Containern ist in Abbildung 6.17 dargestellt. Dadurch dass Container auf den Kernel des Host-Betriebssystems zugreifen können, sind sie schneller in der Initialisierung und benötigen weniger Ressourcen. Denn im Gegensatz zu einer virtuellen Maschine muss kein vollständiges Betriebssystem gestarten werden. Außerdem wird in den Containern die Menge an installierten und laufenden Anwendungen und Bibliotheken nur auf das vom Container benötigte reduziert.

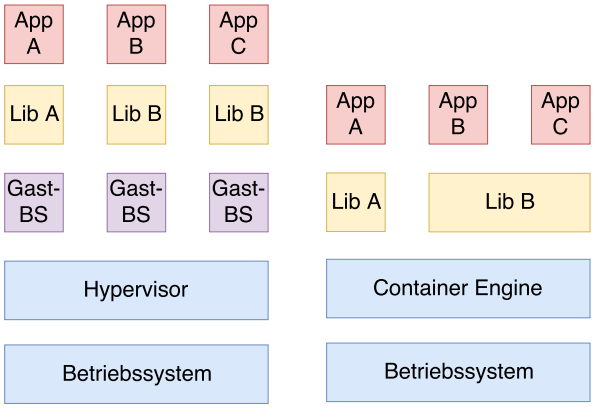


Abbildung 6.17: *Vergleich zwischen virtuellen Maschinen und Linux-Containern beim Bereitstellen von Software*

Ein weiterer Vorteil bei der Benutzung von Containern ist, dass die Definition der Container und ihrer API standardisiert ist. In der Open Container Initiative [150] sind Unternehmen wie Red Hat, IBM, Oracle und Docker vertreten und arbeiten gemeinsam an Container und Image Spezifikationen. Durch eine einheitliche Container Runtime API und Image Spezifikation ist es möglich unabhängig von einer bestimmten Container-Engine Images in den Containern auszuführen. Um die Arbeit mit Containern und Images zu erleichtern und automatisieren, wurde Docker entwickelt.

Docker besteht hauptsächlich aus der Docker Engine und Docker Hub. Die Docker Engine ist eine Kombination aus einem Kommandozeilenprogramm und einem Linux Daemon, die zusammen interagieren, um die Arbeit mit Containern zu vereinfachen. Docker Hub ist eine Cloud Plattform, die die Bereitstellung von Images erleichtert.

Durch eine Dockerfile ist es möglich deklarativ zu beschreiben wie ein Image gebaut werden soll. In dieser Dockerfile können Base Images definiert werden, die als Grundlage für das zu erstellende Image dienen soll und die man anschließend individuell je nach Anforderungen anpassen kann. Diese Images werden in der Regel aus einem Docker Repository geladen, wie zum Beispiel Docker Hub. Nachdem ein Base Image feststeht können über Befehle wie *ADD* oder *COPY* Konfigurations- oder Applikations-Dateien hinzugefügt werden und unter Angabe von *RUN* oder *CMD* kann beschrieben werden, was bei der Ausführung des Images in einem Container geschehen soll.

Innerhalb des Systems wird Docker für die Bereitstellung der Infrastruktur als Ersatz zu einer reinen Vagrant Lösung benutzt. Dafür werden Images für MariaDB, Odysseus, Bestandteile der Confluent Platform, sowie mosaik erstellt. Dadurch dass Docker bereits benutzte Images in einem Cache vorhält, beschleunigt es die Arbeit mit der Infrastruktur, weil so einzelne Teile der Infrastruktur in Isolation bearbeitet werden können. Mit Vagrant müssten Programme in der virtuellen Maschine entweder per Hand verändert werden oder es muss die gesamte Box zerstört und neu instantiiert werden.

Mit Docker ist es möglich einen Container zu löschen und einen neuen Container mit dem jeweiligen Image zu bauen oder bei Bedarf das Image anpassen und anschließend das neue Image in einem Container starten. Die restlichen Container sind von den Änderungen dabei nicht betroffen. Ein weiterer Vorteil davon ist, dass man Container nach Bedarf hinzufügen kann. Dieses kann sowohl für die Skalierbarkeit des Systems als auch für Lasttests des Systems benutzen werden.

Weiterhin ist es möglich bei dem Deployment des Systems auf die gleichen Container zuzugreifen, die während der Entwicklung benutzt werden und diese auf dem System für externe Interaktionen öffentlich zu machen. Dafür muss nur die Konfiguration der Images überschrieben werden, so dass sich die Images zwischen Entwicklung und Produktion nur in Eigenschaften wie zum Beispiel den benutzten Ports unterscheidet.

Ein Problem das bei der Entwicklung mit Docker entstanden ist, ist dass die Confluent Platform Images nicht unter MacOS und Windows in Containern unterstützt wird. Dabei ist das Problem aufgetreten, dass Images der Confluent Platform nicht miteinander kommunizieren konnten. Deshalb wird in dem System eine Kombination von Vagrant und Docker benutzt, indem in Vagrant ein Linux Betriebssystem ausgeführt wird, das Docker und Docker Compose installiert hat und die Images in Containern ausführt.

6.3.6 Swagger

Swagger[49] ist ein Framework mit einer großen Anzahl an unterstützenden Tools für das Designen, Dokumentieren, Erstellen, Testen und Konsumieren von RESTful Webservices. Dabei verwendet es für alle Tools und das Framework die OpenAPI Specification (OAS). Die drei bekanntesten und wichtigsten Tools sind die verschiedenen Versionen des Swagger Editors, der Swagger UI und des Swagger Codegen.

Der Swagger Editor (zu sehen in Abbildung 6.18) dient dazu, per Swagger Syntax oder OpenAPI Specification REST Schnittstellen zu definieren und zu bearbeiten. Dabei geschieht dies entweder im yaml Format, bei dem die Einrückung beachtet werden muss oder im json Format. Die aktuellen Varianten v3 und auch v2 des Swagger Editors stellen über die Navigationsleiste die Funktionalitäten der jeweils aktuellen Version des Swagger Codegens zur Verfügung. Innerhalb des Projektes wird der Swagger Editor lokal bzw. online dazu genutzt, die einheitliche Schnittstellendefinition des PG DAvE und PG E-Stream HTTP Servers zu definieren. Der Swagger Codegen dient zur Generierung

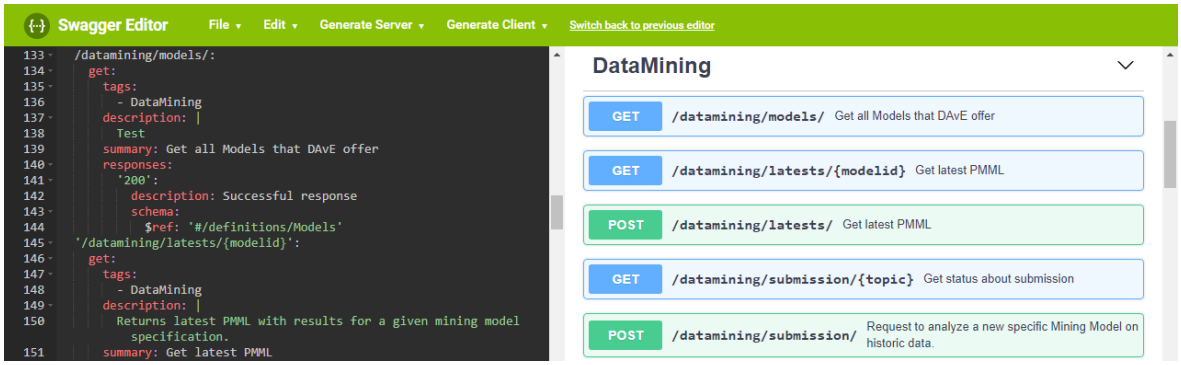


Abbildung 6.18: *Screenshot des Swagger Editors v3, mit geöffnetet PG Dave RESTful Schnittstelle*

von lauffähigen HTTP Servern bzw. HTTP Clients, bezüglich der definierten REST Schnittstelle und einer ausgewählten Sprache und optional genutzten Frameworks. Darunter zum Beispiel Spring als

[49] Swagger Framework. Online: `www.swagger.io`, letzter Zugriff: 19.02.2018

spezifische Java HTTP Server Variante, welche das Spring Framework nutzt oder typescript-angular als Javascript HTTP Client, welcher Typescript und das Angular Framework nutzt. Der Swagger Codegen existiert auch als als CLI Anwendung und als Gradle Plugin. Somit kann zur Entwicklungszeit passend zu unserem Buildsystem, manuell oder automatisiert Quellcode nach Schnittstellenänderungen generiert werden. Dabei können durch Optionen wie das Delegate Pattern bei der Generierung komplette Ordnerstrukturen ersetzt werden (vorzugsweise Unterstruktur eines generated Ordners), ohne dass nachträglich an dem generierten Code etwas abgeändert werden muss, sondern nur an den zum Delegate passenden, sich im nicht generierten Code befindenden, Dateien.

Das dritte wichtige Tool, Swagger UI, wird innerhalb des PG E-Stream Server Projekts dazu genutzt, automatisch auf Basis des programmierten bzw. generierten Spring HTTP Server Codes, eine webbasierte grafische Benutzeroberfläche zur Verfügung zu stellen. Diese dient hauptsächlich dem Testen von Schnittstellenfunktionen während der Entwicklung und begünstig den Workflow beim Entwickeln sehr. Das automatische zur Verfügungstellen der Oberfläche geschieht durch die Einbettung des springfox-Swagger UI Plugins von springfox.io durch Gradle. Dies erstellt eine eigene Ressource zum Erreichen der Swagger UI und durchläuft zur Laufzeit des Programms die durch Spring definierten Endpunkte und generiert auf Basis dieser den passenden Inhalt für Swagger UI (zu sehen in Abbildung 6.19).

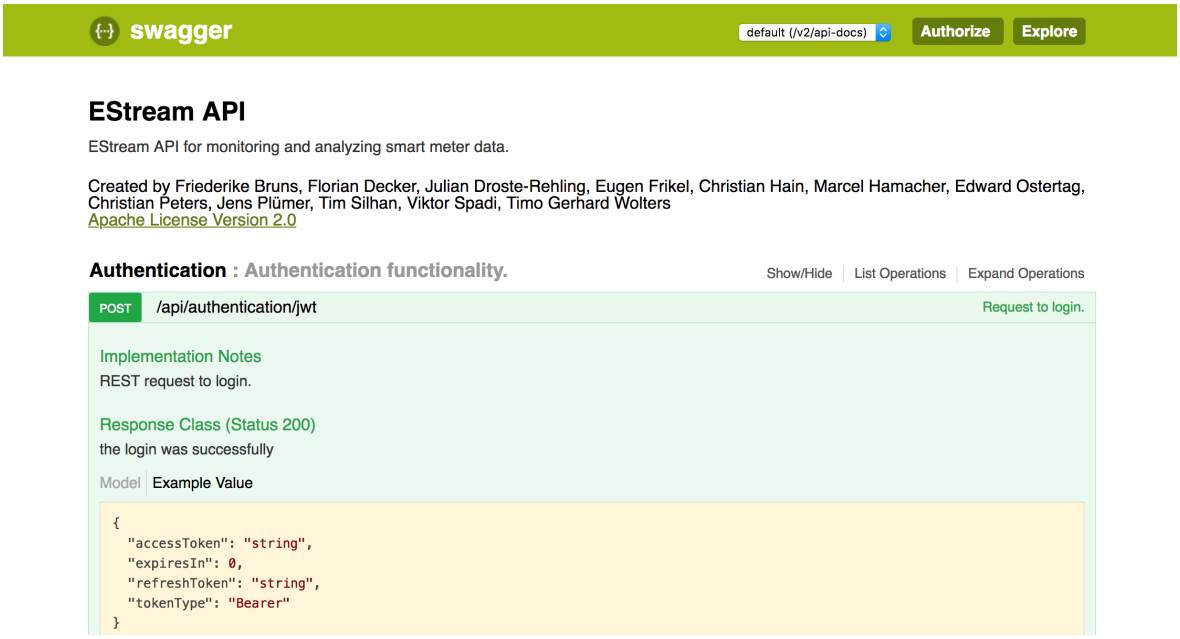


Abbildung 6.19: *Screenshot der Swagger UI von der REST API des E-Stream Backends*

6.3.7 Gradle

Wie bereits im Unterabschnitt 6.1.4 beschrieben, ist Gradle ein Build-Tool, das vor allem im JVM-Ökosystem benutzt wird. In diesem Projekt wurden mit Gradle Aufgaben für das Kompilieren des Systems, Installation und Verwaltung von Dependencies und für die Ausführung von kontinuierlichen

Builds für die Continuous Integration und Continuous Delivery (siehe Unterabschnitt 6.3.9) verwendet, damit dieses nicht manuell ausgeführt werden muss [149]. Zu den Aufgaben, die damit automatisiert werden, zählen beispielsweise der Build des Servers für die lokale Entwicklungs- sowie der Produktionsumgebung, Ausführung von Node.js und NPM Tasks für Produktions-Builds des Clients und das Generieren des Swagger Codes. Außerdem wird mit Gradle das Generieren der Dokumentation (für den Client, Server und die SwaggerUI) ausgeführt. Des Weiteren werden Verifikations-Tasks durch die Ausführung der Unit Tests, Kafka Tests, Local Integration Test und Slow Tests genutzt und das Generieren von TypeScript Interfaces aus den entsprechenden Data Transfer Object Klassen im Server und dem Shared Repository automatisiert.

6.3.8 mosaik Simulation

Mosaik, welches in Unterabschnitt 3.1.1 bereits detailliert in seiner Funktion als Framework zum Erstellen von Smart Grid Simulationen erläutert wurde, wurde in der Projektgruppe in erster Linie als Datenlieferant verwendet. Zu diesem Zweck verwendete die Projektgruppe die Niederspannungssimulation des Projektes *NetzDatenStrom* [211], welche sich zu diesem Zeitpunkt noch in der Entwicklungsphase befand, und passte sie an eigene Bedürfnisse an.

Die Niederspannungssimulation ermöglicht dabei zunächst die Anpassung des verwendeten Szenarios. Da die Simulation im Rahmen der Entwicklung des Projektes häufig neu gestartet werden musste, wurden daher zunächst Topologien verschiedener Größe generiert, um den Startvorgang durch die Verwendung kleiner Topologien beschleunigen zu können. Außerdem wurde hierdurch erreicht, dass die Simulation auf weniger leistungsstarken Systemen lokal ausgeführt werden kann, ohne die für andere Teile des Systems benötigten Ressourcen wesentlich einzuschränken. Die kleinste und während der Entwicklung am häufigsten verwendete Topologie umfasste dabei 52 Smart Meter Gateways und 129 Smart Meter. Die mittlere Topologie umfasste 554 Smart Meter Gateways und 1630 Smart Meter und die größte Topologie 2504 Smart Meter Gateways und 7587 Smart Meter.

Da die Niederspannungssimulation anfangs nur eine Ausgabe der Smart Meter Daten in die Konsole oder eine Datei unterstützte, musste sie von der Projektgruppe zunächst so erweitert werden, dass die Daten mit Odysseus empfangen werden können. Hierfür wurde zunächst eine Ausgabe mit *ZeroMQ* (ZMQ) [60], einer für verschiedene Programmiersprachen verfügbaren Bibliothek für den asynchronen Nachrichtenaustausch, realisiert. Da die Kommunikation zwischen Odysseus und dem E-Stream-Backend eine Kommunikation über Apache Kafka (Unterabschnitt 6.3.3) vorsah, wurde im Verlauf der Projektgruppe auch die Kommunikation zwischen Simulation und Odysseus auf Kafka umgestellt. Dies ermöglichte auch die einfache Bereitstellung der Rohdaten über einen entsprechenden Kafka Topic für die Projektgruppe DAvE. Zu diesem Zweck wurde ein Output- Script in Python entwickelt und in die Simulation integriert.

```
1      def sendRawData(self, message):
2          if (self.first):
3              self.producer = KafkaProducer(bootstrap_servers=[self.bootstrap],
4                  value_serializer=str.encode)
5              self.first = False
6
7          return self.producer.send(self.topic, message)
```

Listing 6.13: *Funktion zum Senden der Rohdaten an Kafka-Broker*

Dieses wurde im weiteren Verlauf immer wieder angepasst und um einen Wrapper erweitert, welcher auch für die Bereitstellung der jeweils verwendeten Topologie auf einem Topic zuständig ist. Listing 6.13 und Listing 6.14 zeigen die relevanten Funktionen zur Übertragung der Smart Meter Daten (sendRawData) und der Topologie (sendTopology). Hierbei ist insbesondere zu beachten, dass die Daten in unterschiedlicher Form übertragen werden, da die Topologie im JSON-Format vorliegt, wird sie dementsprechend serialisiert. Die Smart Meter Daten dagegen werden in der Simulation zunächst in ein XML-Schema überführt und anschließend als XML-String übertragen.

```
1      def sendTopology(self, message, blocking = True):
2          self.producer = KafkaProducer(bootstrap_servers=[self.bootstrap],
3                  value_serializer=lambda m: json.dumps(m).encode('ascii'))
4          future = self.producer.send(self.topic_topology, message)
5          if(blocking):
6              # Block for 'synchronous' sends
7              try:
8                  result = future.get()
9                  log.debug("result = future.get()")
10                 print(result)
11             except KafkaError:
12                 log.exception()
13                 pass
14             self.producer.flush()
```

Listing 6.14: *Funktion zum Senden der Topologie an Kafka-Broker*

```
1  {
2      "automatic_export": false,
3      "debug": false,
4      "events": "../../simulation_scenarios/events.json",
5      "maxStarttimeDifference": 450,
6          "output" : "KafkaXML",
7          "output_options" : {
8                          "step_size" : 900,
9                          "server"        : "134.106.56.15",
10                         "port"          : 29092,
11                         "topic"         : "rawdata",
12                         "topic_topology": "topology"
13                 },
14     "rt_factor": 0.1,
15     "substations": [
16         {
17                 "topology_folder" : "../../simulation_scenarios/
                       estream_scenario/topologie_trafo/"
18         }
19     ],
20     "timeinterval": 900,
21     "until": 31536000
22 }
```

Listing 6.15: *Konfiguration der Nierspannungssimulation*

Zur Verwendung des Kafka-Wrappers war es dabei auch notwendig, die in der Simulation verwendete Konfigurationsdatei um die für den Kafka-Producer benötigten Parameter zu ergänzen. Hierzu gehören die Angaben zur IP-Adresse und den Port des Kafka-Brokers, an welchem die Daten geschickt werden sollen. Darüber hinaus müssen die zu verwendenden Topics für die Rohdaten sowie für die

Topologie angegeben werden. Die schließlich in der Projektgruppe verwendete Konfiguration ist in Listing 6.15 dargestellt.

Da es die erwähnte Überführung der Smart Meter Daten in das COSEM-konforme XML-Format zu Beginn der Projektgruppe noch nicht gab, wurden die Daten zunächst nur als JSON übertragen. Hierdurch ergaben sich weitere Anpassungen, die für die Projektgruppe notwendig waren. So fiel durch die Umstellung die Angabe der ID der jeweiligen Smart Meter Gateways weg, welche im JSON-Format noch mit jedem Datensatz übertragen wurde. Da diese an vielen Stellen des E-Stream-Produktes verwendet wurde (unter Anderem bei der Prognose von Verbrauch und Erzeugung), wurde die XML-Ausgabe zunächst so erweitert, dass sowohl die Gateway-ID, als auch der Typ eines jeden Smart Meters (in Form von „Consumer“ und „Producer“) in der übertragenen XML enthalten war. Da dies jedoch nicht dem COSEM-Standard entspricht, wurde diese Lösung später durch das Enrichment der Daten in Odysseus ersetzt und stattdessen die dem COSEM-Standard entsprechende Ausgabe der Simulation verwendet.

6.3.9 Jenkins

Jenkins ist eine Software zur kontinuierlichen Integration von Software. Es stellt eine webbasierte Anwendung dar. Über den Browser können alle wichtigen Informationen bezogen werden und alle wichtigen Konfigurationen getätigt werden.

Jenkins ist als ein Fork von Hudson hervorgegangen, welches ebenfalls ein Tool zur kontinuierlichen Integration von Software darstellt. Der große Vorteil von Jenkins ist, dass es eine Vielzahl von Plugins unterstützt, sodass sich alle gängigen Tools nutzen lassen (Git, SVN, Apache Maven, Gradle usw.). Durch die große Erweiterbarkeit von Jenkins wird dieses für eine Großzahl von verschiedenen Projekten eingesetzt.

Die Idee von Jenkins oder von Continuous Integration (CI) ist es frühzeitig im Entwicklungsprozess eine lauffähige Version des Produktes hervorzubringen. Dazu werden Entwickler angehalten oft ihren Projektfortschritt an einer Stelle zu commiten, sodass Jenkins aus den verschiedenen Commits, der verschiedenen Entwicklern die Anwendung automatisiert zusammenbaut und Tests durchführt. Sodass Entwickler zeitnah Rückmeldung erhalten, ob die neu entwickelten Features Fehler enthalten. Durch dieses Vorgehen ist es gegeben, dass immer eine lauffähige Version der Anwendung vorhanden ist.

Nach der Installation von Jenkins wurde eine Reihe von wichtigen Plugins installiert. Die wichtigsten darunter sind die folgenden

Slave Setup Plugin Diesen Plugin wurde genutzt, um einen Jenkin-Slave zu konfigurieren. Dadurch wurde der Build der Dokumentation auf eine andere virtuelle Maschine ausgelagert.

Buckminster Wird für den Build unseres eigenen Odysseus-Produkt benötigt.

Build Blocker Dient dem Synchronisieren der verschiedenen Jenkins-Builds.

Build Timeout Wird ebenfalls wie *Build Blocker* zum Synchronisieren von Jenkins-Builds benötigt.

Publish over SSH Für die Continuous Delivery wird dieses Plugin benötigt. Damit wird das Produkt automatisiert auf dem Demosystem deployed.

Slack Notification Um über Fehlschläge der Builds oder fehlgeschlagene Tests zeitnah informiert zu werden wurde *Slack Notification* installiert. Damit wird es möglich Notifications der Jenkins-Jobs in einem Slackchannel zu pushen.

AnsiColor Die Standardinstallation von Jenkins bietet leider kein Syntax-Highlighting der Konsolenausgabe. Daher wurde entschieden dieses Plugin zu installieren.

In der Abbildung 6.20 ist eine Übersicht aller Jenkins-Jobs zu sehen die im Rahmen dieser Projektgruppe entstanden sind. Unter diesen Tasks befindet sich einer zum Erstellen unseres Produktes. Dieser wird alle 60 Minuten ausgelöst, wenn sich Änderungen am Code ergeben haben. Weitere wichtige Jobs sind das automatisierte Ausführen der Tests und das Erstellen unseres eigenen Odysseus-Produktes. Selbstverständlich existiert auch ein Job zum automatisierten Deployen unseres Produktes.

Alle +

S	W	Name ↓	Letzter Erfolg	Letzter Fehlschlag	Letzte Dauer	Built On
✓	☀	DEPRECATED Odysseus - EStreamProduct - Build1	14 Tage - #146	Unbekannt	3 Minuten 25 Sekunden	jenkins_slave
✓	☀	E-Stream - Build	2 Stunden 56 Minuten - #101	Unbekannt	3 Minuten 39 Sekunden	jenkins_slave
✓	☀	E-Stream - Dokumentation - Build	1 Tag 2 Stunden - #120	1 Monat 10 Tage - #106	4 Minuten 24 Sekunden	jenkins_slave
✓	⛅	E-Stream - e2e Test	2 Stunden 50 Minuten - #161	22 Stunden - #160	13 Minuten	jenkins_slave
✓	⛅	E-Stream - PingTest	2 Stunden 18 Minuten - #51	22 Stunden - #49	0.87 Sekunden	jenkins_slave
✓	☀	E-Stream - Production Deployment	2 Stunden 26 Minuten - #36	Unbekannt	5 Minuten 13 Sekunden	jenkins_slave
✓	☀	E-Stream - Release	10 Minuten - #3	Unbekannt	9 Minuten 17 Sekunden	jenkins_slave
✓	⛅	E-Stream - SlowTest	2 Stunden 30 Minuten - #57	2 Stunden 50 Minuten - #56	4 Minuten 1 Sekunde	jenkins_slave
✓	☀	E-Stream - Test	2 Stunden 52 Minuten - #84	Unbekannt	1 Minute 31 Sekunden	jenkins_slave
✓	☀	Odysseus - EStreamProduct - Build2	14 Tage - #281	Unbekannt	4 Minuten 30 Sekunden	jenkins_slave

Abbildung 6.20: *Übersicht der Jenkins-Jobs*

7 Projektfortschritt

In diesem Kapitel wird der chronologische Fortschritt des Projekts beschrieben. Hierzu wird das Kapitel in vier Abschnitte unterteilt, die verschiedene Projektphasen dokumentieren. Diese sind der Projektstart, die Seminarphase, die inkrementelle Entwicklung des Produkts und schließlich der Projektabschluss.

Dieses Kapitel bietet eine Übersicht über den Fortschritt des Projekts, beschreibt jedoch nicht detailliert die Inhalte der einzelnen Phasen. Es ist dazu gedacht, die Konzepte der Phasen zu erfassen, um die grundlegende chronologische Struktur des Projekts zu verdeutlichen.

7.1 Projektstart

Die Projektgruppe E-Stream wurde am 04. April 2017 mit einem ersten Gruppentreffen gestartet. Im Rahmen dieses Treffens wurde das Ziel und die Motivation für die Projektgruppe noch einmal durch die Betreuer vorgestellt. Dies beinhaltete unter Anderem die Vorstellung der Seminarthemen. Anschließend haben sich die Gruppenmitglieder vorgestellt und ihre Erwartungen an das Projekt geäußert. Nach dem Kennenlernen wurde über die Verteilung von Rollen, die den Ablauf des Projekts erleichtern sollen gesprochen. Hierzu gehörten z. B. die Rollen Scrum-Master, Projektleiter und ein Kommunikationsbeauftragter. Des Weiteren wurden grundlegende Regeln für die Gruppe festgelegt und ein Termin für regelmäßige Gruppentreffen gefunden. Es folgte eine Diskussion über die Anforderungen an den Stack, mögliche Technologien und die Strukturierung der Dokumentation. Abschließend wurde die Einrichtung eines E-Mail Verteilers, sowie einer Telegram-Gruppe für die Kommunikation innerhalb der Gruppe beschlossen.

7.2 Seminarphase

Die Seminarphase diente zur Einarbeitung in die theoretischen Grundlagen des Projekts. Sie begann mit dem Kick-Off Meeting am 04. April 2017. Die Themen umfassten unter anderem Datenströme und Data-Mining sowie ihre Visualisierung und die Protokolle, die für den Austausch der Daten notwendig sind. Eine Arbeit befasste sich speziell mit Teammanagement und der Organisation des Projektablaufes.

Jedem Gruppenmitglied wurde eins dieser Themen zugewiesen. Diese Person sollte Expertenwissen in dem Bereich erlangen und die Erkenntnisse schriftlich als Seminararbeit festhalten und den anderen Mitgliedern in Form eines Vortrags näher bringen. Am 04. Mai 2017 fand das Blockseminar statt, bei dem die Themen vorgestellt wurden. In Tabelle 7.1 ist die vollständige Zuordnung der Seminarthemen zu sehen.

Zusätzlich zur Ausarbeitung der Seminarthemen fanden wöchentliche Treffen statt, die sich überwiegend mit organisatorischen Themen befassten. Es wurde festgelegt, dass alle Dokumente, die für das Projekt relevant sind, im Confluence persistiert werden oder direkt in die Dokumentation eingepflegt werden. Die Dokumentation wird in LaTeXüber ein eigenes Git-Repository verwaltet. Die Verwaltung des Programmcodes erfolgt ebenfalls über ein Git-Repository. Die Repositories werden über Bitbucket verwaltet, da es optimale Integration mit Jira und Confluence ermöglicht. Des Weiteren

Seminarthema	Bearbeitet von
Mosaik	Eugen Frikel
CIM	Christian Peters
Datenstromverarbeitung	Timo Wolters
Data Mining auf Datenströmen	Tim Silhan
Odysseus	Marcel Hamacher
PV-Vorhersage	Friederike Bruns
Windvorhersage	Julian Droste-Rehling
PMML	Viktor Spadi
Visualisierung	Florian Decker
Teammanagement	Christian Hain

Tabelle 7.1: *Zuweisung der Seminarthemen*

wurden grundlegende Regeln innerhalb der Gruppe festgelegt, die sich mit der Planung der Sprints und der Scrum Durchführung befassen. Außerdem wurde beschlossen die Arbeitszeit mit Hilfe des Tempo Tools von Jira zu erfassen.

Die Kommunikation zu den Projektgruppen DAvE und NetzDatenStrom wurde hergestellt und eine Rolle für die Kommunikation mit der DAvE Projektgruppe definiert. Darüber hinaus wurde mit dem Einrichten einer Testumgebung für das Projekt, in Form von Virtuelle Maschinen, begonnen. Zusätzlich wurde ein Raum reserviert, in dem die Mitglieder sich treffen können, um gemeinsam an dem Projekt weiterzuarbeiten. Es wurden außerdem Regeln für die Erstellung von Aufgaben im JIRA festgelegt. Als Nächstes wurde beschlossen, ein Corporate Design, zunächst in Form eines Logos, sowie eine Präsentationsvorlage für die Projektgruppe zu erstellen.

Abschließend wurde am 11. Mai 2017 ein Brain-Storming zum Projektziel abgehalten. Dies beinhaltete die Verantwortungsbereiche der Projektgruppe sowie die zu entwickelnden Komponenten. In diesem Kontext wurden auch Anforderungen und Use-Cases definiert und die Grenze zur Projektgruppe DAvE festgelegt. Die Ergebnisse des Brain-Stormings sind dem entsprechenden Protokoll und der Dokumentation zu entnehmen. Das Brain-Storming diente zusätzlich der Planung des ersten Sprints.

7.3 Entwicklung

Nach dem Abschluss der Seminarphase startete die Projektgruppe mit dem Scrum Prozess. Die Ergebnisse der einzelnen Sprints werden im folgenden Abschnitt zusammengefasst und dokumentiert. Hierbei wird jeweils auf die Planung und Zielsetzung des Sprints sowie die Ergebnisse des Reviews und der Retrospektive eingegangen. Die Gruppentreffen, insbesondere am Tage eines Sprintabschlusses beziehungsweise Sprintstartes, wie in Unterabschnitt 4.2.2 beschrieben, durchgeführt.

7.3.1 Sprint 1

Der erste Sprint startete am 16. Mai 2017 und endete am 30. Mai 2017. Der Fokus dieses Sprints lag auf der Evaluierung und Auswahl der Technologien für die einzelnen Komponenten des Projekts. Hierfür wurden die Technologiealternativen für fünf Komponenten in Teams mittels Nutzerwertanalyse

evaluiert. Die Komponenten umfassten Backend, View Frameworks, Visualisierung, Datenbanken und Build Tools (siehe Abschnitt 6.1). Die Ergebnisse wurden in dem Gruppentreffen präsentiert.

Des Weiteren wurde an der Umsetzung des Logos für die Projektgruppe gearbeitet und Lösungen iterativ verbessert. Außerdem wurde weiter an der Deploy- und Test-Pipeline für das Projekt gearbeitet und die Struktur der Dokumentation verbessert. Einige Tasks konnten aufgrund fehlender VMs nicht abgeschlossen werden, sodass sie in den neuen Sprint übernommen werden.

Positiv wurde angemerkt, dass die Projektarbeit gut, motiviert und strukturiert vorangetrieben wird und bisher eine gute Aufwandsschätzung betrieben wurde. Als negativ wurde die Organisation der Tickets in JIRA und Kommunikationsprobleme aufgeführt.

Um die Probleme angehen zu können, wurde vereinbart, die Kommunikation innerhalb der Projektgruppe zu intensivieren. Es sollen außerdem genug Aufgaben ohne Bearbeiter in den neuen Sprint aufgenommen werden, da nach Beendigung einer Aufgabe keine freien vorhanden und einige Gruppenmitglieder nicht ausgelastet waren. Die Projektdokumentation soll über Jenkins gebaut und für die folgende Retrospektive nach der Starfish-Methode vorgegangen werden (siehe Unterabschnitt 4.5.2).

7.3.2 Sprint 2

Der zweite Sprint startete daraufhin am 30. Mai 2017 und endete zwei Wochen später am 13. Juni 2017. In diesem Sprint war das Ziel eine Grundlage für die weitere Entwicklung zu schaffen, das bedeutet ein Projekt-Setup, ein erster Architekturentwurf und eine Dokumentations-Grundlage zu entwickeln.

Zunächst wurde dafür festgelegt, dass übernommene Tasks freigegeben werden, wenn sie nicht in absehbarer Zeit weiter bearbeitet werden und noch nicht abgeschlossen sind. Fragen bezüglich einzelner Task sind an den Reporter oder in der Gruppe zu stellen. Hierdurch wurde versucht, den beschlossenen Actions aus dem letzten Sprint-Review entgegenzukommen.

Mit Ende des Sprints am 13. Juni 2017 wurde beschlossen, dass die Rolle des Git-Masters (Tim Silhan) eingeführt wird, welcher dafür Sorge trägt den GitFlow Arbeitsfluss einzuhalten. Außerdem wurde entschieden, dass auf die Branches *master* beziehungsweise *develop* nur über Pull Requests, nach dem ein finales Review durchgeführt wurde, gemergt werden darf.

In der Sprint Retrospektive wurden Punkte herausgestellt, die zu stark vorangetrieben wurden, sodass beschlossen wurde, dass Entscheidungen getroffen werden können, diese jedoch der Gruppe anschließend kommuniziert werden sollten. Das Gleiche betrifft auch das Löschen von Dateien aus dem Repository. Bei den bestehenden Problemen wurde beschlossen, die Anwesenheit bei Gruppentreffen zu steigern und die Dokumentation über Jenkins zu bauen, sobald das technisch möglich ist (es gab Probleme mit den VMs).

7.3.3 Sprint 3

Der dritte Sprint startete am 13. Juni 2017 und endete drei Wochen später am 04. Juli 2017. In diesem Sprint lag das hauptsächliche Ziel in der Erstellung einer Anforderungsanalyse, dem initialen Erstellen eines Usermanagements für die Sicherung des Backends und der damit einhergehenden Bereitstellung von Nutzern für Projektpartner und der Restrukturierung beziehungsweise Übernahme von bestehenden LaTeXDokumenten.

Christian Hain verkündet zu Beginn des letzten Gruppentreffens, dass er die Projektgruppe verlässt. Edward übernimmt daher Christians Rolle als Kommunikationsbeauftragten und als Release-Engineer.

Während der Retrospektive stellte sich heraus, dass vier Actions die im vorherigen Sprint beschlossen wurden nicht oder nicht komplett umgesetzt wurden, darunter das Vermeiden der falschen Verwendung von Git, das JIRA Sprintboard wurde nicht immer mit einer größeren Variation von Tasks aktuell gehalten und zusätzlich wurde die Zukunftsplanung/Meilensteinplanung nicht fortgeführt. Diese Punkte wurden für den nächsten Sprint mit übernommen und zusätzlich wurden Actions eingeführt, um den GitFlow zu verbessern, auf die Scrumkonformität (das Backlog füllen, im Weekly Scrum auch Vorhaben für die nächste Woche erwähnen, assignen/unassignen) zu achten und zum wiederholten Male angemerkt, dass die Tasktypen mehr in Richtung Programmierung gehen sollten. Außerdem wurde eingeführt, dass für die nächsten Gruppensitzungen jede Person eine rote Karte erhält, um so unnötige Diskussionen lautlos anmerken zu können und diese zum Aufhören zu bewegen.

Des Weiteren wurde beschlossen, dass in Zukunft zur Einschätzung der benötigten Zeit für Tasks das Magic Estimation Verfahren verwendet wird, sodass dabei eine Zeitersparnis bei dem Meetings erzielt werden kann (siehe Unterabschnitt 4.5.1). Aus verschiedenen Gründen (der Möglichkeit einen Bot und Channels zu verwenden, und zur besseren Teamkoordination auch mit der PG DAvE) wurde als neues Kommunikationsmedium Slack eingeführt. Für Starfish wurde demnach ein Bot erstellt, sodass jedes Teammitglied Kommentare einfach anonym eintragen kann und diese teilweise bei Überschneidungen schon zusammengeführt werden. Es ist außerdem für den Scrum-Master und den Product-Owner einsehbar, wie viele Einträge von wie vielen Teilnehmern vorgenommen wurden.

7.3.4 Sprint 4

Der vierte Sprint startete am 04. Juli 2017 und endete zwei Wochen später am 18. Juli 2017. Bei der Sprintplanung wurden die Zeiten wie lange ein Task ungefähr dauern wird mithilfe von Magic Estimation bestimmt. Das Ziel dieses Sprints war, dass alle Komponenten vom Gesamtkonzept für sich funktionsfähig sind und gegebenenfalls schon eine Kommunikation zu den anderen Komponenten erstellt wird. Das Gesamtkonzept beinhaltet die Komponenten des Backends, Kafka, Odysseus, Mosaik und Frontends, wobei beim Frontend erstmal nur ein Prototyp Dashboard entwickelt werden soll.

Außerdem wurde bei einem Treffen mit der Projektgruppe DAvE besprochen, welche gemeinsamen Use Cases existieren. Das Ziel der Projektgruppe DAvE ist es vorrangig ein Konzept zur Langzeitarchivierung von möglichst verschiedenen Daten zu erstellen und auf diesen Daten Data-Mining Modelle zu trainieren. Damit entstehen für beide Projektgruppen die folgenden zwei Anwendungsfälle. Der erste Use Case ist, dass wir die Rohdaten vom erzeugten und verbrauchten Strom der PG DAvE senden sowie auch wieder abfragen können. Damit erhält die Projektgruppe ihre zu speichernden Daten und kann gezielt historische Daten abfragen. Der zweite Use Case ist, dass von der Projektgruppe DAvE die Ergebnisse und die resultierenden Modelle vom Data-Mining bereitgestellt werden bzw. abgefragt werden können. Des Weiteren wurde beim Treffen schon abgestimmt, dass zum Datenaustausch voraussichtlich Kafka verwendet werden soll.

Im Sprint Review wurde die Aufgabe der Kommunikation von Odysseus und Mosaik über ein XML-Schema zurückgestellt. Denn die Kommunikation funktioniert über JSON und das in Mosaik verwendete XML-Schema ist noch nicht endgültig fertig. Somit wird das XML Schema erst später final implementiert.

Das eigens gesetzte Sprintziel wurde übertroffen, da neben der grundsätzlichen Funktion von Mosaik, Odysseus und Kafka auch schon die Kommunikation zwischen diesen Komponenten hergestellt wurde. Neben der Erstellung eines prototypischen Dashboards wurde auch die Authentifizierung mittels JWT in das Backend integriert. Zudem wurde während dieses Sprints das Usermanagement erweitert und am Anfang des Sprints wurde ein Meilensteinplan (siehe Abschnitt 4.4) erstellt.

Die Ergebnisse des Starfishs waren, dass bei der Taskplanung die Aufgaben genauer beschrieben und die Tasknamen aussagekräftiger werden sollen. Außerdem sollen funktionsfähige Development Branches auf den Master Branches released werden. Das JIRA Sprintboard wurde im letzten Sprint zu inkonsequent gepflegt und soll in Zukunft aktueller gehalten werden.

7.3.5 Sprint 5

Der fünfte Sprint startete am 18. Juli 2017 und endete zwei Wochen später am 1. August 2017. Das Sprintziel war, die ersten Möglichkeiten zum Visualisieren von Daten im Frontend zu implementieren und eine Verbindung zwischen Frontend und Backend, Kafka und Odysseus sowie Kafka und Backend herzustellen. Außerdem sollte das Konzept für die Verwendung von PMML bzw. JPMML entwickelt werden.

Am 19. Juli fand ein großes Estimation Treffen statt, in dem alle Aufgaben aus dem Product Backlog mithilfe von Magic Estimation geschätzt wurden. Da die meisten Aufgaben im Product Backlog nicht so zeitaufwändig waren, konnten diese einfach in das Sprint Backlog gezogen werden.

Am 21. Juli fand ein Treffen mit der PG DAvE statt, in welchem über den jeweiligen aktuellen Stand gesprochen wurde. Es wurde beschlossen, dass die von uns entwickelte REST-API mit Hilfe von Swagger dokumentiert und die mit Swagger erstellte YAML Datei mit der Beschreibung der API an die PG DAvE weitergegeben wird. Außerdem wurde ein neues Slack Team für Förderung der gemeinsamen Kommunikation erstellt.

Im Frontend wurden Komponenten für das Anzeigen von Liniendiagrammen, Balkendiagrammen und Tabellen implementiert. Auch Kafka wurde in diesem Sprint erfolgreich integriert. Zusätzlich wurden nachträglich einige Tests im Backend geschrieben. Das Sprintziel wurde demnach erreicht.

Im Sprint Review wurde die Login Seite vorgestellt. Dabei wurde angesprochen, dass es bei falschen Eingaben kein aussagekräftiges Feedback gibt und noch einige Funktionen fehlen. Anschließend hat Christian das Konzept vorgestellt, welches beschreibt wie die PG DAvE mit uns kommunizieren kann.

Die Ergebnisse der Retrospektive waren, dass das Team mehr Reviews zeitnah durchführen sollte. Hierbei sollte eine kurze Einleitung gegeben sein, wie ein Review durchgeführt werden kann. Es wurde beschlossen, dass nicht mit einem neuen Task anfangen werden sollte, wenn noch Reviews offen sind. Reviews sollen gründlicher durchgeführt werden und zusätzlich nach dem Merge/Rebase soll nochmals überprüft werden, ob dadurch nichts kaputt gemacht wurde. Nach dem Merge in Jenkins soll überprüft werden, ob der Build auf Jenkins erfolgreich war.

7.3.6 Sprint 6

Der sechste Sprint begann am 1. August 2017 und endete am 15. August 2017. Die wesentlichen Ziele des Sprints waren die Vorbereitung von Data Mining sowie der Ausbau der Schnittstelle zu PG DAvE.

Es sollten insbesondere mögliche Verfahren zur Prognose des Netzzustandes sowie die bisher vorhandenen Data Mining Operatoren in Odysseus evaluiert werden, um festzustellen, welche Verfahren mit welchem Aufwand einsetzbar sind. Zudem sollte das zuvor entwickelte Konzept zur Verwendung von PMML zum Austausch von Modellen implementiert und eine entsprechende Schnittstelle zum Datenaustausch definiert werden.

Zu Beginn des Sprints wurde darüber hinaus beschlossen, dass sämtliche Kommunikation des entwickelten Systems künftig über Kafka Connect stattfinden soll, um eine klare und einheitliche Schnittstelle des Systems zu gewährleisten. Hierzu wurde mit der Einrichtung von Kafka Connect sowie der Implementierung einer entsprechenden Schnittstelle begonnen.

Des Weiteren ergab sich in diesem Sprint die Notwendigkeit, bestimmte Dateien wie Schema-Klassen und Schnittstellendefinitionen als Abhängigkeit in verschiedenen Komponenten des Systems verwenden zu können und diese auch PG DAvE zugänglich zu machen. Zu diesem Zweck wurde ein öffentliches Shared Repository über GitHub[1] eingerichtet, welches künftig für geteilte Daten verwendet werden sollte.

Jedoch konnten nicht alle Aufgaben in diesem Sprint abgeschlossen werden. Insbesondere Aufgaben zur Simulation von Anfragen der PG DAvE konnten aufgrund von Abhängigkeiten zu anderen Aufgaben innerhalb des Sprints nicht rechtzeitig begonnen werden und wurden daher in den nächsten Sprint übernommen.

Letzteres wurde auch in der Retrospektive, welche erneut mit der Starfish-Methode durchgeführt wurde, angemerkt und darauf hingewiesen entsprechende Abhängigkeiten künftig zu berücksichtigen. Zudem wurden aus der Retrospektive das Hinzufügen eines erfolgreichen Gradle builds zur Definition of Done und die Durchführung gemeinsamer Code Reviews in den Gruppensitzungen als Actions abgeleitet.

Allerdings zeigte sich in der Retrospektive auch, dass nicht alle Actions der vorherigen Retrospektive konsequent umgesetzt wurden. Insbesondere die zeitnahe Ausführung von übernommenen Aufgaben sowie das Freigeben von Tasks wurden erneut bemängelt und somit erneut als Actions aufgenommen. Um die Einhaltung der entsprechenden Regeln besser durchsetzen zu können, wurde vorgeschlagen dies künftig nicht nur in der Retrospektive, sondern auch im laufenden Sprint zu überprüfen.

Als positiv wurden in der Retrospektive die anhaltend gute Kommunikation innerhalb der Projektgruppe, das gute Teamwork, eine gute Mischung der verschiedenen Aufgaben und die inzwischen kurzen Reviewzyklen hervorgehoben.

7.3.7 Sprint 7

Vom 15. August bis zum 29. August fand der siebte Sprint statt. Die relevanten Ziele dieses Sprints waren unter anderem die weitere Zukunftsplanung. Dies äußerte sich in Tasks, welche sich mit weiteren Anforderungsanalysen auseinandergesetzt haben. Insbesondere wurden die Grenzen des Systems definiert und die Beschreibung des bestehenden Systems wurde aktualisiert. Des Weiteren ist es aufgrund des fortschreitenden Projektstandes nötig gewesen, die bereits bestehenden Epics zu validieren und diese durch die neu gewonnenen Erkenntnisse anzupassen.

[1] Shared Repository für Schnittstellendefinitionen und Modelle. Online unter: `https://github.com/PG-EStream/shared`, letzter Zugriff 28.03.2018.

Direkt einen Tag nach dem Start des Sprints fand ein Treffen mit der Projektgruppe DAvE statt. In diesem Treffen stellten sie ihren eingesetzten Technologie Stack vor. Darüber hinaus wurden die definierten Schnittstellen bezüglich der Funktionalität und Modellierung von der PG DAvE abgesegnet. Jedoch wurden seitens der PG DAvE Bedenken geäußert, die gewünschten Schnittstellen nicht innerhalb des Projektzeitraums realisieren zu können. Dieses Szenario wurde bereits durch die präventive Maßnahmen, wie die Erstellung von Mocks, berücksichtigt. Dennoch soll in einem weiteren Treffen die PG DAvE auf das Bereitstellen von obligatorischen Endpunkten verpflichtet werden.

Eine weitere große Aufgabe in diesem Sprint stellte das Überarbeiten des Glossars der Dokumentation dar. Dafür war es nötig die gesamte bereits erstellte Dokumentation zu sichten und wichtige Begrifflichkeiten zu erläutern. Im Rahmen dieser Sichtung wurden viele weitere Mängel behoben oder dokumentiert, sodass diese in einem folgenden Task validiert und ggf. korrigiert werden können.

Des Weiteren legte dieser Sprint den Grundstein für die Kommunikation zwischen Odysseus und der Kernkomponente dieser Projektarbeit, den Springboot Server. Denn innerhalb eines Tasks wurden verschiedene Ansätze für eine störungsfreie Kommunikation zwischen diesen Komponenten evaluiert und realisiert.

Am Ende des Sprints wurde die Retrospektive des Sprints durchgeführt. Kritisiert wurde u. a. die Anzahl der Tasks bzw. Issues im Backlog sowie die Anzahl der Tasks im Sprint. Zudem wurde kritisiert, dass sich gegenseitig blockierende Tasks im Sprint befanden. Außerdem wurde die Qualität der Beschreibungen der Tasks beanstandet. Lobend erwähnt wurden dagegen die Kommutation und das Teamwork der gesamten Gruppe sowie die insgesamt sehr zufriedenstellende Sprint Planung. Als Reaktion auf die Kritikpunkte wurde beschlossen, dass zu allgemein gehaltene Taskbeschreibungen direkt durch eine spezifischere Beschreibung ergänzt oder ausgetauscht werden. Durch ein weiteres Treffen zum Erstellen von Tasks soll das Backlog mit weiteren Tasks ausgestattet werde.

7.3.8 Sprint 8

Der achte Sprint begann am 29. August 2017 und endete am 12. September 2017. Ziele für diesen Sprint waren die Einbindung von geographischen und abstrakten Karten in das Frontend, eine Verbindung für den Datenaustausch zwischen Backend und Frontend sowie erste simple Berechnungen auf den Daten, die sich zunächst auf Minimum-, Maximum- und Durchschnittsberechnungen beschränken sollten.

Am 30. August fand ein erneutes Treffen mit der PG DAvE statt. Dieses Mal zusammen mit den jeweiligen Betreuern der Projektgruppen. Bei diesem Treffen wurde beschlossen, dass die beiden Gruppen sich auf gemeinsame Meilensteine einigen werden und es wurden vier gemeinsame Use Cases ausgearbeitet. Es wurde beschlossen, dass die Use Cases zur „Abfrage von Rohdaten“ (in beide Richtungen) sowie die „Speicherung von Rohdaten“ (von PG E-Stream an PG DAvE) Minimalziele sind, die erfüllt werden müssen. Die Use Cases „Modellbau auf Anfrage“ und „Einbinden von Wetterdaten“ wurden als optional eingestuft, die vom Fortschritt des Projekts abhängig sein sollen.

Das erste Code-Review innerhalb der Gruppe fand am 1. September statt. Hier gab es eine kurze Einführung in Angular und eine Vorstellung der Codebasis des Frontends. Die Vorstellung, die von Viktor geführt wurde, kam sehr positiv an und gab einen guten Einblick darin, wie das Frontend aufgebaut ist und wie daran weitergearbeitet werden kann. Als weitere teambildende Maßnahmen ging die Gruppe anschließend auf das Oldenburger Stadtfest.

In der Gruppensitzung der zweiten Woche des Sprints am 5. September wurde damit angefangen, die in den Retrospektiven entstandenen Actions zu wiederholen und zu schauen, ob die Projektgruppe sich daran hält diese umzusetzen oder ob daran noch mehr gearbeitet werden müsse.

Am 6. September hat sich ein Teil der Gruppe zusammengesetzt, um neue Tasks aus den bestehenden Anforderungen abzuleiten. Dabei wurde zunächst die Sprintplanung für kommende Sprints überarbeitet und neue Ziele definiert, die in den kommenden Sprints erfüllt werden sollten. Anschließend wurden neue Tasks definiert und mittels Planning Poker (siehe Unterunterabschnitt 4.1.2.2) geschätzt.

Am 12. September ging der achte Sprint zu Ende und es wurde der Projektfortschritt präsentiert. Dabei konnte gezeigt werden, dass in diesem Sprint die Sprintziele erfüllt wurden. Für die ersten Berechnungen auf den Daten wurde ein Odysseus-Skript geschrieben. Weiterhin wurden die Kartendarstellungen im Frontend implementiert. Außerdem wurde erfolgreich eine Verbindung zwischen Backend und Frontend über Websockets eingerichtet. Damit ist es nun möglich, dass Daten über Kafka von der Simulation über das Backend hin an das Frontend geleitet und zunächst in einer Tabelle angezeigt werden. Ein weiterer Fortschritt in dem Sprint war der Beginn der Umsetzung für die Generierung von Anfragen vom Backend an Odysseus.

Bei der Sprintretrospektive ist erneut herausgekommen, dass die Motivation, Teamarbeit und die Hilfsbereitschaft untereinander sehr gut sind. Weiterhin wurde es als positiv aufgenommen, dass im Sprint frühzeitig über anstehende Aufgaben nachgedacht wurde. Als negative Punkte wurden dieses Mal erwähnt, dass sich Aufgaben innerhalb des Sprints blockiert haben und es einige relativ große Aufgaben gab, die sich durch den Sprint durchgezogen haben, so dass am Ende wenig kleine Aufgaben im Sprint waren. Diese Punkte wurden anschließend bei der Sprintplanung berücksichtigt.

7.3.9 Sprint 9

Der Sprint neun wurde im Zeitraum vom 12. September 2017 bis zum 26. September 2017 durchgeführt und hatte zum Ziel die Ausarbeitung der Dokumentation voranzutreiben. Darunter sollten zum einen neue Texte entstehen, die den Scrum-Prozess der PG beschreiben und andererseits sollten neue Anforderungen ausgearbeitet sowie bereits bestehende Anforderungen überarbeitet, validiert und erneut verabschiedet werden. Zu den Anforderungen sollten neue Anwendungsfälle entwickelt und bestehende überarbeitet werden. Außerdem sollten verschiedene Evaluationskonzepte entwickelt werden. Beispielsweise soll die Usability des Frontends durch Experten evaluiert werden können. Die Ausarbeitung der Zwischenpräsentation (für den Prototyp I) stand ebenfalls auf der Agenda. Hierfür mussten auf der Implementierungsebene noch fehlende UI-Komponenten entwickelt und getestet werden. Weitere zentrale Implementierungsziele waren: Refactoring der Web-Socket-Verbindung sowie die Anfrageerstellung und das Anfragemanagement im Backend.

Insgesamt wurden viele Aufgaben in diesem Sprint nicht abgearbeitet, da die Zeiteinschätzung für die Bearbeitung einiger Aufgaben falsch bemessen wurde. Beispielsweise wurden die meisten Arbeitsstunden für die Bearbeitung und Vorbereitung der Zwischenpräsentation verwendet. Die Probevorträge für die Zwischenpräsentation waren insbesondere sehr zeitaufwändig, da hierbei viele Teammitglieder teilgenommen haben.

In der Retrospektive wurden Zusammenarbeit, Kommunikation und eine harmonische Gruppendynamik als positive Merkmale herausgestellt. Negativpunkte waren zum einen der JIRA-Workflow, welcher durch Mehrfachaufgabenzuweisung zu einer unkoordinierten und fehlerhaften Arbeitsweisegeführt

hat, und zum anderen die Nichtfertigstellung vieler Aufgaben. Als Ergebnis lässt sich also festhalten, dass die gesetzten Ziele für diesen Sprint nur teilweise erfüllt werden konnten. Dennoch wurden alle kritischen Aufgaben erledigt, welche den Projektfortschritt bis hin zur Zwischenpräsentation blockiert hätten. Die Arbeitszeit wurde insofern sinnvoll genutzt, dass die blockierenden Aufgaben bearbeitet wurden, weshalb der Sprint als Erfolg zu verbuchen ist.

7.3.10 Sprint 10

Der zehnte Sprint startete am 26. September 2017 und endete zwei Wochen später am 10. Oktober 2017. Die wesentlichen Ziele des Sprints waren die Erstellung der Zwischenpräsentation und die Fertigstellung der dafür wichtigen Tasks im Backlog. In diesem Sprint sollte auch erstmals der vertikale Durchstich des Systems funktionieren. Alle dafür erforderlichen Komponenten wurden unabhängig voneinander in den vorherigen Sprints erledigt. Zudem wurden weitere Evaluationsaufgaben, deren Ergebnisse für die Zwischenpräsentation relevant waren, eingeplant. Hierzu gehörten zum Beispiel das Evaluationskonzept für den Performanztest, die Überprüfung der Korrektheit der Daten und die Usability des Systems. Des Weiteren sollten Dokumentationen für die Wikiseite des Odysseus sowie für die Projektdokumentation der entstandenen Operatoren für Odysseus vervollständigt werden. In Zusammenarbeit mit der PG-DAvE wurden gemeinsame Use Cases hervorgebracht.

In der Gruppensitzung der ersten Woche wurde der bisherige Fortschritt der Zwischenpräsentation anhand einer Vorabversion der Präsentation gezeigt und in der Gruppe konstruktiv in Diskussion weiterentwickelt. Anders als ursprünglich in der Sprintplanung vorgesehen, wurde nicht weiter an der Anforderungsanalyse gearbeitet, da die Erstellung der Zwischenpräsentation wegen des bevorstehenden Termins eine höhere Priorität hatte. Zu diesem Zeitpunkt wurde auch verkündet, dass (E-Stream) Odysseus-Produkt fortan über das releng Projekt verfügbar ist und benutzt werden kann. Mit der Verfügbarkeit von diesem Odysseus-Produkt, welches die von uns entwickelten Operatoren enthält, die für unser System relevant sind, konnte auch die Integration des Produkts in die Vagrantbox eingeleitet werden, welche für die lokale Entwicklung des Projekts notwendig ist.

Am 10. Oktober 2017 war das Sprintende und in der Gruppensitzung wurde die finale Version der Zwischenpräsentation sowie einige der erledigten Aufgaben präsentiert. Im Sprint Review hat sich herausgestellt, dass viele Aufgaben nicht bearbeitet wurden und somit in den nächsten Sprint verschoben werden mussten. Als Grund dafür hat sich eine inadäquate Aufwandseinschätzung der einzelnen Aufgaben herausgestellt. Viele Aufgaben haben viel länger gedauert als ursprünglich geplant und wurden aufgrund der zeitintensiven Arbeiten zur Vorbereitung der Zwischenpräsentation nicht oder nur teilweise erledigt. Entsprechende Actions wurden in der Sprint Retrospektive für die Nachverfolgung angelegt. Ebenso wurde in der Retrospektive festgestellt, dass das Teamwork, die Motivation, die interne Kommunikation und die gegenseitige Hilfe weiterhin als positiv von der Projektgruppe wahrgenommen wurden. Aufgrund der bereits erwähnten nicht erledigten Aufgaben, was zusätzlich beim Starfish erwähnt worden ist, wurde diese Problem als Action niedergelegt. Zukünftig sollen Tasks im Backlog besser priorisiert werden und es soll darauf geachtet werden, dass nicht zu viele Tasks im Review sind, bzw. diese abgearbeitet werden, bevor mit neuen Tasks begonnen wird. Anschließend soll ermittelt werden, wie die genannte Problematik, bis auf die schlechte Zeitplanung auftreten konnte, um sie zukünftig zu vermeiden.

7.3.11 Sprint 11

Vom 10. Oktober 2017 bis zum 25. Oktober 2017 fand der elfte Sprint statt. Ziel des Sprints war ein Überarbeiten der Code-Basis, sodass Fehler und Bugs behoben und die Probleme isoliert werden, da das Gesamtsystem nicht bei allen Gruppenmitgliedern lief. Zusätzlich wurde am Ende des Sprints ein neuer Release erstellt, welcher den vertikalen Durchstich enthält. Die Anforderungsanalyse sollte darüber hinaus als Basis für die Zukunftsplanung erweitert, das Domänenwissen innerhalb der Projektgruppe vertieft und die Anwendungsfälle verfeinert werden. Des Weiteren sollte das Backlog wieder mit Aufgaben gefüllt und Kafka-Connect auf der Demo-VM installiert werden.

Im Sprint wurde daher vorgesehen, die e2e-Tests in die CI-Umgebung zu integrieren. Um den Client der Anwendung nicht auf eine Sprache zu begrenzen, wurde zusätzlich ein Konzept für die Internationalisierung aufgestellt, das im folgenden Sprint umgesetzt werden soll. Zusätzlich sollte das Produkt als Download im Confluence zur Verfügung stehen und die REST-Schnittstelle für Kafka-Connect angepasst werden. Darüber hinaus war der generische Filter zu erweitern, sodass er in Kombination mit beliebigen Visualisierungen und mit beliebigen Daten eingesetzt werden kann. Da im letzten Sprint das Odysseus-Produkt fertiggestellt wurde, wurde in diesem Sprint die Dokumentation der hinzugefügten Features vorgenommen. Für das Füllen des Backlogs, ist ein Zusatztreffen vereinbart worden. Darüber hinaus waren einige Fehlerbehebungen im Client und der CI-Umgebung im Sprint vorgesehen.

Durch das Zusatztreffen wurde das Jira-Backlog weiter gefüllt und es wurden einige Fehler mit den e2e-Tests und der CI-Umgebung behoben. Außerdem wurde mit dem Ende des Sprints ein neues Release erstellt und als Download im Confluence-Wiki angeboten. Das Domänenwissen wurde durch eine Papersammlung zum Thema Smart Grid erweitert, wird aber im nächsten Sprint durch eine Vorstellung von wichtigen, allgemeinen Konzepten in der Energiedomäne fortgesetzt. Weiterhin wurde die Anforderungsanalyse nicht weiter ausgearbeitet und die Einrichtung von Kafka-Connect funktionierte nicht so wie erwartet. Diese Ziele werden in den nächsten Sprint übernommen.

In der Retrospektive wurden wieder die gute Teamarbeit, Kommunikation, Zusammenarbeit und Motivation hervorgehoben. Zusätzlich wurde die gute Aufgabenvielfalt angesprochen. Negativ sind fehlende Kommentare über den Fortschritt von Aufgaben aufgefallen. Ebenso gab es Aufgaben, die keine Beschreibung haben, sodass das Ziel der Aufgabe nicht klar wurde. Außerdem sollen Aufgaben mit höherer Priorität zuerst bearbeitet werden, damit wichtige Aufgaben am Ende des Sprints nicht liegen bleiben. Um diesen Problemen entgegenzuwirken wurde festgehalten, dass Taskbeschreibungen mindestens das Ziel und eine Problembeschreibung beinhalten und dass Aufgaben mit höherer Priorität zuerst bearbeitet werden sollen. Sollte eine Aufgabe innerhalb des Sprints nicht angefangen werden, dann soll in der Gruppe geklärt werden, warum dies nicht gemacht werden konnte. Außerdem sollen alle Tasks nach Möglichkeit vor Ende des Sprints auf Done stehen.

7.3.12 Sprint 12

Vom 26. Oktober 2017 bis zum 07. November 2017 fand der zwölfte Sprint statt. Ziel des Sprints war das Entwickeln eines Konzepts für eine erweiterbare Topologie-Anzeige für Smart Meter/Verbünde. Zusätzlich sollten die übrigen Aufgaben aus dem Bereich Cleanup der Codebase ausgeführt werden. Des Weiteren sollte ein Parser für die Übersetzung von COSEM-Daten in das XML-Format implementiert werden. Außerdem sollte eine Frontend-Komponente für das Erstellen von Queries entwickelt

werden. Es soll die Implementierung für eine Internationalisierung des Frontends entwickelt, ein WebSocket-Konzept erstellt und dieses anschließend implementiert werden.

Mit Ende des Sprints konnte das Konzept für die Anzeige der Topologie vorgestellt werden. Hierbei wurde beschlossen, dass der ForceDirectedGraph eine geeignete Darstellung sei und es wurde eine Priorisierung vorgenommen, nach der die topologische Darstellung wichtiger als die geographische Darstellung ist. Ebenso verfahren wurde mit der Confluent Plattform, für welche Anpassungen im Vagrant-File vorgenommen wurden. Des Weiteren wurden die Testmodelle für den PG DAvE-Mock vorgestellt und ein PMML/XML Dummy erstellt, welcher in Odysseus ausführbar ist. Da die Aufgabe, den XML-Converter in die Simulation einzubinden, lediglich fast fertig ist, wird diese in den folgenden Sprint gezogen. Es wurde ein Konzept für die Entwicklung eines Frontend-Moduls für das Managen von Queries erstellt. Die Implementierung für eine Internationalisierung wurde abgeschlossen und es wurde ein WebSocket-Konzept erstellt. Außerdem wurde die Anforderungsanalyse weiter ausgearbeitet und begonnen die Use Cases genauer zu definieren. Diese Aufgabe wird jedoch in dem folgendem Sprint beendet. Die Dynamisierung der Frontend Line Charts konnte nur fast beendet werden. Die Implementierung des Datenaustausches zwischen Frontend und Backend sowie das Anpassen der Connector Swagger Schnittstelle und das Testen des JDBC Connectors wurden noch nicht begonnen und daher in den folgenden Sprint gezogen.

Als besonders positiv in der Sprint Retrospektive wurden die Code Reviews genannt. Außerdem wurden als positiv die gute Kommunikation, Hilfsbereitschaft, Teamwork und soziale Gruppentreffen, sowie Motivation, innovative Ideen und die schnelle Taskbearbeitung bei einer größeren Aufgabenvielfalt genannt. Als negativer Punkt wurde aufgeführt, dass zu viele neue Actions eingeführt werden, wenn Probleme auftreten. Unter Start Doing findet sich, zu prüfen, warum viele Aufgaben zum Sprintende noch nicht erledigt sind, dass versucht werden sollte, einen Sprint, ohne sich gegenseitig blockierende Tasks, zu starten und dass es mehr Reviews von Konzepttasks geben sollte, sodass alle das Konzept kennen und mehr Input als nur von dem Reviewer gegeben werden kann. Als Critical markierte Aufgaben sollten zuerst bearbeitet werden, insbesondere Konzeptaufgaben, wenn auch die Implementierung im Sprint ist. Insgesamt sollte auf mehr Sauberkeit in den Git-Repositories geachtet werden. Um Problemen entgegenzuwirken, wurde beschlossen, dass beachtet wird, bei Änderungen auch JIRA zu aktualisieren. Außerdem, dass blockierende Aufgaben als „Blocker“priorisiert werden, die Gitbeauftragten bei bestehenden Problemen die entsprechenden Personen direkt kontaktieren und dass Aufgaben bereits bei ihrer Erstellung priorisiert werden. Bis auf darauf zu achten, alte Sprintziele zu erreichen und den Backlog zu priorisieren konnten alle alten Actions erfüllt werden.

7.3.13 Sprint 13

Der 13. Sprint wurde im Zeitraum vom 08. November 2017 bis zum 22. November 2017 durchgeführt. Die Sprintziele bestanden darin, ein Template für Regressionsanfragen auf dem Smart Meter Datenstrom zu entwerfen, angelehnt an das von uns entwickelten Query Schema. Außerdem sollte das Konzept implementiert werden, welches sich mit der Visualisierung der Topologie als Navigation-Tree und als abstrakte Karte beschäftigte. Zusätzlich sollten die nicht erledigten Sprintziele des letzten Sprints bewältigt werden. Außerdem wurde beschlossen, dass in diesem Sprint mit der Umstellung auf Docker begonnen wird.

Während des abschließenden Sprint Reviews stellte sich heraus, dass das Sprintziel der Umsetzung des Topologie Konzeptes nicht komplett fertig gestellt werden konnte, da es in diesem Sprint zu vielen

Krankheitsfällen kam. Zudem traten bei einem vorherigen Sprintziel, dem Bereitstellen der Topologie, noch unvorhergesehene Probleme auf. Außederdem blockierte diese Aufgabe fälschlicherweise die Aufgabe zur Visualisierung der abstrakten Topologie, welche daher ebenfalls nicht umgesetzt wurde.

Bei der ersten Gruppensitzung am 15. November 2017 wurde nach langer Wartezeit ein Code Review des QueryBuilderService/QueryManagerService durchgeführt. Hierbei stellte sich heraus, dass bei der Programmierung wieder vermehrt an den erstellten Styleguide gedacht werden sollte. Außerdem stellte Michael den aktuellen Fortschritt seiner Arbeit im Bezug auf die Topologie Verarbeitung vor und zeigte die Integration in Odysseus und die dafür verwendeten Bibliotheken. Die Zusammenarbeit mit der PG DAvE befindet sich weiterhin im Stillstand, da auf mehrere E-Mails zur Terminfindung für die Erklärung des Verzugs, von der PG DavE noch keine Rückmeldung geschah.

Bei der Retrospektive wurde wieder die gute Teamarbeit, das Erledigen von Gruppenaufgaben, die Kommunikation und das Kommentieren von laufenden Aufgaben in Jira als positiv bewertet. Negativ fielen unter anderem erneut die schlechte Internetverbindung im Gruppenarbeitsraum, sowie zum ersten Mal die fehlende Motivation einzelner Gruppenmitglieder und die spärliche Anwesenheit bei den optionalen Gruppentreffen auf. Außerdem wurde negativ angemerkt, dass mehr Kommentare für den Zwischenstand von längeren Aufgaben in Jira erstellt werden sollten und das versucht werden sollte, die Abhängikeit von Aufgaben im Sprint zu vermeiden. Zusätzlich wurde angesprochen, ob die Zeitschätzung nur positive Aspekte hat. Nach einer Diskussion und Abstimmung durch die Anwesenden wurde festgesetzt den nächsten Sprint Scrum unkonform zeitlich ungeschätzt zu starten, es wurden lediglich die Schwierigkeitsgrade der einzelnen Aufgaben geschätzt.

7.3.14 Sprint 14

Im Zeitraum vom 23. November bis zum 06. Dezember fand der 14. Sprint statt. Die Ziele des Sprints waren Test-Modelle für den PG DAvE-Mock zu erstellen. Außerdem ist es möglich die Test-Modelle über den Client anfordern zu können, was das Zusammenspiel aller Systeme, sowie des Mocks für die PG DAvE testet. Des Weiteren sollte Docker für alle Systeme eingerichtet werden, um die Skalierbarkeit und die Usability des Gesamtsystems zu erhöhen. Der Client sollte um einen Playground-Bereich erweitert werden, der für Testzwecke genutzt werden kann. Des Weiteren sollte ein Konzept für die Projektstruktur im Client ausgearbeitet werden. Letztlich sollten noch übrig gebliebene Aufgaben des vorherigen Sprints bearbeitet werden.

Die Test-Modelle für den PG DAvE-Mock wurden erstellt, allerdings fehlen noch Teile für die Integration aller beteiligten Systeme. Diese werden im folgenden Sprint weiter bearbeitet. Docker wurde für alle Systeme eingerichtet, jedoch gab es Probleme damit, die Container auf Windows-Systemen auszuführen. Daher wird Docker nun in einer Vagrant-Box ausgeführt, so dass es auch auf Windows-Systemen problemlos läuft. Der zusätzliche Aufwand durch Vagrant ist gering, da die Vagrant-Box nur ein einziges mal aufgesetzt werden muss und generell lediglich als Container für Docker dient. Der Playground-Bereich wurde eingerichtet. Es wurde ein Konzept für die Projektstruktur des Clients ausgearbeitet, was in einem der folgenden Sprints umgesetzt werden soll. Wegen eventuellen Problemen wird das Konzept womöglich inkrementell umgesetzt, um Konflikte zu vermeiden.

In der Gruppensitzung am 29. November 2017 wurde ein Code Review der Filterkomponente im Client durchgeführt, bei welchem diese um Buttons zur „Reapply“und „Reset“Funktionalität erweitert wurde. Des Weiteren wurde das Konzept des Dashboards vorgestellt und diskutiert. In der Sitzung am 06. Dezember wurde die Docker Infrastruktur präsentiert. Außerdem wurden neue Funktionalitäten,

wie zum Beispiel der Playground-Tab, die Baumstruktur für die Topologie, der Gradle Task für die Codegenerierung und Annotationen für Swagger vorgestellt. Anschließend wurde der nächste Sprint geplant.

In der Retrospektive wurden besonders Teamwork, Kommunikation und Hilfsbereitschaft positiv hervorgehoben. Des Weiteren wurde nun positiv angemerkt, dass mehr in den Projektgruppenräumen gearbeitet wird und die Teilnahme an den Gruppentreffen gut ist. Code Reviews und Konzepte sollten ebenfalls beibehalten werden. Allgemein wurden in dieser Retrospektive wenige negative Punkte genannt. Es wurde jedoch angemerkt, dass mehr auf Sauberkeit geachtet werden soll. Hierzu gehört sowohl die Sauberkeit und Vorbereitung bei der Versionskontrolle und der Pull-Requests, als auch die Qualitätssicherung des Codes. Außerdem soll mehr auf Tests und Dokumentation geachtet werden. Letztlich wurde noch angemerkt, dass die Internetverbindung in den Projektgruppenräumen zu langsam ist und so die Arbeit in den Räumen beeinträchtigt. Die negativen Punkte der letzten Retrospektive wurden überwiegend verbessert. Bei einem der übriggebliebenen Punkte, Tasks bei Bedarf während des Sprints aufzuteilen, wurde angemerkt, dass dies bei den Tasks in diesem Sprint nicht nötig war. In Zukunft soll vermehrt auf Sauberkeit und Qualität des Codes und des Taskworkflows geachtet werden. Vor Allem sollen die Commitnachrichten aussagekräftiger werden, damit daraus automatisch ein Changelog generiert werden kann. Ebenfalls sollen Bugs mit höherer Priorität eingestuft werden, da sie häufig die Arbeit an anderen Tasks behindern.

7.3.15 Sprint 15

Im Zeitraum vom 07. Dezember bis zum 20. Dezember fand der 15. Sprint statt. Die Hauptziele dieses Sprints waren die Fehlerbereinigung des E-Stream Produkts und die Aktualisierung der Dokumentation. Da die Dokumentation in den vorherigen Sprints immer wieder als nicht kritisch eingestuft wurde, wurden andere Aufgaben vorrangig bearbeitet. Konkret wurde neben der Aktualisierung der Dokumentation auch geplant, dass Klassen-, Sequenz- und Aktivitätsdiagramme zur richtigen Dokumentation des Quellcodes erstellt werden sollen.

Das Hauptziel des Sprints wurde fast vollständig erreicht, denn es konnten alle Bugfix-Tasks bis auf den Slow-Test Bugfix und den e2e-Test Bugfix erledigt werden. Das weitere Ziel des Sprints war die Erledigungen der Dokumentationsaufgaben, da diese immer wieder nach hinten verschoben worden waren. Auch in diesem Sprint wurden diese erneut aus Prioritätsgründen nicht erledigt.

Bei der ersten Gruppensitzung am 13. Dezember 2017 wurde das Konzept zur Überarbeitung der Projektstruktur des E-Stream Clients vorgestellt. In der Sprintabschlusssitzung am 20. Dezember wurde der neue Sprint- und Meilensteinplan für die restliche Projektzeit vorgestellt und der erste Ansatz zur Implementierung der Anzeige der Netzdatentopolgie wurde präsentiert. Hierbei wurden zudem Problemstellen bei der Umsetzung erläutert und in der Gruppe diskutiert. Außerdem wurde in diesem Sprint die Umsetzung auf Docker in der Produktionsumgebung sowie in der Entwicklungsumgebung fertiggestellt. Die umgesetzten Bugfix-Tasks haben Fehler beim Jenkins E-Stream Build, QueryManager/QueryBuilder Integrationstests, Hibernate Datenbank Anbindung, Swagger Schnittstellengenerierung und Gradle ProductionBuild korrigiert.

In der Retrospektive wurde die Kommunikation, die privaten gemeinsamen Aktionen und das Teamwork als besonders positiv hervorgehoben. Als weitere positive Aspekte der Gruppenarbeit wurde noch das Dokumentieren der Tasks in der Kommentarfunktion von Jira, die Code-Reviews in den Sitzungen sowie das Erstellen von ausführlichen Konzepten genannt.

Als nächstes wurden die Punkte angesprochen, die in Zukunft umgesetzt werden sollen. Hierbei wurde konsequenterweise das Thema der weiteren Dokumentationsaufgaben genannt, da diese Aufgaben in diesem Sprint wieder bewusst runterpriorisiert wurden und dies folgerichtig nicht in weiteren Sprints so weiterlaufen sollte. Außerdem wurde vorgeschlagen als Gruppe einen gemeinsamen Hackathon zu veranstalten, um weiterhin gemeinsame Aktionen zu unternehmen. Der Hackathon kann sich um das Thema der Projektgruppe bewegen oder auch etwas komplett anderes behandeln.
Der einzige negativ genannte Punkt ist, dass zur Zeit einige Gruppenmitglieder ihre Aufgaben alleine bzw. zu Hause erledigen, anstatt einige Themen gemeinsam im Gruppenraum zu bearbeiten.

7.3.16 Sprint 16

Der 16. Sprint wurde im Zeitraum vom 20. Dezember 2017 bis zum 03. Januar 2018 durchgeführt. Dieser Sprint war also in der Ferienzeit. Da in dieser Zeit Weihnachten und Silvester war, hatte das Team eine Woche Urlaub bekommen. Dadurch war dieser Sprint vergleichsweise kurz. Aus diesem Grund hat das Team beschlossen, dass in dieser Zeit an den noch offenen Aufgaben weitergearbeitet wird. Wenn keine Aufgaben mehr da sind, konnten beliebige Aufgaben aus dem Backlog in den Sprint reingezogen werden.

Die Sprintzeile waren das Anpassen der Client Projektstruktur sowie die Erweiterung der Dokumentation. Das Hindernis in diesem Sprint war, dass Bitbucket und die DemoVM oft nicht funktioniert hatten. Außerdem konnte Docker aufgrund eines Fehlers im Startup-Script nicht gestartet werden. Dadurch hatte das Team Schwierigkeiten mit mehreren Personen an einer Aufgabe zu arbeiten und unser Backend konnte nicht mit dem production-Profil gestartet werden.

In diesen Sprint wurde der Docker-Startup Fehler korrigiert und die Review Checkliste überarbeitet. Außerdem wurde das Konzept für die Client Projektstruktur in die Dokumentation übernommen und zum Großteil auch im Quellcode umgesetzt. Auch wurde an der Implementierung der Topologie für die Anzeige der Netzdaten weitergearbeitet und es gab erste Versuche die Blockierung der Tests durch Kafka zu beheben.

Beim Sprint Review wurde die angepasste Review Checkliste vorgestellt und die bisherigen Fortschritte bei der Behebung der Kafka Blockierung vorgestellt. Außerdem wurde nochmal gezeigt, wie lokal mit Docker getestet werden kann. Eine Retrospektive wurde nicht durchgeführt, da die DemoVM und somit auch der Scrumbot nicht funktioniert hatten.

7.3.17 Sprint 17

Der 17. Sprint startete am 03. Januar 2018 und wurde am 17. Januar 2018 abgeschlossen. Wesentliche Ziele des Sprints waren der Abschluss von Aufgaben, welche noch aus dem Vorjahr offen geblieben waren, sowie das Erarbeiten von konkreten Konzepten zum Data Mining mit Odysseus und die Evaluation der Performance des Systems.

Von den ursprünglichen Zielen des Sprints konnte nur das Konzept zur Evaluation der Performance vollständig umgesetzt werden. Darüber hinaus wurde jedoch auch die Dokumentation erweitert und die Überarbeitung der Projektstruktur im Frontend endgültig abgeschlossen. Hierzu wurde auch ein Code-Review mit der gesamten Gruppe abgehalten. Außerdem wurden Bugs zur Blockierung der Tests durch Kafka behoben und mit der Implementierung der Query Manager GUI begonnen. Aufgrund

diverser Probleme mit Kafka-Connect wurde in der Gruppe beschlossen dies nicht mehr zu verwenden, sondern sich stattdessen auf wichtigere Probleme zu fokussieren. Zusätzlich fand in diesem Sprint ein erneutes Treffen mit der PG-DAvE statt. Hierbei kam es zunächst zu Unstimmigkeiten darüber, ob die Umsetzung der Zusammenarbeit tatsächlich verpflichtend ist. Es wurde daher unsererseits beschlossen, zunächst ein weiteres Treffen mit der PG-DAvE abzuwarten bevor weitere Arbeitszeit in die Entwicklung und Umsetzung der Schnittstelle investiert wird. Dieses Treffen sollte dann in Anwesenheit der Projektbetreuer zu Beginn des 18. Sprints erfolgen. Außerdem wurden in diesem Sprint innerhalb der Gruppe Kleingruppen gebildet, welche für einzelne Bereiche des Systems aufgabenübergreifend verantwortlich sein sollten. Die Gruppen zu „Angular und GUI“, „Data Mining und Odysseus“, „Spring, Backend und Mock Server“sowie „Kafka, Infrastruktur und Evaluation“sollten dabei insbesondere darauf achten, dass Aufgaben zum jeweiligen Bereich auch tatsächlich erledigt und nicht immer weiter verschoben werden.

In der Retrospektive des Sprints wurden Teamwork und die gute Motivation innerhalb der Gruppe erneut positiv angemerkt. Dabei ist jedoch aufgefallen, dass dies inzwischen offenbar nicht mehr von sehr vielen Gruppenmitgliedern so empfunden wird. Im Gegensatz zu früheren Retrospektiven wurde nun auch auf eine mangelnde Kommunikation, insbesondere in Form des Antwortverhaltens auf Slack, aber auch in Bezug auf fehlende Kommentare in Jira, hingewiesen. Hierzu wurden entsprechende Actions beschlossen, um die Kommunikation innerhalb des Teams wieder zu stärken.

7.3.18 Sprint 18

Der 18. Sprint fand vom 17. bis zum 31. Januar 2018 statt und umfasst wie üblich somit zwei Wochen. Ziel dieses Sprints war ursprünglich die Umsetzung des Data-Mining Konzepts, die Fertigstellung der dynamischen Dashboard-Komponente für das Frontend und die Dokumentation durch das Erstellen neuer Kapitel und Überarbeiten bereits vorhandener Kapitel weiter voranzutreiben.

Aufgrund der Verzüge aus vorherigen Sprints, war es nicht möglich, alle Ziele zu erreichen. Dennoch wurde ein erheblicher Teil der dynamischen Dashboard-Komponente fertiggestellt und es konnten zwei Dokumentationstasks erfolgreich abgeschlossen werden. Ein weiterer wichtiger Punkt, welcher in diesem Sprint erreicht wurde, ist, dass die andauernden Prombleme mit der Firewall unseres Demosystems dauerhaft gelöst werden konnten. So war es zuvor des Öfteren vorgekommen, dass das System von außerhalb nicht erreichbar war oder einzelne Komponenten des Systems, wie beispielsweise Odysseus oder Kafka nicht erreichbar waren. Zudem wurde ein wesentlicher Teil der QueryManager GUI in diesem Sprint umgesetzt.

Am 28.01.18 gab es ein weiteres Treffen mit der PG-DAvE. Ziel dieses Treffens war die Finalisierung der Schnittstellen zwischen den Projektgruppen. Dazu wurden während des Treffens alle API-Endpunkte durchgegangen und überprüft ob diese den Ansprüchen der Projektgruppen genügen.

Des Weiteren wurde in diesem Sprint klargestellt, dass die Zusammenarbeit der PG-E-Stream und PG-DAvE obligatorisch ist, aber die Abschlusspräsentationen separat stattfinden.

In der Retrospektive wurde der *Hackathon* wieder angesprochen. Dieser soll nun in den folgenden Wochen konkretisiert werden. Neben den üblichen Punkten wurde auch angesprochen, dass neben der Projektgruppe teambildende Maßnahmen gefördert werden sollen, um die Motivation und Kommunikation zwischen den Projektgruppenmitgliedern zu steigern.

7.3.19 Sprint 19

Vom 01. Februar 2018 bis zum 14. Februar 2018 fand der 19. Sprint statt. Ziel dieses Sprints war es das Konzept für Data Mining, die begonnenen Tasks für das Frontend des Systems abzuschließen und mit der Implementierung des Data Minings anzufangen.

Die Produktivität dieses Sprints war stark reduziert, da der Großteil der Gruppe sich Urlaub wegen der Klausurenphase genommen hat. In der Woche vom 05.02. bis 11.02. hat nur ein Gruppenmitglied durchgehend gearbeitet. Dennoch wurde beschlossen sich am 07.02. zu treffen, um den Fortschritt der Aufgaben aus dem letzten Sprint zu besprechen und etwaige Probleme zu klären.

Durch die Abwesenheit des Großteils der Gruppe wurden die Sprintziele in diesem Sprint nicht erreicht. Dennoch konnte unter Anderem die Umstellung der Simulationsdaten von JSON auf XML abgeschlossen werden. Auf Grundlage davon und eines Bugfixes im Odysseus Docker Image wurde das Konzept für das Data Mining begonnen und parallel dazu ebenfalls die Implementierung der Anfragetemplates für das Data Mining angefangen. Weiterhin wurden die Use Cases der Projektgruppe in die Dokumentation eingepflegt. Außerdem wurde die mit PG DAvE vereinbarte Schnittstelle für die Freigabe von Rohdaten implementiert, so dass externe Datenarchive die eingehenden Daten von einem Kafka Topic auf Nachfrage auslesen können.

Am Ende des Sprints wurde innerhalb der Retrospektive angesprochen, dass der Fortschritt dieses Sprintes stagnierte, da der Großteil der Gruppe Urlaub hatte und somit kaum Aufgaben erledigt wurden. Als einzig Positives wurde die Reaktion auf Fragen in Slack erwähnt und die Hilfsbereitschaft. Ansonsten wurden die mangelnde Motivation, fehlende Kommunikation über den Fortschritt bei Aufgaben, die Vernachlässigung von Aufgaben in der Projektgruppe wie zum Beispiel Protokolle anfertigen und nicht erbrachte Arbeitsstunden genannt. Deswegen wurde beschlossen, dass in Sprint 20 eine Sondersitzung stattfinden soll, um diese Punkte in der ganzen Gruppe anzusprechen und mit einem anschließenden gemeinsamen Essen die Motivation zu steigern.

7.3.20 Sprint 20

Sprint 20 fand vom 15. Februar bis zum 7. März statt und weichte somit mit drei Wochen von der üblichen Länge ab. Aufgrund der geringeren Produktivität des vorherigen Sprints, wurden für diesen Sprint Wege überlegt, die Produktivität zu steigern.

Eine Konsequenz daraus war, dass beschlossen wurde *Hackathon*s zu veranstalten. Dies bedeutet, dass an einem Tag möglichst alle Teilnehmer der Projektgruppe sich zusammensetzen und vorher festgelegte Ziele versuchen zu erreichen. Das geschah sowohl in Einzel- als auch in Gruppenarbeit. Um die Motivation und Stimmung hoch zu halten wurden die *Hackathons* immer mit mindestens einer gemeinsamen Mahlzeit verbunden. Der Zeitraum von einem Hackathon umfasste in der Regel zehn bis zwölf Stunden.

Eine weitere Konsequenz war, dass ein neuer Meilensteinplan ausgearbeitet werden musste, damit alle wichtigen Ziele noch erreicht werden können. Zudem wurde beschlossen, dass am Ende jeder Sitzung jeder Teilnehmer kurz mitteilt, mit welchem Thema er sich als nächstes beschäftigen möchte.

Die Ziele für den 20. Sprint waren unter anderem die Implementierung aller Features, Fortschritte bei der Projektdokumentation und bei der Umsetzung des Evaluierungskonzepts.

Am 1. März fand der erste *Hackathon* statt. Ziele die definiert wurden, waren:

- Den Mock, welcher die Interaktion mit der Projektgruppe DAvE simulieren soll, vorzubereiten und zu verifizieren.
- Prüfen ob auf Kafka Broker außerhalb unseres Netzes zugegriffen werden kann.
- Schreiben weiterer PMML Anfrage Templates.
- erfolgreiche Anfrageergebnisse visualisieren.
- Ausreißererkennung in unser Produkt integrieren.
- diverse kleinere Verbesserungen und Bugfixes.

Der erste Hackathon war sehr erfolgreich. Alle zu Beginn definierten Ziele konnten erreicht werden und alle Teilnehmer der Projektgruppe haben sich gut eingebracht. Innerhalb dieses Sprints fand am 6. März ein weiterer *Hackathon* statt. Diesmal wurden die folgenden Ziele definiert: Zusammenführen verschiedener offener Featurebranches, Fertigstellen des Mocks der PG DAvE, das Umstrukturieren des Frontends sowie kleinere Verbesserungen und Bugfixes.

Am Ende des Sprints befanden sich keine Tasks mehr im Sprint Board in der Spalte *TODO*. Die Retrospektive ergab, dass keine neuen Features mehr implementiert werden sollen. Außerdem ist mehrmals der Wunsch nach weiteren *Hackathons* geäußert worden. Zudem soll keine weitere Energie in den Ausbau der Infrastruktur gesteckt werden. Obwohl die *Hackathons* im Allgemeinen gut angenommen worden sind, ist der Wunsch geäußert worden die Ablenkung durch beispielsweise Handys zu reduzieren. Abschließend kann der Sprint 20 als ein sehr produktiver Sprint betrachtet werden.

7.3.21 Sprint 21

Vom 07. März 2018 bis zum 21. März 2018 fand der 21. Sprint statt. Zu den Zielen für diesen Sprint gehörte neben der Vorbereitung für die Präsentation auch das ausführliche Testen und Fehlerbehebung des Produkts. Zusätzlich wurden viele Aufgaben für die Dokumentation angedacht, welche JavaDoc und JavaScriptDoc umfassten. In der Hauptdokumentation sollten Latex-Fehler behoben werden und der gesamte Text zum derzeitigen Stand korrigiert und überarbeitet werden. Die geplante Umsetzung der Usability-Studie wurde aufgrund von Zeitmangel auf den nächsten Sprint verschoben, da noch keine Testpersonen zur Verfügung standen. Die benötigten Testpersonen für die Tests wurden dann aber am Ende von Michael im Rahmen von Offis Mitarbeitern und Studenten vorgeschlagen.

Für eine effektive kollektive Fertigstellung der Aufgaben wurde am 14. März 2018 ein weiterer Hackathon abgehalten. Aus dem vorherigen Sprint wurde die Datenverarbeitung im Client (Data-Service) sowie die Topologievisualisierung übernommen. Des Weiteren musste ein Fehler behoben werden, welcher dafür sorgte, dass die Tests auf dem Jenkins fehlgeschlagen sind.Da am Tag des Gruppentreffens der ersten Sprintwoche gleichzeitig der Hackathon abgehalten wurde und Michael sich im Urlaub befand, ist dieses ausgefallen. Gleichzeitig gab es aufgrund der genauen Aufgabenverteilung kaum zu besprechende Tagesordnungspunkte.

Am Ende des Sprints ist in der Retrospektive aufgefallen, dass einige der Doku-Tasks zwar noch offen waren, jedoch teilweise schon durch überschneidende Themen abgearbeitet wurden. Diese wurden als erledigt markiert, um Missverständnisse zu vermeiden. Die Usability-Tests sind aus der Sicht der

Fragestellung soweit fertig gestellt worden. Auch die notwendigen Anfragen zur Visualisierung des Data Minings wurden entsprechend erzeugt und können mit den Probanden durchgegangen werden. Es waren jedoch noch kleine Anpassungen der angegebenen Parameter notwendig.

Im Client wurden die Navigationstabulatoren umstrukturiert, sodass die Topologie in einem eigenen Tab angezeigt wird und gleichzeitig auch an anderer Stelle im Client dargestellt werden kann. Zusätzlich wurden die Beschreibungen für die Anfrage-Templates eingefügt und die geografische Karte wurde prototypisch umgesetzt. Des Weiteren wurden veraltete Komponenten ausgegliedert und durch die generischen ausgetauscht. Die Benutzerverwaltung, im Sinne der Anzeige der Tabulatoren im Dashboard, die jeweils für den Benutzer sichtbar sein dürfen, wurden implementiert. Beispielsweise ist der Playgroun-d und der Monitor-Tab nur für Administratoren sichtbar, da er ansonsten für normale Benutzer nicht zugänglich sein soll. Die letzten Fehlerbehebungen an den Odysseuskomponenten, der Mosaik-Topologie und der Anreicherung der Topologie sind noch in Arbeit und wurden entsprechend im nächsten Sprint weiter bearbeitet.

7.3.22 Sprint 22 und Projektabschluss

Sprint 22 fand vom 21. März bis zum 06. April und damit dem Ende der Projektgruppe statt. In diesem letzten Sprint war die Fertigstellung der Abschlusspräsentation und Zeit für den Feinschliff sowie unvorhergesehene Probleme eingeplant. Da die Usability Studie sich verzögert hat, wurde sie außerdem in diesem Sprint durchgeführt. Zusätzlich wurde die Dokumentation vervollständigt. Um die Umsetzung dieser Ziele voranzutreiben, insbesondere im Hinblick darauf, dass dies der Endspurt ist und anschließend alle geplanten Aufgaben vollendet sein müssen, wurde ein letzter *Hackathon* beziehungsweise *Dokuthon* durchgeführt. Die Ziele hierfür waren die folgenden:

- Präsentation überarbeiten
- Usability Tests durchführen
- Performance Evaluation abschließen
- Topologieanreicherung
- Anforderungsanalyse prüfen/überarbeiten/korrigieren
- Dokumentation fertigstellen

Von diesen wurde bis auf die vollständige Prüfung der Anforderungsanalyse und die Fertigstellung der Dokumentation alles erledigt. Da also noch letzte Inhalte der Dokumentation hinzugefügt werden mussten und der letzte Feinschliff für die Präsentation vorgenommen werden sollte, fand am Dienstag den 03.04.18 ein weiteres Treffen statt. Am folgenden Tag wurde die Abschlusspräsentation vorgestellt und letzte Texte vervollständigt beziehungsweise letzte Textprüfungen durchgeführt. Somit wurde die Projektgruppe erfolgreich beendet.

8 Produktbeschreibung

In diesem Abschnitt werden die Konzepte und deren Implementierung im Projekt behandelt. Dabei wird insbesondere auf die Architektur des Gesamtsystems, das Backend, das Frontend und Odysseus eingegangen.

8.1 Architektur

Die Architektur des Gesamtsystems besteht aus den fünf Komponenten Kafka, Odysseus, Backend, Frontend und einer Datenbank. Deren Zusammenhang und Kommunikation untereinander und mit externen Systemen wird in Abbildung 8.1 dargestellt.

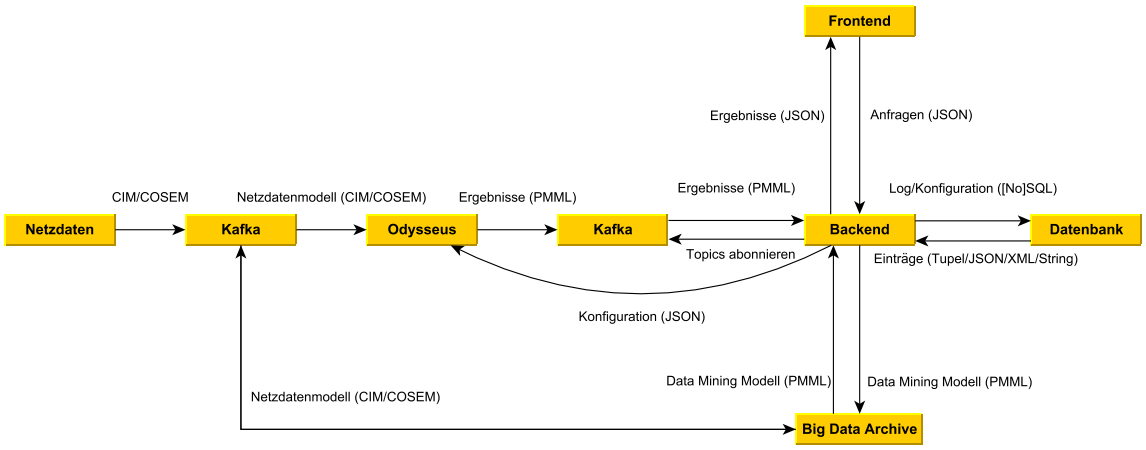


Abbildung 8.1: *Grobarchitektur des Gesamtsystems*

Hier werden Netzdaten durch eine mosaik-Simulation erzeugt und in ein Kafka-Cluster geschrieben. Die eingehenden Daten werden an ein OdysseusNet übertragen und dort verarbeitet. Die ermittelten Ergebnisse werden zurück in das Kafka-Cluster geschrieben und somit anderen Teilsystemen zur Verfügung gestellt. Durch die Wahl von Kafka und Odysseus als verarbeitende Systeme kann das System mit steigenden Anforderungen auch verteilt betrieben und somit skaliert werden.

Das Backend kann auf alle Daten im Kafka Cluster zugreifen und ist die Schnittstelle für andere Systeme. Insbesondere werden die Daten aus der Verarbeitung durch das Frontend visualisiert und über Anfragen des Frontends können weitere Daten angefragt und das System gesteuert werden. Durch eine Benutzerverwaltung werden allerdings nicht allen Benutzern alle Operationen zur Verfügung gestellt.

Des Weiteren ist es der PG DAvE so auch möglich auf weitere Rohdaten für das Archiv zugreifen zu können oder neue Modelle an das Backend zu übergeben.

In der Datenbank sind Templates für PQL Anfragen und die Nutzer persistiert. Darüber hinaus werden auch Warnungen für zum Beispiel Lastspitzen oder Ausfälle dauerhaft gespeichert, damit sie bei Bedarf abgefragt werden können. Im Folgenden werden die zentralen Komponenten mit ihrer eigenen Architektur detaillierter dargestellt.

8.1.1 Odysseus

Die Verarbeitung wird in Abbildung 8.2 dargestellt. Für die Datenquelle wird neben dem Kafka *TransportHandler* ein DLMS/COSEM *ProtocolHandler* und ein Tupel *DataHandler* benutzt [209]. Die bereits vorhandenen Datamining Operatoren können für die Verarbeitung benutzt werden und die zustandsbehafteten Operatoren werden so verändert, dass zur Laufzeit ein neues Modell über die REST-Schnittelle gesetzt werden kann. Als Datensenke wird wieder eine Kafka-Senke benutzt, die mit Hilfe einer PMML-Komponente Ergebnisse in einem standardisierten Format zurückschreiben kann.

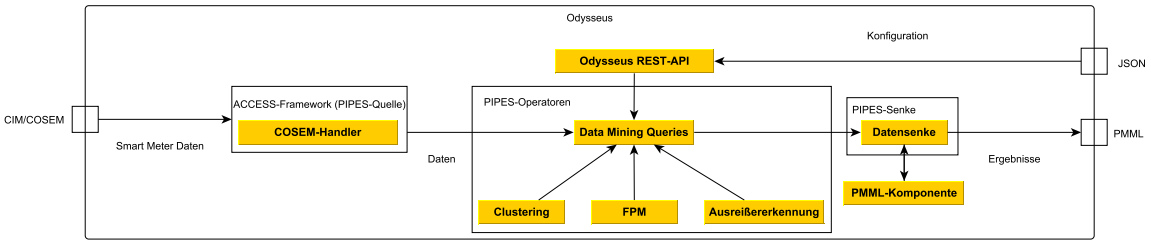


Abbildung 8.2: *Wichtige Komponenten in Odysseus*

8.1.2 Backend

Das Backend besitzt vier wichtige Schnittstellen: die REST-API, die Schnittstellen zu Kafka und Odysseus sowie die Datenbankanbindung. Über die REST-API kommunizieren andere Systeme mit dem Backend, während das Backend über die entsprechenden Komponenten mit den verarbeitenden Systemen kommuniziert. Zusätzlich besitzt das Backend eine PMML-Komponente zur Verarbeitung wie Abbildung 8.3 zeigt. Durch die Kapselung der Schnittstelle für externe Systeme im Backend können alle Anfragen durch eine Benutzerverwaltung eingeschränkt werden, wodurch sich die Gesamtsicherheit des Systems erhöht.

8.1.3 Frontend

Das Frontend besitzt eine große Anzahl an Visualisierungskomponenten, die über einen *DataService* mit Daten versorgt werden.

8.1.3.1 Angularstruktur

In diesem Abschnitt wird die Architektur des Clientprojekts vorgestellt. Diese beinhaltet sowohl die Struktur der Bausteine, die durch Angular zur Verfügung gestellt werden, als auch die Ordnerstruktur des Projekts. Die hier vorgestellte Architektur basiert größtenteils auf dem Style Guide von Angular [12]. Die drei wichtigsten Bausteine für die Projektstruktur in Angular sind Module, Komponenten und Services.

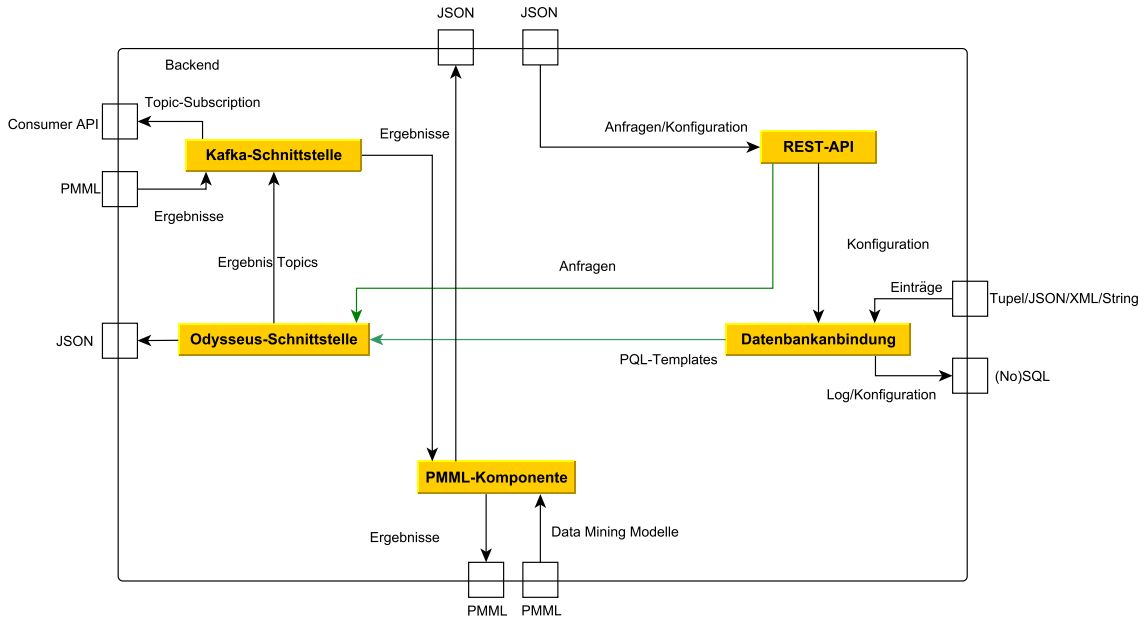


Abbildung 8.3: *Komponenten im Webserver-Backend*

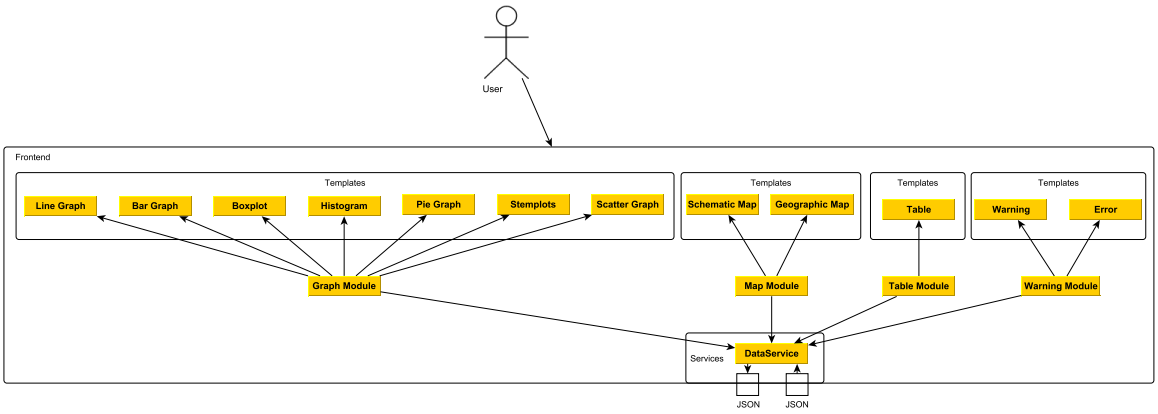


Abbildung 8.4: *Komponenten im Webserver-Frontend*

Module

Module werden in der Dokumentation von Angular als „cohesive block of code dedicated to an application domain, a workflow, or a closely related set of capabilities“ beschrieben. Sie beinhalten Komponenten und Services und gruppieren sie so in logische Einheiten. Dieser Zusammenhang sollte sich auch in der Ordnerstruktur widerspiegeln. Module können des Weiteren andere Module importieren und bestimmte Funktionalitäten dieser importierten Module in einem sogenannten Convenience Module bündeln und erneut zu Verfügung stellen. In der Angular Dokumentation werden Module grob in fünf Klassen unterteilt. Diese sind Domain, Routed, Routing, Service und Widget [10].

- Domain Module exportieren in der Regel eine Top-Level-Komponente und deklarieren sonst nur private Helferkomponenten. Sie benutzen eher selten Services. Ein Beispiel für ein Domain Modul ist ein Modul zum Bearbeiten eines Nutzers.
- Routed Module sollten nichts exportieren, da sie lediglich durch einen Router über ihren Typ angezeigt werden. Alle lazy-loaded Module sind automatisch Routed Module. Sie sollten in der Ordnerstruktur zusammen mit ihrem Router gruppiert werden.
- Routing Module sind Module, die das Routing für andere Module übernehmen. Dies erhöht die Übersichtlichkeit des gerouteten Moduls und bleibt dem Single Responsibility Prinzip treu.
- Service Module bestehen im besten Fall nur aus Providern. Sie sollten keine Deklarationen oder Exporte enthalten. Service Module werden in der Regel nur im Root-Modul importiert.
- Widget Module stellen Widgets bereit und sollte daher ihre Widgets exportieren. Sie sollten keine Provider besitzen. Widget Module werden da importiert, wo ihre Widgets benötigt werden.

Eine detailliertere Erklärung zu den Modulgruppen kann in der Angular Dokumentation gefunden werden [11].

Feature Module	Declarations	Providers	Exports	Imported By	Examples
Domain	Yes	Rare	Top component	Feature, `AppModule`	`ContactModule` (before routing)
Routed	Yes	Rare	No	Nobody	`ContactModule`, `HeroModule`, `CrisisModule`
Routing	No	Yes (Guards)	`RouterModule`	Feature (for routing)	`AppRoutingModule`, `ContactRoutingModule`, `HeroRoutingModule`
Service	No	Yes	No	`AppModule`	`HttpModule`, `CoreModule`
Widget	Yes	Rare	Yes	Feature	`CommonModule`, `SharedModule`

Abbildung 8.5: *Modularten nach dem Angular Style Guide*

Zwei spezielle Modularten, die die Übersichtlichkeit im Projekt verbessern, sind das Shared Module und das Core Module. Das Shared Module beinhaltet Komponenten, Direktiven und Pipes, die an sehr vielen Stellen benötigt werden. Das Shared Module wird in allen Feature Komponenten importiert, auch in den Komponenten, die lazy-loaded werden. Es sollte allerdings keine Provider enthalten. Mehr Informationen hierzu können in der Angular FAQ für Module gefunden werden. Das Core Module beinhaltet nur Provider, die applikationsweit benötigt werden. In Ausnahmefällen kann es sinnvoll sein, auch Komponenten in das Core Modul aufzunehmen. Es wird ausschließlich im Root-Modul importiert [11].

Komponenten

Eine Komponente „controls a patch of screen called a view“. Sie ist demnach mit einer View, in der Regel in Form eines Templates und eventuell zugehörigen Styles verbunden. Eine Komponente beinhaltet Logik, die die View unterstützt und sollte alle nichttrivialen Aufgaben an Services delegieren, um die Übersichtlichkeit zu wahren. Das heißt, dass eine Komponente hauptsächlich Properties und Methoden für das Data-Binding bereitstellt. Aus der Dokumentation: „A good component presents properties and methods for data binding. It delegates everything nontrivial to services.“. Die Zusammengehörigkeit von Komponenten und ihren Templates sollte auch in der Ordnerstruktur sichtbar sein. Services hingegen werden nicht unbedingt nur von einer Komponente genutzt und sind daher schwächer an die Komponente gekoppelt [9].

Services

Services sind Klassen, die eine wohldefinierte Aufgabe erfüllen. Die Aufgabe sollte dabei relativ feingranular gehalten werden bzw. größere Aufgaben sollten in mehrere wohldefinierte Services aufgeteilt werden. Beispiele für Services beinhalten Logging, Datenbeschaffung, Konfiguration der Applikation oder Berechnung von Steuern [9]. Angular unterstützt Dependency Injection und erleichtert damit die Verwendung von Services. Services sind in der gesamten Applikation verfügbar wenn sie einmal bereitgestellt wurden, egal wo sie bereitgestellt werden. Die Ausnahme hierzu ist, wenn das Modul, das sie bereitstellt, mittels lazy-loading geladen wird. Dann erstellt Angular einen Child-Injector für das Modul, das lazy-loaded wurde.

Single Responsibility

Das Prinzip der Single Responsibility sollte auf alle hier vorgestellten Bausteine angewandt werden. Es hilft dabei, die Applikation sauberer und wartbarer zu halten und verbessert die Testbarkeit. Zu diesem Prinzip gehört auch, dass nur ein Baustein pro Datei definiert wird. Eine Datei beinhaltet demnach entweder eine Komponente oder einen Service oder einen anderen Baustein [278].

Der erste Teil dieses Konzeptes bezog sich hauptsächlich auf die internen Zusammenhänge und Bausteine von Angular und deren Strukturierung. Der Abschnitt sollte die wichtigsten Bausteine grundlegend erläutern. Weitere Information können in der Angular Dokumentation und im Angular Style Guide nachgelesen werden. Im nächsten Abschnitt wird verstärkt auf die Ordnerstruktur des Projekts eingegangen.

8.1.3.2 Projektordnerstruktur

Für die Ordnerstruktur in einem Angularprojekt gibt es mehrere Möglichkeiten. Jede von ihnen ist für einen anderen Projekttyp am besten geeignet. Im Folgenden wird nur die Variante vorgestellt, die sich am besten für das E-Stream Projekt eignet. Sie wird im Angular Style Guide beschrieben und Abbildung 8.6 zeigt eine Übersicht.

- Der gesamte Code für die Applikation sollte sich im Verzeichnis src/ befinden, das direkt unter dem Root-Verzeichnis des Projekts liegt

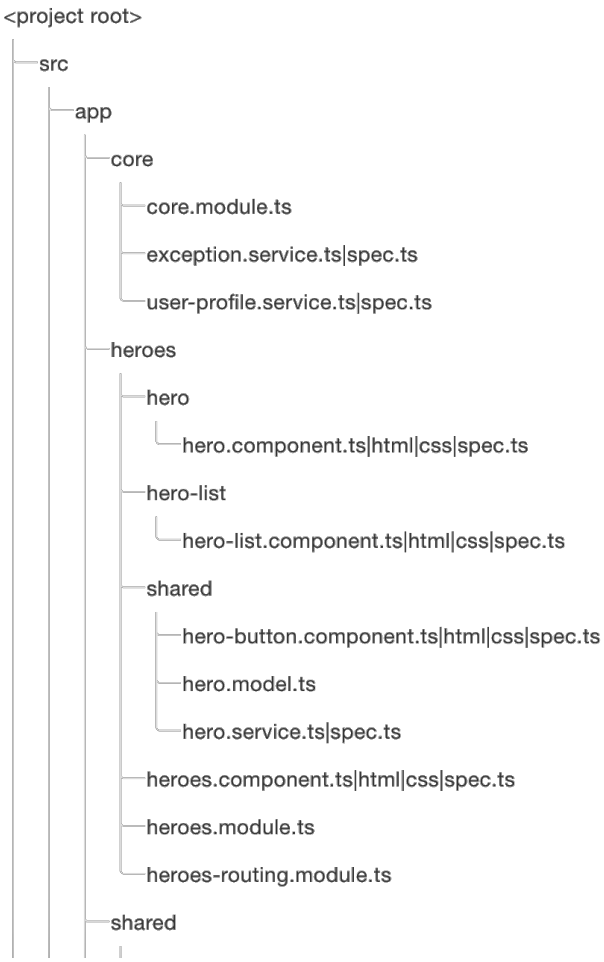


Abbildung 8.6: *Ordnerstruktur Überblick*

- Das Bootstrapping der Applikation sollte in einer Datei main.ts geschehen, die sich direkt im src/ Verzeichnis befindet
- Im src/ Verzeichnis befindet sich ein Verzeichnis assets/, dass Assets wie zum Beispiel i18n-Dateien oder globale Styles enthält
- Im src/ Verzeichnis befindet sich ein Verzeichnis app/, das alle Features beinhaltet
- Direkt unter dem app/ Verzeichnis befindet sich das Modul app.module.ts, welches das Root-Modul der Applikation repräsentiert und die zugehörige Komponente app.component.ts sowie die zugehörigen Dateien. Keine weiteren Module, Komponenten, ... befinden sich direkt unter dem app/ Verzeichnis
- Die Applikation sollte in Features nach der Folders-by-Feature Struktur unterteilt werden
  - Jedes Feature wird in einem Verzeichnis mit dem Featurenamen gruppiert, das sich unter dem app/ Verzeichnis befindet
  - Jedes Feature wird als Modul repräsentiert

  - Features können ein shared/ Verzeichnis enthalten, das Komponenten, Services und Modelle enthält, die von allen anderen Komponenten des Features benötigt werden
  - Features können ein Routing Module enthalten, welches das Routing für das Feature übernimmt. Dieses wird im selben Verzeichnis wie das Feature Module definiert
- Die Applikation sollte ein Shared Modul enthalten Abbildung 8.7
  - Hier werden Komponenten, Direktiven und Pipes definiert, die in vielen anderen Modulen genutzt werden

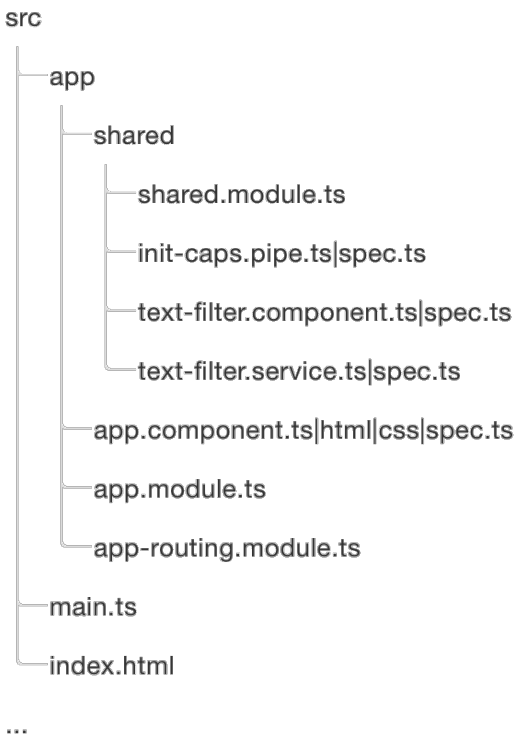


Abbildung 8.7: *Shared Modul Überblick*

- Die Applikation sollte ein Core Modul enthalten, das applikationsweite Services bündelt, die an vielen Stellen benötigt werden Abbildung 8.8
  - Das Core Modul beinhaltet nur Provider
  - Beispiele für Services beinhalten LoggingService, ExceptionService
  - Unter Umständen können auch Komponenten in das Core Modul integriert werden [12]
- Für jede Komponente sollte ein Verzeichnis angelegt werden, dass die verschiedenen Dateien der Komponente (*.ts, *.html, *.css, ...) beinhaltet
  - Der Verzeichnisname sollte die Komponente widerspiegeln bzw. das Äquivalent hierzu nach den Dateinamenkonventionen sein.

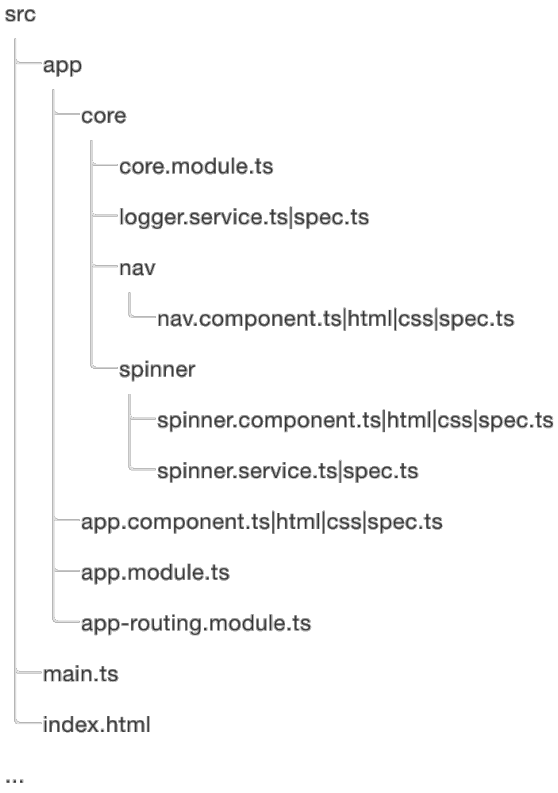


Abbildung 8.8: *Core Modul Überblick*

Shared Modul

In diesem Projekt beinhaltet das Shared Modul wichtige Importe, die die Angular Funktionalitäten ermöglichen, wie zum Beispiel das FormsModule und das CommonModule. Des Weiteren werden einige Material Design Komponenten in das Shared Modul aufgenommen, da sie an sehr vielen Stellen verwendet werden. Außerdem wurde das TranslateModule aufgenommen, um überall die Internationalisierungsmöglichkeiten bereitzustellen. Das autogenerierte Shared Model wird ebenfalls übernommen. Letztlich wurden noch einige Helferklassen aufgenommen, wie zum Beispiel die BaseError Klasse.

Core Modul

Im Core Modul werden applikationsweite Services gebündelt. Die meisten Services im Projekt sind derzeit applikationsweit. Hierzu gehören Services, die die Daten beschaffen, wie zum Beispiel der DataService. Der UserService wird als administrativer Service aufgenommen. Der TopologyService gehört ebenfalls zum Core Modul.

Folders-by-Feature

Durch die Aufteilung des Projekts in die Folders-by-Feature Struktur haben sich sieben Features ergeben, die im Folgenden kurz beschrieben werden.

- Die erste Feature Area ist Topology Feature Area. Sie befasst sich mit allen Belangen der Topologie, die dargestellt wird. Sie beinhaltet demnach sowohl die logische, als auch die visuellen Repräsentationen der Topologie.
- Die Admin Feature Area repräsentiert das zweite Feature. Hier werden administrative Funktionalitäten gebündelt. Sie beinhaltet zum Beispiel die Admin- und Userkomponenten.
- Die nächste Feature Area ist die Charting Area. Diese beinhaltet verschiedene Diagrammtypen, wie zum Beispiel Liniendiagramme, Balkendiagramme oder Tabellen.
- Die Filter Feature Area stellt einen dynamischen Filter bereit, der zusammen mit allen Diagrammen verwendet kann.
- Das Navigation Feature ist für das Routing und die orchestration des Master-Detail Layouts verantwortlich.
- Des Weiteren gibt es ein Monitor Feature, welcher zum Überwachen von Daten genutzt werden kann. Es benutzt die Visualisierungskomponenten und verbindet diese mit Data Services.
- Als letztes gibt es ein Playground Feature, was lediglich für Testzwecke genutzt wird. Hier können neue Komponenten getestet werden, ohne bestehende Seiteninhalte zu verändern. Dieser Bereich ist nur für Administratoren sichtbar.

8.1.3.3 Internationalisierung

Damit das Produkt leichter in andere Sprachen übersetzt werden kann, wurde schon sehr früh eine Internationalisierungsbibliothek verwendet. Dadurch ist es möglich das Frontend in weitere Sprachen zu übersetzen ohne den Quellcode zu verändern. Da das Frontend eine Angular Anwendung ist, wurde die Internationalisierungsbibliothek ngx-translate[1] verwendet.

Um ngx-translate in einem Modul verwenden zu können, muss es das TranslateModule importieren. Dann können Strings in den Komponenten mithilfe der translate pipe übersetzt werden. In Listing 8.1 ist ein Beispiel für die Verwendung der translate pipe. Dabei wird der in „attribute“ abgespeicherte Spring übersetzt. So können auch zur Laufzeit erstellte Stings übersetzt werden.

```
1 <md-option *ngFor="let attribute of filter.attributes" [value]="attribute">
2     {{attribute | translate}}
3 </md-option>
```

Listing 8.1: *Verwendung der translate pipe*

Die Übersetzungen befinden sich in einer JSON Datei. Diese werden im assets/i18n Ordner abgelegt. Diese Datei muss nach dem Code für die Sprache benannt werden. Für Englisch also „en.json“. Da das Frontend nur auf Englisch angezeigt wird, existiert nur eine Übersetzungsdatei.

```
1 {
2   "id": "identification",
3   "symbol": "object attribute",
4 }
```

Listing 8.2: *Einfache Übersetzungen*

[1] Online unter `https://github.com/ngx-translate/core`

In der JSON Datei ist der Name der Schlüssel der Übersetzung und der Wert die Übersetzung. In Listing 8.2 wäre beispielsweise „symbol“ der Schlüssel für die Übersetzung und „object attribute“ die Übersetzung.

Falls eine Übersetzung nicht gefunden wird, wird der Schlüssel (also hier im Beispiel „symbol“) angezeigt. Fehlende Übersetzungen werden zudem auch in der Konsole geloggt. Er werden auch verschachtelte JSON Objekte unterstützt. Listing 8.3 zeigt ein Beispiel für eine verschachtelte Übersetzung. Die Komponenten können dann mit „RangeFilter.anyAttribute“ auf die Übersetzung zugreifen.

```
1 {
2   "RangeFilter": {
3     "anyAttribute": "any attribute"
4   }
5 }
```

Listing 8.3: *Verschachtelte Übersetzungen*

Um weitere Sprachen zu unterstützen, müssen im Frontend weitere Übersetzungsdateien abgelegt werden. Wenn deutsch unterstützt werden soll, muss also im assets/i18n Ordner eine „de.json“ Datei mit den deutschen Übersetzungen abgelegt werden.

8.1.3.4 Style-Guide von NetzDatenStrom

Das Projekt NetzDatenStrom hat einen Style-Guide für die Visualisierungen von verschiedenen Elementen entwickelt, welche auch in diesem Projekt Anwendung finden könnten. In diesem Sinne wurden die Style-Guide Vorgaben betrachtet und überprüft, welche davon zum aktuellen Zeitpunkt des Projektes bereits umgesetzt wurden oder in Zukunft noch übernommen werden könnten. Dabei wurden die Oberpunkte Philosophie, atomare Elemente, Moleküle und Organismen untersucht.

Alle Unterpunkte der Philosophie wurden in unserem Projekt bereits berücksichtigt. Dazu gehören Alarmmeldungen, ein fehlerverzeihendes Design, nachvollziehbare Systementscheidungen, Individualisierbarkeit und Flexibilität, Informationsgehalt, Situationsbewusstsein schaffen, nutzungsorientierte Interoperabilität, sowie Konsistenz und Standardkonformität.

Unterpunkte der atomaren Elemente sind Eingabefelder, Zeichensymbolkatalog, Buttons, Farbpalette, Tabellen, Textelemente, Schriftarten, Listen und Icons. Unterpunkte von Moleküle sind erweiterte Buttons, erweiterte Eingabefelder, Warnhinweise, Tabellendesign, Brotkrumen und Ausklappmenü.

Bezüglich der atomaren Elemente und der Moleküle bestand die Frage, in wie weit sich die Visualisierung unseres Projektes dem Stil von NetzDatenStrom anpasst. Es wurde beschlossen, dass allgemeine Farbkodierungen, wie beispielsweise, dass bei Warnhinweisen die Warnung großflächig in einer Gefahrenfarbe markiert werden muss und außerdem ein Gefahrenzeichen für den Anwender sofort sichtbar sein soll, übernommen werden. Es sollen jedoch nicht alle Designentscheidungen von NetzDatenStrom Anwendung finden, sodass die Eigenständigkeit unseres Projektes weiterhin bestehen bleibt.

Von dem Oberpunkt atomare Elemente wurde der Punkt Eingabefelder bereits umgesetzt, des Weiteren könnte sich mit dem Punkt Zeichensymbolkatalog auseinandergesetzt werden. Unter diesem lassen sich verschiedenste Zeichen und Symbole finden, welche auch in unserem Projekt Anwendung finden könnten (`http://netzdatenstrom.imis.uni-luebeck.de/specs/atoms/symbols/`).

Weitere Punkte aus dem Oberpunkt Moleküle, denen Beachtung geschenkt wird, sind die erweiterten Eingabefelder (bereits umgesetzt), Warnhinweise und die Brotkrumen, welche als Navigationshilfe dienen und das Ausklappmenü (bereits umgesetzt). Bei dem letzten Punkt, Organismen, wurden die Unterpunkte Formulare und Header bereits unabhängig umgesetzt. Der Unterpunkt Topologie Baum könnte ebenfalls betrachtet werden, da das Steuerelement die Möglichkeit bietet, topologische Zusammenhänge eines Stromnetzes hierarchisch darzustellen. Dies geht ebenfalls einher mit der Konzeptidee für die Topologie, welche in unserem Projekt bisher (Stand: November 2017) wie folgt aussieht: Die verschiedenen Node-Typen benötigen verschiedene Visualisierungen inklusive einer Legende für eine schnelle Identifizierung. Nodes sollten markierbar sein und dadurch eine Veränderung des Farbtons oder Schattenwurf hervorrufen. Ein Vorschlag hierfür wäre bei dem Gesamtnetzwerk ein dunkelblau, für das Netzwerkgebiet ein mittelblau, für den Transformator hellblau, für Smart Meter Gateways grau und für Smart Meter schwarz. Diese sollen bis auf Smart Meter jeweils die Form eines Kreises besitzen. Für die Smart Meter Gateways soll die Anzahl der untergeordneten Smart Meter angezeigt werden. Die Kreise sollen jeweils heller werden, sobald mit der Maus hinübergefahren wird.

Insgesamt sind in dem Style-Guide einige Punkte den Best Practices zuzuordnen und benötigen gegebenenfalls nicht extra eine Erwähnung. Außerdem lassen sich in einigen Unterpunkten Überschneidungen oder Zusatzinformationen finden, welche jedoch nicht unbedingt einen eigenen Unterpunkt benötigen (Buttons und erweiterte Buttons, Alarmmeldungen und Warnhinweise, Formulare und (erweiterte) Eingabefelder, ...). Ein Abgleich der entsprechenden Stellen könnte zu einer besseren Übersicht und dem schnelleren Finden von benötigten Vorgaben führen. Ansonsten könnten auch weitere Verweise, zum Beispiel von Alarmmeldungen zu Warnhinweise, hilfreich sein.

8.2 Archivierungsschnittstelle

Dieser Abschnitt behandelt die Definition der Archivierungsschnittstelle, welche zur Kooperation mit einem Langzeitdatenarchiv implementiert werden muss. Die Schnittstelle wurde wie in Unterabschnitt 6.3.6 mit Swagger per OpenApi Specification Version 2 definiert und ist neben weiteren Schnittstellendefinitionen zur Versionierung und zur kollaborativen Arbeit öffentlich auf Github in der Schnittstellendefinitionsdatei „pg_dave_api.yaml“[2] zu finden. Die Schnittstelle kann in zwei Endpunkte `HistoricData` für den Austausch von historischen Daten und `DataMining` für das Anfragen und Erstellen von Data Mining Aufträgen auf Basis der Daten des Langzeitdatenarchives. Diese werden in den beiden nachfolgenden Unterabschnitten genauer erläutert.

Allgemein erwarten alle Request- und Response-Bodies, falls benötigt, den MIME-Type `application/json`. Außerdem definiert jede Ressource im Fehlerfall eine Basisantwort für Fehler, dieser gibt dem REST-Konsumenten speziellere Informationen. Das Fehlerobjekt enthält einen Zeitstempel des Fehlers (timestamp), den HTTP-Request-Error-Code (status), den Pfad der Anfrage (path) sowie eine kurze Fehlernachricht (error). Falls der Fehler durch eine Exception im Server ausgelöst wurde, welche dem Client bekannt sein darf, kann diese ebenfalls mitgeschickt werden (exception). Darüber hinaus wird eine detaillierte Nachricht des Fehlers angegeben (message).

[2] Schnittstelle zum Langzeitdatenarchiv. Online: `https://github.com/PG-EStream/shared/blob/release/0.5.4/interface_design/resources/pg_dave_api/pg_dave_api.yaml`, letzter Aufruf 28.03.2018.

8.2.1 Schnittstelle für historische Daten

Der Abschnitt der Schnittstellendefinition für die historischen Daten erfüllt die Anwendungsfälle *Anfrage von historischen Smart Meter Daten* (siehe Tabelle 14.20) und *Anfrage von historischen Wetterdaten* (siehe Tabelle 14.21). Insgesamt enthält der Endpunkt für die historischen Daten im finalen Stand sieben Ressourcen, wovon vier den Methodentyp `POST` und drei den Methodentyp `GET` erwarten (siehe Abbildung 8.9). Die Ressourcen des Endpunkts sollen nach Definition über den URL-Prefix „/api/data/“ durch den Implementierenden bereitgestellt werden. Der Endpunkt für historische Daten soll es Nutzern des Langzeitdatenarchives ermöglichen vom Langzeitdatenarchiv archivierte Daten abzufragen. Im Idealfall sollte ein Langzeitdatenarchiv dabei dynamisch Anfragen verarbeiten können, um auch angereicherte oder veränderliche Daten zur Verfügung zu stellen. Dies war für die Grundfunktionalität der Kooperation nicht notwendig und so wurde die Schnittstelle statischer gehalten. Somit dient diese Schnittstelle der Anfrage von Rohdaten im Bereich der Smartmeterdaten und Wetterdaten des Deutschen Wetterdienstes. Die einzelnen Ressourcen des Endpunktes werden im Folgenden genauer erläutert.

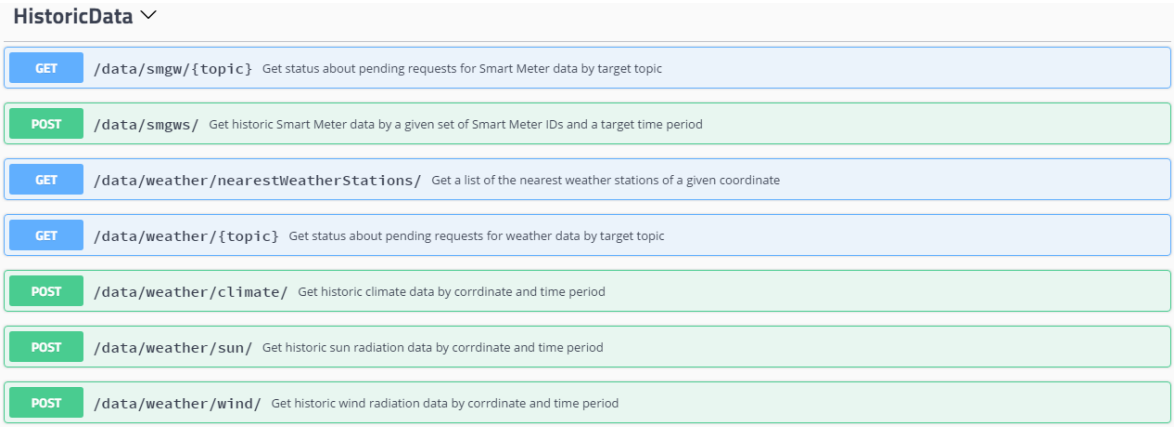


Abbildung 8.9: *Swagger-Screenshot:* `HistoricData`

Die Anfrage auf die erste Ressource „/smgws/“ benötigt ein Objekt, mit der Angabe eines Brokers (broker), welcher derzeitig den Wert „kafka“ für einen Kafka-Broker unterstützt. Außerdem muss ein Intervall für die Daten angegeben werden in der Form von (from), bis (to). Zuletzt müssen noch die Smartmeter-IDs in Form eines Arrays (devices) angegeben werden, welche die historischen Daten erhalten sollen. Bei erfolgreicher Antwort wird ein Topic-Data-Transfer-Objekt zurückgegeben, welches die Adresse (uri) und einen Topic (topic) enthält. Diese Daten können dann von dem Konsumenten dazu genutzt werden auf den Broker und dessen Topic zu abonnieren und die historischen Smartmeterdaten zu lesen.

Die Wetterdaten wurden in drei Kategorien aufgeteilt und sind somit auch über drei Ressource „/weather/climate“ für Klimadaten, „/weather/wind“ für Winddaten, „/weather/sun“ für Sonnendaten erreichbar. Alle drei `POST`-Anfragen besitzen den gleichen Request-Body, für diesen muss ein Broker (broker) und ein Zeitintervall (from, to) angegeben werden. Die Besonderheit liegt darin, dass als weiterer Paramater ein Koordinatentupel (coor) erwartet wird, welches die Längen- (lon) und Breitengraden (lat) besitzt. Dadurch ist es möglich, dass das Langzeitdatenarchiv selbst entscheiden kann,

welche Wetterstationen angefragt werden sollen. Diese Daten, Stationen (stations), Adresse (uri) und der Topic (topic), sind in der Antwort enthalten.

Um den Zustand der historischen Wetterdatenanfragen beziehungsweise der Smartmeterdaten zu erfragen, existiert entweder die „/weather/topic“ oder die „/smgws/“topic“ Ressource. Diese kann mit dem erhaltenen Topic als Pfadparameter jeweils passend angefragt werden. Die Antwort selbst stellt ein Statusobjekt dar, welches ein Statusattribut (status) besitzt, dass die drei Werte RUNNING, SUCCESS, oder FAILURE annehmen kann. Dabei bedeutet RUNNING, dass die Anfrage angenommen wurde, in Bearbeitung ist und das Ergebnis, sobald es zur Verfügung steht auf den angegebenen Topic geschrieben wird. Die Konstante SUCCESS bedeutet, dass die Anfrage bereits vollständig bearbeitet wurde und die Antwort auf dem Topic bereitstehen müsste. Die letzte Statuskonstante FAILURE bedeutet, dass es während der Bearbeitung einen unerwarteten Fehler gegeben hat, der nicht direkt mit dem HTTP-Request und dessen -Body in Verbindung steht.

Die letzte Ressource des historischen Datenendpunktes „/weather/nearestWeatherStations/“, erlaubt es einem Nutzer des Langzeitdatenarchives zu einer Koordinatenangabe wie bei den Wetterdatenanfragen und der optionalen Angabe eines Radius die nächstgelegenen Wetterstationen zu erhalten.

8.2.2 Schnittstelle für Datamining

Der Abschnitt der Schnittstellendefinition für das Datamining auf historischen Daten erfüllt die Anwendungsfälle *Erstellen von Miningmodell-Aufträgen* (siehe Tabelle 14.23) und *Anfrage von Miningmodellen* (siehe Tabelle 14.22). Insgesamt enthält der Datamining-Endpunkt im finalen Stand fünf Ressourcen, wovon zwei den Methodentyp `POST` und drei den Methodentyp `GET` erwarten (siehe Abbildung 8.10). Die Ressourcen des Datamining Endpunkts sollen nach Definition über den URL-Prefix „/api/datamining/“ durch den Implementierenden bereitgestellt werden. Der Endpunkt Datamining soll es Nutzern des Langzeitdatenarchives ermöglichen vorhandene oder aktuelle PMML-Modelle zu erfragen, neue Miningmodelle in Auftrag zu geben und dabei den derzeitigen Stand zu erfragen. Die einzelnen Ressourcen des Endpunktes werden im Folgenden genauer erläutert.

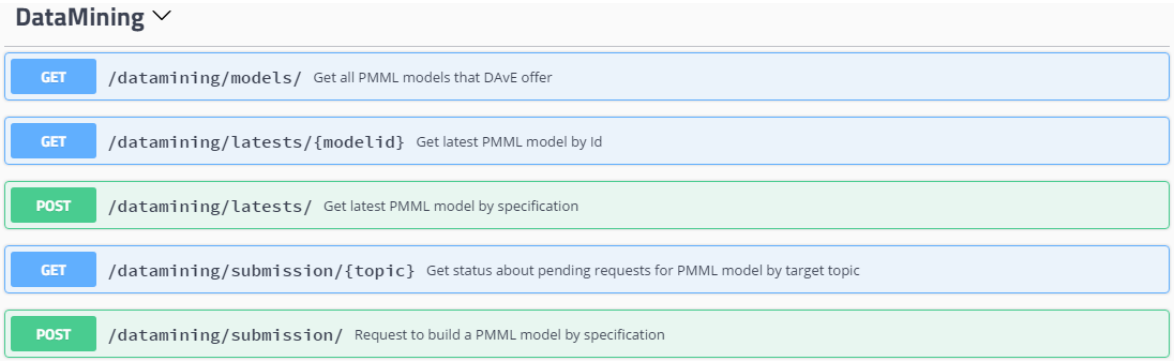


Abbildung 8.10: *Swagger-Screenshot: Datamining*

Die Anfrage auf die erste Ressource „/models/“ liefert als Antwort im HTTP-Body alle vom Langzeitdatenarchiv zur Verfügung gestellten PMML-Modelle. Diese sind jeweils über eine ID referenzierbar (modelId) und enthalten des Weiteren einen Namen des Modells (name), eine Beschreibung (descrip-

tion) und die Zeit des zuletzt durchgeführten Trainings (lastBuild). Über die im Modell enthaltene modelId kann dann über die zweite Ressource „/latest/modelId“ das dazu passende PMML-Modell abgefragt werden. Das PMML-Modell stellt immer ein bereits fertig trainiertes Modell dar.

Um von einem Modell das zuletzt trainierte zu erhalten, kann die dritte Ressource „/submission/latests/“ verwendet werden. Diese erwartet im HTTP-Request-Body ein PMML-Modell, welches mindestens die Schemadefinition enthält und die Angabe eines Brokertyps (derzeitig nur Kafka möglich). Die Schemadefinition beschreibt in diesem Fall die `DataField`-Einträge im `DataDictionary`-Bereich, das zu trainierende Modell, welches beispielsweise ein Regressionsmodell sein kann. Vorzugsweise wird für diesen Knoten auch ein eindeutiger modelName-Attributseintrag erstellt. Die gleiche Anfrage ist auf die Ressource „/submissions/“ möglich, um mit einem PMML-Schema ein neues Modelltraining auf den Daten eines Langzeitarchives in Auftrag zu geben. Dabei erhält man bei beiden Anfragen als Antwort ein Data-Transfer-Objekt, welches die Adresse des Brokerservers (uri) und das dazugehörige Topic (topic) enthält. Daraufhin kann ein Verbraucher das Topic abonnieren. Sobald das Langzeitdatenarchiv das trainierte Modell auf den Topic geschrieben hat, kann das Modell abgefragt werden. Unter Umständen kann es hierbei zu Verzögerungen kommen, wenn das Trainieren eines komplexen Modells viel Zeit in Anspruch nimmt.

Die letzte Ressource „/submissions/topic“ muss einer der erhaltenen Topics übergeben werden woraufhin der aktuelle Status der Datamininganfrage angezeigt wird. Dieser kann derzeitig drei Werte repräsentieren RUNNING, SUCCESS, FAILURE. Dabei bedeutet RUNNING, dass die Anfrage angenommen wurde, in Bearbeitung ist und das Ergebnis, sobald es zur Verfügung steht auf den angegebenen Topic geschrieben wird. Die Konstante SUCCESS bedeutet, dass die Analyse bereits vollständig bearbeitet wurde und die Antwort auf dem Topic bereitsteht. Die letzte Statuskonstante FAILURE bedeutet, dass es während der Bearbeitung einen unerwarteten Fehler gegeben hat, der nicht direkt mit dem HTTP-Request und dessen HTTP-Body zu tun hatte.

8.3 Odysseus Beschreibung

8.3.1 Datamining - Grundlagen

8.3.1.1 Prognoseverfahren

Auf Basis der Seminararbeit über die Vorhersage von Energieverbräuchen von Haushalten, siehe Unterabschnitt 3.4.3, wird die Wahl für ein Verfahren Lastprognosen zu erstellen zum einen anhand der erhaltenen Daten, zum anderen je nach Prognosezeiträumen und nach möglichen Prognoseverfahren und deren Eignung für das gewünschte Ziel getroffen.
Wichtige Daten für Lastprognosen sind demnach physikalische Einflussgrößen, sozioökonomische Faktoren, volkswirtschaftliche Faktoren, technologische Faktoren und eine historische Datenbasis. Durch die mosaik Simulation werden Daten über die Globalstrahlung, die Uhrzeit und die Temperatur erhalten. Es gibt die Möglichkeit Langzeitprognosen für 1-20 Jahre im Voraus (LTFL), Mittelzeitprognosen für eine bis mehrere Wochen (MTLF) und Kurzzeitprognosen für wenige Minuten (STLF) zu erzeugen. Es wurde sich bereits darauf geeinigt nur Kurzzeitprognosen zu behandeln. In der Seminararbeit werden vier verschiedene Prognoseverfahren angesprochen. Dazu gehören die Regressionsanalyse, welche nur auf historischen Daten basiert und dazu neigt ungenauer zu sein, die Verwendung von Fuzzylogik, welche Wissen über Korrelationen innerhalb der Daten erfordert und allein deswegen im Rahmen dieses Projektes eher ungeeignet ist. Des Weiteren gibt es evolutionäre Algorithmen, bei denen es sich

jedoch eher um Optimierungsalgorithmen handelt und daher bei Lastprognosen eine untergeordnete Rolle spielen [238] und neuronale Netze, welche zwar eine hohe Genauigkeit und kurze Antwortzeit bieten, jedoch in der Umsetzung auf Grund der benötigten Trainingsphase sehr umfangreich sind. Alle diese Verfahren sind für Kurzzeitprognosen geeignet.
Daher findet die Erstellung von Lastprognosen mit Hilfe von Regressionsanalysen statt.

Konzept für die Regressionsanalyse Zunächst wird der Use-Case für die Regressionsanalyse aufgestellt, um zu bestimmen in welchem genaueren Kontext diese verwendet werden soll.
Name: Regressionsanalysen
Ziel im Kontext: Prognosen erstellen (Lastprognosen, PV-Prognosen, Windprognosen)
Akteure: Leitwartenpersonal, Administrator, technischer Nutzer (PG-DAvE), Simulation
Trigger: Erhalt neuer Daten über die Simulation oder historischer Daten

Essenzielle Schritte:

- Erzeugung neuer Daten durch die mosaik Simulation, oder Erhalt von Daten durch PG-DAvE
- Die Daten fließen in die Regressionsanalyse mit ein
- Mit Hilfe der Regressionsanalyse werden die neuen Prognosedaten erzeugt
- Die Prognose wird über die Website grafisch dargestellt
- Das Leitwartenpersonal und der technische Nutzer (PG-DAvE) dürfen die Daten ansehen und Prognosen abfragen (aktueller und prognostizierter Verbrauch)

Erweiterungen:
Die Daten können in Echtzeit berechnet und dargestellt werden.

Ein kleiner Einblick in die Regressionsanalyse wurde, wie bereits erwähnt, in Unterabschnitt 3.4.3 gegeben. An dieser Stelle wird jedoch das genauere Vorgehen in Bezug auf die gewählte Architektur in diesem Projekt beschrieben.
Das Ziel der Regressionsanalyse ist die Beschreibung und Beziehung zwischen einer abhängigen Variable und einer (mehreren) unabhängigen Variable(n) durch eine Funktion. Da sie hauptsächlich verwendet wird um Zusammenhänge quantitativ zu beschreiben oder Werte der abhängigen Variablen zu prognostizieren, eignet sich dieses Verfahren für das Erstellen unterschiedlicher Prognosen (Last-, PV-, Windprognosen). Wenn also das Ziel die Prognose, beziehungsweise Vorhersage, ist, kann der durch das Regressionsverfahren ermittelte funktionale Zusammenhang verwendet werden, um ein Vorhersagemodell zu erstellen. Wenn dabei zusätzliche Werte x (unabhängige Variable(n)) ohne dazugehörige Werte y (abhängige Variable) vorliegen, dann kann das angepasste Modell zur Vorhersage des Wertes von y genutzt werden. Dabei gelten einige Voraussetzungen an die verwendeten Daten: Es soll eine Normalverteilung bestehen, die Linearität sollte gegeben sein, die Homoskedastizität, also die Varianz der Verteilung der abhängigen Variablen muss für alle Werte der Unabhängigen konstant sein und die Fälle sowie der Fehler sollten untereinander nicht korrelieren. Es wird davon ausgegangen, dass diese Punkte durch die, zum Beispiel von der Simulation, erhaltenen Daten gegeben sind.
Die allgemeine Form der Regressionsanalyse ist folgende, wobei immer ein Restglied R übrig bleibt (der Fehler):

$$x_t = \alpha_0 + \alpha_1 y_1 t + \alpha_2 y_2 t + ... + \alpha_n y_n t + R_t = \alpha_0 + \sum_{j=1}^{k} \alpha_j y_j t + R_t \tag{8.1}$$

Im Falle der linearen Regression wird die folgende Funktion unter der Bedingung gesucht, dass die Summe der Quadrate der Abstände der tatsächlichen y-Werte von den berechneten $\hat{y}$-Werten ein Minimum hat.

$$\hat{y} = f(\hat{x}) = m * \hat{x} + n \tag{8.2}$$

Die zu minimierende Größe sei demnach allgemein:

$$V = \sum_{i=1}^{k} (y_i - \hat{y}_i)^2 \tag{8.3}$$

Im nächsten Schritt wird Gleichung 8.3 modifiziert und die partiellen Ableitungen $\frac{\vartheta V}{\vartheta m}$ und $\frac{\vartheta V}{\vartheta n}$ gleich null gesetzt, um jeweils das Minimum zu erhalten:

$$V(m,n) = \sum_{i=1}^{k} (y_i - (m * x_i + n))^2 = \sum_{i=1}^{k} (y_i - m * x_i - n)^2 \tag{8.4}$$

Daraufhin wird jeweils eine partielle Ableitung nach m beziehungsweise n durchgeführt.
Odysseus bietet bereits die Möglichkeit Regression mit Hilfe des Classification_learn Operators [117] zu verwenden.

8.3.2 Datamining auf Rohdaten

In diesem Abschnitt wird darauf eingegangen, in welchem Kontext Datamining auf den Rohdaten der mosaik Simulation (siehe Unterabschnitt 3.1.1) betrieben werden kann. Hierfür wird zuerst ein Anwendungsfall spezifiziert, um die Problemstellung zu verdeutlichen.
Name: Simples Datamining auf Rohdaten
Ziel im Kontext: Einfache Kenngrößen für die Eingangsdaten bestimmen, wie zum Beispiel Durchschnitt, Minimum oder Maximum
Akteure: Leitwartenpersonal, technischer Nutzer (PG-DAvE), Simulation
Auslöser: Erhalt neuer Daten durch die Simulation oder historischer Daten durch das Langzeitdatenarchiv der PG-DAvE

Essenzielle Schritte:

- Erzeugung neuer Daten durch die mosaik Simulation oder Erhalt historischer Daten durch PG-DAvE
- Aktualisierung der Kenngrößen (AVG, MIN, MAX)
- Die Kenngrößen werden über die Website grafisch dargestellt. Hierbei können sowohl die aktuellen Werte, als auch der Verlauf der Kenngrößen dargestellt werden.
- Das Leitwartenpersonal und der technische Nutzer (PG-DAvE) dürfen auf die Daten zugreifen und die Kenngrößen für einen bestimmten Zeitpunkt abfragen

Erweiterungen:

- Die Kenngrößen können in Echtzeit berechnet und dargestellt werden
- Es können weitere aussagekräftige Werte hinzugefügt werden, wie zum Beispiel der Median
- Das Darstellungsintervall könnte konfigurierbar sein

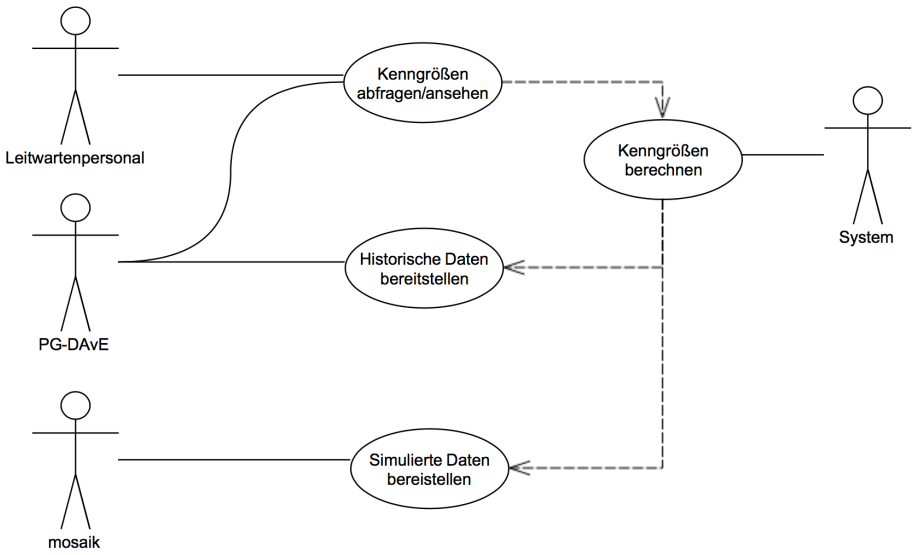


Abbildung 8.11: *Use-Case Diagramm für Datamining auf Rohdaten*

- Die Granularität der Kenngrößen könnte konfigurierbar sein. Hiermit ist gemeint, ob die Kenngrößen zum Beispiel für einzelne Smart-Meter oder gebündelt für Smart-Meter Gateways angezeigt werden.

Abbildung 8.11 zeigt eine Visualisierung des Anwendungsfalls. Mit Hilfe der Kenngrößen, die durch das Datamining ermittelt werden, hat ein Mitarbeiter der Leitwarte einen guten Überblick über das Netz. Ausreißer werden in den minimalen und maximalen Werten widergespiegelt und die durchschnittlichen Werte geben eine Übersicht über das Netz im gesamten Betrachtungsintervall. Je nach Granularität lassen sich bis auf die Ebene einzelner Smart-Meter Ausreißer feststellen. Des Weiteren ist es möglich, die Kenngrößen als Grundlage für komplexere Auswertungen zu nutzen. Der Durchschnittswert könnte etwa als Referenz benutzt werden, um defekte oder manipulierte Anlagen zu erkennen, deren Durchschnittswert signifikant vom Durchschnitt über alle Smart-Meter abweicht. Allerdings müssen für diese Auswertungen auch die Art der Produzenten und deren Standort einbezogen werden, damit valide Aussagen getroffen werden können. Allgemein dienen Kenngrößen dazu, eine große Menge an Daten zu komprimieren und charakterisieren, daher werden sie häufig zur Vorbereitung für weitere Verarbeitungsschritte benutzt und ermöglichen es einem Nutzer, die Daten besser zu verstehen.

8.3.3 Datamining - Konzept

Datamining im E-Stream-System soll grundsätzlich auf zwei Arten erfolgen können: durch die Verwendung eines bereits trainierten Modells, welches in Form einer PMML empfangen wird und über die Datamining Operatoren, die im *Mining Feature* von Odysseus vorhanden sind. Darüber hinaus werden hier auch die Berechnung statistischer Kenngrößen wie Maximum, Minimum oder Durchschnitt als vereinfachte Form des Dataminings verstanden. Damit lassen sich die verwendeten Methoden in die Kategorien „Online Learning“ und „Offline Learning“ unterteilen (siehe Unterabschnitt 3.3.2). „Online Learning“ bezieht sich hier darauf, dass das Modell auf dem Datenstrom gelernt wird, während das Modell beim „Offline Learning“ auf einem zuvor gespeicherten Datensatz trainiert wird. Letzteres ist im Fall des Dataminings mit einem PMML-Modell der Fall.

Online Learning

Unter das „Online Learning“ fallen in diesem Kontext also die Prognose von Verbrauch und Erzeugung auf Basis eintreffender Smart Meter Daten sowie die Ausreißererkennung in Verbrauch und Erzeugung. Für die Prognose stehen jedoch neben den Smart Meter Daten keine weiteren Informationen zur Verfügung. Zwar sehen die Seminararbeiten der Projektgruppe (siehe Abschnitt 3.4) sowie einige der Use Cases auch die Prognose auf Basis von Wetterdaten vor, doch zeigte sich im Laufe der Projektgruppe, dass dies nicht sinnvoll umsetzbar ist. Da die Daten der mosaik Simulation auf Lastprofilen beruhen und nicht von Wetterinformationen beeinflusst werden, ist ihre Prognose über externe Wetterdaten nicht sinnvoll, da hier kein Zusammenhang zwischen den Daten besteht und somit auch kein Modell gelernt werden kann.

Für das Datamining auf dem Datenstrom wurde als Prognoseverfahren die lineare Regression gewählt. Dabei bezieht das „Online Learning“ seine Trainingsdaten aus einem eingehenden Trainingsdatenstrom. Um das Problem mit Speicherüberläufen zu vermeiden (Datenströme sind potentiell unendlich), kann immer nur ein Ausschnitt des Datenstroms betrachtet werden. Hierbei kann nicht davon ausgegangen werden, dass der gerade betrachtete Ausschnitt alle Merkmale des Datenstroms abbildet. Dadurch können die Ergebnisse des gelernten Modells stark variieren. Dabei basiert die Prognose von (Teil-)Netzen auf den aggregierten Messungen aller Smart Meter, die zum betrachteten Netz gehören. Der Gesamtverbrauch, bzw. die Erzeugung eines (Teil-)Netzes ergibt sich durch die Summe aller Verbrauchs- und Erzeugungsmessungen für einen bestimmten Zeitraum. Der gewählte Zeitraum umfasst dabei 15 Minuten, da dies dem Abstand zwischen zwei Übertragungen eines Smart Meters entspricht.

Auch für die Ausreißererkennung ist die Datenverfügbarkeit für die Projektgruppe gering. So wären für die meisten Methoden zur Fehlererkennung bei PV-Anlangen zusätzliche Daten oder gar Laborbedingungen erforderlich [153]. Damit wäre die Erkennung unsachgemäßer Benutzung, bzw. der illegale Betrieb einer PV-Anlage oder Defekte an ihnen, möglich. So könnte durch das Anlegen einer Spannung an die Unterkonstruktion des Strings überprüft werden, ob die Verkabelung und die elektrischen Betriebsmittel korrekt funktionieren oder mit Hilfe einer Wärmebildkamera eine Thermografie zur Erkennung von Hotsports durchgeführt werden [164].

Ein in der Projektgruppe umsetzbares Verfahren, ist dagegen die statistische Ausreißererkennung über die Abweichung des Durchschnitts sowie der Varianz zwischen aktuellen und historischen Daten. Hierfür ist die Betrachtung aktueller und historischer Daten von Verbrauch und Erzeugung auf Basis entsprechender Zeiträume (z. B. Tage oder Monate) notwendig. Dabei kann die Granularität vom einzelnen Smart Meter über Gateways bis zu Teilnetzen oder dem Gesamtnetz reichen. Dieser Ansatz ist zwar anfälliger für Fehlalarme, jedoch aufgrund der begrenzten Datenlage in Odysseus durch die Verwendung des *Outliner-Operators* umsetzbar.

Statistische Kenngrößen

Neben dem Datamining zur Vorhersage von Verbrauch und Erzeugung sowie der Ausreißererkennung, soll das E-Stream-System natürlich auch in der Lage sein, den aktuellen Netzzustand abzubilden. Hierzu sind insbesondere die aggregierten Daten der Smart Meter sowie Smart Meter Gateways relevant. Über den *Aggregation- Operator* von Odysseus [118] ist die Berechnung von Summen, Durchschnitt, minimalen und maximalen Werten sowie der Varianz einfach realisierbar, sodass hierfür lediglich entsprechende Anfragen hinterlegt werden müssen. Da zumindest Teile der Aggregation

auch für die Prognosen (als Eingangsdaten) relevant sind, müssen jedoch auch die Anfragen für die statistischen Kenngrößen in das E-Stream Anfragemanagement (siehe Unterabschnitt 8.4.4) integriert werden, damit sie wiederverwendbar sind.

Datamining mit PMML

Eine weitere Möglichkeit zur Prognose des Verbrauchs und der Erzeugung innerhalb des Netzes ist die Verwendung der Ergebnisse externer Analysen. Hierzu dient das in Unterabschnitt 3.2.3 beschriebene Austauschformat PMML. Wie in Unterabschnitt 8.3.5 beschrieben, wurde hierfür das JPMML-Package in Odysseus integriert. Hierdurch ist es möglich Datamining auf den Netzdaten durchzuführen ohne zunächst auf dem Datenstrom zu lernen. Anfragen zu Prognosen können somit ohne vorherige Wartezeit Ergebnisse liefern.

8.3.4 Datamining - Umsetzung

Wie bereits im Konzept erwähnt, soll sich die Umsetzung am E-Stream Anfragemanagement orientieren und beinhaltet daher in erster Linie das Erstellen neuer Snippets und Templates. Über diese soll ein Nutzer des Systems Datamining Anfragen selbstständig starten und über verschiedene Parameter konfigurieren können. Hierfür wurden Snippets erstellt und in die Datenbank eingepflegt. Dies umfasst folgende Bereiche:

- die Erzeugung von Aggregaten
- das Setzen von Zeitstempeln
- das Zusammenführen (joinen) zweier Datenströme
- die Ausgabe von Ergebnissen im JSON-Format

Um die Verwaltung von Templates gering zu halten, sollte ein Snippet so konstruiert sein, dass es möglichst mehrfach verwendet werden kann. In diesem Abschnitt wird auf die verwendeten Snippets eingegangen, die für die Erstellung der Anfrage-Templates für das Datamining entworfen worden sind.

Die Definition der Quellen für die Anfrage-Templates besteht aus zwei Snippets, welche beide einen *ACCESS-Operator* und einen *TIMEWINDOW-Operator* verwenden. Während der *ACCESS-Operator* beider Snippets äquivalent ist, unterscheiden sich die *TIMEWINDOW-Operatoren* in der Hinsicht, dass einer als Sliding-Window verwendet wird und der andere entweder als Tumbling- oder auch als Jumping-Window definiert werden kann. Je nachdem, welche Art von Fensteroperator verwendet wird, werden die Zeitstempel für die eintreffenden Tupel gesetzt (siehe [120]). Die Differenzierung ist für die Verwendung von Lern-Operatoren relevant, da sich die Semantik je nach Verwendung von Sliding-Window oder Tumbling- Window unterscheiden kann. Bei einem Sliding-Window treten im Vergleich zu einem Tumbling-Window deutlich mehr Gültigkeitsintervalle auf, d.h. dass die Menge an zu betrachteten Tupeln größer ist und daher der Prädiktor unter Umständen häufiger neu berechnet werden muss. Die Verwendung eines Tumbling-Window zusammen mit einem Lern-Operator ergibt daher Sinn.

In Listing 8.4 ist das Anfrage-Snippet für den *Classification-Learn-Operator* dargestellt. Hierbei werden die Query-Keys für input_classifier und input_1 automatisch vom Template-System generiert, wenn eine Anfrage erzeugt wird. Für die Keys class_attribute, ml_framework, ml_algorithm, timestamp_attribute und feature_vector werden Default-Werte in der Datenbank hinterlegt und trai-

ning_data_size ist als verpflichtende Angabe markiert. Für ml_framework ist beispielsweise der in Listing 8.5 dargestellte Eintrag in der Datenbank hinterlegt, sodass standardmäßig das WEKA-Framework verwendet wird.

```
{{input_classifier}} = CLASSIFICATION_LEARN({
        class= '{{class_attribute}}',
        learner = '{{ml_framework}}',
        algorithm = '{{ml_algorithm}}'
    },

    MAP({
        expressions=[
            '{{class_attribute}}',
            ['Timeinterval.start','{{timestamp_attribute}}']
            {{{feature_vector}}}
        ]
    },

        ELEMENTWINDOW({
                    size = {{training_data_size}},
                    advance = 1
                },

                {{input_1}}
        )
    )
)
```

Listing 8.4: *Snippet für Classification-Operator*

```
INSERT INTO `querykey` (`id`, `isclientsink`, `default_value`, `description`,
    `isgenerated`, `iskafka`, `name`, `isobjecttype`, `isreference`,
    `reference_name`, `issink`)

    VALUES (39, false, 'weka', 'Specifies the machine learning framework.',
    false, false, 'ml_framework', false, false, NULL, false);
```

Listing 8.5: *Query Key für Classification-Operator*

Das in Listing 8.4 dargestellte Snippet muss zur tatsächlichen Prognose anschließend mit den in Listing 8.6 beschriebenen Snippets zu einem Template zusammengefügt werden, damit eine vollständige Anfrage für Odysseus daraus vom QueryBuilder erstellt werden kann. Insgesamt besteht das Anfrage-Template damit aus 4 Snippets, die unterschiedliche Aufgaben übernehmen, wobei dem in Listing 8.4 dargestellten Snippet durch das Lernen des Klassifikators die wichtigste Aufgabe zukommt.

```
// Snippet: Datenquelle
{{input_1}} = TIMESTAMP({
        start = '{{timestamp_attribute}}'
    },
    ACCESS({
        source='{{source_name}}',
        protocol='XML',
        transport='Kafka',
        wrapper='GenericPush',
        dataHandler='Tuple',
        metaattribute = ['{{metaattribute}}'],
```

```
12          options=[
13              ['topic', '{{kafka_topic_1}}'],
14              ['messagetype', 'string'],
15              ['bootstrap.servers', '{{kafka_servers}}'],
16              {{{x_path_expression}}}
17          ], schema=[{{{schema}}}]
18      })
19  )
20
21  // Snippet: Zeitfenster
22  {{input_1}} = TIMEWINDOW({
23          size = [{{window_size}}, '{{window_unit}}'],
24          advance = [{{window_advance}}, '{{window_unit}}']
25          }, {{input_2}}
26  )
27
28  // Snippet: Classifier
29  {{input_classification}} = JOIN({
30          predicate = '{{{predicate}}}'
31      },
32          CLASSIFY({
33          classifier='{{classifier}}',
34          classname='{{class_name}}'
35          },
36                  MAP({
37                  expressions=[
38                      ["DoubleToLong({{timestamp_attribute}} + {{forecast}})",
39                       "{{timestamp_attribute}}"]
40                      {{{feature_vector}}}
41                  ]
42                  },
43                          {{input_2}}
44              ),
45                  {{input_classifier}}
46          ),
47                          MAP({
48                          expressions=[
49                    ['{{timestamp_attribute}}','{{timestamp_attribute}}_right']
50                    {{{projection}}}
51                            ]
52                          },
53           {{input_2}}
54      )
55  )
56
57  // Snippet: Output
58  {{output_1}} = SENDER({
59      protocol = 'JSON_COSEM',
60      transport = 'Kafka',
61      sink = '{{sink_name_1}}',
62      wrapper = 'GenericPush',
63      options = [
64          ['topic', '{{kafka_topic_1}}'],
65          ['messagetype', 'string'],
66          ['metadata.broker.list', '{{kafka_servers}}'],
67          ['objectType', '{{object_type_1}}']
68      ]
```

```
69      },
70          MAP({expressions=[
71              ['{{class_name}}','prediction'],
72              ['{{id_attribute}}', 'id'],
73              ['{{type_attribute}}', 'type'],
74              ['toList({{feature_names}})','features']
75                  ]
76          },
77          {{input_classification}}
78      )
79  )
```

Listing 8.6: *Weitere Snippets zur Prognose*

Da das Konzept für die Auswahl von Anfragen vorsieht, dass der Nutzer diese selbst konfigurieren kann (siehe Unterunterabschnitt 8.5.1.2), sind natürlich verschiedene Möglichkeiten der Zusammenstellung denkbar. Listing 8.7 zeigt eine mögliche Anfrage, die auf Basis der gezeigten Snippets für die Prognose vom QueryManager erstellt werden könnte. Dabei wird zunächst der *TIMESTAMP* auf die von der Simulation übergebene *capture_time*, also das in der Simulation verwendete Datum, gesetzt und der Kafka-Broker mit dem Topic rawdata als Datenquelle angelegt. Neben der Server- IP und dem Port werden zusätzlich die X-Path Expressions für das Parsen des übertragenen COSEM-XML Strings sowie das entsprechende Schema angegeben. Anschließend wird ein Zeitfenster angegeben und über die *Selection* nur jene Daten ausgewählt, welche von einem Consumer stammen. Hiermit wird erreicht, dass im Folgenden nur der Verbrauch prognostiziert wird. Anschließend erfolgt die Aggregation in Form der Summenbildung der im zuvor angegebenen Zeitfenster übertragenen Werte der Smart Meter. Die Aggregate werden dann dem Lern-Operator übergeben, der ein Modell trainiert und es anschließend dem Klassifikations-Operator übergibt. Anschließend verwendet der Klassifikations-Operator ein Modell zur Prognose, bevor die Ergebnisse als JSON auf einem entsprechenden Topic in Kafka geschrieben werden. In Listing 8.8 ist die JSON-Ausgabe des Vorhersage-Ergebnisses angegeben.

```
1  #PARSER PQL
2  #ADDQUERY
3
4  // snippet[kafka_xml_input](8)
5  input_1_8TBTH = TIMESTAMP({
6          start = 'capture_time'
7      }, input_enrichment_rawdata_4HGK )
8
9
10 // snippet[sliding_time_window](9)
11 input_1_8CPIY = TIMEWINDOW({
12         size = [5000, 'MILLISECONDS'],
13         advance = [1, 'MILLISECONDS']
14         }, input_1_8TBTH )
15
16
17 // snippet[aggregate](12)
18 input_3_8JJFK  = AGGREGATION({
19         aggregations=[
20         ['FUNCTION' = 'Sum', 'INPUT_ATTRIBUTES' = 'value',
21           'OUTPUT_ATTRIBUTES' = 'sum_value'] ,
22                 ['FUNCTION' = 'Avg', 'INPUT_ATTRIBUTES' = 'value',
23                   'OUTPUT_ATTRIBUTES' = 'avg_value'] ,
24                 ['FUNCTION' = 'Min', 'INPUT_ATTRIBUTES' = 'value',
```

```
                  'OUTPUT_ATTRIBUTES' = 'min_value'],
                ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'value',
                  'OUTPUT_ATTRIBUTES' = 'max_value'],
                ['FUNCTION' = 'Variance', 'INPUT_ATTRIBUTES' = 'value',
                  'OUTPUT_ATTRIBUTES' = 'var_value']]
        }, input_1_8THRF )


// snippet[learner_numeric_with_element_window](26)
input_classifier_8PSRP = CLASSIFICATION_LEARN({
        class= 'sum_value',
        learner = 'weka',
        algorithm = 'linear-regression'
    }, MAP({
        expressions=['sum_value', ['Timeinterval.start','capture_time'] ]
    }, ELEMENTWINDOW({ size = 2, advance = 1 }, input_3_8JJFK ) ) )


// snippet[classifier_and_kafka_output_1](29)
input_classification_8BCCN = JOIN({
        predicate = 'type = "consumer"'
    },  CLASSIFY({
        classifier='classifier',
        classname='prediction'
        },  MAP({
                expressions=[["DoubleToLong(capture_time + 10000)",
                  "capture_time"]] }
                , input_1_8THRF )
                , input_classifier_8PSRP )
                , MAP({ expressions=[
                    ['capture_time','capture_time_right'],
                       "gateway", "sm_id", "value", "scaler",
                        "unit", "status", "type" ]
        }, input_1_8THRF ) )

output_1_8WRGB = SENDER({
    protocol = 'JSON_COSEM',
    transport = 'Kafka',
    sink = 'sink_name_1_8SHTD',
    wrapper = 'GenericPush',
    options = [
        ['topic', 'sooidfjsodvjpsovjpds'],
        ['messagetype', 'string'],
        ['metadata.broker.list', '192.168.33.10:9092'],
        ['objectType', 'PREDICTEDDATA']
    ]
    }, MAP({expressions=[
            ['prediction','prediction'],
            ['gateway', 'id'],
            ['type', 'type'],
            ['toList(capture_time)','features']
        ] }, input_classification_8BCCN ) )
```

Listing 8.7: *Odysseus Anfrage zur Prognose von Verbrauch/Erzeugung*

```
{
  "result":{
    "id":"smgw_dbcc8506-75ca-4456-8a9e-c787e7ec0d63",
    "prediction":"11.0",
    "type":"consumer",
    "features":[
      "5704427",
      "3209762.8111501136",
      "30"
    ],
// meta information
    "timeinterval_start":1519568765930,
    "timeinterval_end":-9223372036854775807,
    "minlstart":-1,
    "maxlstart":-1,
    "lend":-1,
    "latency":-1
  }
}
```

Listing 8.8: *Beispielergebnis der Vorhersage*

Auf die gleiche Art wurde die Ausreißererkennung umgesetzt. Ein Snippet hierzu ist in Listing 8.9 dargestellt. Hierbei wird das *Monitoring Feature* von Odysseus verwendet [115].

```
{{input_1}} = DEVIATIONLEARN({
        trainingmode = \'ONLINE\',
␣␣␣␣␣␣␣␣attribute␣=␣\'{{class_attribute}}\'
␣␣␣␣},
␣␣␣␣{{input_2}}
)

{{input_3}}␣=␣DEVIATIONANOMALYDETECTION({
␣␣␣␣␣␣␣␣interval␣=␣{{deviation_interval}},
␣␣␣␣␣␣␣␣attribute␣=␣\'{{class_attribute}}\'
␣␣␣␣},
␣␣␣␣1:{{input_1}},
␣␣␣␣0:{{input_1}}
)
```

Listing 8.9: *Snippet für Deviation-Detection*

Um die Aggregate AVG, MAX, MIN, SUM, STDDEV und VAR über alle Erzeugnisse der letzten 15 Minuten zu erzeugen, kann das in Listing 8.10 dargestellte Snippet mit den dazugehörigen Werten für die Query-Keys ausgeführt werden. Standardmäßig werden immer alle genannten Aggregate berechnet, da der entsprechende Query-Key dies als default-Wert berücksichtigt. Durch die explizite Übergabe von *snippetId: 12, value: '['AVG','value','avg_value'],['SUM','value','sum_value']', keyId: 33* beim REST-Call werden nur AVG und SUM berechnet. Die Modifikation des Zeitraumes kann auf die gleiche Weise erfolgen. Durch *snippetId: 8, value: 'Minutes', keyId: 30* kann die Zeiteinheit verändert werden. Zulässige Werte sind hier zum Beispiel Minutes, Hours und Nanoseconds.

```
{{input_1}} = SELECT({
        predicate='{{{predicate}}}'
        }, {{input_2}
        }
)

{{input_3}} = AGGREGATE({
        aggregations=[{{{aggregations}}}]
        },{{input_1}
    }
)
```

Listing 8.10: *Snippets für die Aggregation*

Listing 8.11, Listing 8.12 und Listing 8.13 zeigen einmal exemplarisch, wie das in Listing 8.10 dargestellte Snippet zur Aggregation im E-Stream System letztlich verwendet wird. In Listing 8.11 ist dabei zunächst der REST-Call zu sehen, welcher die Parameter zur Erstellung der Anfrage enthält. Da hier in der *snippetValueDtos* keine Einschränkung auf die gewünschte Art des Aggregates gemacht werden, werden wie oben beschrieben alle Aggregate berechnet. Als Kafka-Topic für die eingehen Smart Meter Daten wird *test* gesetzt, das betrachtete Zeitfenster soll *5000 Millisekunden* umfassen und es sollen die Aggregate für *producer*-Smart Meter, also PV-Anlagen, berechnet werden. Anschließend wird festgelegt, dass die Ergebnisse in der Form von *AggregatedDate*, was im E-Stream System eine JSON-Repräsentation entspricht, auf das *aggregate_topic* geschrieben werden sollen.

```
const snippetValueDtos: SnippetValueDto[] = [
  // kafka topic to get cosem data
  {snippetId: 8, value: 'test', keyId: 20},
  // window size 5000 milliseconds
  {snippetId: 9, value: '5000', keyId: 29},
  // predicate for selection
  {snippetId: 12, value: 'type="producer"', keyId: 34},
  // kafka topic for results
  {snippetId: 16, value: 'aggregated_result', keyId: 20},
  // json format
  {snippetId: 16, value: 'AggregatedData', keyId: 23},
];

// GET user id via REST call
    this.userApi.getMe().subscribe(value => {

        // CREATE RegisterQueryDto that bundles all SnippetValueDtos
        // and other data for the query
        const dto: RegisterQueryDto = {
                templateId: 4,                    // select the template
        name: 'aggregations',              // give it nice name
        autorun: false,                    // autostart, yes or no?
        description: 'This a nice query.', //give it a description
        userId: value.id,
        values: snippetValueDtos
        };

        let queryId;
        let webSocketTopicDto: WebSocketTopicDto;

        // EXECUTE REST call to register query where the reponse will be a
```

```
32          // QueryStatusDto.
33          this.queryApi.registerQuery(dto).subscribe(queryStatusDto => {
34                  queryId = queryStatusDto.id;
35          webSocketTopicDto = queryStatusDto.webSocketTopicDto;
36          // WebSocketTopicDto contains all information for creating
37          // and subscribing a WebSocket-topic.
38          webSocketTopicDto.topics.forEach(e => {
39            this.subject = this.dataService.getBehaviourSubject(
40                  this.dataService.createQueryDataEntry(e.topic));
41          })
42        });
43      });
```

Listing 8.11: *REST-Call im E-Stream System zur Aggregation*

In Listing 8.12 ist die PQL-Anfrage dargestellt, welche aus dem gezeigten REST-Call generiert wurde und Listing 8.13 zeigt eine mögliche Ausgabe zu dieser Anfrage.

```
1  #PARSER PQL
2  #ADDQUERY
3
4  // snippet[kafka_xml_input](8)
5  input_1_7PWJD = TIMESTAMP({
6          start = 'capture_time'
7      }, input_enrichment_rawdata_4HGK
8  )
9
10 // snippet[sliding_time_window](9)
11 input_1_7JKVZ = TIMEWINDOW({
12         size = [5000, 'MILLISECONDS'],
13         advance = [1, 'MILLISECONDS']
14         }, input_1_7PWJD
15 )
16
17 // snippet[aggregate](12)
18 input_1_7MPAA = SELECT({predicate='type = "producer"'}, input_1_7JKVZ)
19
20 input_3_7MHEM  = AGGREGATION({
21         aggregations=[
22         ['FUNCTION' = 'Sum', 'INPUT_ATTRIBUTES' = 'value',
23                 'OUTPUT_ATTRIBUTES' = 'sum_value'] ,
24                 ['FUNCTION' = 'Avg', 'INPUT_ATTRIBUTES' = 'value',
25                         'OUTPUT_ATTRIBUTES' = 'avg_value'] ,
26                 ['FUNCTION' = 'Min', 'INPUT_ATTRIBUTES' = 'value',
27                         'OUTPUT_ATTRIBUTES' = 'min_value'],
28                 ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'value',
29                         'OUTPUT_ATTRIBUTES' = 'max_value'],
30                 ['FUNCTION' = 'Variance', 'INPUT_ATTRIBUTES' = 'value',
31                         'OUTPUT_ATTRIBUTES' = 'var_value']]
32         },input_1_7MPAA
33 )
34
35 // snippet[kafka_output_shared_model](16)
36 output_1_7YOAE = SENDER({
37     protocol = 'JSON_COSEM',
38     transport = 'Kafka',
39     sink = 'sink_name_7LRFN',
```

```
40      wrapper = 'GenericPush',
41      options = [
42      ['topic', 'aggregated_result'],
43      ['messagetype', 'string'],
44      ['metadata.broker.list', '192.168.33.10:9092'],
45      ['objectType', 'AGGREGATEDDATA']
46      ]},input_3_7MHEM
47  )
```

Listing 8.12: *Generierte Anfrage zur Aggregation*

```
{
  "result":
  {
    "avg_value":1.8400733853306126E7,
    "max_value":3.67711465E7,
    "min_value":30321.206612253598,
    "sum_value":3.680146770661225E7,
    "stddev_value":2.59796867113447E7,
    "var_value":6.749441216196204E14,
    "timeinterval_start":1518697014015,
    "timeinterval_end":1518697014082,
    "minlstart":-1,
    "maxlstart":-1,
    "lend":-1,
    "latency":-1,
    "smgw_id":"smgw_91f4e068-a71f-4de2-9783-830de605f141",
    "sm_id":"sm_6228a1c3-52c0-4db8-b448-66d6340b58d9",
    "value":30321.206612253598,
    "scaler":-3,
    "unit":30,
    "status":"000",
    "capture_time":4281556,
    "type":"producer"
  }
}
```

Listing 8.13: *Ergebnis zur Aggregation*

Abschließend umfasst die Umsetzung des Datamining-Konzeptes noch die Verwendung des PMML-Operators in Odysseus. Das PMML-Format unterstützt diverse Datamining Verfahren, welche unterschiedliche Anforderungen an die verwendeten Daten stellen. Da sich in der Kooperation mit PG DAvE bereits frühzeitig auf das Verfahren der Regressionsanalyse festgelegt wurde, bezieht sich dieses Konzept auch auf eben jenes Verfahren.

Die Architektur des E-Stream Systems sieht den Empfang von PMML-Dokumenten über einen Kafka-Topic vor. Die entsprechende Schnittstelle ist unter Unterabschnitt 8.2.2 beschrieben. Um das Dokument auch in Odysseus verwenden zu können, musste ein Anfragetemplate zur Erzeugung einer entsprechenden Anfrage erstellt werden. Dieses ist in Listing 8.14 dargestellt. Dabei ist zu beachten, dass *input_1* das PMML-Modell enthalten muss, während *input_2* der Eingang für die Daten, auf die das Modell angewendet werden soll, darstellt.

```
{{input_3}} = PMML_EVAL({
    modelName='{{pmml_model_name}}',
    output='{{pmml_output_mode}}',
    stackSize={{pmml_buffer_size}}
}, {{input_1}}, {{input_2}})
```

Listing 8.14: *Snippet zur Verwendung des PMML-Operators*

Auch wenn die PMML-Modelle von der Projektgruppe DAvE zur Verfügung gestellt werden sollen, wurden zur Entwicklung und zu Testzwecken auch von der Projektgruppe E-Stream entsprechende PMML-Dokumente erzeugt. Diese wurden auf den aggregierten Smartmeter-Daten in 15-minütigen Intervallen trainiert. Dabei wurden zunächst lediglich der Wochentag und die Uhrzeit als unabhängige Variablen berücksichtigt und es wurde je ein Modell für Verbraucher und Produzenten trainiert. Es muss somit für beides eine eigene Anfrage im System installiert werden, wobei sich diese jedoch lediglich im verwendeten PMML-Dokument unterscheiden. Das jeweilige Anfragetemplate ist somit in beiden Fällen identisch.

Die Erstellung der Test-PMML-Dokumente erfolgte mit RapidMiner, einem Framework für maschinelles Lernen und Datamining, und wurde über den in Abbildung 8.12 dargestellten Prozess erzeugt [226]. Dabei wurden zunächst die gespeicherten Smart Meter Daten eingelesen und die für den Lernprozess zu beachtenden Attribute ausgewählt. Um den Vorgang möglichst einfach zu halten, wurde hier lediglich die kodierte Uhrzeit sowie als abhängige Variable der jeweils übertragene Wert ausgewählt. Die Rollen, abhängige und unabhängige Variablen, wurden im Schritt *Set Role* ausgewählt. Um einen Trainings- und einen Testdatensatz zu erhalten, ist in Abbildung 8.12 auch der Schritt *Split Data* eingefügt. Anschließend wurde das Modell gelernt und als PMML-Datei gespeichert.

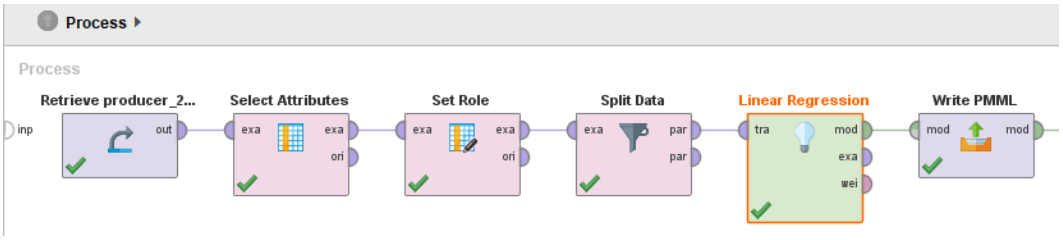


Abbildung 8.12: *Prozess zum Lernen eines Modells in RapidMiner*

8.3.5 JPMML

Damit das Standardformat PMML für Data Mining Modelle verwenden werden kann, wird das Communityprojekt JPMML in Form einer Java Bibliothek in das Odysseus-System integriert. Die Bibliothek unterstützt die Ausführung von PMML Modellen und wird fortlaufend für alle verfügbaren weiterentwickelt. Ein großer Vorteil von JPMML ist die Versionsunabhängigkeit der eingelesenen Modelle. So kann ohne weitere Probleme zum Beispiel Version 4.1 oder 4.3 einlesen und ausgeführt werden, ohne das zwischen Versionsunterschieden differenziert werden muss. In der Regel sind die Modelle abwärtskompatibel, sodass in neueren Versionen prinzipiell neue ausführbare Modellvarianten hinzukommen. Da die Ausführungs-API sich während der Versionssprünge nicht verändert, kann die

Anbindung an die Bibliothek in der Odysseusimplementierung einfach durch eine neue Version ersetzt werden, ohne Funktionalitäten zu verlieren.

8.3.5.1 Initiales Konzept für PMML Extension

PMML kann an vielen (fast allen) Stellen (Elementen) erweitert werden. Diese Erweiterungen werden in Form von Extensions als Kind-Elemente an die entsprechenden Stellen angefügt.[3] Die XSD für eine Extension sieht komplett eigene Elemente wie im Listing 8.15 vor.

```
<xs:element name="Extension">
  <xs:complexType>
    <xs:complexContent mixed="true">
      <xs:restriction base="xs:anyType">
        <xs:sequence>
          <xs:any processContents="skip" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
        <xs:attribute name="extender" type="xs:string" use="optional"/>
        <xs:attribute name="name" type="xs:string" use="optional"/>
        <xs:attribute name="value" type="xs:string" use="optional"/>
      </xs:restriction>
    </xs:complexContent>
  </xs:complexType>
</xs:element>
```

Listing 8.15: *XSD für eine Extension*

Ein Beispiel für eine Extension (*DataFieldSource*), die in *DataField* eingebettet wird, zeigt Listing 8.16.

```
<DataField name="foo" dataType="double" optype="continuous">
  <Extension>
    <DataFieldSource sourceKnown="yes">
      <Source>derivedFromInput</Source>
    </DataFieldSource>
  </Extension>
</DataField>
```

Listing 8.16: *Beispiel für eine Extension*

8.3.5.2 Konzept

Nach Untersuchung der Aufgabenstellung sollten allgemeine Extensions für den Header definiert werden. In dem Task wurde zum Beispiel der Kafka Topic erwähnt, von dem aus Daten für die auszuführende PMML bezogen werden sollen. Ein Beispiel hierfür könnte folgender Ausschnitt in Listing 8.17 darstellen:
KafkaTopics kapselt die Datenquellen für jeweils ein Datenschema. Um mehr als einen Topic bündeln zu können, könnte man mit beliebig vielen Topic Elementen Unions erstellen, die als Datenquellen dienen.

[3] Nähere Informationen sind unter `http://dmg.org/pmml/v4-3/GeneralStructure.html#xsdElement_Extension`.

```
<Header>
   <Extension>
      <KafkaTopics model="modelA">
         <Topic name="exampleTopic" />
      </KafkaTopics>
      <KafkaTopics model="modelB">
         <Topic name="exampleTopic2" />
         <Topic name="exampleTopicFoo" />
      </KafkaTopics>
   </Extension>
<!-- other Elements -->
</Header>
```

Listing 8.17: *KafkaTopics*

8.3.5.3 Struktur

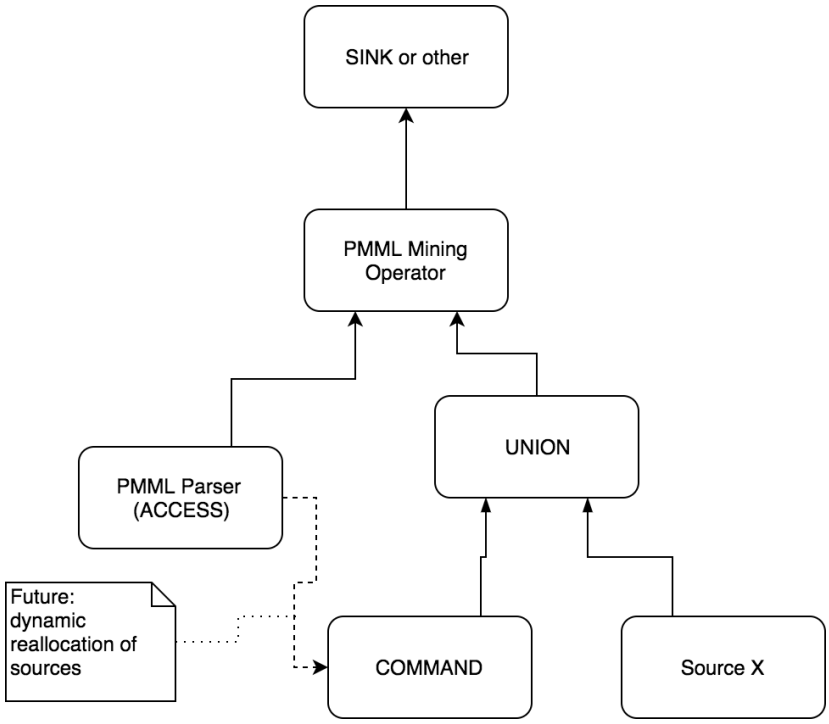


Abbildung 8.13: *Mögliche Zugriffstruktur*

Das PMML Dokument wird über einen ACCESS Operator geladen (zum Beispiel von einem Kafka Topic, oder sonstigem) und durchläuft alle notwendigen Extensions um alle notwendigen Informationen zu erhalten. Ein MiningOperator, der als einen Eingang das Model übergeben bekommt, kann dann aktualisierte Modelle (wenn über einen Topic eine neue PMML ankommt) weiterleiten, ohne dass die Anfrage neu gestartet werden muss. Wenn man einen UNION Operator vor den Mining Operator schaltet, könnte zukünftig sogar über den COMMAND Operator auf weitere Topics zugegriffen werden, sofern diese sich zur Laufzeit ändern sollen (solche Überladungen können wir in den Extensions definieren, wie oben gezeigt).

8.3.5.4 Korrektes Produkt ausführen

- Entweder Odysseus mit dem E-Stream Branch auschecken.
- Aus dem Package heraus de.pgestream.odysseus.studio das Produkt ausführen (damit werden alle notwendigen Bundles automatisch mitgeladen).

8.3.5.5 PMML Operatoren

ProtocolHandler:

Um auf ein PMML Dokument zuzugreifen kann man "PMML" als Protocolhandler angeben. Es sind keine weiteren Parameter notwendig, das PMML Model wird dann als Ausgabe über Port 0 ausgegeben. Hier kann man als Transport zum Beispiel Kafka verwenden um PMML Dokumente zur Laufzeit von Anfragen neu laden zu lassen.

Parameter:

- `fireOnce`: wenn der Wert auf "true" gesetzt wird, dann verarbeitet der Protocolhandler nur das aller erste Dokument vom TransportHandler gelesen und der Status des Operators wird anschließend auf "Done" gesetzt. Sollte dann der InputStream irgendwo anders enden, wird die Anfrage auch entsprechend beendet.

Das Beispiel in Listing 8.18 zeigt den Zugriff auf ein von der DMG zur Verfügung gestelltes PMML Dokument.

```
1  models = ACCESS({
2      source='pmml-data',
3      wrapper='GenericPull',
4      transport='HTTP',
5      protocol='PMML',
6      dataHandler='Tuple',
7      options=[
8          ['uri', 'http://dmg.org/pmml/pmml_examples/rattle_pmml_examples/AuditKMeans.xml'],
9          ['method', 'get'],
10         ['scheduler.delay', '100000'],
11         ['fireOnce', 'true']
12     ],
13     schema=[
14         ['models', 'Object'],
15     ]
16 })
```

Listing 8.18: *ProtocolHandler*

Operator:

Ein PMML Dokument kann potentiell mehrere Mining-Modelle enthalten und ist somit außerhalb des Transporthandlers angesiedelt. Über den Operator PMML_EVAL kann unter Angabe der oben genannten Beschaffung der Modelle eins durch den Namen ausgewählt und ausgeführt werden. Der Input auf port 1 (data) sollte dann Daten enthalten, die mindestens dem Schema von den im PMML Dokument angefragten Daten entsprechen. Sollten mehr Attribute existieren, werden diese natürlich ignoriert. Damit kann man verschiedene Modelle mit den gleichen Daten ausführen, ohne mehrere

Operatoren zur Filterung dazwischen zu schalten, oder mehr Meta-Daten mit den ausgewerteten Daten zu transportieren.

Parameter:

- `modelName`: Jedes Modell innerhalb eines PMML Dokuments hat einen eindeutigen Namen, sollte man „default“ angeben, wird einfach das erste Modell (in der XML-Reihenfolge) ausgeführt.
- `output` (optional, default: NONE): Beschreibt den Ausgabemodus des Operators, hier kann zwischen none, model und input gewählt werden
  - `NONE`: Nur das Ergebnis wird ausgegeben, das Attribut heißt dann „prediction“
  - `MODEL`: Alle vom Model verwendeten Felder werden zusätzlich zur „prediction“ ausgegeben, prediction wird ans Ende der Spalte gehängt
  - `INPUT`: Alle bei Data übergebenen Werte werden mit ausgegeben
- `stackSize` (optional, default: 100): Beschreibt den FIFO Zwischenspeicher in Form eines Stacks. Dieser wird verwendet, wenn DATA Elemente liefert, jedoch noch kein Evaluations Modell eingegangen ist. Sobald ein Modell empfangen wird, verarbeitet der Operator alle im Stack vorhandenen Elemente vor den nächsten eintreffenden Elementen (bis zur maximalen Größe des Stacks). Für jedes neue Modell wird der Stack um Fehler zu vermeiden gelehrt

Das Beispiel in Listing 8.19 zeigt die Verwendung des PMML_EVAL Operators.

```
1 results = PMML_EVAL({
2     modelName='KMeans_Model',
3     output='input',
4     stackSize=5
5 }, models, data)
```

Listing 8.19: *PMML_EVAL Operator*

8.3.6 Verarbeitung der Topologie

Ein zentrales Unterscheidungsmerkmal der Smart Meter für die Vorhersage des Netzzustandes ist der Typ des jeweiligen Smart Meters. So können sie entweder Strom verbrauchen, oder diesen beispielsweise als Photovoltaikanlage in das Netz einspeisen. Da die Informationen über den Typ eines Smart Meters nicht im Datenstrom der *mosaik* Simulation enthalten sind, müssen sie über die Topologiedaten bezogen werden. Diese liegen jedoch in der Simulation lediglich in Form von JSON-Dateien vor, welche mit Odysseus nicht ohne Weiteres verarbeitet werden können. Wie in Unterabschnitt 6.3.8 bereits beschrieben, war es hierfür zunächst erforderlich die Simulation so anzupassen, dass die Topologiedaten auch über einen Kafka Topic bereitgestellt werden. Darüber hinaus musste ein Parser für das JSON-Schema erstellt und entsprechende Anfragen in das E-Stream-System integriert werden, was im Folgenden erläutert wird.

8.3.6.1 Parser für die Topologie

Die Topologie der *mosaik* Simulation liegt in Form eines JSON-Schemas vor, welches exemplarisch in Listing 8.20 dargestellt ist. Dieses musste zur Verarbeitung mit Odysseus zunächst in ein Tupel-Schema

überführt werden. Zu diesem Zweck wurde ein neuer Protocol-Handler für Odysseus implementiert, welcher das JSON-Schema in das für Odysseus handhabbare Tupel-Schema übersetzt.

```
{
    "links": [
        {
            "source": "transformator_9828a31b-2588-4b2e-b683-7bd042f29c9f",
            "target": "node_05a67aa16-4091-407f-894d-53e9147b3544"
        }
    ],
    "smgw": [
        {
            "coordinates": {
                "lat": 53.1875114,
                "lon": 8.1670191
            },
            "id": "smgw_a50e2672-1f88-482a-8eca-be9ce5f49d3d",
            "node": "node_05a67aa16-4091-407f-894d-53e9147b3544",
            "smartmeters": [
                {
                    "id": "sm_5243aae8-ec2b-4604-b38d-38b4d98a376b",
                    "models": [
                        {
                            "name": "HouseHold",
                            "options": {
                                "annualConsumption": 3966,
                                "residents": 3
                            }
                        }
                    ]
                }
            ],
            "start_difference": 337,
            "tags": {
                "addr:city": "Wiefelstede",
                "addr:country": "DE",
                "addr:housenumber": "6",
                "addr:postcode": "26215",
                "addr:street": "Metjendorfer Landstra e",
                "addr:suburb": "Metjendorf",
                "amenity": "fast_food",
                "name": "Alexander Grill"
            }
        }
    ]
}
```

Listing 8.20: *JSON-Schema der Topologiedaten*

Der Protocol-Handler für die Topologie befindet sich in *de.uniol.inf.is.odysseus.estream.topology* und beinhaltet mit *TopologyProtocolHandler* den eigentlichen Protocol-Handler, den *TopologyParser* zum Übersetzen der Daten, sowie *SmartmeterTopologyModel* und *LinkTopologyModel*, welche als Modelle für den Aufbau der Tupel-Schemata dienen.

Auch wenn für die Projektgruppe eigentlich nur das einem Smart Meter zugrundeliegende Modell zur Bestimmung des Typs relevant wäre, wurde der Parser für die Topologie so implementiert, dass

auch die meisten anderen Informationen aus den Topologiedaten ausgelesen werden können. Die Ausgabe des Parsers kann daher durch das Schema bei der Definition der Anfrage sowie die *type*-Option bei der Erstellung einer entsprechenden Anfrage angegeben werden. Eine vollständige Anfrage zur Verwendung des Parsers ist in Listing 8.21 dargestellt. Dabei müssen nicht alle Attribute im Schema angegeben werden. Das Attribut *type* im Schema ist ein zusätzliches Attribut, welches für eine einfachere Unterscheidung der Smart Meter vom Parser erzeugt wird. Der Wert des Attributes ist immer *producer*, wenn das zugrundeliegende *model* einer PV-Anlage entspricht. Andernfalls ist *type* immer *consumer*.

Sollen die Beziehungen zwischen den Elementen in Odysseus verarbeitet werden, kann als *type* *links* und als Schema *source* und *target* angegeben werden. Die Ausgabe beinhaltet dann Tupel aus *source* und *target*.

```
#PARSER PQL
#RUNQUERY
topology := ACCESS({
        source='topology_mosaik',
        protocol='TOPOLOGY',
        transport='Kafka',
        wrapper='GenericPush',
        dataHandler='Tuple',
        options=[
            ['topic', 'topology'],
            ['messagetype', 'string'],
            ['bootstrap.servers', 'localhost:9092'],
            ['type', 'smartmeters']
        ] ,
                schema = [ ['smartmeter','String'],
                           ['gateway', 'String'],
                           ['node', 'String'],
                           ['type', 'String'],
                           ['model', 'String'],
                           ['country', 'String'],
                           ['city', 'String'],
                           ['postcode', 'String'],
                           ['coordinates', 'String'] ]
})
```

Listing 8.21: *Anfrage zur Verwendung des Protocol-Handlers für die Topologie*

8.3.6.2 Anfragen zur Topologieverarbeitung

Der in Unterunterabschnitt 8.3.6.1 beschriebene Protocol-Handler kann zwar die Topologie der *mosaik* Simulation in ein Tupel-Schema überführen, doch müssen die Daten anschließend noch mit dem COSEM-Datenstrom der Simulation zusammengebracht werden. Da für die Projektgruppe in erster Linie von Interesse ist, welchem Typ ein Smart Meter angehört, liegt es nahe, die Daten über die Smart Meter ID zu verknüpfen. Hierfür wird der *Enrich Operator* von Odysseus, wie in Listing 8.22 abgebildet, verwendet [119]. Hierbei werden die Topologiedaten mit den COSEM-Daten verknüpft, wenn das *smartmeter*-Attribut der Topologie mit dem *sm_id*-Attribut der COSEM-Daten übereinstimmt. Im Ergebnis entstehen Tupel mit den Attributen der geparsten Topologie sowie allen Attributen der COSEM-Daten.

```
1 #PARSER PQL
2 #RUNQUERY
3 output = ENRICH({
4         MINIMUMSIZE = 1,
5         PREDICATE = 'smartmeter = sm_id'
6         },
7         topology,
8         raw_data
9 )
```

Listing 8.22: *PQL-Anfrage zur Verwendung des Enrich Operators*

Dabei sollte der *Enrich Operator* eigentlich die geparste Topologie im Cache vorhalten, da diese nur einmal zu Beginn übertragen wird. Im Rahmen von Tests stellte sich jedoch heraus, dass dies nicht wie erwartet funktioniert. Daher werden die Daten nun direkt nach der Anreicherung auf einem weiteren Kafka Topic bereitgestellt und von dort für die weitere Verarbeitung bezogen.

8.4 Backend Beschreibung

Die Hauptkomponenten des Backends sind die `Benutzerauthentifizierung`, die `Benutzerverwaltung`, das `KafkaManagement`, der `QueryWebSocketService`, der `QueryManagerService` und der `QueryBuilderService`. Die Benutzerauthentifizierungsschnittstelle kann genutzt werden um sich einen gültigen Zugangstoken erstellen zu lassen. Über die Benutzerverwaltungsschnittstelle können Nutzer verwaltet werden. Über das KafkaManagement können Kafka Topics verwaltet werden. Das QueryWebSocketService wird genutzt um auf einem Topic ankommende Daten auf den richtigen WebSocketTopic weiterzuleiten, damit das Frontend die Daten verwenden kann. Das QueryManagerService ist für das Verwalten und der QueryBuilderService ist für das Erstellen von Queries zuständig. Diese Komponenten werden in den folgenden Abschnitten noch genauer beschrieben.

8.4.1 Benutzerauthentifizierung

Damit Nutzer nur Aktionen ausführen, die sie mit ihren in Unterabschnitt 8.4.2 beschriebenen Rechten ausführen dürfen, wurden die Rest Endpunkte gesichert. Um auf diese Endpunkte zugreifen zu können, muss sich der Nutzer zunächst authentifizieren. Im Folgenden wird der serverseitige Authentifizierungsprozess beschrieben.

Zunächst müssen die Anmeldedaten an den „/api/authentication/jwt“ Endpunkt gesendet werden. Die Anmeldedaten bestehen aus dem Benutzernamen und dem Passwort. Wenn ein Nutzer zu den übergeben Anmeldedaten existiert, also in der Datenbank abgespeichert ist, wird ein Zugangstoken zurückgesendet. Dieser Zugangstoken ist signiert. Es ist also sehr schwierig die Zugangstokens zu fälschen. Dadurch können die Zugangstokens gut zur Authentifizierung genutzt werden. Sie sind aber aus Sicherheitsgründen nur für eine bestimmte Zeit gültig. Sobald ein Zugangstoken nicht mehr gültig ist, kann dieser nicht mehr zur Authentifizierung genutzt werden. Der Nutzer muss sich dann einen neuen Zugangstoken von dem Backend erstellen lassen.

Dabei werden JSON Web Tokens[4] als Zugangstokens verwendet. In diesen Zugangstoken wird der Benutzername und die Rollen des Nutzers abgespeichert. Der Zugangstoken kann dann zum Beispiel im Frontend entschlüsselt werden, um so herauszufinden welche Rollen der Nutzer hat. Der Vorteil ist, dass das Backend so die wichtigsten Daten hat, die es zur Ausführung der Anfragen benötigt. Dadurch können viele Datenbank-Aufrufe vermieden werden, wodurch auch die Performanz erhöht wird. Bei einer HTTP-Abfrage muss der Zugangstoken im sogenannten „Authorization“ Header übergeben werden. Außerdem muss das Präfix „Bearer “ vorangestellt werden. Die Autorisierung, also wie ein bestimmter Endpunkt im Backend gesichert werden kann, ist bereits in Unterabschnitt 8.4.2 beschrieben.

8.4.2 Benutzerverwaltung

Das zu entwickelnde System soll über eine Benutzerverwaltung verfügen, um einen auf den Nutzer abgestimmten Zugang zu gewissen Funktionen des Systems zu ermöglichen. Zunächst wurde ein Konzept für das User- und Identity Management ausgearbeitet, wodurch Rollen der Nutzer definiert und Technologien evaluiert werden. Die Rollenhierarchie, die sich beim ersten Brainstorming ergeben hat, ist in Tabelle 8.1 zu sehen.

Dabei ergibt sich folgende Hierarchie:

$$\begin{aligned} \text{Admin} \rightarrow \text{Leitwarten Personal} &\rightarrow \text{PMML Model Benutzer} \\ &\rightarrow \text{PMML Model Lieferant} \end{aligned}$$

Eine solche Hierarchie wird benötigt, damit Nutzer beim Aktualisieren keine höheren Rollen als ihre eigenen angeben können. Ein Nutzer darf sich als Leitwartenpersonal beispielsweise nicht selbst zum Admin heraufstufen. Daher findet eine Fehlerbehandlung für invalide Rollenzuweisungen statt.

Implementierung

Für die Implementierung der in Tabelle 8.1 aufgeführten Rollen des entwickelten Spring Boot Backendsystems, wurde auf die bereitgestellten Funktionalitäten von Spring Security zurückgegriffen[5]. Auf Ebene der Datenschicht wurden für die zu verwaltenden Nutzer und die entsprechenden Rollen zunächst in einer MariaDB Datenbank, zur Projektstruktur passende, Schemata angelegt Abbildung 8.14. Ein `User` umfasst dabei Informationen bezüglich des einzigartigen Nutzernamens, der Email, des Passworthashs und der optionalen Vor- und Nachnamen. Außerdem existiert eine Rollentabelle (Authority), welche als Menge von einzigartigen Rollenbezeichnern dient. Einem `User` können mehrere Rollen zugewiesen werden, weshalb eine Zuweisungstabelle für die Beziehung zwischen den Nutzern und Rollen existiert.

Danach wurde unter Verwendung des Spring Security Frameworks eine initiale Sicherheitskonfiguration implementiert. Diese stellt durch den von Spring bereitgestellten IoC-Container eine BCrypt Spring Bean für das Kodieren und Dekodieren von Passwörtern zur Verfügung. Hierbei wurde die

[4] `https://jwt.io/`
[5] `https://projects.spring.io/spring-security/`

Rollenname	Rollenbezeichner	Rechte
PMML Model Benutzer	ROLE_MODEL_USER	• darf PMML ansehen • darf PMML in PQL umwandeln und an Odysseus schicken
PMML Model Lieferant	ROLE_MODEL_PROVIDER	• darf PMML versenden
Leitwarten Personal	ROLE_USER	• darf Daten ansehen (aktueller/prognostizierter Verbrauch) • darf Störungen einsehen, Kommentare dazu schreiben, etc. • darf die Netztopologie abrufen • darf Meldungen über illegale Anlagenbetreibungen einsehen
Technischer Nutzer	ROLE_TECH_USER	• PG-DAvE • darf PMML abfragen • darf anonymisierte Verbrauchsdaten, Prognosen, etc. abfragen
Admin	ROLE_ADMIN	• darf User anlegen/löschen/sperren/etc. • darf alle Konfigurationsparameter des Servers ändern

Tabelle 8.1: *Nutzer- und Rollenkonzept*

zu durchlaufende Rundenanzahl von BCrypt auf 11 erhöht, um eine gute Balance zwischen Brute Force Sicherheit und der Wartezeit bei realen Anmeldeversuchen zu erreichen. Damit ein Nutzer die REST API verwenden kann, muss er sich zunächst authentifizieren. Daher ist die Route /login mit den HTTP-Verben GET und POST auch für nicht authentifizierte Benutzer freigeschaltet. Auf Ebene der Datenzugriffsschicht werden die von Spring bereitgestellten JpaRepositories zur automatischen Generierung von CRUD-Operationen in Verbindung mit Hibernate für das objektrelationale Mapping

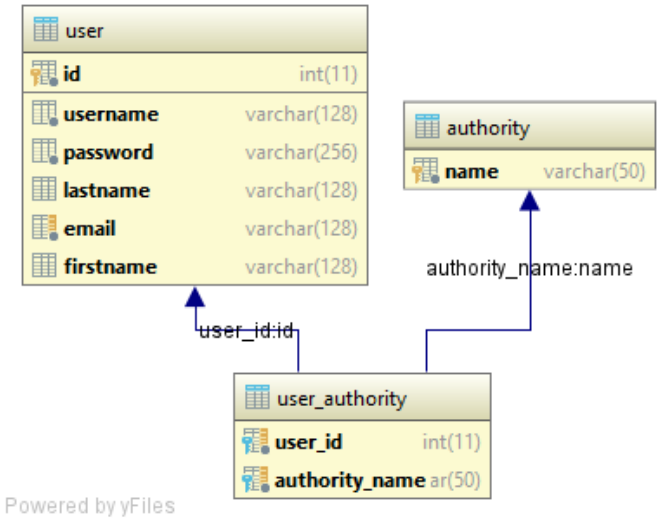


Abbildung 8.14: *Datenbankausschnitt der Benutzerverwaltung*

in passende Entitätenklassen verwendet. Für die User wurde somit eine UserEntity Klasse erstellt und für die Rollen eine AuthorityEntity Klasse, welche für das Mapping benötigt werden.

Passend dazu wurden die Repositories angelegt, das UserRepository für die Nutzeroperationen und das AuthorityRepository für die Operationen auf der Rollentabelle. Da die hier vorgestellte Implementierung einer Nutzerverwaltung eine eigens erstellte Datenbankstruktur besitzt, muss für das Zusammenspiel mit dem Spring Security Framework eine neue Spring Bean mit dem Namen *userDetailsService* vom Typ UserDetailsService angelegt werden. Dies wurde durch die Annotation `@Component` an der von UserDetailsService abgeleiteten Klasse und dem Überschreiben der Methode „UserDetails loadUserByUsername(String username)“ umgesetzt. Diese Methode wird automatisch von Spring Security aufgerufen, um die Details eines User (Rechte, Username, Passwort, . . .) passend zu einem einzigartigen Nutzernamen zurückzugeben, um so das Anmelden und die Rechteverifikation durchzuführen. In dieser Methode wird durch das im Konstruktor injizierte UserRepository nach einem UserEntity Objekt passend zu dem übergebenen Namen gesucht und dieses bei Erfolg in eine eigens erstellte SecurityUserDetails Klasse gekapselt. Des Weiteren wird eine UserNotFoundException geworfen. Die erstellte SecurityUserDetails Klasse dient dazu, den passenden Rückgabetyp der *loadUserByUsername* Methode von Spring Security zu entsprechen und dabei das UserEntity Objekt zu Kapseln, um diese später durch den SecurityContextHolder von Spring Security wieder zugreifbar zu machen, um somit den eingeloggten Benutzer bei Bedarf in jeder Anfrage erhalten zu können.

Um die HTTP-Anfragen bezüglich der Benutzerverwaltung verarbeiten zu können, wurde eine UserResource Klasse erstellt, welche als Rest-Controller agiert. Diese bildet alle Anfragen welche das Pfadpräfix „/api/users“ besitzen auf die implementierten Methoden ab. Die implementierten Methoden entsprechen der Auflistung alle Funktionalitäten in Tabelle 8.1. Die Dokumentation der REST Methoden geschieht über Swagger und die bereitgestellten Annotationen `@ApiOperation` und `@ApiResponses`/`@ApiResponse`. Dabei können Beschreibungen für sowohl die Datentransferobjekte (DTO), als auch für die reine Funktionalität selbst eingefügt werden und es können Beschreibungen und Strukturen für die Rückgaben, speziell im Fehlerfall, getätigt werden Listing 8.23. Für einfache

Rollenbeschränkungen wird die Annotation `@Secured("ROLE_X")` verwendet. Für komplexere Rollenbeschränkungen, wie es zum Beispiel beim Aktualisieren eines Nutzers der Fall ist, kann die Annotation `@PreAuthorize` in Verbindung mit der Spring Expression Language verwendet werden. Ein Nutzer darf die update Operation nur verwenden, wenn dieser mindestens Adminrechte besitzt, oder wenn der zu aktualisierende Benutzer er selbst ist, siehe Listing 8.23.

```
1  @ApiOperation(notes = "Tries␣to␣update␣an␣existing␣user␣corresponding␣to␣the␣passed␣data.",
2                value = "Update␣a␣user",
3                authorizations = @Authorization(value = Swagger2Config.securitySchemaJWT))
4     @ApiResponses({
5        @ApiResponse(code = 200,
6            message = "User␣was␣updated␣successfully.", response = UserDto.class),
7        @ApiResponse(code = 404,
8            message = "Not␣found!␣This␣could␣happen␣if␣an␣admin␣tries␣to␣update␣a␣non␣existing␣user.",
9            response = BaseErrorResponse.class)
10    })
11    @PreAuthorize("hasRole(T(de.estream.security.AuthoritiesConstants).ADMIN)
12 ␣␣␣␣␣␣␣␣␣␣or␣#updateUserDto.username␣==␣principal.username")
13    @PutMapping("/")
14    public ResponseEntity<UserDto> updateUser(@Valid @RequestBody UserDto updateUserDto)
15           throws URISyntaxException {
16       UserEntity userEntity = userService.updateUser(updateUserDto);
17
18       return ResponseEntity
19              .created(new URI("/api/users/" + userEntity.getId()))
20              .body(new UserDto(userEntity));
21    }
```

Listing 8.23: *Implementierung der REST Anfrage zum Updaten eines Benutzers*

Außerdem ist, wie in Listing 8.23 zu sehen, die komplette Domänenschicht in eigene Serviceklassen ausgelagert. Dies dient der Testbarkeit und Abstraktion der Operationen. Fehler werden durch das werfen von eigens angefertigten Exceptions signalisiert. Um die gesamte REST Schnittstelle liegen eigens implementierte ControllerAdvices, die diese Fehler gegebenenfalls abfangen, die Exceptions auswerten und daraufhin passende Fehlermeldungen in einem strukturierten Format zurückgeben.

Keycloak als Authentifizierungsprovider

Keycloak ist eine flexible, open-source und technologieunabhängige Möglichkeit des Outsourcing für Authentifizierung und Autorisierung. Hiermit kann der wesentliche Aspekt jeder Applikation, der Sicherheit, in einer eigenen Infrastruktur eingesetzt und angepasst werden. Keycloak ist nicht nur ein Authentisierungsserver, sondern bietet beispielsweise auch ein komplettes Managementsystem. Keycloak kann dabei leicht in Form eines Spring Boot Adapters verwendet werden. Es wurde beschlossen, dass Keycloak als Authentifizierungsprovider bei Bedarf ergänzt werden soll, da es zunächst für die Entwicklung unseres Systems nicht essentiell ist[6].

[6] https://developers.redhat.com/blog/2017/05/25/easily-secure-your-spring-boot-applications-with-keycloak/

8.4.3 KafkaManagement

Die Verwaltung von Kafka lässt sich in zwei Bereiche unterteilen. Der erste Bereich wird durch `SpringKafka` abgedeckt und verwaltet die statische Verarbeitung von Kafkatopics, d.h. Topics, die immer existieren, und auf denen kontinuierlich Daten ankommen. Der zweite Bereich wird durch den `KafkaService` abgedeckt, der die dynamische Verarbeitung von Kafkatopics verwaltet. Dies umfasst Topics, die zur Laufzeit verwaltet werden müssen. Außerdem kann es sein, dass die Nachrichten auf dem Topic auch nicht zur gesamten Laufzeit des Servers benötigt werden, sodass der Topic auf Anfrage abonniert und zu einem späteren Zeitpunkt deabonniert werden muss.

Dafür benutzt der `KafkaService` die offizielle Kafka-API. Damit soll auch die Problematik der sich entwickelnden API von Kafka beginnend mit der Version `0.10.0` zur Version `1.0.0` adressiert werden, da Änderungen der Kafka-API nur durch Änderungen im `KafkaService` übernommen werden können und so vom restlichen System gekapselt werden. In der Version `0.10.0` wurde `SpringKafka` genutzt, um die Rohdaten auszulesen und durch eine WebSocket-Verbindung an den Client zu senden. Da das Topic bereits zur Kompilierzeit bekannt ist und das Topic nach dem Erstellen auch nicht weiter geändert werden muss, reicht eine statische Verarbeitung für die Rohdaten aus. Zusätzlich wurden Funktionen im `KafkaService` wie das Erstellen, Löschen und Abonnieren von Topics hinzugefügt. Diese Funktionen können auch über die REST-API von einem Administrator-Nutzer angesprochen werden. Zusätzlich wurde eine weitere Ebene eingeführt, durch die sich andere Systemteile an Nachrichten von bestimmten Topics anmelden können, welche unabhängig vom Zustand der Topics sind, sodass auch asynchrone Verarbeitung möglich ist und der Zustand des Kafka-Clusters unabhängig vom Zustand des Servers ist. Außerdem ist es möglich zum Serverstart Standardtopics zu erstellen.

Neben der Anpassung für die Version `0.11.2` wurde bei der `SpringKafka`-Verarbeitung auch das Anfragen der Topologie hinzugefügt, da diese ebenfalls nur auf einem festen Topic vorhanden ist und keine dynamische Konfiguration benötigt. Der `KafkaService` wurde um die Möglichkeit erweitert, für jedes Topic eine eigene Konfiguration zu erstellen und bestehende Konfigurationen zu ändern. Darüber hinaus wurde durch die Instabilität von Kafka in dieser Version ein Circuit-Breaker-Pattern für die Kafka-Operationen hinzugefügt, da von der Kafka-API geworfene Fehler normalerweise auf einen fehlerhaften Handshake zurückzuführen waren, sodass die Operation erfolgreich im Cluster ausgeführt wurde, aber durch den falschen Handshake als fehlerhaft erkannt wurden und den Server zum Absturz brachten. Durch das Absichern des Services wurde nur die Sicherung geöffnet und das System konnte selbst entscheiden wie mit dem Fehler umgegangen wird.

Eine Aktualisierung auf die Version `1.0.0` von Kafka war nicht möglich, da `SpringKafka` eine neuere Spring-Version benötigt. Eine Anpassung des `KafkaServices` ist theoretisch möglich, aber durch die vielen Änderungen der API muss der Service vollständig neu implementiert werden, was zeitlich innerhalb der Projektarbeit nicht mehr möglich war. Daher wurde die Möglichkeit für den `KafkaService` geschaffen, externe Broker einzubinden, sodass über die lokale Architektur hinaus durch externe Kafka-Cluster mit anderen Systemen kommuniziert werden kann.

8.4.4 QueryManagerService

Die Aufgabe des Anfragemanagements ist es, die vom Client angefragte Anfrageverarbeitung zu koordinieren, zu verwalten und zu überwachen. Hierfür wird eine entsprechende REST-Schnittstelle

zur Verfügung gestellt, welche die Kommunikation zwischen Client und Anfragemanagement ermöglicht. Die Schnittstelle ermöglicht das Installieren, Starten, Stoppen, Löschen und die Statusabfrage von Anfragen. Bei der Installation einer Anfrage wird ein `RegisterQueryDto` übergeben, welches Informationen über das Anfrage-Template beinhaltet und in Abbildung 8.15 abgebildet ist. Die Template-Informationen werden dann in der Datenbank gesichert und an die Anfragegenerierung weitergegeben, welche versucht aus den Informationen eine PQL-Anfrage zu generieren. Wurde eine Anfrage erfolgreich generiert, wird das Anfragemanagement aufgefordert alle notwendigen Schritte für die Datenübertragung durchzuführen. Die wichtigsten Schritte betreffen die Registrierung von Kafkatopics, die Installation von Kafkatopic-Listenern und die Erstellung von WebSocketverbindungen. Alle weiteren Aktionen, die die Schnittstelle bereitstellt, können anschließend über eine zurückgelieferte Anfrage-ID angefragt werden. Eine Übersicht über den Ablauf bei der Installation einer Anfrage ist in Abbildung 8.16 sehen.

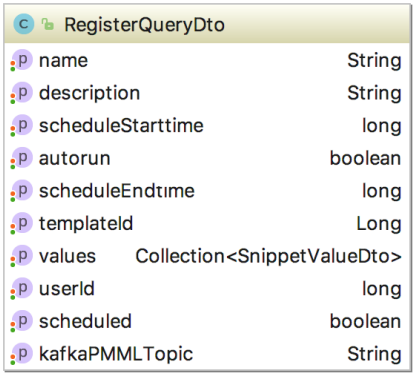


Abbildung 8.15: *`RegisterQueryDto`-Klasse*

In Abbildung 8.17 ist die Klassenhierarchie zu sehen, welche das Anfragemanagement beschreibt und seine einzelnen Bestandteile beschreibt. Die Hauptklasse des Anfragemanagements ist `QueryManagerService`, welche alle weiteren deklarierten Klassen als sogenannte Helferklassen verwendet und für deren Koordination zuständig ist. Die `QueryManagerService`-Klasse implementiert die beiden Java-Interfaces `IQueryManagerService` und `IQueryStringListener`. Das `IQueryStringListener`-Interface deklariert eine Methode, die von der `QueryBuilderService`-Klasse (vgl. Unterabschnitt 8.4.5) als asynchroner Callback verwendet wird, um eine generierte PQL-Anfrage an die `QueryManagerService`-Klasse zurückzuliefern.

Die `QueryDatabaseService`-Klasse stellt Methoden zum Erstellen, Speichern und Löschen von `QueryEntity`, `QueryMappingEntity` und `QueryWebSocketEntity`-Objekten bereit, welche in einem beliebigen Jpa-Repository gesichert werden können. Die `QueryWebSocketService`- Klasse ist für die Verwaltung von WebSocketverbindungen zuständig, die anhand der Anfrage-Informationen erstellt werden müssen. Die Klasse erstellt daher für jede Verbindung einen `QueryResultListener`, der auf einem angegebenen Kafkatopic horcht (siehe Unterunterabschnitt 8.4.4.2). Zusätzlich verwendet die Klasse den `QueryTopicService` um die benötigten Kafkatopics im System zu registrieren (vgl. Unterabschnitt 8.4.3). Die `QueryActionService`- und `QueryREST`-

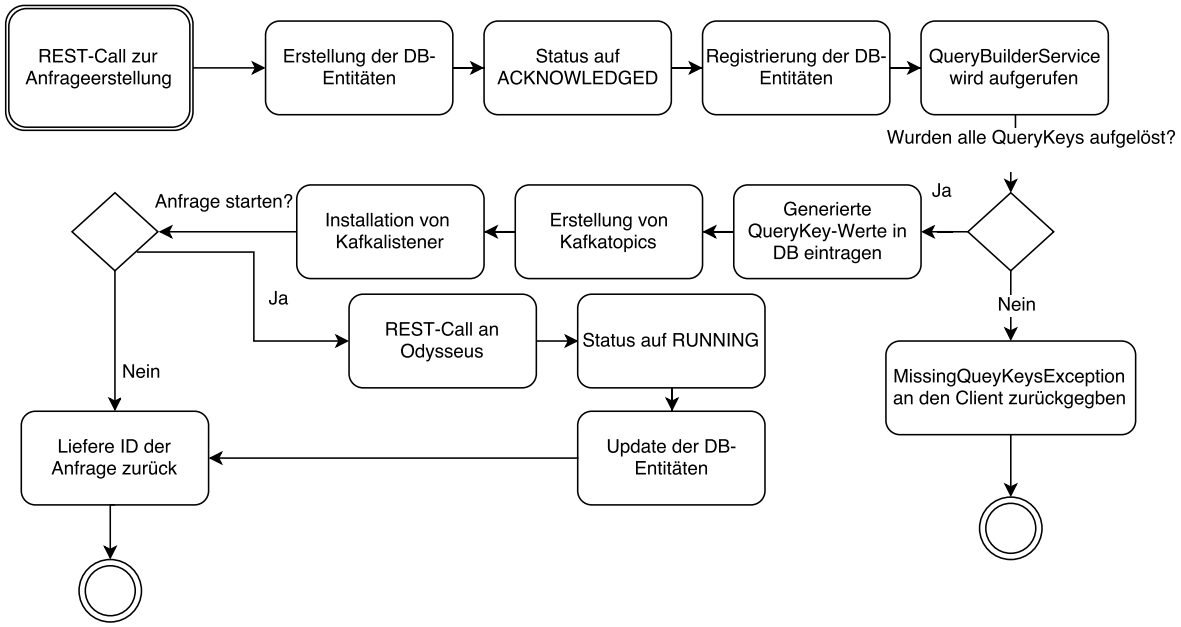


Abbildung 8.16: *Ablauf bei der Installation einer Anfrage*

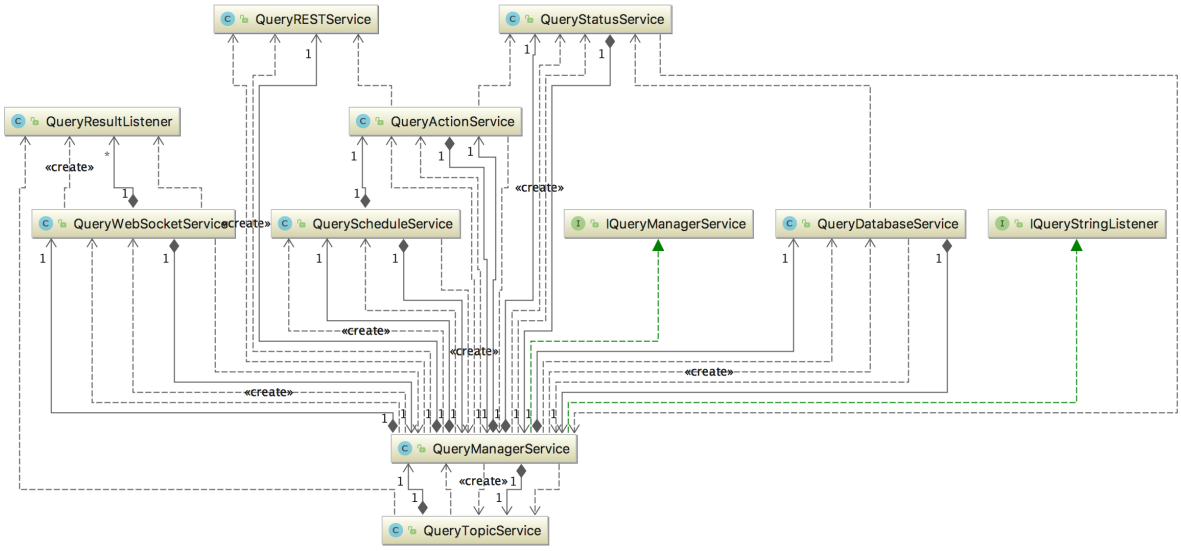


Abbildung 8.17: *Klassendiagramm Anfragemanagement*

`Service`-Klasse stehen in enger Beziehung zueinander, da zunächst eine übergebene Aktion auf Gültigkeit geprüft wird und anschließend die Aktion durch einen entsprechenden REST-Call zu einer Odysseus-Instanz ausgeführt wird. Eine Aktion bezieht sich hierbei auf das Installieren, Starten, Stoppen und Löschen einer Anfrage auf einer gegebenen Odysseus-Instanz. Der `QueryActionService` ist also dafür zuständig zu prüfen, ob eine Aktion ausgeführt werden darf. Beispielsweise kann eine Anfrage erst gestartet werden, wenn diese zuvor installiert wurde. Der `QueryRESTSer-`

vice hingeben ist allein für die Kommunikation und das Aufrufen von REST-Calls zuständig. Der QueryScheduleService soll für das Scheduling von Anfragen verwendet werden, also die automatische Anfragesteuerung nach Zeitplänen ermöglichen soll. Bislang wird diese Klasse allerdings nicht verwendet, da keine Implementierung vorhanden ist.

8.4.4.1 Template-System

Das Anfragesystem basiert auf einem Template-System, welches durch verschiedene Komponenten ein flexibles und erweiterbares System zur Definition von Anfragen darstellt. Die Komponenten werden durch Datenbanktabellen realisiert, welche in Abbildung 8.18 abgebildet sind. Hierbei erlaubt das Template-System die Definition einer Anfrage über sogenannte QueryTemplates (als Tabelle querytemplate angegeben) und QuerySnippets (als Tabelle querysnippet angegeben). Ein Snippet repräsentiert ein PQL-Skript, welches verschiedene PQL-Operatoren beinhalten kann. Innerhalb eines Snippets können QueryKeys verwendet werden, die zur Parametrisierung eines Snippets genutzt werden können. Die Zuweisung von Snippet und Keys wird über die Join-Tabelle querysnippet_key realisiert. Ein QueryKey dient als eine Referenz für einen Parameter, der beispielsweise vom Client spezifiziert wird.

Ein Template wiederum besteht aus einem oder mehreren Snippets, die über die querytemplate_snippet Tabelle miteinander verbunden werden. Zusätzlich lässt über die Tabelle querysnippet_connect spezifizieren, ob der Wert eines Keys aus einem Snippet auch in einem anderen Snippet verwendet werden soll. Beispielsweise lassen sich so die Ergebnisse von Operatoren außerhalb eines Snippets wiederverwenden. Außerdem besitzen sowohl Templates als auch Snippets einen Typ, der über die Tabellen querytemplatetype und querysnippettype referenziert wird.

Die Aufgabe des QueryManagerService im Zusammenhang mit dem Template-System besteht nun darin, die vom Client übergebenen Parameter in entsprechende Objekte zu verwandeln und in der Datenbank abzulegen. Hierfür existieren die drei Tabellen query, querymapping und querywebsocket. Während die query-Tabelle Metainformationen zu einer Anfrage darstellt, wie Erstellungszeit oder Status (beispielsweise INSTALLED, RUNNING, etc.), beschreibt die querymapping-Tabelle die Wertezuweisung von Querykeys zu einem Snippet. Die querywebsocket-Tabelle speichert Informationen, die für die Datenübertragung der Anfrageergebnisse relevant sind. Darüber hinaus besitzt eine query-Entität eine Referenz auf die Anmeldedaten für eine Odysseusinstanz (odysseuscredential und odysseusserver). Der QueryManagerService kann entscheiden auf welcher Instanz die Anfrage installiert werden soll. Wurden die entsprechenden Entitäten in der Datenbank registriert, wird der QueryBuilderService aufgerufen und es wird versucht die Anfrage anhand der Informationen aus der Datenbank zu generieren.

8.4.4.2 WebSocketService

Die QueryWebSocketService-Klasse wird über den QueryManagerService aufgerufen, wenn der Client eine Anfrage an den Server stellt. Im Client wird erwartet, dass die durch die Anfrage verarbeiteten Daten über ein entsprechendes WebSockettopic übertragen werden. Anschließend sollen die Daten im Client durch verschiedene Komponenten visualisiert werden können, wobei das vorgesehen wird, dass die Anzeige unterschiedliche Datentypen unterstützt. Die entsprechende Definition der Datentypen erfolgt über Jackson-Klassen (JSON-Schema), die bei der Anfragedefinition angegeben

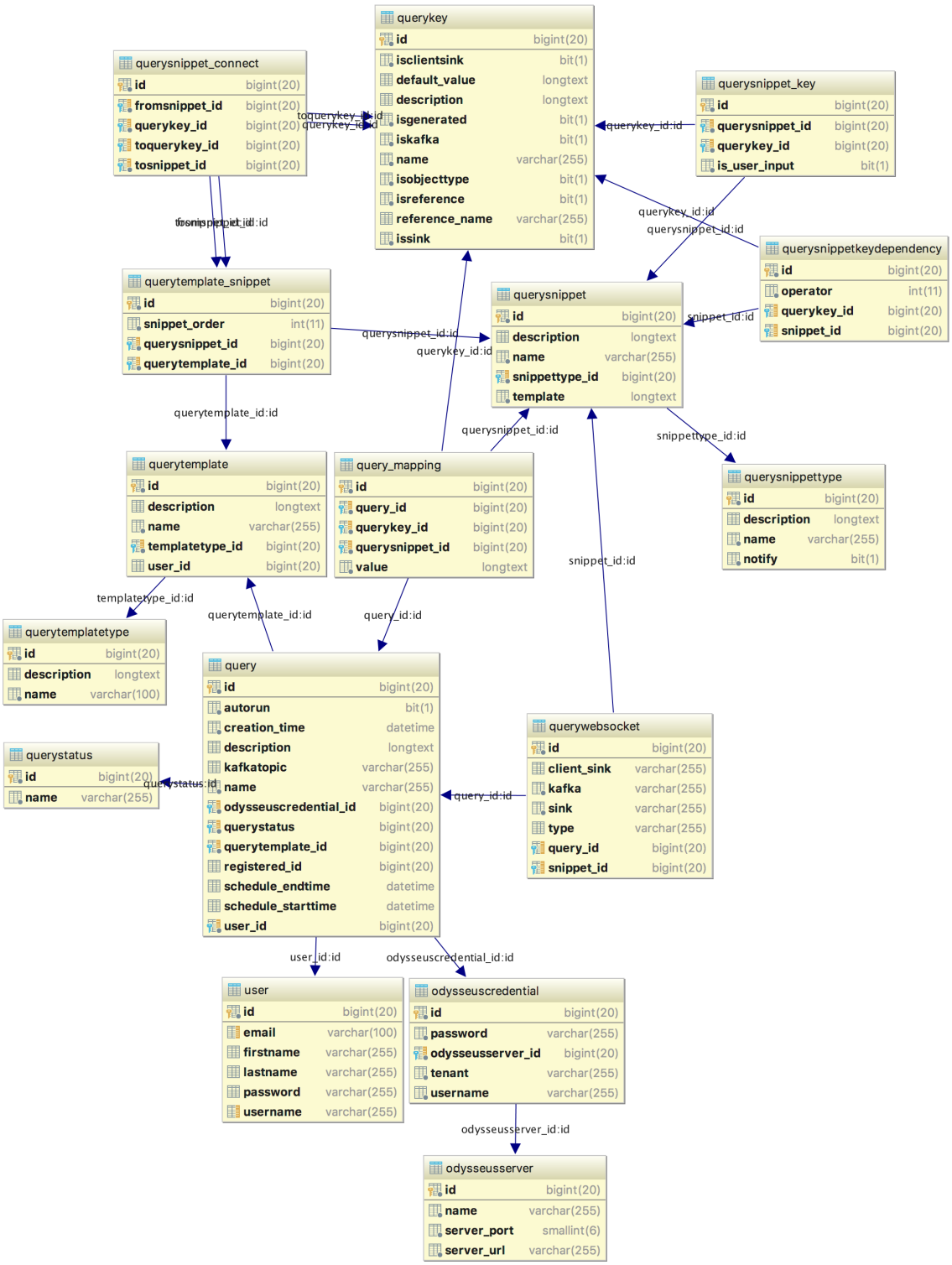


Abbildung 8.18: *Datenbankschema für das Anfragesystem*

werden müssen. Um die Ergebnisdaten einer Anfrage zum Client übertragen werden können, müssen entsprechende Listener im Backend installiert werden. Die Klasse `QueryResultListener` wird für die Zuordnung zwischen einem Kafkatopic und einer WebSocketsession verwendet. Der `QueryResultListener` horcht auf dem Kafkatopic und überträgt Ergebnisdaten zum Client.

Eine Anfrage kann mehrere Senken-Operatoren definieren, die das Ergebnis auf verschiedene Kafkatopics schreiben. Dadurch kann es vorkommen, dass für eine Anfrage mehrere `QueryResultListener`-Instanzen installiert werden müssen. Zusätzlich kann es sein, dass jede Datensenke ein anderes Datenschema definiert. Der in Listing 8.24 dargestellte Senken-Operator zeigt, wie das JSON-Schema **RawData** über den Parameter `objectType` deklariert wird.

```
output = SENDER({
            sink='output',
            wrapper='GenericPush',
            transport='Kafka',
            protocol='JSON_COSEM',
            options=[
                ['objectType','RawData'],
                ['topic', 'test'],
                ['messagetype', 'string'],
                ['metadata.broker.list', 'localhost']
            ]
          },
         pipe
        )
```

Listing 8.24: *PQL-Anfrage zum Schreiben auf Kafkatopic*

Es ist theoretisch denkbar, dass mehrere Senken-Operatoren innerhalb eines Snippets definiert werden. Daher ist es möglich komplexe Anfrage-Konstrukte zu formulieren, was die Flexibilität und Mächtigkeit des Anfragesystems unterstreicht. Zur Folge dessen, können Abhängigkeiten zwischen QueryKeys auftreten, da beispielsweise verschiedene Keys miteinander assoziiert werden müssen, wenn diese im selben Operator verwendet werden (zum Beispiel `json_schema_1` und `kafka_topic_1`). Ansonsten kann der `QueryWebSocketService` beim Installieren der Listener keine Datentypen oder den zu einer Senken dazugehörigen Kafkatopic identifizieren. Daher beinhaltet das Datenbankschema die Tabelle `querywebsocket` (vgl. Abbildung 8.18). Ein weiterer Punkt ist der Typ eines Snippets, der sich auf den Senken-Typ bezieht. Implizit wird definiert, dass alle Senken-Operatoren den gleichen Senken-Typ haben (beispielsweise `KAFKA_OUT`). Ansonsten hätte die Typ-Definition eines QuerySnippets wenig Bedeutung. Es muss daher als Konvention festgehalten werden, dass die Senken-Operatoren innerhalb eines QuerySnippets den gleichen Senken-Typ haben müssen.

Die QueryResultListener-Instanzen werden dann von der Klasse QueryWebSocketService erstellt, wobei der Service durch die Entitäten in der Tabelle websockettopic und querssnippetkeydependency die Abhängigkeiten einfach auflösen kann. Die Erstellung erfolgt wie in 8.4.4.2 zu sehen ist.

```
list types;//contains all snippet type ids
           //that should be associated with
           //a websocket session

for each Snippet in Template:
  if(types.contains(Snippet.querySnippetTypeId))
    // iterate over all keys that are marked as a JSON-Schema
    for each QueryKey in getAllJsonSchemaKeys(Snippet):
        // get all depending keys for the given query key
        list QueryKeyDependencies = getQueryKeyDependencies(QueryKey)
        //checks if there is already a
        //QueryResultListener for this schema
        if(!existsSession(QueryKey)
            //create QueryResultListener and
            //start listening on kafka topic
            createWebSocketSession(QueryKey, QueryKeyDependencies)
        else
            //add QueryResultListener and
            //start listening on kafka topic
            associateWebSocketSession(QueryKeyDependencies)
```

Listing 8.25: *Peusdocode WebSocketService*

Es soll für jede QuerySnippet-Entität eines Anfrage-Templates, welches eine entsprechende QuerySnippetTypeId besitzt, eine Websocketsession assoziiert werden. Es wird dabei entweder ein neuer QueryResultListener erstellt und auf den Kafkatopic des QuerySnippets gehorcht oder es wird ein vorhandener QueryResultListener verwendet. Das ist der Fall, wenn zwei Senken-Operatoren innerhalb eines QuerySnippets einer Anfrage den gleichen Wert für ihren json_schema_x haben, also das gleiche JSON-Schema verwenden. Gleiches gilt für zwei QuerySnippets innerhalb einer Anfrage. Das bedeutet, wenn ein Senken-Operator das gleiche JSON-Schema verwendet wie ein Senken-Operator in einem anderen QuerySnippet, werden die Senken mit dem selben Kafkatopic assoziiert.

8.4.5 QueryBuilderService

Wie bereits in Unterabschnitt 8.4.4 beschrieben, ist der QueryBuilderServicefür die eigentliche Erzeugung der Anfrage, die als String an Odysseus übertragen wird, zuständig. Hierzu wird die abstrakte Klasse AbstractQueryBuilderService abgeleitet und implementiert. Diese kapselt im Wesentlichen den Aufruf zum Bauen einer Anfrage durch Eingabe des entsprechenden QueryEntity Objekts ab.

8.4.5.1 Ermittlung von QueryKey-Werten

Die QueryKeys können basierend ihrer Voreinstellung auf verschiedene Weisen ermittelt werden. Standardmäßig hat ein QueryKey einen Namen, eine Beschreibung und mehrere Flags, die den Ursprung vermitteln. Sollte kein Flag gesetzt sein, muss vom Benutzer ein Wert übergeben werden. Für Keys, die keinen Wert zugewiesen bekommen haben, aber ein Standardwert vorliegt, wird dieser

genommen, da er automatisch vom System gesetzt wird. Ansonsten wirft der Builder einen Fehler, da dieser Wert von nirgends sonst ermittelt werden kann. Soll das System den Wert automatisch erstellen (z. B. für Senken-Namen in Odysseus, die für den Benutzer ohne Belang sind, jedoch über das System hinweg einmalig sein müssen), kann er als automatisch generiert definiert werden. Diese generierten Schlüssel dienen zusätzlich dazu, verschiedene `QuerySnippets` mit einander zu verbinden. Eine weitere Möglichkeit Schlüsselwerte zu ermitteln ist eine Referenz. Diese sagen dem System, dass der eigentliche Wert erst später geliefert wird, weil er z. B. aus einer externen Quelle, oder vom System selbst kommt.

8.4.5.2 Asynchrone Daten über einen Kafka Topic

Da eine Anfrage potentiell nicht sofort gebaut werden können muss, wird auf EventListener zurückgegriffen, welche man durch die Implementierung des `IQueryStringListener` dem `QueryBuilderService`übergeben kann. Dieser bekommt dann, beim erfolgreichen Bauen durch das Interface, die `QueryEntity` und den entsprechenden String, der die Anfrage darstellt, zurück. Wenn die `build` Methode des `QueryBuilderService`aufgerufen wird, liefert sie eine Map mit den fehlenden Key Werten, die der Builder braucht, zurück. Sollte in der `QueryEntity` ein Kafkatopic angegeben sein, so wird über den `KafkaService` auf diesen abonniert. Durch den im Abonnement gegebenen Topic können weitere Daten verarbeitet werden. Für jeden `QueryKey`, der als Referenz markiert ist, wird der `reference_name` auf den angekommenen Daten ausgewertet und im Cache gespeichert. Wenn der Builder ein Template übergeben bekommt, sichert er vorerst für alle Referenzen die kompilierten XPath Expressions ab. Anschließend werden die Daten aus dem Kafkatopic als XML interpretiert um dann nach den kompilierten Expressions durchsucht. Sollte ein Wert gefunden werden, wird das Ergebnis im Cache abgelegt. Zusätzlich ist das System derartig erweiterbar, dass auch weitere Formate nachträglich eingefügt werden können. Beispielsweise könnte man auch Daten im JSON-Format implementieren und als Referenzschlüssel den Pfad zum Attributnamen des Tupels angeben.

8.4.5.3 SnippetBuilder

In Unterabschnitt 8.4.4 wurde bereits kurz aufgefasst, dass ein Template aus verschiedenen Snippets besteht. Da die verwendete Syntax erweiterbar und austauschbar gestaltet werden soll, ist jedes Snippet durch eine eigene `SnipptBuilder` Instanz gekapselt. Diese hat keinerlei Informationen darüber, welchem Template es angehört, oder welche Anfrage tatsächlich gebaut wird. Da ein Snippet in mehreren Templates verwendet werden kann, gibt es einen `SnippetBuilder` Cache, der aus einer `SnippetEntity` ein Objekt erzeugt, welches zur Laufzeit nur noch Werte übergeben bekommen muss, um ein Teil eines Templates zu übersetzen. Somit hängt die tatsächliche Erzeugung des Strings von der Implementierung der `build`-Methode des `SnippetBuilders` ab. In unserem Fall haben wir die sogenannte Mustache Syntax gewählt. Bei der Mustache Syntax wird eine zu interpolierende Variable von zwei geschweiften Klammern umgeben. Beispiel Listing 8.26 zeigt direkt in der ersten Zeile die Verwendung der Variable `outName`. Damit wird signalisiert, dass dieser Teil der Vorlage durch den eigentlichen Wert ersetzt werden soll. Wir verwenden für die Umsetzung die gleichnamige Bibliothek Mustache [202], welche schon genau die Funktionalitäten mitbringt, die wir benötigen. Gleichzeitig ist das Klassendesign so, dass man theoretisch auch andere Bibliotheken verwenden könnte, ohne die restliche Implementierung anfassen zu müssen. Zusätzlich bietet Mustache, wie viele andere Template-Bibliotheken weitere Features, über das reine Ersetzen von Werten hinaus.

Man kann z. B. Schleifen laufen lassen, wenn man Arrays von Werten einsetzen möchte, oder durch Java-Reflections sehr komplexe Objekte erstellen. Zusammengefasst ist der `SnippetBuilder` ein Adapter, dem man ein Objekt mit Werten (wie oben beschrieben) gibt, die auch alle vollständig sein müssen, da sonst eine Exception geworfen wird. Es gibt also keinerlei Mechanismen, an dieser Stelle Schlüsselwerte zu ermitteln.

```
{{outName}} = KeyValueToTuple( { schema = [
        ['id', 'String'], ['created_time', 'String'], ['message', 'String'] ],
        type = 'message', keepinput=false },
        ACCESS( {
        source = '',
        wrapper = 'GenericPull',
        transport = 'Facebook',
        protocol = 'Facebook',
        datahandler = 'keyvalueobject',
        options = [
                [ 'page', {{pageName}} ],
                [ 'accessToken', {{accessToken}} ],
                [ 'limit', {{messageLimit}} ],
                [ 'scheduler.delay', {{schedulerDelay}} ]
        ]
} )
```

Listing 8.26: *Beispiel eines Snippets mit Mustache Syntax*

8.4.5.4 TemplateBuilder

Der `TemplateBuilder` ist im Wesentlichen zur Verwaltung von `SnippetBuildern` verantwortlich. Da wie bereits erwähnt, ein Snippet mehreren Templates angehören kann, wurde es ohne weitere Abhängigkeiten konzipiert. Der `TemplateBuilder` verwaltet also alle Instanzen von Templates, sowie den dazugehörigen `SnippetBuildern`. In einem Cache, der bei Änderung eines Templates/Snippets für die jeweiligen `SnippetBuilder` und `TemplateBuilder` invalidiert wird, werden dann die notwendigen Instanzen abgespeichert. Im Prinzip entspricht das dem FlyWeight Pattern, wo von einer konkreten Instanz nur genau eine vorhanden sein muss, da sie für alle Templates, die das gleiche Snippet referenzieren, gleich ist. Da die Reihenfolge der einzelnen Snippets festgelegt ist und über eine Nummer für jedes zugewiesene Element sortiert wird, manipuliert man nur die Ausführungsreihenfolge der einzelnen `SnippetBuilder`. Wenn eine Anfrage gebaut werden soll, wird als erstes das verwendete Template ermittelt. Sollte dieses nicht im Cache vorliegen, wird der `TemplateBuilder` angewiesen, alle notwendigen Snippets zu sammeln, ggf. die einzelnen `SnippetBuilder` zu erzeugen und die jeweiligen Mustache-Templates zu kompilieren. Anschließend wird die Instanz des Builders abgespeichert. Für den Fall, dass die Anfrage aus einem Grund fehlerhaft war oder Daten nachgeliefert werden, muss das Template oder ein anderer Bestandteil der Vorlage (z. B. ein Snippet) nicht erneut ermittelt werden. Für das häufige Verwenden von gleichen Templates und Snippets wird somit der benötige Arbeitsspeicher sehr gering gehalten, da viel wiederverwendet wird. Der `TemplateBuilder` läuft dann in der gegebenen Reihenfolge durch alle Snippets, ermittelt welche Werte gefordert werden, übergibt diese Werte und fügt die Ergebnisse zusammen. Sollte während der Verarbeitung keine `MissingKeyException` geworfen werden, war das eigentliche Bauen der Anfrage, zumindest aus syntaktischer Sicht, erfolgreich.

8.5 Frontend Beschreibung

8.5.1 Frontend Konzept

Das Frontend dient als einzige Benutzerschnittstelle zu dem gesamten E-Stream System. Dort kann der Benutzer sich die Daten visualisiert anzeigen lassen sowie Einstellungen im System vornehmen. Im folgenden Konzept wird die geplante Menüstruktur schematisch dargestellt, welche die vorgesehenen Funktionen auf der Webseite strukturiert, um ein möglichst einfaches und intuitives Bedienkonzept zu realisieren. Dafür wurde auf der Startseite ein Dashboardkonzept gewählt, sodass der Benutzer seine Ansicht vollständig individuell gestalten und konfigurieren kann.

8.5.1.1 Dashboard

Nach dem erfolgreichen Einloggen in das System, startet die Bedienung mit einem benutzerdefinierten Dashboard. Hier hat der Benutzer die Möglichkeit die Inhalte auf dem Dashboard selbst zu definieren. Es können mehrere verschiedene Visualisierungen und welche Daten auf der jeweiligen Darstellung angezeigt werden sollen ausgewählt werden.
Des Weiteren kann der Benutzer nicht nur ein Dashboard konfigurieren, sondern er kann verschiedene individuelle Dashboards anlegen. Damit kann der Benutzer für verschiedene Anwendungsszenarien verschiedene Dashboards erstellen, um schnell zwischen den verschiedenen Ansichten umschalten zu können. In der Abbildung 8.19 wird eine schematische Ansicht der Dashboard-Startseite dargestellt. Auf der linken Seite des Dashboards befindet sich eine Sidebar, über die kann der Benutzer das gewünschte Dashboard anzeigen, neuerstellen oder konfigurieren.

Abbildung 8.19: *Konzept der Dashbboardansicht*

Dashboard Kacheln

Wie zuvor bereits erwähnt soll es verschiedene vom Nutzer konfigurierbare Dashboards geben. In der Abbildung 8.20 wird veranschaulicht, dass es im Dashboard die Möglichkeit geben soll über beispielsweise ein „Plus“ oder „Hinzufügen“ ein Visualisierungselement in einer Kacheldarstellung

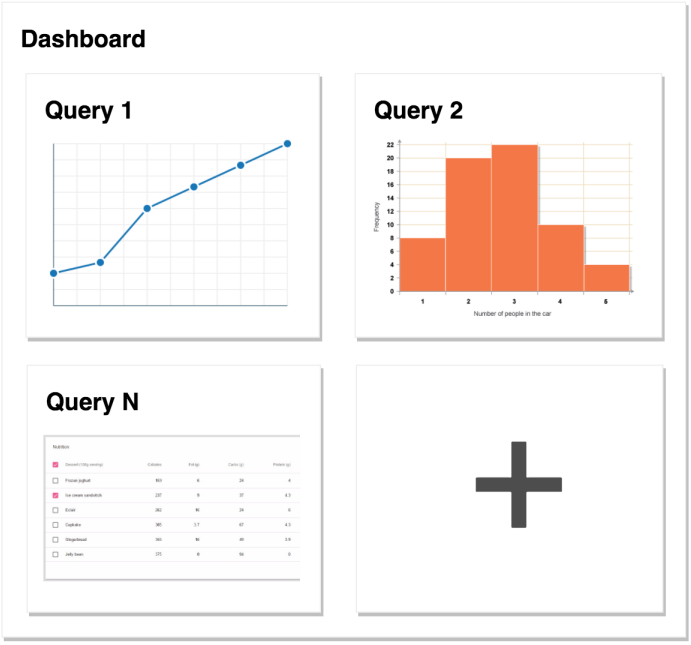


Abbildung 8.20: *Dashboard Kacheln*

hinzuzufügen. An dieser Stelle soll eine Auswahl möglicher Visualisierungen anhand laufender Queries erhalten werden können.

8.5.1.2 Queries Ansicht

Auf der Queries Ansicht in Abbildung 8.21 kann der Benutzer sich unter *Running* alle im E-Stream System laufenden Anfragen anzeigen lassen. Damit kann der Anwender zum einen sehen, welche Anfragen er eventuell ins Dashboard übernehmen möchte und zum anderen kann er so überprüfen, ob eine von ihm benötigte Anfrage schon läuft oder noch gestartet werden muss. Falls eine Anfrage noch gestartet werden soll, kann er auf der Sidebar den nächsten Menüpunkt *Templates* auswählen. In diesem Untermenü kann der Anwender aus vordefinierten Anfragetemplates auswählen, die letzten gewünschten Parameter der Anfrage konfigurieren und diese daraufhin starten. Die fehlenden Parameter, die hier der Anwender noch auswählen kann, sind zum Beispiel eine vordefinierte Datenquelle oder das gewünschte Zeitfenster.

Das Konzept für das Managen von Queries teilt sich im Grunde in zwei verschiedene Bereiche auf:

- Aus Benutzersicht: Das Auswählen verfügbarer Vorlagen und das Starten sowie visualisieren (User View)
- Aus Administratorsicht: Das Anlegen von Snippets und das Verbinden von Snippets zu Templates sowie die dazugehörige Key Verwaltung (Template Edit View)

Im Folgenden werden beide Bereiche näher erläutert. Einige Komponenten werden entsprechend genauer beschrieben, da sie in vielen Teilen der Views wiederverwendet werden sollen und entsprechend nur einmal implementiert werden müssen, sodass sie allen folgenden Szenarien entsprechen.

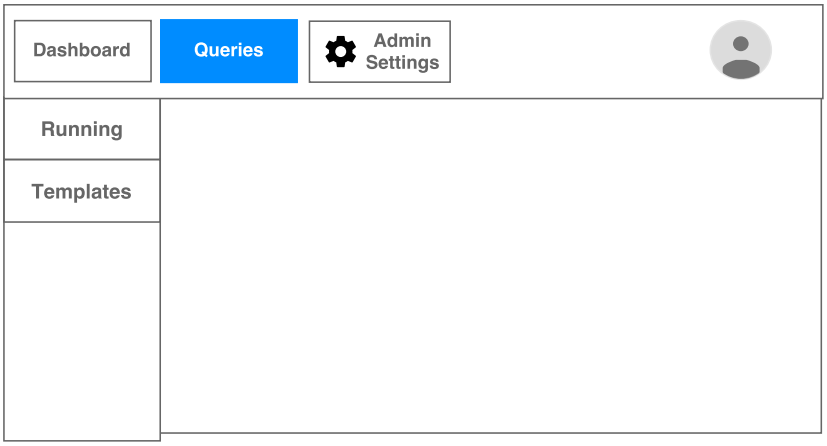


Abbildung 8.21: *Konzept der Query Anzeige*

User View

Liste laufender Queries / Erstellbarer Queries Es soll eine Komponente geben, die Listen anzeigen kann. Zum einen, damit alle vorhandenen Templates sichtbar sind, die ausgeführt werden können (im Prinzip alles, was als Data-Mining über das System E-Stream bereitgestellt wird) und zum anderen eine Liste der Queries, welche momentan im System (oder vom Benutzer) am Laufen sind, siehe Abbildung 8.22.

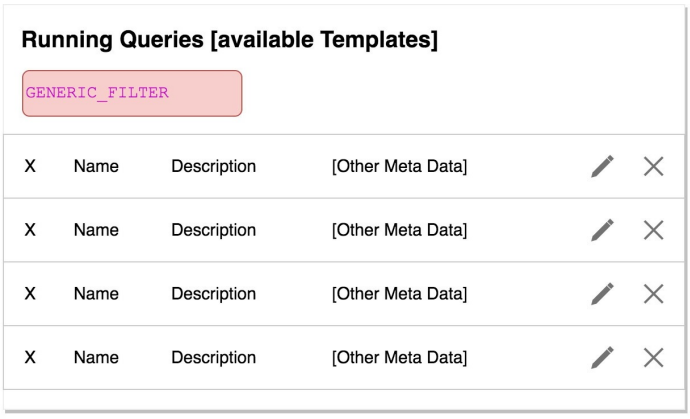


Abbildung 8.22: *Liste laufender Queries*

Starten einer Query Wenn ein Template ausgewählt wurde und dieses gestartet werden soll, werden in der Regel verschiedene Werte von Variablen benötigt. Einige dieser Keys können Standardwerte haben (müssen also nicht zwangsläufig überschrieben werden) andere müssen eventuell auf jeden Fall ausgefüllt werden. Die View, die das Starten eines Templates ermöglichen soll, sollte alle benötigten Werte abfragen (zum Beispiel Zeitraum der Anfrage etc.), siehe Abbildung 8.23.

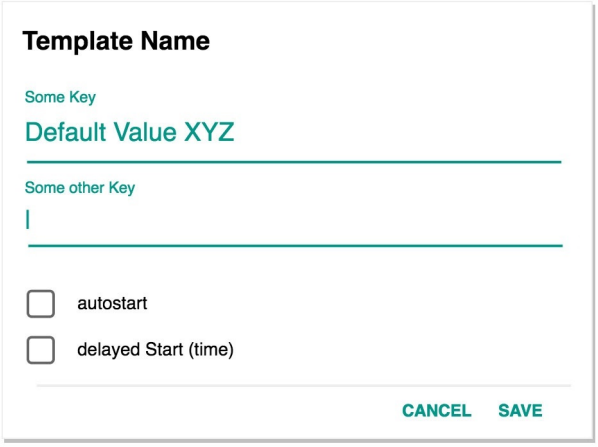


Abbildung 8.23: *Starten einer Query*

Status einer Query Aus der Übersicht der laufenden Queries heraus soll durch einen Klick eine solche Detailansicht angezeigt werden. In dieser soll neben einigen Meta-Informationen und eventuell Logs (abhängig von der Rolle des Nutzers), auch eine Anzahl verschiedener Visualisierungsmöglichkeiten zur Verfügung gestellt werden. Durch Klicken auf eine der Visualisierungen soll der entsprechende Graph angezeigt werden. Alternativ kann außerdem ein „Plus“-Button hinzugefügt werden, durch den eine Anheftung der Visualisierung an das Dashboard ermöglicht wird. Darüber hinaus könnten zusätzlich Einstellungen getätigt und die Query beendet werden.

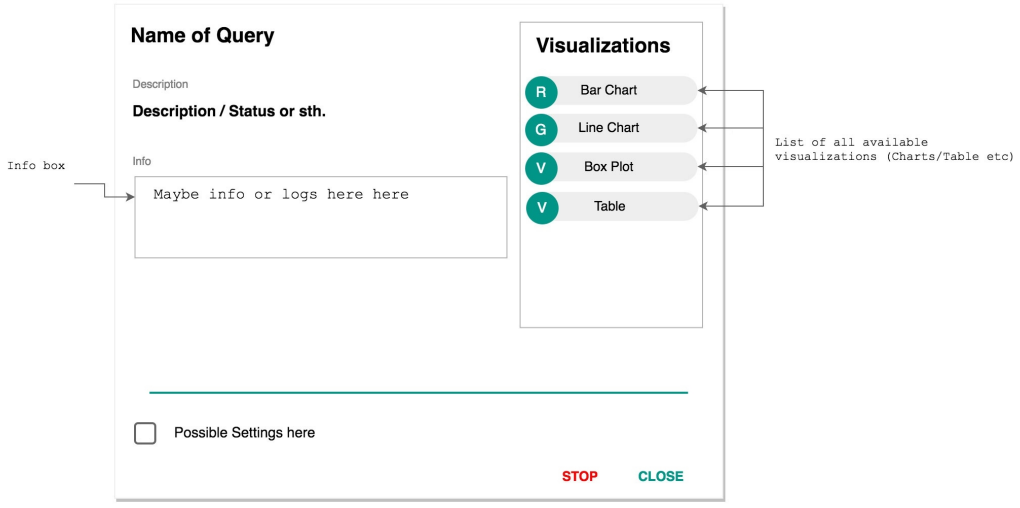


Abbildung 8.24: *Status einer Query*

Template Edit View

Sidebar Layout Bezüglich dieser Aufteilung könnte nachfolgende Abbildung 8.25 in zwei Gruppen unterteilt werden, „Snippets“ und „Templates“, so dass jeweils die Snippets mit dazugehörigem Key in einer Gruppierung stehen und alle zum Template gehörigen Views dort angesiedelt werden können.
Damit ohne erst in die View gehen zu müssen direkt bestimmte Funktionen ausgeführt werden können, könnte die „Add“-Funktion von überall aus direkt aufrufbar sein. Denkbar wären zusätzliche Funktionen, welche sich möglicherweise zur Entwicklungs-/Benutzungszeit ergeben werden.

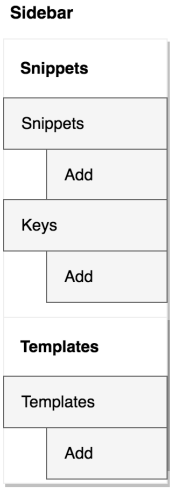


Abbildung 8.25: *Sidebar*

Übersicht von Templates/Snippets für die Bearbeitung Wie bereits zuvor in Sidebar Layout beschrieben, soll es eine Auflistung aller vorhandenen Snippets und Templates geben, so dass gezielt mit Hilfe der generischen Filterkomponente nach bestimmten Elementen gesucht werden kann (siehe Abbildung 8.26. Von hier aus soll dann direkt in den Editiermodus der jeweiligen Komponente übergeleitet werden. Diese Komponente soll wiederverwendbar sein.

Bearbeitung von Snippets Wenn zum Beispiel über den „Bearbeiten“-Button der oben genannten Liste ein Snippet ausgewählt wird, öffnet sich eine neue View, welche die vollständigen Informationen des jeweiligen Snippets anzeigt und bearbeitbar macht. Neben dem Namen und der Beschreibung (zusätzlich sollte noch ein Dropdown für den Typ des Snippets eingerichtet werden), wird in der Mitte ein Code-Editor mit Syntax-Highlighting angezeigt.
Die Snippets können Variablen mit Hilfe der Mustache-Syntax (aka. Handlebars oder Double Curly Braces) beinhalten. Alle gefundenen Variablen werden mit den im Server bereits vorhandenen Variablen verglichen (solche REST Calls sind leicht zu implementieren, weil sie nur alle Werte im TemplateKeyRepository abfragen müssen). Wird der Name bereits verwendet, soll rechts direkt die Referenz angezeigt werden, andernfalls handelt es sich um eine neue Variable und der Benutzer kann das Snippet erst speichern, wenn er den Key gespeichert hat (in der Abbildung nicht vorhanden, jedoch

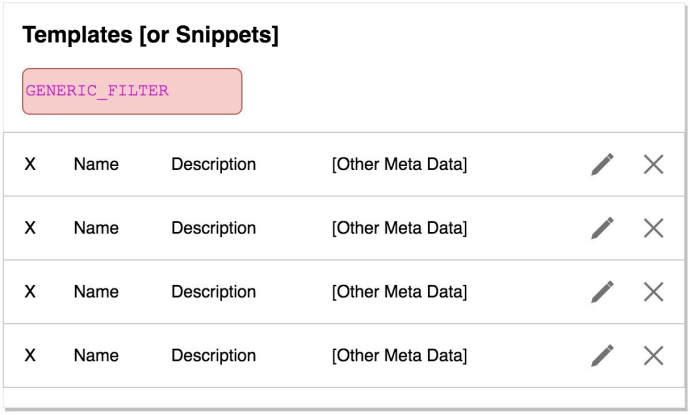


Abbildung 8.26: *Templates/Snippets Übersicht*

steht da statt dem Edit Button ein „+“, der folgende Punkt „Bearbeitung von Snippet Keys“ klärt weitere Details). Hilfreich wäre hier auch eine Auflistung bereits definierter Keys (beispielsweise über einen „Plus“-Button über dem Code Editor, welcher eine Liste anzeigt).
Soll ein neuer Key erstellt werden folgt die Darstellung der Bearbeitung der Keys, welche im nächsten Punkt zu sehen ist. Die Rücknavigation soll dann über Breadcrumbs/automatische Rückleitung zum Editieren des Snippets nach dem Speichern des Keys bewerkstelligt werden.
Snippets können gegebenenfalls noch weitere Einstellungen haben, diese sollen unterhalb der Linie als klare Abgrenzung zum oberen Inhalt stehen, damit sie nicht übersehen werden. Das Speichern oder Abbrechen der Bearbeitung kann dann über die unten rechts platzierten Buttons geschehen.

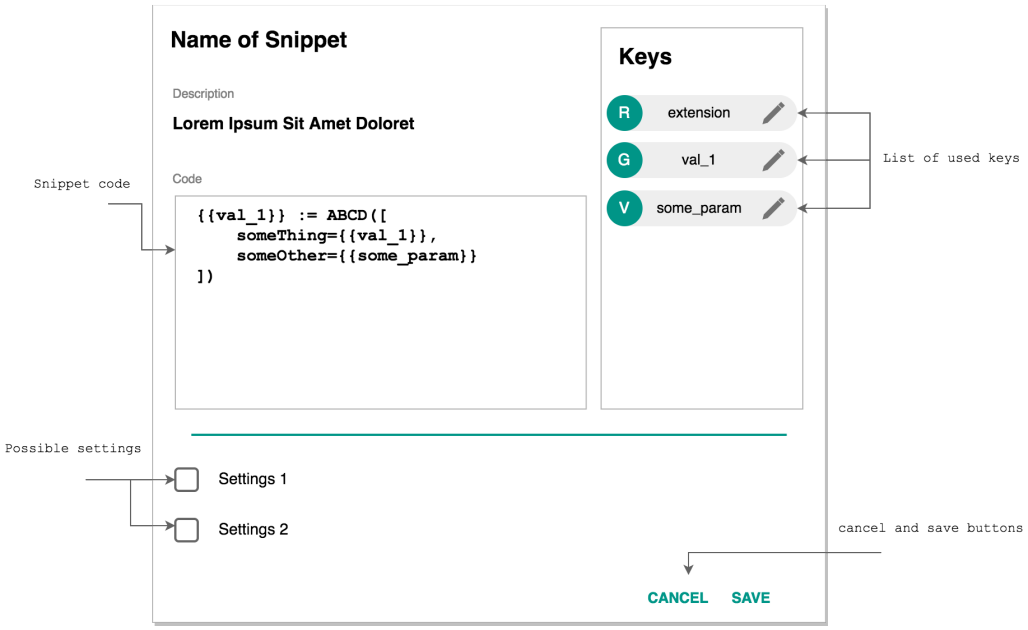


Abbildung 8.27: *Bearbeiten von Snippets*

Bearbeitung von Snippet Keys Wie bereits zuvor erwähnt, können Keys nicht nur über die gegebene Auflistung, sondern auch aus dem Snippet-Editor heraus verwendet werden, siehe Abbildung 8.28. Für die Darstellung eines Keys sollte nach Möglichkeit ein modales Fenster verwendet werden, da nicht sehr viel Platz benötigt wird alle notwendigen Einstellungen anzuzeigen bzw. editierbar zu machen.

In der Editieransicht, in der Abbildung 8.28 rechts zu sehen, werden dann Parameter wie beispielsweise Name, Beschreibung, Standardwert, Autogeneriert und Referenz (und dazugehörig Referenzschlüssel) zur Bearbeitung angezeigt.

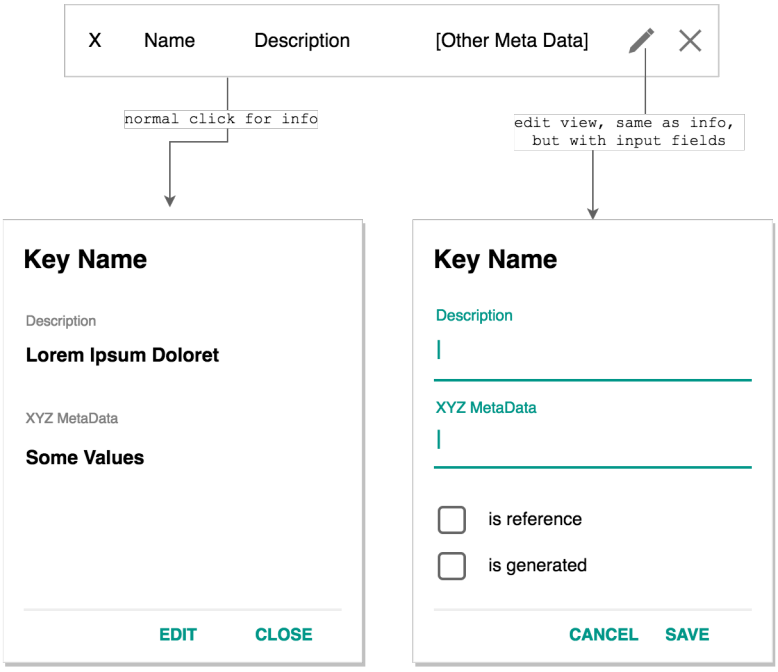


Abbildung 8.28: *Bearbeiten von Snippet Keys*

Bearbeitung von Templates Da Templates aus einer Anhäufung von Snippets bestehen, müssen diese miteinander verbunden werden, siehe Abbildung 8.29. Snippets hingegen können sogenannte autogenerierte Werte enthalten (Variablen, die zur Laufzeit vom Server generiert werden, dessen Werte für die Verwendung nicht direkt relevant sind und die der Benutzer auch nicht kennen muss). Um wissen zu können, welche generierten Werte von welchem Snippet an welches Snippet weitergereicht werden sollen, wird eine Liste von Snippets angezeigt, in der die Namen der generierten Namen beinhaltet sind. Der Benutzer kann dann die einzelnen Ein-/Ausgänge mit Hilfe von Pfeilen via Drag&Drop miteinander verbinden. Alle Verbindungen werden daraufhin angezeigt. Snippets, die keine Eingangswerte für generierte Variablen haben, sollen farblich hervorgehoben werden. Damit ist eindeutig, dass dieser Wert an der gegebenen Stelle automatisch generiert wird, ohne von einem anderen Snippet zu kommen (dies kann jedoch durchaus gewollt sein).

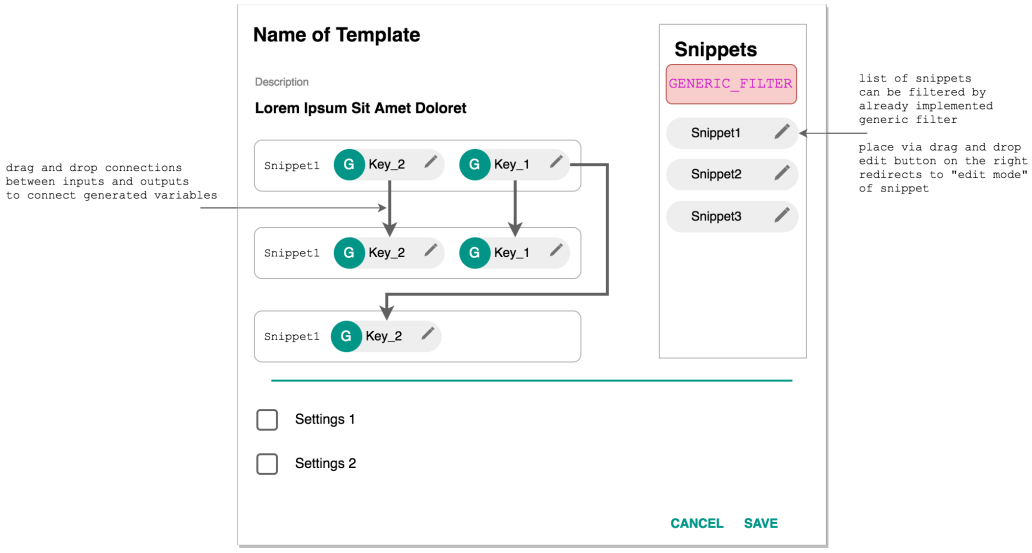


Abbildung 8.29: *Bearbeiten von Templates*

8.5.1.3 Administratoren Menü

Das Administratoren Menü wird nur Benutzern angezeigt, die auch über Administratorenrechte verfügen. Das Sidebarmenü des Administrators in Abbildung 8.30 besteht aus drei Oberpunkten: Die Benutzerverwaltung, der Query Manager und die Rohdatenansicht.
In der Benutzerverwaltung kann der Administrator einen neuen Benutzer anlegen, einen bestehenden Benutzer verändern oder einen Benutzer löschen. Beim Ändern eines Benutzers besteht die Möglichkeit ihm Administratorenrechte zu geben oder wieder zu entziehen.
Im Query Manager verfügt der Administrator über die Möglichkeit Anfragetemplates zu erstellen. Dies hat den Zweck, dass der Standardbenutzer möglichst wenig technisches Hintergrundwissen über das Anfragekonzept des E-Stream Systems benötigt und bei der Verwendung des E-Stream Systems direkt vordefinierte Anfragen verwenden kann. Templates bestehen aus mehreren Snippets, die zuvor im Menüpunkt *Snippets* erstellt werden können.
Als letzten Unterpunkt *RawData* kann sich der Administrator zur weiteren Information oder zur Fehlerdiagnose die Rohdaten des Systems ansehen.

8.5.1.4 Benutzer Menü

Der Benutzer benötigt kein vollwertiges Menü, wie die zuvor dargestellten Seiten. Denn das E-Stream System ist durch das Dashboard Konzept sehr benutzerindividuell und braucht keine weiteren Konfigurationsmöglichkeiten für den Anwender. Daher ist hier ein einfaches Dropdown Menü vorgesehen, bei dem der Benutzer die Benutzeroberflächensprache ändern kann und sich aus dem E-Stream System ausloggen kann.

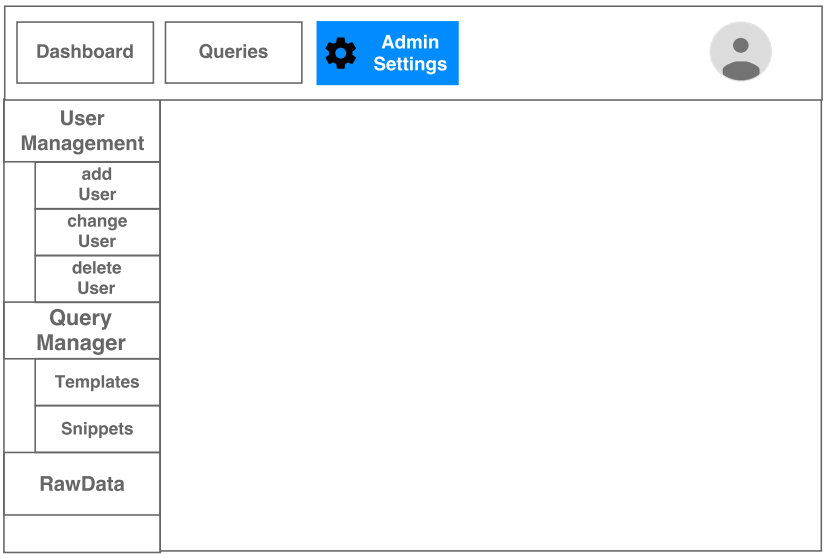


Abbildung 8.30: *Konzept der Administratorenansicht*

Abbildung 8.31: *Konzept des Benutzer Dropdown Menüs*

8.5.2 Anzeige der Rohdaten

Eine Grundfunktion, die eine datengetriebene Software haben sollte, welche nicht nur für einfache Endanwender gedacht ist, ist das Darstellen von Rohdaten. Mit Rohdaten sind die Daten gemeint, die dieses System von außerhalb empfängt. Denn mithilfe des Anzeigens von unverarbeiteten Daten, kann geschultes Personal Übertragungsfehler detektieren und Störungen besser analysieren. Dies wird dann notwendig, wenn das System bei der Verarbeitung von Daten keine sinnvollen Ergebnisse mehr liefert. In diesem Projekt sind Rohdaten die Werte von Smart Metern, die aus der mosaik Simulation stammen oder später direkt von den Smart Metern beziehungsweise von Smart Meter Gateways kommen.

Anwendung 1 - Tabellenform

Zur Darstellung von Rohdaten bietet es sich an, dass die empfangenen Werte in eine Tabelle eingetragen werden. Dabei werden alle neuen Werte direkt an die Tabellen angehängt, um immer Zugriff auf die aktuellsten Werte zu erhalten. Jeder empfangene Datensatz, wird in einer eigenen Zeile und jedes Attribut wird in einer Spalte dargestellt.

Anwendung 2 - Graph

Da es sich bei den Rohdaten um einen Datenstrom handelt, ist es schwierig, in einer Tabelle immer die neusten Werte zu sehen beziehungsweise wahrzunehmen. Deshalb gibt es eine weitere Option diese Werte darzustellen. Mithilfe eines Graphen ist es für einen Menschen viel einfacher Zahlenbereiche und Schwankungen zu erkennen. Daher werden bei der Rohdatenanzeige durch einen Graphen, die empfangenen Werte einfach aneinandergereiht und visualisiert. Da allerdings mehrere verschiedene Daten übertragen werden, kann man das zu betrachtende Attribut auswählen oder alle Datensätze aller Attribute in einem Graphen anzeigen lassen.

8.5.3 Visualisierung der abstrakten Topologie

Mithilfe unseres Produktes, ist es möglich die Hierarchie eines Smart Grids, also die Beziehungen zwischen dem Gesamtnetzwerk, Netzwerkgebiet, Transformator/ Umspannwerk bis hin zum Smart Meter Gateway, als Topologie darzustellen. Hierbei besitzen Knotenpunkte, wie beispielsweise Transformatoren oder Smart Meter Gateways keine feste Position, wie zum Beispiel auf einer Karte, sondern werden abstrakt dargestellt, damit ein möglichst planarer Graph entsteht. Zudem ist die Navigation über eine Seitenleiste in der Topologie möglich.

```
1  // overall grid
2  {
3    "id": "1",
4    "name": "Germany",
5    //sub grids
6    "children": [
7      {
8        "id": "sg_01",
9        "name": "XY",
10       //transformators
11       "children": [
12         {
13           "id": "",
14           "type": "",
15           //smart meter gateways
16           "children": [
17             {
18               "id": "smgw_8ebaa821-3dc4-469b-8a1d-b7d6c87fabb8",
19               "coordinates": {
20                 "lat": 53.1409915,
21                 "lon": 8.2093512
22               },
23               "tags": {
24                 "addr:city": "Oldenburg",
25                 "addr:country": "DE",
26                 "addr:housenumber": "19",
```

```
27                  "addr:postcode": "26122",
28                  "addr:street": "Wallstra\u00dfe",
29                  "addr:suburb": "Innenstadt",
30                  "amenity": "pub",
31                  "contact:email": "info@fiddlersgreen-oldenburg.de",
32                  "contact:phone": "+49 441 16622",
33                  "name": "Fiddlers Green - Irish Pub",
34                  "smoking": "isolated",
35                  "url": "http://www.fiddlersgreen-oldenburg.de/",
36                  "wheelchair": "limited"
37                },
38                //smart meters
39                "children": [
40                  {
41                    "id": "sm_85c38e67-025a-49d6-8957-4ffc44fc5cbd",
42                    "name": "HouseHold",
43                    "options": {
44                      "residents": 4,
45                      "yearConsumption": 4689
46                    }
47                  }
48                ]
49              }
50            ]
51          }
52        ]
53      }
54    ]
55  }
```

Listing 8.27: *Beispiel einer Topologie*

Dieses Beispiel soll einen Eindruck darüber geben, in welchem Format Daten in unserem Client ankommen. Die Kommentare im obigen Beispiel sind selbstverständlich nicht JSON konform und werden im tatsächlichen Produkt auch nicht generiert. Sie dienen nur dem Verständnis, um welche Ebene im Smart Grid es sich bei welchem Objekt handelt. Durch einen API-Aufruf wird eine Datei vergleichbar mit 8.5.3 erhalten.

Um die Anzeige möglichst generisch zu halten, ist es möglich, verschiedene zusätzliche Informationen zu verschiedenen Knotenpunkten zu speichern. Außerdem ist es möglich, über einen so genannten Navigationsbaum die darzustellenden Knoten zu selektieren. Der im Hintergrund ausgeführte Algorithmus wird über eine Anzeige ausgewählt. Wird als Anzeige das reine Darstellen der Topologie gewählt, wird im Hintergrund ein Algorithmus ausgeführt, welcher versucht, die Knoten so positionieren, dass alle Knoten derselben Ebene in etwa die gleiche Kantenlänge besitzen und sich möglichst wenig Kanten kreuzen.

Die verschiedenen Knoten (Smart Meter Gateways, Transformatoren, etc.) besitzen unterschiedliche Darstellungen. Diese Knoten sind anklickbar, sodass diese sich markieren lassen. Da zum Zeitpunkt der Projektgruppe leider noch keine Empfehlung der Netzdaten Strom zur Visualisierung der Topologie vorlag, wurde diese komplett von uns selber entwickelt. Die Darstellung der Knoten wird in Tabelle 8.5.3 festgehalten.

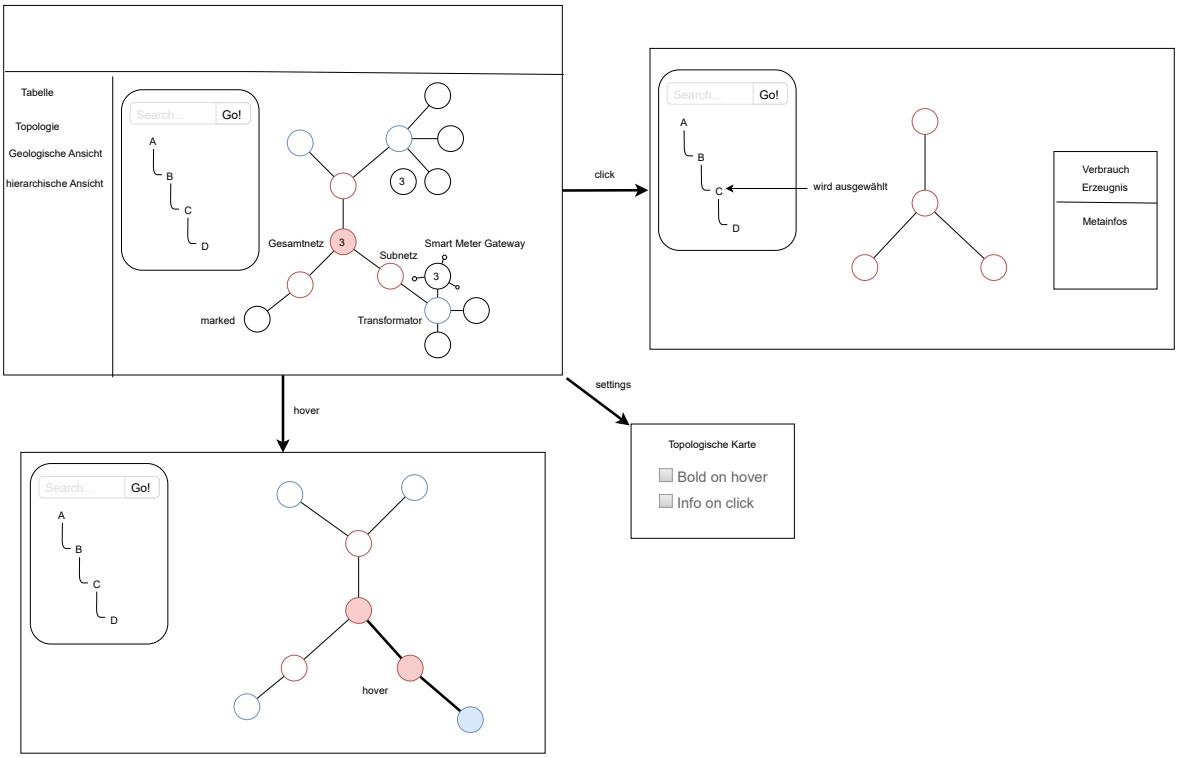


Abbildung 8.32: *Konzept der topologischen Karte*

8.5.3.1 Navigation Tree

Durch den Navigation Tree wird dem Nutzer eine einfache Möglichkeit geboten die Knoten der abstrakten Hierarchie zu filtern oder zu selektieren. Auch wird die Navigation mittels eines Navigations Tree vom Styleguide des NetzDatenStrom empfohlen[7]. Ergänzt wird die Filterung durch ein Texteingabefeld in dem durch ein String ein bestimmter Knoten gesucht werden kann, sodass nur der passende Knoten angezeigt wird.

Node Type	**Form**	**Farbe**	**Farbe Hover?**
Gesamtnetzwerk	Kreis	000e7f	heller werden
Netzwerkgebiet	Kreis	394de5	heller werden
Transformator	Kreis	7f8eff	heller werden
Smart Meter Gateway	Kreis mit Zahl für Anzahl der Children Smart Meter	a3a7cc	heller werden
Smart Meter	—	—	—

[7] vgl. `http://netzdatenstrom.imis.uni-luebeck.de/specs/organisms/topology-tree/`

8.5.4 Dynamisches Dashboards

Die Umsetzung der dynamischen Dashboards aus Unterunterabschnitt 8.5.1.1 lässt sich in verschiedene Bereiche aufteilen:

1. die Speicherung,
2. die Verwaltung mehrerer Dashboards,
3. die Konfigurierbarkeit.

Ein Überblick über die Komponenten und Services, die diese Bereiche ist in Abbildung 8.33 dargestellt. Der `DashboardStorageService` übernimmt zusammen mit der `DashboardComponent` die Speicherung und der `DashboardService` verwaltet alle Dashboards sowie stellt Operationen zur Manipulation zur Verfügung. Alle Komponenten übernehmen einen Teil der Konfigurierbarkeit der Dashboards. Die Komponenten und Services werden im Folgenden erläutert.

8.5.4.1 Speicherung

Die Speicherung der Dashboards wird automatisch übernommen sobald der Nutzer die `DashboardComponent` verlässt. Darüber hinaus gibt es auch eine Möglichkeit die Speicherung über eine entsprechende Schaltfläche durchzuführen. Der `DashboardStorageService` besitzt ein allgemeines Interface, um verschiedene Speicherungsarten zu unterstützen und ist so implementiert, dass die Dashboards für jeden Nutzer gesondert im Local-Storage des Browsers gespeichert, sodass sich mehrere Nutzer an einem Browser nicht eine Konfiguration teilen müssen.

Die Dashboardkonfiguration besitzt zudem eine Version, durch die Fehler nach Änderungen an der Datenstruktur vermieden werden sollen. Dafür können nach dem Strategy-Pattern weitere Strategien implementiert werden, welche die alte Datenstruktur in eine neue Struktur überführt. Bisher wird bei Versionsänderung die aktuelle Konfiguration verworfen und die Standardkonfiguration geladen, sodass in der Dashboardverwaltung immer mit gültigen Daten gearbeitet werden kann.

8.5.4.2 Verwaltung

Bei der Verwaltung der Dashboards liegt das Modell aus Abbildung 8.34 zugrunde. Es gibt das Aggregationsobjekt `Dashboards`, um eine Serialisierungswurzel zu besitzen und die Speicherversion zugreifbar zu machen. In der Klasse `DashboardConfiguration` befindet sich die Konfiguration für ein Dashboard. Dies besitzt einen Namen und eine Größe, welche der Menge an Elementen im Raster in einer Zeile entspricht. Der `editMode` beschreibt, ob die Graphen im Dashboard bearbeitet werden können. Außerdem enthält die Konfiguration `DashboardGraph`en.

Ein `DashboardGraph` hat eine Höhe und Breite im Raster und besitzt einerseits einen Namen (`queryName`) und andererseits einen Schlüssel, durch den das System an Daten für den Graphen gelangt. Die `GraphConfiguration` des Graphen enthält alle Informationen wie diese Daten angezeigt werden sollen. Der Schlüssel `visualizationType` wird dazu benutzt, eine generische Visualisierungskomponente zur Anzeige zu finden. Die Daten selber werden als Tupel verstanden und durch das `Column`-Feld modelliert. Für das Tupel-Schema kann dabei für jedes Element die Sichtbarkeit und Text zur Anzeige eingestellt werden, während der Standardtext dem Tupelattributsnamen

entspricht. Für Graphen gibt es zusätzlich auch die `ChartConfiguration`, mit der der x-Wert bestimmt wird. Dafür kann ein beliebiges Tupelattribut benutzt werden. In jeder Komponente wird ein Teil dieses Modells konfigurierbar gemacht.

8.5.4.3 Konfigurierbarkeit

Die hierarchische Struktur des Modells wird durch eine entsprechend hierarchische Struktur in den Dashboard-Komponenten wiedergespiegelt, wie Abbildung 8.35 zeigt. Die `DashboardComponent` setzt für die Anordnung der Graphen im Dashboard ein Drag & Drop System um und zeigt bei Bedarf alle anderen benötigten Komponenten an. In der `EditDashboardComponent` werden generelle Dashboard-Funktionen wie zum Beispiel das Speichern, Zurücksetzen, Ändern des Dashboardnamens und Hinzufügen von Komponenten zum aktuellen Dashboard zur Verfügung gestellt. In der `DashboardTileComponent` kann die Größe des Graphen angepasst werden, wenn der `editMode` aktiviert ist.

Das Laden der generischen Visualisierungskomponenten wird in der `DashboardVisualizationManagerComponent` vorgenommen, während die Anpassung der Tupel- und Graphenkonfiguration durch die `DashboardVisualizationManagerConfigurationComponent` durchgeführt wird. Diese Komponente unterstützt für die Reihenfolge der Tupelelemente ebenfalls Drag & Drop-Funktionalität, die durch ein Icon auf der linken Seite benutzt werden kann.

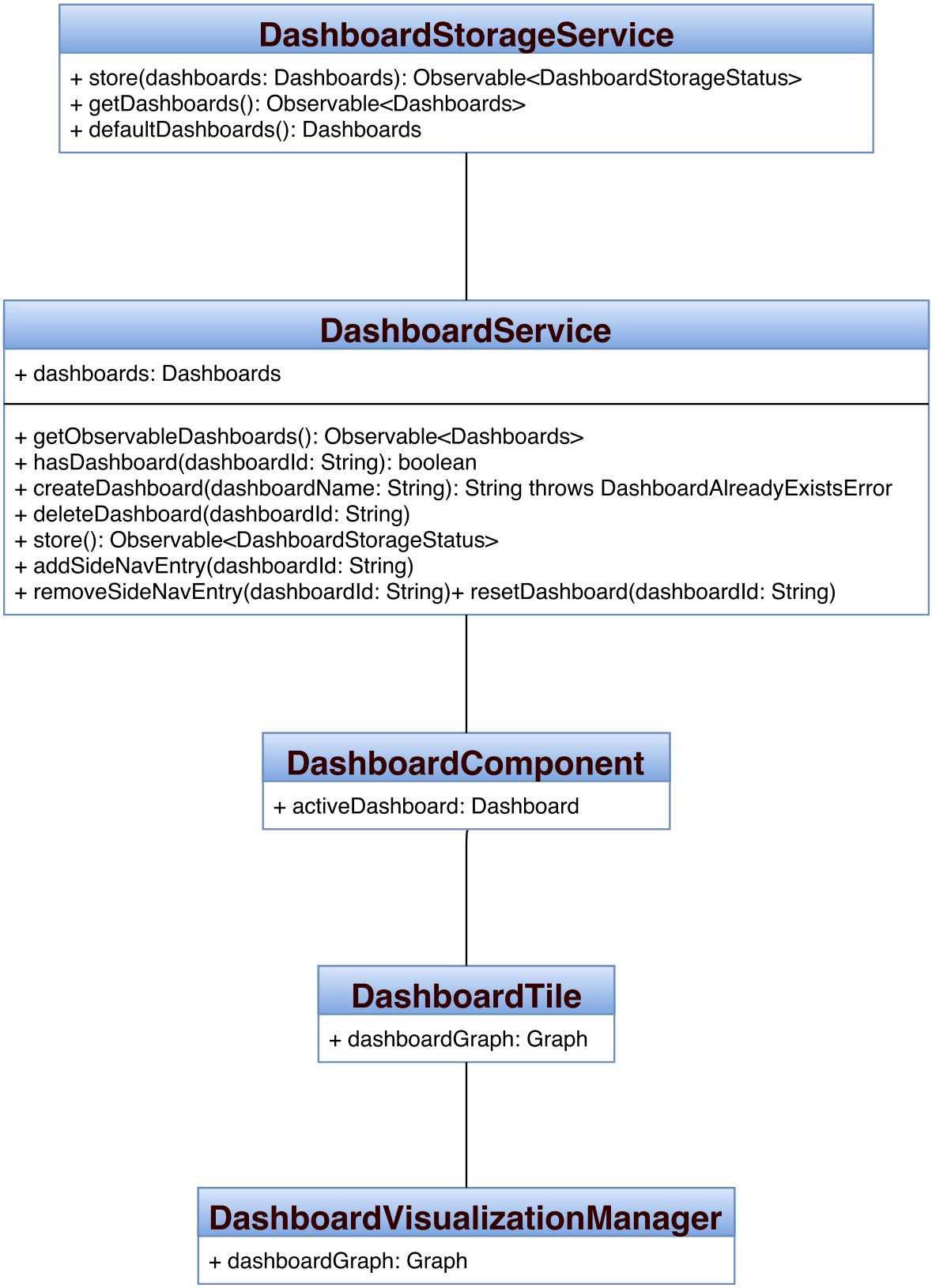


Abbildung 8.33: *Übersicht über einen Teil der Dashboard-Komponenten*

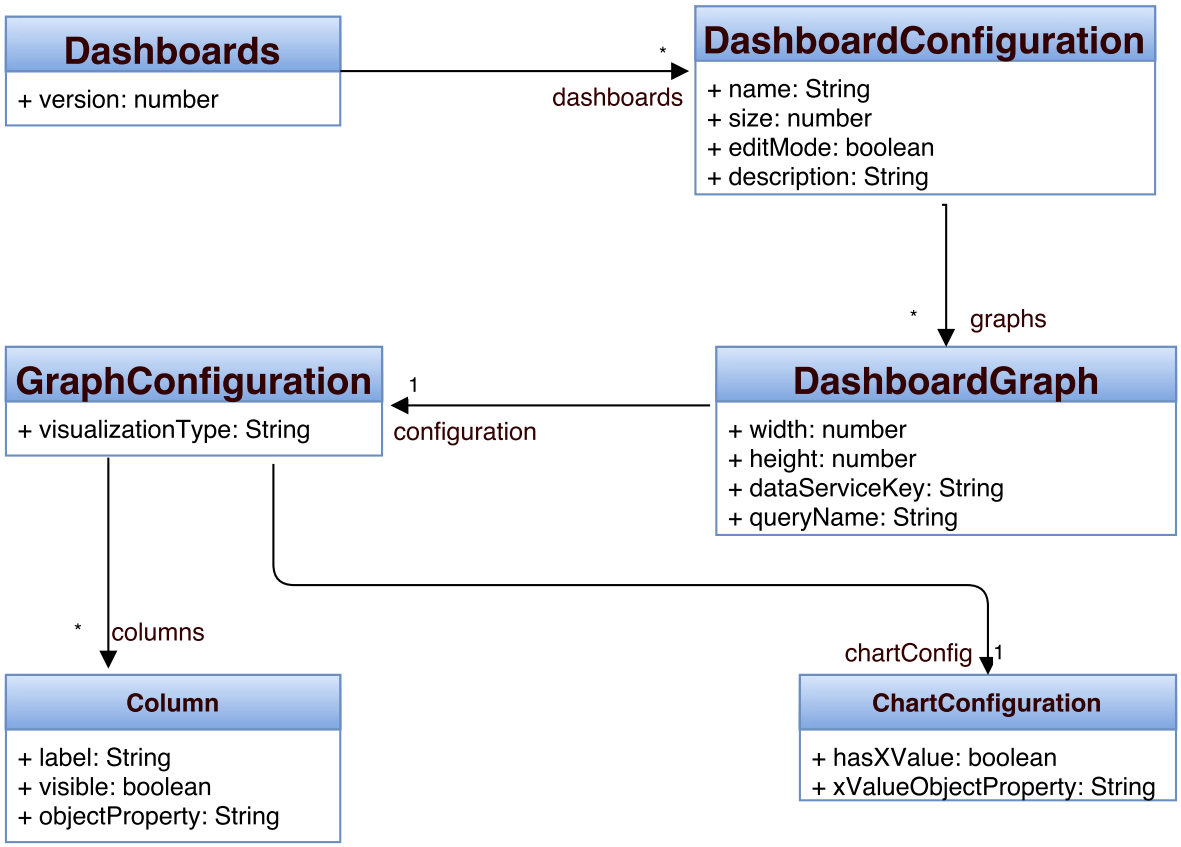


Abbildung 8.34: *Modell für die dynamischen Dashboards*

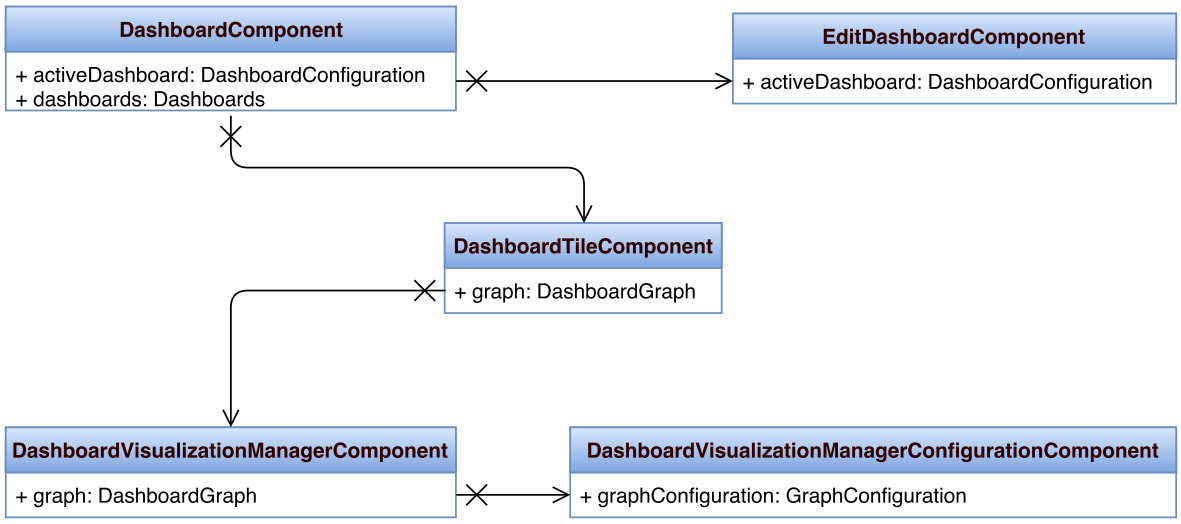


Abbildung 8.35: *Komponentenzusammenhang für die Dashboardkomponenten*

8.6 Benutzeranleitung

In diesem Abschnitt werden die Mindestanforderungen für das Verwenden des E-Stream Systems, sowie dessen Inbetriebnahme dargestellt.

8.6.1 Mindestanforderungen

Für die Inbetriebnahme des Systems müssen bestimmte Mindestanforderungen erfüllt sein. Eine der wichtigsten Mindestanforderungen für das Starten des Gesamtsystems ist der Arbeitsspeicher. Um das gesamte E-Stream System zu betreiben, sollte man über mindestens 8 GB RAM verfügen. Für eine optimale Nutzung sollte das Computersystem, auf dem E-Stream läuft, über 16 GB Arbeitsspeicher verfügen. Ein reines Starten des Systems kann zwar mit 8 GB geschehen, jedoch ist die Benutzung des Systems und das Ausführen von Mining-Anfragen mit 16 GB deutlich performanter und deshalb zu empfehlen. Als Betriebssystem wird eine Linux Variante empfohlen. Während der Entwicklung des Systems wurde Ubuntu 16.04 verwendet. Bei MacOS und Windows wurde mit Vagrant eine virtuelle Maschine mit Ubuntu 16.04 Server erstellt. In dieser virtuellen Maschine lief die benötigte Infrastruktur, während das Backend und das Frontend auf dem Host-System gebaut wurden. Die Bereitstellung der gesamten Infrastruktur, die für das System benötigt wird, geschieht mit Hilfe von Docker und Docker Compose (siehe Unterabschnitt 6.3.5). Um fehlerhaftes Verhalten zu verhindern ist es notwendig VirtualBox und Vagrant zu installieren, bevor das System gestartet werden kann. Dadurch wird gewährleistet, dass das System mitsamt der Infrastruktur genau so ausgeführt wird, wie es während der Entwicklung geschehen ist. Außerdem hat das den Vorteil, dass Docker und Docker Compose in der virtuellen Maschine vorinstalliert werden.

8.6.2 Einrichtung der Infrastruktur

Bevor das System benutzt werden kann, muss es zunächst aus dem Git-Repository ausgecheckt werden. Die Verzeichnisstruktur des E-Stream Git-Repositories hat die in Abbildung 8.36 dargestellte Struktur. In diesen Verzeichnissen befinden sich folgende Komponenten des Systems:

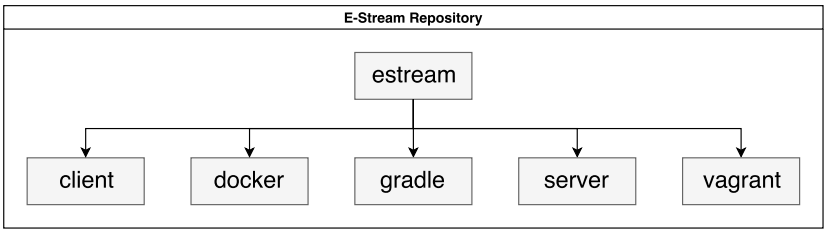


Abbildung 8.36: *Verzeichnisstruktur des E-Stream Repositories*

- client - Das Frontend des Systems
- docker - Dockerfiles und Docker Compose YAML Konfigurationsdateien, sowie ein Bash-Skript für das Starten der Infrastruktur
- gradle - Gradle Wrapper für das gesamte System

- server - Das Backend des Systems
- vagrant - Vagrantfile für das Starten einer virtuellen Maschine unter Windows oder MacOS

Bei der Benutzung eines abweichenden Betriebssystems und zur besseren Isolation der benutzten Infrastruktur von dem System wird mit Vagrant (siehe Unterabschnitt 6.3.4) eine virtuelle Maschine erstellt, auf der die notwendige Infrastruktur installiert wird. Dafür ist es notwendig in das *vagrant* Verzeichnis zu wechseln und mit `vagrant up` die virtuelle Maschine zu starten, wie in Listing 8.28 zu sehen ist.

```
1  $ ./vagrant up
2  Bringing machine 'default' up with 'virtualbox' provider...
3  ==> default: Checking if box 'bento/ubuntu-16.04' is up to date...
4  ==> default: Clearing any previously set forwarded ports...
5  ==> default: Clearing any previously set network interfaces...
6  ==> default: Preparing network interfaces based on configuration...
7      default: Adapter 1: nat
8      default: Adapter 2: hostonly
9  ==> default: Forwarding ports...
10     default: 3306 (guest) => 3306 (host) (adapter 1)
11     default: 2181 (guest) => 2181 (host) (adapter 1)
12 [...]
13 ==> default: Machine already provisioned. Run [...]
14 ==> default: flag to force provisioning. Provisioners [...]
15 $
```

Listing 8.28: *Starten der Vagrant-Box*

Nachdem die virtuelle Maschine hochgefahren ist, kann mittels `vagrant ssh` eine SSH-Verbindung mit der virtuellen Maschine aufgebaut werden. Anschließend kann, wie weiter unten in Listing 8.30 dargestellt, die Infrastruktur für das System mit der Ausführung des Bash-Skripts bereitgestellt werden. Hierbei ist zu beachten, dass die hier benutzte Vagrant Box den Benutzernamen *vagrant* und das Passwort *vagrant* besitzt.

```
1  $ ./vagrant ssh
2  vagrant@127.0.0.1's password: vagrant
3  Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-87-generic x86_64)
4
5   * Documentation:  https://help.ubuntu.com
6   * Management:     https://landscape.canonical.com
7   * Support:        https://ubuntu.com/advantage
8
9  96 packages can be updated.
10 40 updates are security updates.
11
12
13 Last login: Thu Mar 29 08:46:03 2018 from 10.0.2.2
14 vagrant@vagrant:~$ ./startup_infrastructure.sh start
```

Listing 8.29: *Verbindung mittels SSH und Bereitstellung aller Docker Images und Container*

Um die Infrastruktur für das System zu starten, befindet sich in dem *docker* Verzeichnis das Bash-Skript *startup_infrastructure.sh*. Dieses wird durch die gemeinsame Nutzung des Verzeichnisses zwischen dem Host-System und der virtuellen Maschine in das Home-Verzeichnis des vagrant Nutzers in die virtuelle Maschine kopiert.

```
1  vagrant@vagrant:~$ ./startup_infrastructure.sh
2  Missing arguments!
3  Usage: ./startup_infrastructure [Option] [Profile]
4
5  Options:
6          clean    Remove the simulation repositories from the docker [...]
7          start    Install and starts up the complete infrastructure [...]
8          startm   Starts up the infrastructure without mosaik [...]
9          stop     Stops every running Docker Container
10
11 Profile:
12         dev      Start infrastructure in dev mode for local development. [...]
13         prod     Start infrastructure in prod mode for running [...]
```

Listing 8.30: *Ausführung startup_infrastruce.sh ohne Parameter*

Mit der Ausführung von *startup_infrastructure start* werden Docker Container für folgende Software gebaut und gestartet:

- Zookeeper
- Kafka Broker
- Odysseus E-Stream Server
- LV-Simulation (mosaik)
- MariaDB

Da in diesem Schritt ebenfalls die von der Projektgruppe angepasste LV-Simulation gebaut wird, wird diese, sofern sie noch nicht im *docker* Verzeichnis existiert, aus dem Git Repository ausgecheckt und in ein Docker Image kopiert, das anschließend in einem Container in Dauerschleife ausgeführt wird. Sobald alle Docker Images gebaut sind, werden die Container gestartet. Hier ist es wichtig zu beachten, dass die Images nicht im *detached* Modus gestartet werden. Das heißt auch, dass die Ausgabe aller Container zusammen in dem Terminal ausgegeben wird. Deshalb sollte das Terminal-Fenster nicht geschlossen werden, solange man mit dem E-Stream System arbeitet. Erst durch das Drücken von *Strg+c* werden die Container gestoppt.

Es ist auch möglich, Änderungen bei dem Starten der Infrastruktur vorzunehmen, indem zum Beispiel die LV-Simulation ausgelassen wird. Das kann durch das Starten des Skripts mit dem Parameter `startm` geschehen. Weiterhin ist es möglich, die Infrastruktur ebenfalls in einem *Production* Profil zu starten. Dafür wird eine zweite Docker Compose Datei benutzt, die die Standardwerte überschreibt. Somit ist es möglich, die gleiche Infrastruktur für zwei unterschiedliche Umgebungen zu benutzen. Das starten in dem Production Profil geschieht über Übergabe eines zweiten Parameters, nämlich `prod`, bei der Ausführung des Skripts. Hierbei ist zu beachten, dass dafür die IP-Adressen der Infrastruktur in den Konfigurationsdateien für das System angepasst werden müssen.

8.6.3 Starten des Systems

Nachdem die Infrastruktur hochgefahren ist, ist es möglich das E-Stream System zu starten. Dafür muss außerhalb der virtuellen Maschine in das *server* Verzeichnis gewechselt werden. In diesem Verzeichnis befindet sich ein Gradle-Wrapper mit dem Dateinamen *gradlew*. Das System hat den Gradle-Task

fullBuild, der sowohl das Frontend als auch das Backend mit einem Befehl bauen kann. Dafür muss dieser Gradle-Wrapper mit *fullBuild* als Parameter ausgeführt werden, wie es in Listing 8.31 zu sehen ist.

```
$ ./gradlew fullBuild
:client:nodeSetup UP-TO-DATE
:client:npmSetup SKIPPED
:client:npmInstall
[...]
:test
:check
:build
:fullBuild

BUILD SUCCESSFUL in 3m 45s
14 actionable tasks: 11 executed, 3 up-to-date
```

Listing 8.31: *Ausführung des Gradle-Tasks fullBuild im server Verzeichnis*

Nachdem das System vollständig kompiliert ist, befindet sich die *server-0.1.jar*, mit der das System gestartet werden kann, in dem neu erstellten Verzeichnis *build/libs/*. Wie in Listing 8.32 zu sehen ist, kann das gesamte System über das Starten dieser JAR-Datei ausgeführt werden.

```
$ java -jar build/libs/server-0.1.jar

  _______  _____ _                                      _____
 |  _____|/ _____| |                                    |  __ \
 | |__   | (___ | |_ _ __ ___  __ _ _ __ ___ _______| |  | | _____   __
 |  __|  \___ \| __| '__/ _ \/ _` | '_ ` _ \_______| |  | |/ _ \ \ / /
 | |____ _____) | |_| | |  __/ (_| | | | | | |     | |__| |  __/\ V /
 |_______|_____/ \__|_|  \___|\__,_|_| |_| |_|     |_____/ \___| \_/
:: EStream v0.1 :: Running Spring Boot v1.5.6.RELEASE ::
```

Listing 8.32: *Starten des Systems*

Nachdem das System gestartet ist, kann das Frontend des Systems über *http://localhost:8080/* erreicht werden und das System ist einsatzbereit, wie in Abbildung 8.37 zu sehen ist.

8.6.4 Bedienung des Systems

Ist das gesamte System gestartet, kann ein Nutzer sich über seine Zugangsdaten in dem System anmelden und ein neues Dashboard anlegen. Dieses ist in Abbildung 8.38 dargestellt. Damit in dem Dashboard Visualisierungen angezeigt werden, muss zunächst eine Query ausgesucht und gestartet werden, die die gewünschten Ergebnisse ausgibt. Dafür muss in dem oberen Bereich des Dashboards die Queries Ansicht ausgewählt werden. In dieser Ansicht können in dem *Query Templates* Abschnitt die gewünschten Queries über das Upload Symbol gestartet werden (siehe Abbildung 8.39). Beim Klicken auf das Symbol wird zunächst die Query Configuration Ansicht geöffnet, in der die Parameter der Query angepasst werden können. Zur sofortigen Ausführung der Query sollte die Checkbox mit Autostart ausgewählt werden. Das ist an dem Beispiel einer Query für eine Aggregation auf dem

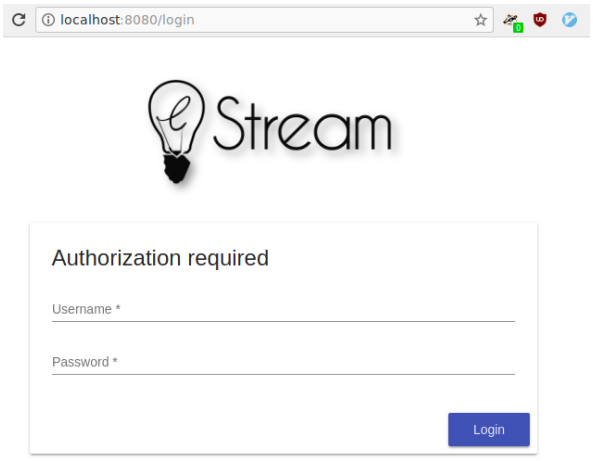


Abbildung 8.37: *Login-Seite des laufenden Systems*

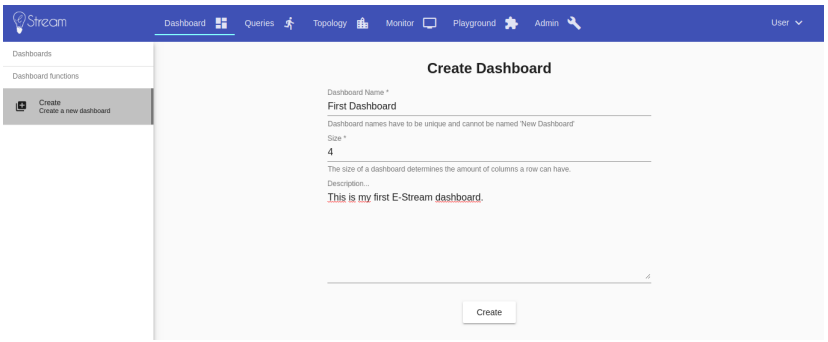


Abbildung 8.38: *Erstellen des ersten Dashbaords*

Abbildung 8.39: *Vordefinierte Queries*

Datenstrom in Abbildung 8.40 zu sehen. Beim Klicken auf Start wird die Query gestartet und ist in dem Abschnitt Running Queries zu sehen, was in Abbildung 8.41 dargestellt ist. Anschließend kann über die Auswahl der Dashboard Ansicht die Visualisierung für diese Query einem Dashboard hinzugefügt werden. Dafür muss das erstellte Dashboard ausgewählt und der Edit Button gedrückt werden. Anschließend ist das Bearbeitungsmenü des Dashboards zu sehen und es kann mit einem Klick auf das + Symbol eine neue Visualisierung zum Dashboard hinzugefügt werden. Das Menü für das Bearbeiten eines Dashboards ist in Abbildung 8.42 zu sehen. Ist das geschehen, kann die

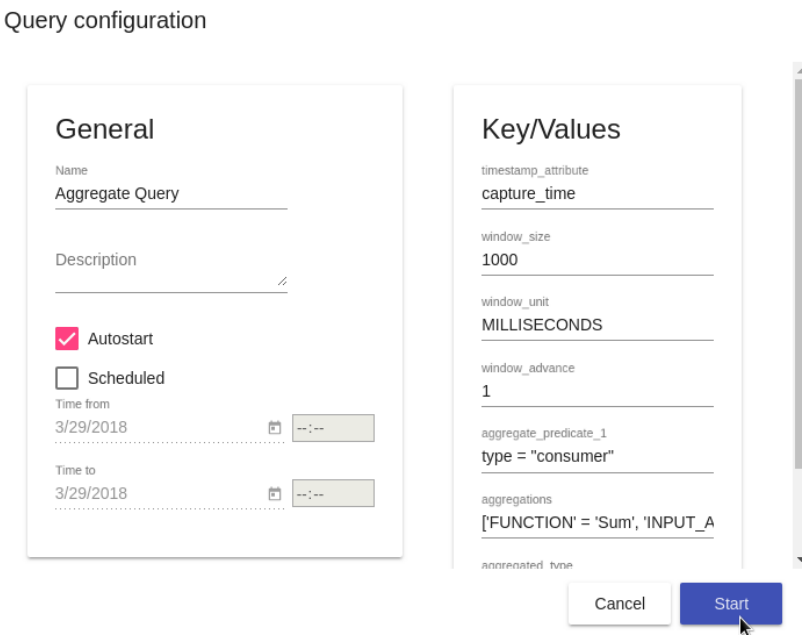


Abbildung 8.40: *Konfiguration und Starten einer Query*

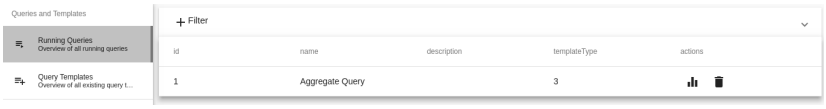


Abbildung 8.41: *Alle durch den Benutzer gestarteten Queries*

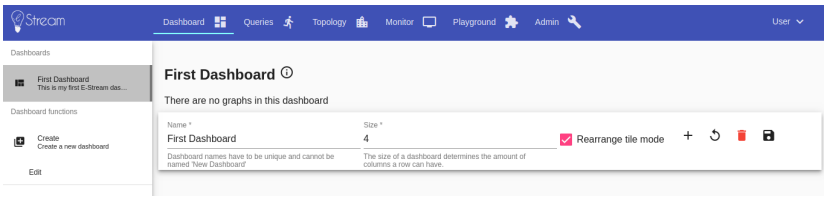


Abbildung 8.42: *Bearbeitungsmenü des erstellten Dashboards*

laufende Query ausgesucht werden und durch die Art der Visualisierung für die Darstellung der Ergebnisse dieser Query ausgewählt werden (siehe Abbildung 8.43). Abschließend kann die Größe der Visualisierung verändert werden, wenn die Option *Rearrange tile mode* ausgewählt ist, so dass das Dashboard ähnlich zu dem in Abbildung 8.44 ist. Somit hat der Benutzer sein erstes Dashboard konfiguriert.

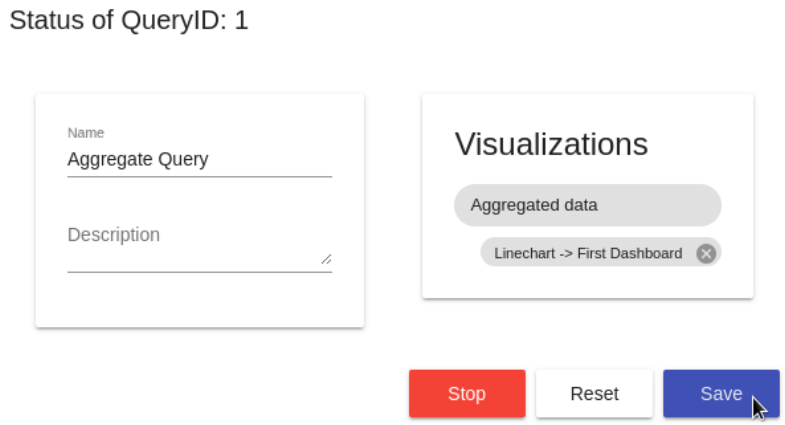


Abbildung 8.43: *Auswahl der Art der Visualisierung für eine laufende Query*

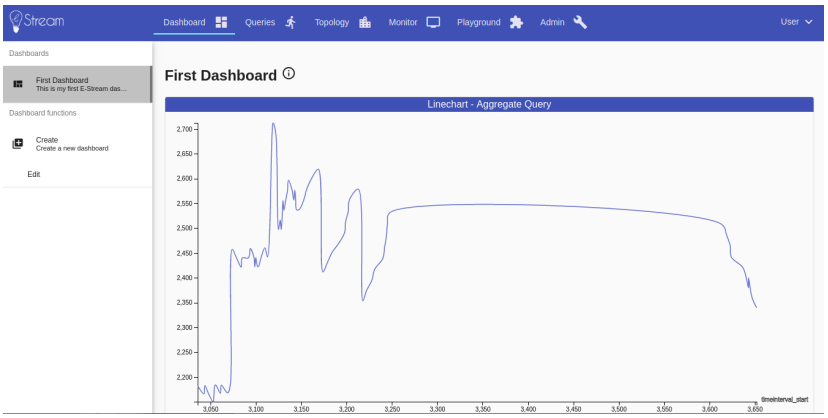


Abbildung 8.44: *Fertiges Dashboard mit einer Visualisierung*

9 Test

9.1 Übersicht Testmethoden

Um sicherstellen zu können, dass die entwickelte Software so funktioniert, wie erwartet, ist es unumgänglich die Software zu testen. Dabei kann die Software manuell bei jeder Änderung ausprobiert werden, um zu überprüfen, ob diese Änderungen ungewolltes Verhalten erzeugen. Eine bessere Möglichkeit ist es automatisierte Tests zu nutzen. Diese werden von Entwicklern oder Testern eigens dafür geschrieben, um die Software auf Fehlverhalten zu überprüfen.

Traditionell werden bei der Softwareentwicklung die Rollen von Entwicklern und Testern getrennt [16]. Dadurch entsteht allerdings eine Verzögerung beim Testen von Features, wenn diese schneller in einem System entwickelt als getestet werden können. Entwickler, die ihre Features zum Testen bereitstellen müssen je nach Entwicklungsprozess bis zu einigen Wochen warten, bis sie eine Rückmeldung von einem dedizierten Testteam erhalten [16]. Gerade in einem agilen Entwicklungsprozess passt eine solche Wartezeit auf die Testergebnisse nicht zu dem Rest der iterativen Entwicklung, die auf schnelle Rückmeldungen ausgelegt ist. Deshalb übernehmen in agilen Teams die Entwickler immer mehr die Verantwortung für das Testen ihres eigenen Quelltextes oder es arbeiten Tester mit den Entwicklern im Rahmen eines Sprints zusammen am Testen der Software.

9.2 Unit-, Integration- und End-To-End-Tests

Bevor neue Software getestet werden kann, sollte zunächst klar sein, welchen Teil der Software die Tests abdecken. Tests können von einer einzelnen Funktion über eine Klasse bis hin zum gesamten System alles abdecken. Dabei kommt man gewöhnlich zu drei Kategorien, die diese Art des Testens beschreiben:

- **Unittests** sollen sich beim Testen auf den kleinsten Interaktionsumfang mit dem System beschränken. In Java kann ein Unittest zum Beispiel eine Klasse abdecken und innerhalb des Tests wird die Logik der Klasse getestet.
- **Integrationsstests** decken eine große Menge an Einheiten ab, die getestet werden sollen. Ein einzelner Integrationsstest kann zum Beispiel aus mehreren Klassen oder Interfaces bestehen, die zusammen interagieren müssen.
- **End-To-End-Tests** testen im Allgemeinen das gesamte System ohne dabei innere Funktionen des Systems zu benutzen. Sie sollen die Interaktion des Nutzers simulieren und zum Beispiel GUI-Interaktionen ausführen, um so das System darauf zu überprüfen, ob es die funktionalen Anforderungen erfüllt.

Damit Tests Entwickler effektiv unterstützen können, müssen Sie schnell ausführbar und zuverlässig sein. Außerdem müssen Fehler in der Software durch Tests ausfindig gemacht werden. Durch zu große Ausführungszeiten der Tests sinkt die Wahrscheinlichkeit, dass Entwickler diese nach Änderungen an der Software ausführen. Somit können Sie nicht sicher sein, dass etwaige Fehler behoben oder neue Fehler verursacht wurden. Sind Tests unzuverlässig indem Sie zum Beispiel falsche Fehlschläge anzeigen oder keine, obwohl Fehler vorliegen, senken Sie das Vertrauen von Entwicklern in die Tests. Dies führt ebenfalls dazu, dass Sie weniger ausgeführt werden oder ihre Ergebnisse ignoriert

werden. Wenn Tests nicht die Ursache für einen Fehlschlag eingrenzen können, müssen Entwickler die Fehlerursache finden. Dies kann die Entwicklung verzögern.

Gerade End-To-End-Tests sind sehr fragil, da beispielsweise kleine Änderungen an einer GUI die Tests fehlschlagen lassen und aufeinanderfolgende Tests, die eine Nutzerinteraktion simulieren beim ersten Fehlschlag weitere Fehler anzeigen, obwohl diese evtl. nur auf Grund des ersten Fehlschlags entstanden sind. Auch tendieren solche Tests dazu sehr lange zu dauern, da alle Komponenten des Systems benötigt werden. Aus diesem Grund gibt es einen ungefähren Richtwert, dass die Tests, die am wenigsten das System abdecken, am häufigsten vorkommen sollen. Das heißt, dass ein gutes Verhältnis aus Tests am meisten aus Unittests, anschließend aus Integrationstests und einigen im Vergleich wenigen End-To-End-Tests bestehen sollte [271].

9.3 Verwendete Tests

In diesem Projekt wurden verschiedene Arten von Tests geschrieben. Da es verschiedene Arten von Integrationstests gibt, wurden diese in weitere Kategorien unterteilt. Im Backend gibt es Java Schnittstellen für die Kategorien „FastTest“, „SlowTest“, „KafkaTest“ und „LocalIntegrationTest“. Diese befinden sich im Paket `de.estream.category`. Mithilfe der JUnit `@Category` Annotation wird dem Test eine Kategorie zugewiesen. Listing 9.1 zeigt eine beispielhafte Verwendung der JUnit `@Category` Annotation.

```
1 @Category(de.estream.category.KafkaTest.class)
2 public class KafkaTest {
3
4 // der Test...
5
6 }
```

Listing 9.1: *Zuweisen einer Kategorie*

Zusätzlich gibt es für die verschiedenen Kategorien eigene gradle Tasks. Diese suchen nach Tests die mit der jeweiligen Kategorie markiert wurden und führen nur diese aus.

Im folgenden werden die einzelnen Kategorien und die Verwendung in Projekt beschrieben.

9.3.1 Unittests

Unit Tests gehören zu der Kategorie „FastTest“. Mit diesen wurde das Verhalten einzelner Services getestet. Alle Abhängigkeiten, die nichts mit dem jeweiligen Service zu tun hatten, wurden mithilfe von Mockito[1] aufgelöst. So konnte die Logik der Services unabhängig von anderen Klassen getestet werden. Außerdem muss das Backend so nicht gestartet werden. Dadurch waren diese Tests vergleichsweise sehr schnell. Vor allem Services, die für die Nutzerverwaltung und die Verwaltung der Queries zuständig sind, wurden mit Unit Tests getestet.

[1] online unter `http://site.mockito.org/`

9.3.2 Integrationstest

Bei den Integrationstests wird der gesamte Server mit einer In-Memory-Datenbank gestartet. Dadurch sind diese Tests um einiges langsamer. Der Vorteil ist aber, dass so zum Beispiel auch die REST-Schnittstelle getestet werden kann. Diese Tests sind allerdings instabiler und schlagen manchmal nach Änderungen fehl obwohl nichts fehlerhaft ist.

Um die stabileren Integrationstests von den instabileren abzugrenzen, wurden sie in drei Unterkategorien aufgeteilt. Diese werden im Folgenden beschrieben.

9.3.2.1 Slow-Test

In die „SlowTest“ Kategorie fallen alle stabilen Integrationstests. Wenn ein Slow-Test fehlschlägt, ist also sehr wahrscheinlich etwas fehlerhaft. Entweder wurde der Test nach einer Änderung nicht angepasst oder es handelt sich um einen Fehler im Quellcode. Slow-Tests wurden für die Authentifizierung und für das UserRepository geschrieben.

9.3.2.2 Kafka-Test

Kafka-Tests prüfen Klassen, die von Kafka abhängen. Im Verlauf des Projektes wurden oft festgestellt, dass Kafka Tests oft fehlschlagen, obwohl nichts fehlerhaft ist. Dies liegt häufig an den zu kleinen Timeouts für den Test. Wenn der Test erneut ausgeführt wird, funktioniert es oft wieder. Vor allem der KafkaService wurde sehr umfassend getestet.

9.3.2.3 Localintegration-Test

Localintegration-Tests sind Tests, die nicht auf dem Jenkins ausgeführt werden können, sondern ein lokales Setup benötigen. Dies liegt daran, dass es aufwändig wäre diese Tests so zu schreiben, dass sie lokal und auf dem Jenkins funktionieren. Denn das Produktivsystem und die Developmentumgebung verwenden unterschiedliche Kafka Broker- und Odysseus Adressen sowie unterschiedliche Ports. Somit wurde die „LocalIntegrationTest“ Kategorie eingeführt, die auf die lokale Developmentumgebung angepasst wurde.

9.3.3 Ping-Test

Der Ping-Test ist ein Jenkins Job, welcher den Client aufruft und überprüft ob der Client einen sinnvollen HTTP-Code zurückgibt. Er überprüft also ob der Client erreichbar ist.

9.3.4 Schnittstellentests

Für die Kooperation mit Langzeitarchiven wurde zusammen mit PG DAvE eine gemeinsame Schnittstelle entworfen mit der der Datenaustausch zwischen dem System und einem Langzeitdatenarchiv erfolgt. Damit die Entwicklung der Systeme unabhängig voneinander stattfinden kann, wurde beschlossen ein Testsystem zu schreiben, dass das Verhalten des jeweils anderen Systems emulieren soll.

Um die öffentlichen Schnittstellen eines Langzeitarchivs zu testen, wurde ein Testsystem geschrieben, das zum Beispiel bei einer Anfrage nach einem trainierten Data Mining Modell eine PMML-Datei auf einen Kafka Topic sendet und als Antwort die URI des Kafka Brokers und den Topic-Namen zurückgibt. Dafür wurde ein neues Spring Boot Projekt mit einem eigenen Kafka Cluster aufgestellt. Weiterhin wird das Testsystem auf einer virtuellen Maschine in Docker Containern ausgeführt.

9.3.5 e2e-Tests

End-To-End oder e2e-Tests sind eine spezielle Art von Integrationstests, die darauf abzielen Fehler zu finden, die bei der Integration mehrerer Komponenten auftreten und daher nicht mit Hilfe von Unit-Tests gefunden werden können. Sie ermöglichen es, die Interaktion eines echten Nutzers mit der Website zu simulieren und automatisierte UI-Tests zu erstellen [13].

In Angular Applikationen werden die e2e-Tests in der Regel mit Hilfe von Protractor ausgeführt [223]. Protractor wurde im Kontext von Angular entwickelt und bietet daher sehr gute Integration in Angular Apps. Protractor benutzt Selenium WebDriver, um den Browser automatisiert zu steuern. Demnach stellt Protractor die Schnittstelle zwischen der Angular Applikation und dem Automatisierungsframework dar. Die Struktur der Tests ist sehr ähnlich zu einem regulären Unit-Test, wie in Abbildung 9.1 zu sehen ist. Der Unterschied ist lediglich, dass Protractor verwendet wird, um mit den UI-Elementen zu interagieren. Das Besondere hierbei ist, dass das Klicken eines Elements möglichst realistisch simuliert wird. Wird also versucht auf ein Element zu klicken, das von einem anderen Element verdeckt wird und das Event nicht weiterleitet, schlägt der Klick fehl. Würde zum Beispiel der Klick durch aufrufen der Click-Methode des Elements mittels JavaScript geschehen, würde diese Methode auch ausgeführt werden, wenn das Element verdeckt oder sogar unsichtbar wäre. Durch e2e-Tests wird ein realistischeres Verhalten simuliert, was dabei hilft, Fehler in konkreten Anwendungsfällen der Nutzer der Website zu entdecken [13].

Im Falle dieses Projekts werden mit Hilfe von e2e-Tests einfache Aufgaben bei der Benutzung der Website getestet. Hierzu gehört das Anmelden und Abmelden von der Website, sowie die Verifikation, dass nur korrekte Anmeldedaten funktionieren. Außerdem wird die Navigation auf die verschiedenen Bereiche getestet. Es wird also ein Nutzer simuliert, der alle Bereiche der Website einmal öffnet und somit sicherstellt, dass alle Bereiche verfügbar sind und die Website sich nicht aufhängt. Weitere kompliziertere Szenarien sind möglich, wie z.B. das Erstellen eines neuen Nutzers oder das Konfigurieren des Dashboards. Hierbei muss bedacht werden, dass es sich um Integrationstests handelt. Der Server muss also verfügbar sein und Änderungen an der Datenbank, wie z.B. durch das Erstellen eines Nutzers, wirken sich auf das Produktionssystem aus, wenn keine spezielle Konfiguration für die e2e-Tests genutzt wird. Es muss auf die Nebenwirkungen von Aktionen geachtet werden bzw. Veränderungen rückgängig gemacht oder eine separate Umgebung verwendet werden.

9.4 Automatisierte Tests

Damit Unittests schnell eine Rückmeldung geben können, ob das System noch weiterhin so funktioniert, wie das vor einer Änderung geschah, müssen Tests automatisiert laufen. Manuelle Eingaben von Werten und händisches Testen von Logik kann zwar in einer explorativen Phase stattfinden, um möglicherweise sinnvolle Randwerte zu finden. Sobald aber die Testparameter klar sind, sollten diese wiederholbar

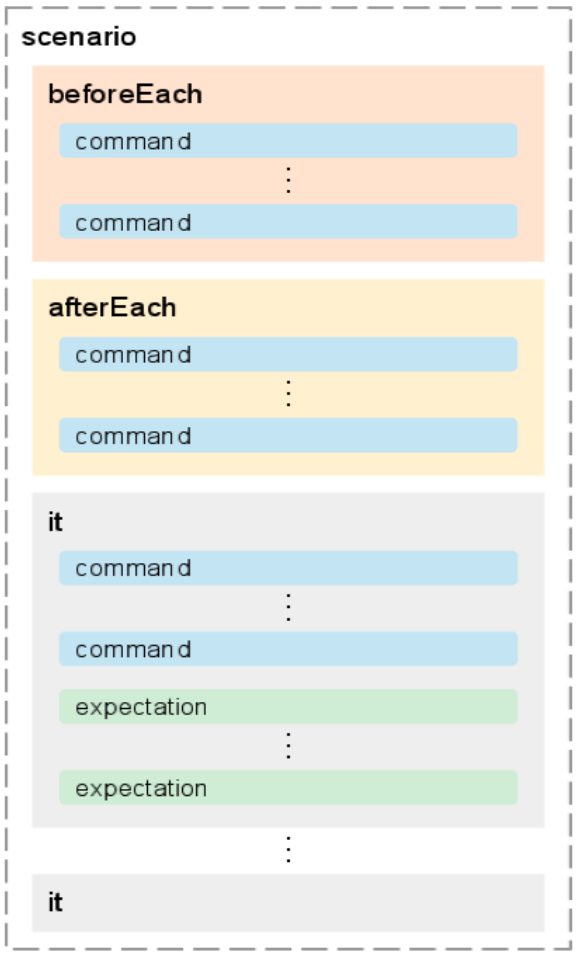


Abbildung 9.1: *Struktur eines e2e-Tests im Jasmine Test-Framework*

immer wieder aufgerufen werden können. Martin Fowler beschreibt das mit Self-Testing Code, wo automatisierte Tests gegen das zu testende System ausgeführt werden [102]. Zusammen mit einer kontinuierlichen Integration des Systems können diese Tests nach jeder Änderung ausgeführt und so beispielsweise bei Fehlschlägen in den Tests, die letzten Änderungen verworfen werden. Dabei spielt es keine Rolle, ob die Tests nach der Implementierung der Einheit oder davor implementiert werden, wie etwa bei der Test-Driven Development Methode geschieht, so lange die Einheiten von Tests abgedeckt werden und sie den drei vorher genannten Anforderungen an Unit Tests entsprechen.

10 Evaluation

10.1 Usability Tests

Usability beschreibt die Eigenschaft eines Produktes, die es Nutzern erlaubt, effektiv und intuitiv Aufgaben zu erfüllen. Die folgenden vier Punkte spielen hierbei eine besondere Rolle [72]:

1. Usability legt den Fokus auf den Nutzer.
2. Die Nutzer benutzen das Produkt, um produktiver zu sein.
3. Nutzer versuchen Aufgaben zu erledigen.
4. Die Nutzer entscheiden, ob das Produkt einfach zu nutzen ist.

Der erste Punkt bezieht sich auf den zukünftigen und potentiellen Nutzer, der bei der Entwicklung des Produktes direkt beteiligt wird. Dadurch soll ein gebrauchstaugliches Produkt entstehen, wobei der Entwicklungsprozess direkt von den Nutzern beeinflusst wird. Es wird angenommen, dass die Entwickler des Produktes selbst keine geeigneten Kandidaten für eine Evaluation sind [72]. Ein Produkt kann auf verschiedene Weisen produktiv aus der Sicht eines Nutzers sein. Wenn es nicht viel Zeit beansprucht, sich mit dem Produkt vertraut zu machen, wenn nicht viele Schritte notwendig sind, um das gewünschte Ziel zu erreichen oder wenn es einfach ist, neue Aufgaben mit Hilfe des Produkts zu bewältigen, ohne lange nach einer Lösung zu suchen [72].

Usability wird häufig mit Produktivität verbunden. Dies liegt daran, dass ein Produkt mit einer hohen Gebrauchstauglichkeit in der Regel auch die größte Zeitersparnis mit sich bringt. Das Produkt ist lediglich ein Hilfsmittel, welches dem Nutzer erlaubt, seine Arbeit produktiver zu erledigen. Es ist wichtig, dass diese Erkenntnis bei der Entwicklung im Fokus ist. Außerdem sollte eine Dokumentation zur Verfügung gestellt werden, die vom Nutzer zu Hilfe gezogen werden kann, wenn er vor einem neuem Problem steht. Ein Beispiel für ein Problem ist das Adressieren von Briefen mit Textverarbeitungsprogrammen. Viele Unternehmen benutzen Textverarbeitungsprogramme um Briefe zu erstellen, adressieren diese jedoch handschriftlich, weil den Nutzern die Funktion zum Adressieren von Briefen im Textverarbeitungsprogramm nicht bekannt ist oder sie es für zu schwierig halten und dann die Alternative, sie handschriftlich zu adressieren, wählen [72].

Der letzte Punkt kann so interpretiert werden, dass die Nutzer die Gebrauchstauglichkeit des Produkts bewerten und nicht die Entwickler oder Designer des Produkts. Es muss des Weiteren beachtet werden, dass nicht nur die Usability der Funktion zählt, nachdem man gelernt hat sie zu benutzen, sondern auch die Zeit, die dafür aufgewendet werden muss, diese Funktion zu beherrschen. Häufig ist die Lernkurve von Produkten zu steil für die Anwender, was sie dann dazu veranlasst, bei den Basisfunktionalitäten zu bleiben und alle weitergehenden Funktionen zu ignorieren [72]. Im Folgenden werden verschiedene Testmethoden vorgestellt, mit denen die Usability eines Produkts evaluiert werden kann.

10.1.1 System Usability Scale

Es ist schwierig, Usability in konkreten Werten auszudrücken, da Subjektivität stets eine Rolle spielt. Des Weiteren muss Usability immer im Kontext der Anwendung gesehen werden. Speziell für Informationssysteme ist die Usability in ISO-9241-11 festgelegt. Um die inhärente Subjektivität abzubilden, müssen geeignete Einheiten beziehungsweise eine geeignete Skala bei der Evaluierung verwendet

werden. Der ISO-9241-11 nennt drei verschiedene Klassen von Messungen, die bei der Evaluation der Usability beachtet werden sollten [158].

1. Effektivität (Die Fähigkeit eines Nutzers Aufgaben mit dem Produkt zu erledigen und die Qualität der Ergebnisse)
2. Effizienz (Die Anzahl der verbrauchten Ressourcen beim Bearbeiten der Aufgaben)
3. Zufriedenheit (Subjektive Bewertung der Zufriedenheit beim Benutzen des Systems durch den Nutzer)

Die Bedeutung von Effektivität ist kontextabhängig und die Einheiten variieren demnach je nach System. Dies hat auch zur Folge, dass ein Vergleich der Usability verschiedener Systeme schwer möglich ist. Gleiches gilt für die Effizienz des Systems. Im dritten Punkt, der Zufriedenheit, ist es schon eher möglich Systeme zu vergleichen. Hier werden systemunabhängige Skalen verwendet, um diese subjektive Bewertung widerzuspiegeln. Ein Beispiel für eine solche Skala ist die Likert-Skala [158]. Die Likert-Scala ist rangskaliert und nicht kardinal, wird in der Praxis jedoch oft durch natürliche Zahlen kodiert. Allgemein dienen sie dazu, die Übereinstimmung der Meinung eines Testers mit einer Aussage festzustellen. Hierbei können die Antwortmöglichkeiten sowohl eine gerade, als auch eine ungerade Anzahl besitzen. Bei einer ungeraden Anzahl gilt die mittlere Möglichkeit häufig als neutral. Falls es nicht möglich sein soll, neutral zu antworten, sollte eine gerade Anzahl an Antwortmöglichkeiten verwendet werden [234].

Ein Testverfahren, bei dem die Likert-Skala angewendet wird, ist die *System Usability Scale* oder auch *SUS*. Die SUS beinhaltet zehn Punkte, die der Nutzer, nachdem er mit dem Produkt gearbeitet hat, nach der Likert-Skala bewerten soll. Es werden zwischen fünf und sieben Antwortmöglichkeiten vorgegeben. Es ist relevant, dass ein Nutzer nach diesem Schema das Produkt bewertet. Diese zehn Fragen wurden aus einer Auswahl von 50 Fragen ausgewählt, da sie bei den Nutzern die höchste Übereinstimmung zeigten. Es wurden zwei Softwaresysteme miteinander verglichen, wobei eines eine hohe und das andere eine geringe Usability aufweist. Die Tester beantworteten alle 50 Fragen zu dem System das sie testen sollten. In der Auswertung wurden dann die Fragen selektiert, in denen die Nutzer die größte Übereinstimmung hatten. Nur so ist eine Basis für eine wissenschaftliche Auswertung der Ergebnisse gegeben, da die Streuung oder Subjektivität bei den anderen Fragen zu hoch ist. Das Ergebnis dieser Analyse ist die SUS mit den zehn Fragen die nun in der Regel benutzt werden. Die SUS sieht folgendermaßen aus [158]:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.
5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.

10. I needed to learn a lot of things before I could get going with this system.

Die Antwortmöglichkeiten reichen von *Strongly disagree* bis hin zu *Strongly Agree* auf einer fünf Punkte Skala. Die SUS sollte direkt im Anschluss nach der Benutzung des Produkts stattfinden, ohne nach der Benutzung mit den Entwicklern gesprochen zu haben. Des Weiteren sollte nicht zu lange über die Fragen nachgedacht werden, sondern die erste Reaktion auf die Frage festgehalten werden [158]. Die Auswertung der SUS geschieht über eine Gewichtung der Antworten auf die einzelnen Fragen. Das Ergebnis ist ein einzelner Wert im Bereich von 0 - 100. Die einzelnen Antworten liegen in einem Bereich von 0 - 4 (Trifft überhaupt nicht zu bis trifft auf jeden Fall zu). Um das Ergebnis zu erhalten werden zuerst die Antworten der Fragen 1, 3, 5, 7 und 9 betrachtet. Es wird 1 von ihrem Skala-Wert subtrahiert und die Ergebnisse summiert. Danach werden die Antworten der Fragen 2, 4, 6, 8 und 10 betrachtet. Ihr Beitrag ist $5 - scaleValue$. Auch sie werden summiert und es wird eine Gesamtsumme gebildet. Diese liegt in einem Bereich von 0 bis 40. Die Summe wird daher mit 2,5 multipliziert, um einen Wertebereich von 0 - 100 zu erhalten. Dieser Wert stellt das Ergebnis der SUS dar. Der Grund für die unterschiedliche Bewertung der Fragen liegt daran, dass einige Fragen positiv und andere negativ formuliert sind. Dies soll den Nutzer dazu veranlassen jede Frage genau zu lesen [158].

Die SUS ist frei verfügbar und hat sich in der Praxis als wertvoll erwiesen. Sie korreliert auch mit anderen Usability Skalen, was ihre Aussagekraft bestätigt [158]. In diesem Projekt bietet die SUS eine einfache und kostengünstige Methode, die Usability unseres Produkts zu evaluieren und sollte daher angewandt werden. Sie setzt allerdings voraus, dass die Testpersonen die Gruppe potentieller Anwender darstellen, was im Kontext dieses Projekts schwierig ist. Die Durchführung der Usability Analyse mit anderen Nutzern mindert die Aussagekraft des Tests, kann jedoch trotzdem gute Einblicke in die Usability geben.

10.1.2 Clickstream Analysis

Clickstream Analysis bezeichnet ein breites wissenschaftliches Feld, das sich damit beschäftigt Informationen aus dem Web-Log eines Servers zu ziehen. Aus dem Web-Log ist ersichtlich, auf welchem Weg ein Nutzer die Webseite traversiert, wie lange er sich auf den einzelnen Seiten aufgehalten hat und vieles mehr. Die Zuordnung eines Nutzers zu einer Anfrage geschieht zum Beispiel über die IP-Adresse. Clickstream Analyse wird mittlerweile von sehr vielen großen Firmen betrieben. Es ist außerdem möglich die Analyse von einem anderen Anbieter ausführen zu lassen, indem Zugriffe auf die eigene Webseite durch einen Proxy-Server des Analyseanbieters geschickt werden. Die Anwendungsbereiche von Clickstream Analysis reichen von der Erkennung von Robotern bis hin zum besseren Verständnis des Kundenverhaltens. Eine Fluggesellschaft könnte beispielsweise das Ziel verfolgen, dass ein Kunde fähig sein soll innerhalb von drei Minuten einen Flug zu buchen. Mittels Clickstream Analysis kann danach genau verfolgt werden, in welchem Schritt des Prozesses wie viel Zeit verstreicht. Es kann außerdem ermittelt werden, welcher Anteil der Besucher letztendlich ein Produkt oder eine Dienstleistung erwirbt [167].

Ein Teil der Clickstream Analyse ist die Pfadanalyse. Die einzelnen Seiten der Webseite sind über Hyperlinks miteinander verbunden. Ein Nutzer navigiert üblicherweise über diese Links zu anderen Teilen der Webseite. Dadurch ergibt sich ein Pfadgraph, an dem zu erkennen ist, welche Teile der Webseite häufig besucht werden und welche nur sporadisch vorkommen. Verbindet man dies mit der Zeit, die auf den jeweiligen Seiten verblieben wird, kann man wertvolle Informationen darüber erhalten, wie gut die Usability der Seite ist [167].

Generell bietet die Clickstream Analysis viele Möglichkeiten um das Benutzerverhalten zu analysieren. Dies ist vor allem im E-Commerce von Bedeutung und findet daher in Online-Shops die größte Anwendung. Die Bewertung von Usability mittels Clickstream Analysis ist nur ein kleiner Teil dieses großen Feldes. Die Anwendung und Auswertung ist nicht so leicht, wie zum Beispiel bei der SUS und erfordert eine größere Anwendergruppe um gute Ergebnisse zu liefern. Für die Messung der Usability wäre etwa Eye-Tracking von größerem Vorteil, da so gezielter die Elemente identifiziert werden können, die die Nutzer verwirren beziehungsweise bei denen sie sich lange aufhalten. Des Weiteren könnte dadurch festgestellt werden, welcher Teil des Produkts für den Anwender sehr wichtig ist und welche Elemente er nicht einmal bemerkt. Im Rahmen dieses Projektes ist die Realisierung und Auswertung von Clickstream Analysen wahrscheinlich zu aufwändig, da die Tests länger dauern und kein Budget zur Verfügung steht, um Tester für den Aufwand zu vergüten. Da das Aufsetzen eines Usability Labors zu aufwändig ist, bieten sich einfache Testverfahren, wie die SUS für dieses Projekt an, um einen Überblick über die Usability des Produkts zu bekommen.

10.2 Durchführung der Usability Tests

Die Durchführung der Usabililty Tests für das Produkt geschah durch eine Feldstudie, die mit Hilfe eines System Usability Scale Fragebogens ausgewertet wurde. In dieser Studie wurden vier Probanden eingeladen, um an der Studie teilzunehmen. Die Anzahl wurde auf maximal fünf gesetzt, denn nach Jakob Nielsen können mit dieser Anzahl an Probanden schon die wichtigsten Probleme in der Usability des Systems aufgedeckt werden und weitere Probanden verbessern nicht zwangsweise das Ergebnis [206]. Die Probanden wurden zunächst von mehreren Studienleiter empfangen und haben eine Einwilligungserklärung zum Unterzeichnen bekommen, die eine Zusammenfassung der Thematik der Studie enthält und den Ablauf der Studie beschreibt. Etwaige Fragen der Probanden, die dabei aufkamen wurden von den Studienleitern beantwortet.

Nachdem die Probanden die freiwillige Teilnahme an der Studie unterschrieben haben, wurden sie an einen Arbeitsplatz mit dem bereits laufenden System gesetzt. Anschließend wurde den Probanden die erste Aufgabenbeschreibung vorgelesen und die Probanden haben mit der Aufgabe begonnen. Nach Abschluss einer Aufgabe wurde die Beschreibung für die nächste Aufgabe vorgelesen. Dieser Vorgang hat sich so lange wiederholt bis alle Aufgaben erledigt waren. Danach wurde ein Fragebogen an die Probanden verteilt, mit dem sie die Usability des Systems beurteilen sollten. Abschließend wurden die Probanden nach weiteren Punkten gefragt, die ihnen bei der Benutzung des Systems aufgefallen sind und danach verabschiedet.

10.2.1 Probanden

Die Auswahl der Probanden fiel auf zwei Mitarbeiter aus der NetzDatenStrom Arbeitsgruppe des OFFIS sowie auf zwei Studierende aus einer fachfremden Projektgruppe. Die Teilnehmer waren alle männlich und zwischen 27 und 34 Jahre alt. Somit war die Hälfte der Probanden grundsätzlich mit dem Energiebereich vertraut und die andere Hälfte war zwar fachfremd im Bezug auf den Energiebereich. Dennoch waren es Studierende des Fachbereiches Informatik und kannten sich dadurch mit verschiedenen Arten von Bedienkonzepten für Informationssystemen aus. Somit waren alle Teilnehmer mit der Bedienung informationstechnischer Systeme vertraut, sodass sie adäquat die Umsetzung des E-Stream Produkts bewerten können.

10.2.2 Studienverlauf

Zu aller erst bekamen die Probanden eine kleine Einweisung, in der das Projekt und das Dashboard, welches die Probanden im laufe des Tests erstellen sollen, kurz vorgestellt wurde. Anschließend haben die Probanden die Aufgabenstellung gelesen und haben mit der Bearbeitung der Aufgaben begonnen. Dabei wurde das vorgehen der Probanden festgehalten. Bei fragen der Probanden zum System, wurden diese von uns Beantwortet, ansonsten mussten die Probanden die Aufgabenstellung selbstständig bearbeiten. Anschließend mussten die Probanden die Bedienbarkeit des Systems bewerten. Die Aufgaben an die Probanden waren:

1. Anmelden und Erstellen eines Dashboards.
2. Erstellen und Visualisieren einer Query.
3. Filtern der Rohdaten der Smart Meter

10.2.3 Auswertung der Studie

Die Auswertung des System Usabililty Scales hat gezeigt, dass das E-Stream Produkt keine intuitiv Bedienung besitzt. Mit dem einem Mittelwert von 60% wurde das System unterdurchschnittlich bewertet. Der SUS-Test prüft die Gebrauchstauglichkeit eines Systems, wobei das E-Stream System auch zum Abschluss des Projekts kein marktreifes Produkt, sondern ein Prototyp ist. Daher wurde zu Beginn der Evaluation angenommen, dass die Bedienbarkeit geringfügig ausfällt. Des Weiteren ist die Zielgruppe des E-Stream Systems spezielles Fachpersonal, welches vor einer dauerhaften Anwendung des Systems eine Schulung erhalten würde.

PID **Question**	1 Answer	2 Answer	3 Answer	4 Answer
1	5	2	2	4
2	4	2	4	2
3	2	4	4	2
4	4	2	3	2
5	5	4	2	3
6	1	1	3	4
7	3	5	2	5
8	3	2	4	1
9	2	4	3	3
10	2	1	3	2
Result	57,5	77,5	40	65

Tabelle 10.1: *System Usability Scale Score*

Die Probanden haben angegeben, dass das System zur besseren und schnelleren Bedienung mehr Hilfetexte enthalten sollte. Außerdem wurde angemerkt, dass die Dialoge in der Weboberfläche explizite Schaltflächen für *Speichern* und *Schließen* oder *Abbrechen* haben sollte, damit der Anwender seine Interaktion konkret bestätigen kann.

10.3 Konzept für Performance Evaluation

Für die Performanceevaluation wird zunächst die Klasse des Systems benötigt. Durch die Entkopplung in Form von Kafka von Odysseus und dem Backend ist es naheliegend, dass es sich um ein verteiltes System handelt und daher nach den abgeleiteten Kriterien von [37] aus [3] evaluiert werden sollte. Auf der anderen Seite ist das Backend in Kombination mit der Datenbank und dem Frontend monolithisch aufgebaut und kann daher diese Kriterien nicht erfüllen. Die Performanz dieses Teilsystems als verteiltes System ist auch insofern nicht sinnvoll, da der Anwendungsbereich als Prototyp für Netzleitsysteme im Kontext des Smart Grids eine geringe Nutzerzahl (< 1000) erwarten lässt. Die Performanzevaluation sollte daher in 3 Bereiche, visualisiert durch die Abbildung, unterteilt werden: das verwaltende System als monolithisches System - bestehend aus dem Backend, dem Frontend und der Datenbank -, das verarbeitende System - bestehend aus Odysseus und Kafka - als verteiltes System und die Kommunikationsmechanismen zwischen den Systemen. Dabei ist zu beachten, dass nach [4] die allgemeine Performanz des Systems immer von der benutzen Hardware und verteilten System zusätzlich von der Lokalität der Komponenten abhängt.

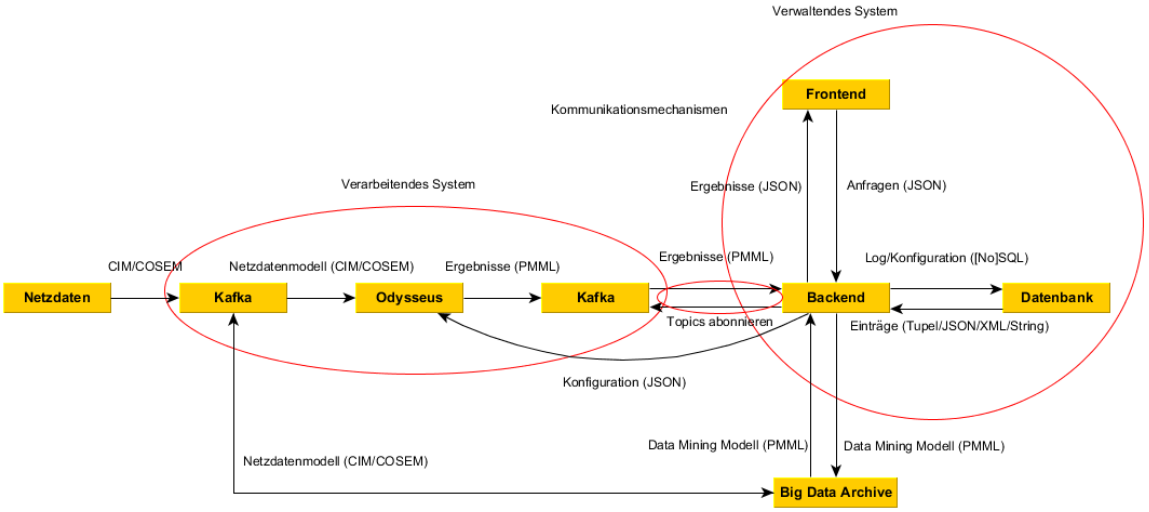


Abbildung 10.1: *Unterteilung der Architektur in Komponenten, die zusammen überprüft werden.*

10.3.1 Vewaltender Bereich

Der verwaltende Bereich ist eine monolithische Webanwendung. Dabei muss zwischen der Performanz des Servers (Backend) und der Performanz des Clients (Frontend) unterschieden werden, da diese als unabhängige Anwendungen entwickelt wurden, die miteinander kommunizieren können. Außerdem werden unterschiedliche Anforderungen an den Server in Form von Reaktivität und Latenz als an den Client gestellt.

10.3.1.1 Backend Performance

Die Performanz dieser Anwendung kann dabei laut [37] durch die Latenz von Anfragen unter Last evaluiert werden. Die Evaluation kann direkt über das Frontend evaluiert werden, wenn die Request-Response Zeit gemessen wird. Dies bedeutet, dass man eine fest gesetzte Anzahl an Anfragen pro Sekunde an das System schickt und die mittlere Antwortzeit in Millisekunden misst. Da die Antwortzeit aber unter anderem auch von den Anfragen abhängt, müssen verschiedene Anfragetypen isoliert werden: Anfragen an das Backend ohne Seiteneffekte, d.h. ohne Einbindung der Datenbank, eines Datenarchivs, Kafka oder Odysseus; Anfragen mit Seiteneffekten, die sich aus den kombinatorischen Möglichkeiten ergeben. Zusammengefasst ergeben sich folgende Möglichkeiten:

- Anfragen ohne Einbezug anderer Systeme (zum Beispiel das Anfragen der Startseite)
- Anfragen mit Einbezug der Datenbank (zum Beispiel Login)
- Anfragen mit Einbezug von Kafka (zum Beispiel Anfragen der Topologie)
- Anfragen mit Einbezug von Odysseus (Anfragen zum Starten/Stoppen einer Odysseus-Query)
- Anfragen mit Einbezug der Datenbank und Kafka (sofern vorhanden)
- Anfragen mit Einbezug der Datenbank und Odysseus (sofern vorhanden)
- Anfragen mit Einbezug der Datenbank, Odysseus und Kafka (Installieren von Queries mit automatischer Ausführung und anschließender Visualisierung im Frontend)

10.3.1.2 Frontend Performance

Des Weiteren sollte die Reaktivität des Frontends getestet werden. Für visualisierende Systeme ist ein zentrales Performanzkriterium nach [280] die gezeichneten Bilder pro Sekunde (fps). Dabei ist es unabdingbar, dass 30 fps überschritten werden, da es sonst zu wahrnehmbaren Bildstörungen kommen kann. Darüber hinaus werden alle JavaScript-Operationen in einem Thread ausgeführt. Sollten die fps nicht verfügbar sein, dann kann auf einen Reaktivitätstest der Oberfläche zurückgegriffen werden: ein Knopf wird gedrückt, der eine einfache Änderung an der GUI durchführen soll (zum Beispiel Anzeigen eines Texts). Dabei kann gemessen werden wie lange das Ausführen dieser Methode dauert, während visualisierende Komponenten ebenfalls angezeigt werden. Das Problem bei dieser Methode liegt darin, dass in den Ablauf des Systems eingegriffen wird und somit die Ergebnisse minimal verfälscht werden. Die angezeigte Datenfenstergröße und die Anzahl an visualisierenden Komponenten kann ebenfalls unterschieden werden. Die Datenfenstergröße erhöht die Komplexität eines Graphen und verringert somit die Performanz, während viele Graphen problematisch sind, da alle eine gewisse Komplexität aufweisen. Zusammengefasst gibt es folgende kombinatorische Möglichkeiten der Frontendperformanceevaluation:

- Ein Graph mit einem kleinen Fenster (zum Beispiel 100 Elemente) und ein FPS-Test
- Ein Graph mit einem kleinen Fenster (zum Beispiel 100 Elemente) und ein Reaktivitäts-Test
- Ein Graph mit einem großen Fenster (zum Beispiel 10000 Elemente) und ein FPS-Test
- Ein Graph mit einem großen Fenster (zum Beispiel 10000 Elemente) und ein Reaktivitäts-Test
- Viele Graphen (8-16) mit einem kleinen Fenster und ein FPS-Test

- Viele Graphen mit einem kleinen Fenster und ein Reaktivitäts-Test
- Viele Graphen mit einem großen Fenster und ein FPS-Test
- Viele Graphen mit einem großen Fenster und ein Reaktivitäts-Test

10.3.2 Verarbeitender Bereich

Der verarbeitende Bereich ist ein verteiltes System und daher lässt sich die Performanz durch die Umsetzung der Kriterien aus [3] und der daraus folgenden Skalierbarkeit evaluieren. Skalierbarkeit für ein verteiltes System ist der Schlüsselpunkt der Performanz des Systems, da so bei Bedarf weitere Hardwareressourcen zugeschaltet werden können, wenn die klassischen Performanzanforderungen aus monolithischen Systemen einen Schwellwert überschreiten. Dabei sind diese Kriterien zentral für die Skalierbarkeit des verarbeitenden Systems. Voraussetzungen, bzw. Konzepte zur Erfüllung dieser Kriterien werden in [36] beschrieben. Zusätzlich muss die Performanz der Datenstromverarbeitung evaluiert werden. Die zentralen Konzepte sind dabei Reaktion, Widerstandsfähigkeit, Elastizität und Nachrichten-getrieben. Reaktion äußert sich dabei durch eine Antwort des Systems nach möglichst kurzer Zeit. Widerstandsfähigkeit bedeutet, dass das System mit Fehlern umgehen und sich ohne äußere Einwirkung reparieren kann (self-healing). Elastizität umfasst die Tatsache, dass die Lokalität eines Systems nicht von Belang ist und dieses System ohne Probleme auf andere Knoten verschoben werden kann. Nachrichtengetriebene Systeme kommunizieren durch nicht blockierende Kommunikationsprotokolle mit asynchronen Nachrichten und Transparenz von Lokalität*. Zusammenfassend kann die Performanz transitiv durch die folgenden Kriterien evaluiert werden:

- Reaktion auf Anfrage (wie im monolithischen System als Messung der Antwortzeit auf eine Anfrage)
- Widerstandsfähigkeit
- Elastizität
- Nachrichten-getrieben

Skalierbarkeit ist dabei nicht diskret, sondern kontinuierlich zu betrachten, da skalierbare Systeme durchaus eine Obergrenze für ihre Skalierbarkeit erreichen können, da sonst zum Beispiel die Netzwerklast durch viele Teilnehmer zu groß wird und das System zusammenbricht. Kriterien für ein unendlich skalierbares System befinden sich in [133]. Darunter befinden sich (Entitäten sind dabei Teilnehmer eines verteilten Systems und werden manchmal auch als Aggregate bezeichnet):

- Entitäten sind eindeutig definiert (analog zum Primary Key in Datenbanken)
- Mehrere unabhängige Anwendungsbereiche von Serialisierung (Transaktionen, Single Responsibility Principle)
- At-Least-Once messaging
- Nachrichten werden an Entitäten geschickt
- Entitäten verwalten den Zustand einer Konversation pro Gruppe (Menge von Entitäten)
- Verschiedene Indizes sind in unterschiedlichen Serialisierungsbereichen (Umkehrschluss: ein Index befindet sich in genau einem Serialisierungsbereich)

- Nachrichten zwischen Entitäten sind unverbindlich

10.3.3 Spezielle Performanzkriterien für Datenströme

Datenströme sollten darüber hinaus durch weitere Kriterien evaluiert werden, da an Datenströme weitere Kriterien gestellt werden. Nach [105] beschreibt eine geringe Latenz eine gute Performanz. Sie kann dadurch gemessen werden, dass die Zeit nach dem Eintreffen des ersten Datenelements bis zur Produktion des ersten Ergebniselements gemessen wird. Diese hängt nicht nur von den Eigenschaften des Datenstroms, sondern auch von der Anfrage [113]. Eine komplexe Anfrage, bestehend aus vielen zustandsbehafteten Operatoren, wird langsamer verarbeitet als eine einfache Anfrage, die die Daten nur durchleitet. Dabei ist nicht nur die logische Anfrage, sondern auch der physische Anfrageplan in Odysseus von Bedeutung. Daher können zur Evaluation Anfragen auf den Smart Grid Daten installiert werden, während die Zeit vom Eintreffen der Daten in das Kafka-Cluster bis zum Eintreffen der Ergebnisse ins Kafka-Cluster gemessen wird.

Die Daten des Datenstroms nach [4] haben auch einen Einfluss auf die Performanz der Verarbeitung, insbesondere die Typen der Attribute und die Anzahl der Attribute, da die Verarbeitung auf Attributen und die Übertragung der Elemente die Verarbeitung verlangsamen, wenn die Datenelemente insgesamt größer ausfallen. Dafür können Anfragen mit Smart Grid Daten und Daten aus dem Nexmark-Auction-Server genutzt werden, um die Latenzunterschiede zu messen.

Durch eine Verteilung des DSMS sollte auch eine unterschiedliche Verteilung der Anfragepläne evaluiert werden. Die Anfragen aus dem Evaluationsteil für Anfragen ohne DSMS-Cluster können hier nach unterschiedlichen Strategien verteilt werden. Dafür kann jeder Operator auf eine eigene Instanz gebracht werden, während inkrementell weitere Operatoren auf eine Instanz gebracht werden. Dabei kann die Performanz der unterschiedlichen Verteilungen miteinander verglichen werden, sodass am Ende der Evaluation eine ideale Verteilung für die evaluierten Anfragen zustande kommt.

Durch die kontinuierlichen Anfragen sollen auch kontinuierlich Ergebnisse produziert werden. Daher lässt sich die Performanz des Datenstroms laut [105] auch durch die Ergebnisproduktionsrate evaluieren. Dafür kann die Produktionsrate durch unterschiedliche Anfragen überprüft werden. Wichtig ist dabei, dass die Ergebnisrate des gesamten Systems betrachtet werden sollte und daher von den Typen der einzelnen Ergebnisse abhängt.

Load Shedding ist der Mechanismus für DSMS, um mit Überlastung zurecht zu kommen. Dabei ist ein System performant nach [113], wenn es unter Last, d.h. in diesem Fall mit einer sehr hohen Eingangsrate, eine geringe Load Shedding Rate hat und so performanter ist. Durch nicht deterministische Komponenten des DSMS werden auch Ergebnisse ungenauer, daher ist nach [4] ein indirektes Performanzkriterium eine hohe Genauigkeit der Daten nach der Berechnung. Diese kann durch eine endliche Datenmenge überprüft werden, die durch das DSMS verarbeitet wird, und zusätzlich wird das genaue Ergebnis dieser Datenmenge berechnet oder das Ergebnis ist bereits bekannt, um dann die Abweichung der Ergebnisse voneinander zu bestimmen. Zusammengefasst ergeben sich die folgenden zusätzlichen Kriterien für die Datenstromevaluation:

- Verzögerung von Datenausgabe zur Dateneingabe (output delay)
  - Abhängig von der Art der Anfrage
    - Abhängig von der Zahl der Operatoren

        * Abhängig vom Anfrageplan
        * Abhängig von den Operatortypen
    - Abhängig von den Daten des Stroms
        * Abhängig vom Typ der Attribute
        * Abhängig von der Anzahl an
    - Abhängig von der Datenstromverteilung
- Produzierte Ergebnisse pro Zeit (output rate)
    - Abhängig von der Art der Ausgabe: Typ und Anzahl an Ausgaben
- Nicht verarbeitete Tupel (load shedding rate)
- Genauigkeit von Ergebnissen

10.3.4 Kommunikationsmechanismen

Kommunikationsmechanismen um Performanzproblemen vorzubeugen sind zum Beispiel backpressure, bzw. Circuit Breaker aus [155], da ein skalierendes System (das verarbeitende System) mit einem nicht unbedingt skalierbaren System (verwaltendes System) das nicht skalierbare System überlasten kann. Durch Kommunikationsmechanismen ist die Performanz des verwaltenden Systems garantiert. Backpressure ist ein Konzept, bei dem der Empfänger eines Datenstroms einen Kommunikationskanal zum Sender hat und bei Überlastung des Empfängers dieses durch eine Nachricht an den Sender signalisieren und der Sender nach Konvention seine Datenrate senkt bis der Empfänger nicht mehr überlastet ist. Circuit Breaker ist eine Lösung für Überlastung und Fehler, falls der Sender die Konvention nicht einhält. Ein Circuit Breaker ist ein endlicher Automat (FSM) mit den Zuständen Geschlossen, Halb-offen, Offen. Der Standardzustand ist Geschlossen. In diesem Zustand unternimmt der Circuit Breaker nichts und lässt alle Nachrichten des Senders durch. Wird der Empfänger überlastet, dann geht der Circuit Breaker in den Zustand Offen und lässt keine Nachrichten mehr durch. Nach einem Timeout geht der Circuit Breaker in den Zustand Halb-Offen und lässt einen Teil der Nachrichten durch. Wird der Empfänger immer noch überlastet, dann geht der Circuit Breaker wieder in den Zustand Offen. Sollte alles in Ordnung sein, dann wechselt der Circuit Breaker wieder in den Zustand Geschlossen. Für eine gute Performanz sollten beide Konzepte umgesetzt sein, sollte dies nicht möglich sein, sind die Circuit Breaker wichtiger, da dieser Mechanismus nur den Client betrifft.

10.4 Umsetzung der Performance Evaluation

10.4.1 Verwendete Hardware

Die Evaluation des Systems wurde auf einem Rechner ausgeführt, der zur Ausführung der Simulation verwendet wurde und eine virtuelle Maschine beinhaltet. Dabei besitzen der Rechner folgende Kenngrößen:

- **Prozessor** Intel Core i7-4710HQ mit 4 Kernen und 2,50 GHz
- **Arbeitsspeicher** 16GB DDR3

- **Festplatte** Samsung SSD 850 Pro 512 GB
- **Grafikkarte** Intel HD Graphics 4600

und die verwendete virtuelle Maschine:

- **Prozessor** Intel Xeon E5645 mit 2 Kernen und 2,40 GHz
- **Arbeitsspeicher** 16GB DDR3
- **Festplatte** 400GB HDD

10.4.2 Durchführung der Performance Evaluation für das Gesamtsystem

Die Evaluation der Performance des E-Stream Systems sollte insbesondere die benötigte Zeit von der Erstellung eines Datensatzes in der mosaik Simulation bis zur Visualisierung der Daten im Frontend umfassen. Dafür wird zunächst ein entsprechender Zeitstempel in der Simulation gesetzt, welcher zusammen mit den Smart Meter Daten über den Kafka-Producer an Odysseus gesendet und dort verarbeitet wird. Die in Odysseus verarbeiteten Daten sollten anschließend wieder über Kafka an das Backend und von dort aus über eine WebSocket-Verbindung an das Frontend gesendet werden. Um dies zu erreichen, wurden mehrere Anpassungen an der Simulation und dem E-Stream Backend sowie die Ausführung entsprechender Anfragen zur Messung der Performance durchgeführt. Die Experimente wurden mehrfach wiederholt und dementsprechend der Durchschnittswert für die Messergebnisse gebildet.

Um die Ausgabe der Simulation mit einem Zeitstempel versehen zu können, musste zunächst ein passender Zeitpunkt für das Setzen des Zeitstempels gefunden werden. Da die Simulation selbst nicht Bestandteil der Evaluation sein sollte, musste der Zeitstempel hier möglichst spät gesetzt werden. Daher wurde im *cosem_xml.py*-Script ein entsprechendes Attribut dem COSEM-Schema hinzugefügt. Damit entspricht die Ausgabe der Simulation zwar nicht dem COSEM-Standard, doch da dieses zusätzliche Attribut ausschließlich in der Evaluationsphase genutzt wurde und in der finalen Version nicht mehr vorhanden ist, wurde dies als unkritisch betrachtet. In Listing 10.1 ist das Setzten des Zeitstempels in der *cosem_xml.py* dargesellt.

```
1     attribute = ET.SubElement(attributes, 'attribute')
2     attribute.set('id', "6")
3     value = ET.SubElement(attribute, 'value')
4     doublelongunsigned = ET.SubElement(value, 'double-long-unsigned')
5     doublelongunsigned.text = str(int(round(time_.time() * 1000)))
```

Listing 10.1: *Zeitstempel in der Simulation für die Evaluation*

Die erweiterte COSEM-Nachricht wird über den Kafka-Producer an einen Kafkatopic gesendet, welche anschließend vom Odysseus-System gelesen und verarbeitet wird. Die entsprechende Anfrage ist in Unterunterabschnitt 10.4.2.1 zu sehen. Bei der Datenverarbeitung wird am Ende ein weiterer Zeitstempel gesetzt und die verarbeiteten Daten auf ein Kafkatopic geschrieben. Um die Daten im Frontend auswerten zu können, wurde eine neue Komponente angelegt. Die Komponente berechnet aus dem aktuellen Zeitpunkt und dem letzten gesetzten Zeitstempel die Differenz, um die Dauer der Verarbeitung zu bestimmten. Hierbei wird die Zeit vom Erstellen eines Datensatzes (Simulation) bis hin zum Empfang der Daten (Frontend) betrachtet.

Im Gegensatz dazu, ist es denkbar, dass das System auch vollständig verteilt evaluiert werden könnte, indem die Simulation, Odysseus, das Backend-System und das Frontend separat betrachtet werden. Hierfür müssten die unterschiedlichen Teilsyteme auf eigenen Maschinen ausgeführt werden. Die Kommunikation zwischen den Teilsystemen ist durch die Verwendung von Apache Kafka und WebSockets für diesen Anwendungsfall ausgelegt. Die Evaluation basiert jedoch auf den Zeitstempeln der jeweiligen Systemzeiten, daher wurde die verteilten Evaluation nicht umgesetzt, um eine mögliche Ungenauigkeit bei den Messungen zu vermeiden. Hierfür müssten nämlich alle Maschinen synchronisiert werden, um gleiche Systemzeiten zu erhalten. Damit die Ergebnisse dennoch den zu erwartenden Werten bei Einsatz eines verteilten Systems entsprechen, fand die Kommunikation über einen externen Kafka-Broker statt, sodass die Dauer auch das Senden der jeweiligen Datenpakete über das Internet beinhaltet.

10.4.2.1 Anfragen für Odysseus

Für ein aussagekräftiges Ergebnis sollten verschiedene Durchläufe zu unterschiedlichen Szenarien durchgeführt werden. Das einfachste Szenario besteht aus der reinen Übertragung der Daten auf ein Kafkatopic, ohne dass eine Datenverarbeitung durch das Odysseus-System durchgeführt wird. Das zweite Evaluationsszenario betrachtet Bildung von Aggregationen, wie Summe, Durchschnitt und Varianz. Das Dritte Szenario betrachtet die eine Regression, wie sie auch vom Produktivsystem zur Prognose von Verbrauch oder Erzeugung verwendet wird. In Listing 10.2 sind die Datenquellen zum Empfang der Rohdaten sowie der Topologie abgebildet, welche sich von den sonst im System verwendeten Datenquellen lediglich durch die verwendeten Topics (*eval_rawdata* und *eval_topology* sowie dem zusätzlichen Attribut *sending_time* unterscheidet. Bei letzterem handelt es sich um den in der mosaik Simulation gesetzten Zeitstempel.

```
#PARSER PQL
#RUNQUERY
input_topology_eval := ACCESS({
                source='input_topology_eval',
                protocol='Topology',
                transport='Kafka',
                wrapper='GenericPush',
                datahandler='Tuple',
                        options=[
                                ['topic', 'eval_topology'],
                                ['messagetype', 'string'],
                                ['bootstrap.servers', '134.106.56.15:29092'],
                                ['auto.offset.reset', 'earliest'],
                                ['type', 'smartmeters']
                                ], schema=[
                                        ['smartmeter','String'],
                                        ['gateway', 'String'],
                                        ['type', 'String']
                                        ]
        }
)

#PARSER PQL
#RUNQUERY
input_rawdata_eval := ACCESS({
        source='input_rawdata_eval',
        protocol='XML',
```

```
28          transport='Kafka',
29          wrapper='GenericPush',
30          datahandler='Tuple',
31          options=[
32                  ['topic', 'eval_rawdata'],
33                  ['messagetype', 'string'],
34                  ['group.id', 'cosem_mosaik_1521654083892'],
35                  ['auto.offset.reset', 'latest'],
36                  ['bootstrap.servers', '134.106.56.15:29092'],
37                  ['sm_id','/cosem/ldevs/ldev/@id'],
38                  ['obis_id','/cosem/ldevs/ldev/objects/object/attributes/
39                          attribute[@id=\'1\']/value/string'],
40                  ['value','/cosem/ldevs/ldev/objects/object/attributes/
41                          attribute[@id=\'2\']/value/float64'],
42                  ['scaler','/cosem/ldevs/ldev/objects/object/attributes/
43                          attribute[@id=\'3\']/value/integer'],
44                  ['unit','/cosem/ldevs/ldev/objects/object/attributes/
45                          attribute[@id=\'3\']/value/enum'],
46                  ['status','/cosem/ldevs/ldev/objects/object/attributes/
47                          attribute[@id=\'4\']/value/enum'],
48                  ['sending_time','/cosem/ldevs/ldev/objects/object/attributes/
49                          attribute[@id=\'6\']/value/double-long-unsigned'],
50                  ['capture_time','/cosem/ldevs/ldev/objects/object/attributes/
51                          attribute[@id=\'5\']/value/double-long-unsigned'],
52                  ['path', '/cosem/']
53                  ],
54          schema=[
55                  ['sm_id','String'],
56                  ['obis_id','String'],
57                  ['value','Double'],
58                  ['scaler','Integer'],
59                  ['unit','Integer'],
60                  ['status','String'],
61                  ['sending_time','Long'],
62                  ['capture_time','Long']
63                  ]
64          }
65  )
```

Listing 10.2: *Datenquellen für die Simulation*

In Listing 10.3 ist die Anfrage für das erste Szenario dargestellt. Zunächst werden die Daten der Simulation mit den Daten der Topologie angereichert. Dies ist zwar für die weitere Ausführung der Anfrage nicht relevant, doch wird dieser Schritt auch bei der Ausführung normaler Anfragen ausgeführt und die benötigte Zeit sollte daher ebenfalls bei der Evaluation berücksichtigt werden. Anschließend wird in der dargestellten Anfrage lediglich die *sending_time* in ein Key-Value-Schema überführt, um sie im JSON-Format übertragen zu können. Die in Listing 10.4 und Listing 10.5 abgebildeten Anfragen für das zweite und dritte Szenario beginnen ebenfalls mit der Anreicherung der Daten und Enden mit der Umwandlung des Ergebnisses in das Key-Value-Schema sowie das Senden an einen Kafkatopic, weshalb diese Schritte im Folgenden nicht weiter betrachtet werden.

```
#PARSER PQL
#RUNQUERY
enrich_eval = ENRICH({
        minimumsize = 1, predicate = 'smartmeter = sm_id'
        }, System.input_topology_eval,
           System.input_rawdata_eval
)

result_to_kv = TOKEYVALUE(MAP({
        expressions = [ 'sending_time' ]
        }, enrich_eval
        )
)

send_evaluation_a = SENDER({
        protocol = 'JSON',
        transport = 'Kafka',
        sink = 'send_evaluation_a',
        wrapper = 'GenericPush',
        options = [
                ['topic', 'evaluation'],
                ['messagetype', 'string'],
                ['bootstrap.servers', '134.106.56.15:29092']
                ]
        }, result_to_kv
)
```

Listing 10.3: *Evaluation der Rohdaten*

Die in Listing 10.4 dargestellte Anfrage entspricht im Wesentlichen einer Anfrage, wie sie vom E-Stream Templatesystem auch für das einfache Data-Mining generiert werden kann. Unterschiede können sich insbesondere durch Anpassungen, welche der Nutzers in Bezug auf die gewählte Größe des Fensters durchführen könnte, abbilden. Zusätzlich wurde der Aggregation noch die maximale *sending_time* hinzugefügt, um diese im Frontend verwenden zu können.

```
///
input_data_aggregation = TIMESTAMP({
        start = 'capture_time'
        }, enrich_eval
)

input_data_window = TIMEWINDOW({
        size = [1100, 'MILLISECONDS'],
         advance = [1, 'MILLISECONDS']
         }, input_data_aggregation
)

input_selection = SELECT({
        predicate='type = "consumer"'
        }, input_data_window
)

input_aggregation  = AGGREGATION({
        aggregations=[
                ['FUNCTION' = 'Sum', 'INPUT_ATTRIBUTES' = 'value',
                        'OUTPUT_ATTRIBUTES' = 'sum_value'],
```

```
22                  ['FUNCTION' = 'Avg', 'INPUT_ATTRIBUTES' = 'value',
23                          'OUTPUT_ATTRIBUTES' = 'avg_value'],
24                  ['FUNCTION' = 'Min', 'INPUT_ATTRIBUTES' = 'value',
25                          'OUTPUT_ATTRIBUTES' = 'min_value'],
26                  ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'value',
27                          'OUTPUT_ATTRIBUTES' = 'max_value'],
28                  ['FUNCTION' = 'Variance', 'INPUT_ATTRIBUTES' = 'value',
29                          'OUTPUT_ATTRIBUTES' = 'var_value'],
30                  ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'sending_time',
31                          'OUTPUT_ATTRIBUTES' = 'sending_time']
32                  ]
33          }, input_selection
34  )
35  ///
```

Listing 10.4: *Evaluation der Aggregation*

Auch Listing 10.5 entspricht einer Anfrage, wie sie vom E-Stream System generiert werden kann. Es ergeben sich die gleichen Einschränkungen, die bereits für Listing 10.4 erläutert wurden. Abweichungen können sich also im Wesentlichen durch eine andere Fenstergröße ergeben, welche in diesem Szenario auf 5000 Millisekunden gesetzt ist und dementsprechend auch Einfluss auf das Ergebnis haben kann.

```
1   ///
2   classification_eval_a = TIMESTAMP({
3           start = 'capture_time'
4           }, enrich_eval
5   )
6
7   classification_eval_b = TIMEWINDOW({
8           size = [5000, 'MILLISECONDS'],
9           advance = [1, 'MILLISECONDS']
10          }, classification_eval_a
11  )
12
13  classification_eval_c = SELECT({
14          predicate='type = "consumer"'
15          }, classification_eval_b
16  )
17
18  classification_eval_aggregate  = AGGREGATION({ aggregations=[
19          ['FUNCTION' = 'Sum', 'INPUT_ATTRIBUTES' = 'value',
20                  'OUTPUT_ATTRIBUTES' = 'sum_value'],
21          ['FUNCTION' = 'Avg', 'INPUT_ATTRIBUTES' = 'value',
22                  'OUTPUT_ATTRIBUTES' = 'avg_value'],
23          ['FUNCTION' = 'Min', 'INPUT_ATTRIBUTES' = 'value',
24                  'OUTPUT_ATTRIBUTES' = 'min_value'],
25          ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'value',
26                  'OUTPUT_ATTRIBUTES' = 'max_value'],
27          ['FUNCTION' = 'Variance', 'INPUT_ATTRIBUTES' = 'value',
28                  'OUTPUT_ATTRIBUTES' = 'var_value'],
29          ['FUNCTION' = 'Max', 'INPUT_ATTRIBUTES' = 'sending_time',
30                  'OUTPUT_ATTRIBUTES' = 'sending_time']
31      ]
32     }, classification_eval_c
33  )
```

```
34
35 classification_eval_learn = CLASSIFICATION_LEARN({
36         class= 'sum_value',
37         learner = 'weka',
38         algorithm = 'linear-regression'
39         },
40     MAP({
41         expressions=[
42         'sending_time', 'sum_value',
43         ['Timeinterval.start','capture_time'] ]
44         },
45         ELEMENTWINDOW({
46         size = 2,
47         advance = 1
48         },
49         classification_eval_aggregate )
50     )
51 )
52
53 classification_eval_e = JOIN({
54         predicate = 'type = "consumer"'
55     },
56     CLASSIFY({
57                 classifier='classifier',
58         classname='prediction'
59     },
60     MAP({
61                 expressions=[
62                 ["DoubleToLong(capture_time + 10000)", "capture_time"]
63                 ]
64         },
65         classification_eval_c),
66         classification_eval_learn
67         ),
68         MAP({
69                 expressions=[
70                 'sending_time', ['capture_time','capture_time_right'],
71                 "gateway", "sm_id", "value", "scaler", "unit", "status",
72                 "type" ]
73         },
74         classification_eval_c )
75 )
76 ///
```

Listing 10.5: *Evaluation der Klassifikation*

10.4.2.2 Ergebnisse für das Gesamtsystem

Nach der Ausführung der unterschiedlichen Szenarien wurden die ersten 1000 erzeugten Zeitwerte genommen, um die Latenz des Systems, von der Übertragung eines Datenwertes von der Simulation bis hin zur Darstellung im Frontend, zu bestimmen. Dabei konnte festgestellt werden, dass je nach Bearbeitung der Datensätze die Latenz bis zur Anzeige im Frontend steigt. Durchschnittlich braucht ein Datensatz ohne weiterer Bearbeitung 0,232 Sekunden bis es im Frontend angezeigt wird. Bei einer Verarbeitung mit einer Aggregation auf den Datensätzen steigt die Dauer bis zur Anzeige im Frontend

auf durchschnittlich 1,136 Sekunden. Die längste Dauer wird bei der Verwendung der Regression erreicht. Dort dauert die Spanne von der Erzeugung bis zu Anzeige im Frontend durchschnittlich 4,386 Sekunden, was in Abbildung 10.2 dargestellt ist.

Ebenfalls ist die Mindest- und Höchstdauer der Datensätze zu beachten. Während bei Rohdaten und aggregierten Daten die Mindestdauer bei wenigen Millisekunden liegt, 9ms und 12ms, liegt die Mindestdauer bei Anfragen mit einer laufenden Regression bei 2,664 Sekunden. Die Höchstdauer bis zur Darstellung im Frontend ist bei allen drei Anfragetypen auf über 4 Sekunden. Bei Rohdaten ist die höchste gemessene Zeit bei 4,667 Sekunden, bei einer laufenden Aggregation bei 5,572 Sekunden und und bei einer laufenden Regression liegt die Höchstdauer bei 6,291 Sekunden. Gerade bei keiner weiteren Verarbeitung der Daten, sondern der bloßen Anzeige der Rohdaten, fällt die Differenz zwischen Minimum und Mittelwert im Vergleich zum Maximum sehr groß aus. Da bei dieser Messung alle Komponenten des Systems in Betrieb sind, wäre eine weitere Untersuchung der Differenz bei einer Fortführung des Projekts interessant, um eine Abgrenzung treffen zu können, welche Komponente des Gesamtsystems für diese hohe Latenz verantwortlich ist.

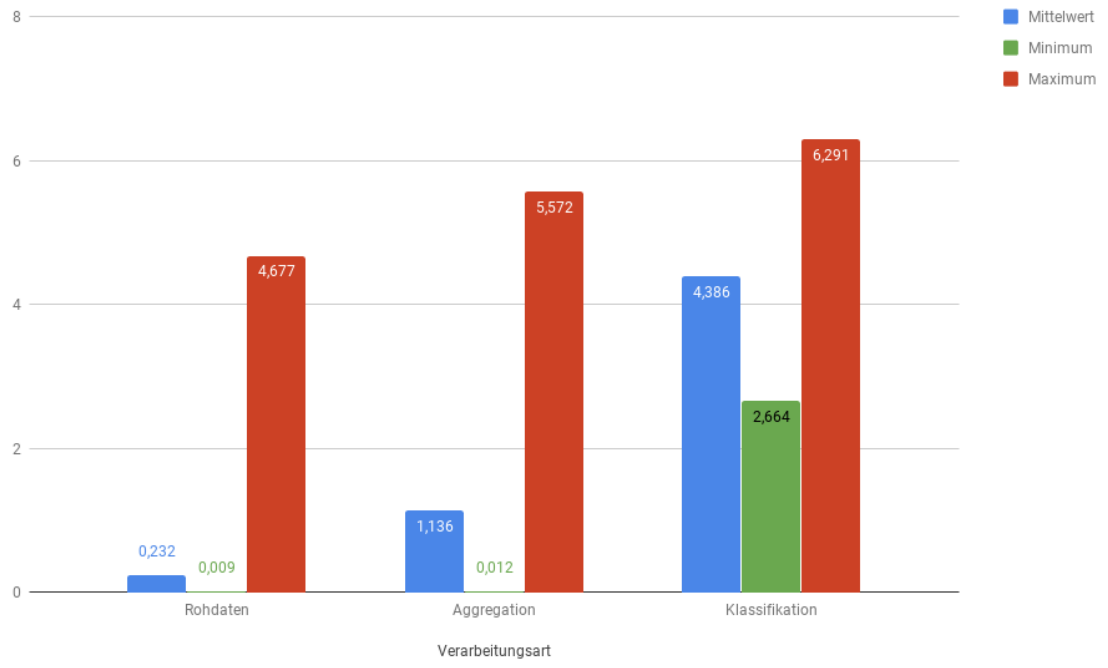


Abbildung 10.2: *Dauer von der Übermittlung eines Datenwertes bis zur Darstellung im Frontend*

10.4.3 Durchführung der Performance Evaluation für das Frontend

Neben der Latenzmessung des Gesamtsystems, wurde für die Evaluation ebenfalls die Performance des Frontends getestet, um zu überprüfen, ob bei verschiedener Auslastung des Frontends das System weiterhin für den Benutzer bedienbar bleibt. Dafür wurde die Anzahl der Bilder pro Sekunde gemessen.

Im ersten Szenario wurde beginnend mit einer Linechart-Visualisierung in einem Dashboard die Anzahl der Linechart-Visualisierungen um jeweils eine erhöht. Dabei wurden für zwei Minuten die Bildrate in fps gemessen. Im zweiten Szenario wurde die Performance des Frontends gemessen, indem in der Topologie-Ansicht unterschiedliche Filter vorgenommen wurden und dementsprechend unterschiedlich viele Elemente auf der Karte zur gleichen Zeit sichtbar sind. Es wurden die fps gemessen, solange sich die gezeichnete Karte noch nicht stabilisiert hat. Denn es gilt zu beachten, dass sobald sich die Karte stabilisiert hat, gibt es keine Änderungen mehr in der dargestellten Ansicht und somit ist die Bildrate konstant bei 60fps. Deswegen wurde für die Evaluation nur der Zeitpunkt überprüft, in der die Topologie-Ansicht neu gezeichnet wird, da das einen Einfluss auf die Wahrnehmung des Benutzers haben kann. Wie in [194] beschrieben, hat eine Abnahme der Bildrate auf unter 2fps eine Beeinträchtigung der Ausführung der Aufgaben des Benutzers zur Folge. Wohingegen die Arbeit ab 4fps zwar auch beeinträchtigt ist, diese aber nicht so stark ins Gewicht fällt, wie bei unter 2fps. Bildraten von 15 fps und höher haben weniger bis gar keinen Einfluss auf den Nutzer, so dass die Bearbeitung von Aufgaben unbeeinträchtigt geschehen kann, weshalb eine stetige Bildrate bei 15fps erwünscht ist.

Bei der Evaluation der Performance im Dashboard, wurde wie oben beschrieben, damit angefangen eine Linechart-Visualisierung zum Dashboard anzuzeigen und die fps für zwei Minuten aufgezeichnet. Anschließend wurde eine weitere Linechart hinzugefügt, die die Ergebnisse der gleichen laufenden Query visualisiert und ebenfalls die Bildrate der Ansicht mit diesen zwei Visualisierungen gemessen. Dies wurde solange durchgeführt bis acht Linechart-Visualisierungen im Dashboard zu sehen waren. In Abbildung 10.3 ist das Dashboard mit einer Visualisierung zu sehen, was für die Evaluation benutzt wurde und in Abbildung 10.4 das Dashboard mit vier Visualisierungen. Ab vier Visualisierungen waren nicht mehr alle Komponenten des Dashboards auf der Ansicht zu sehen.

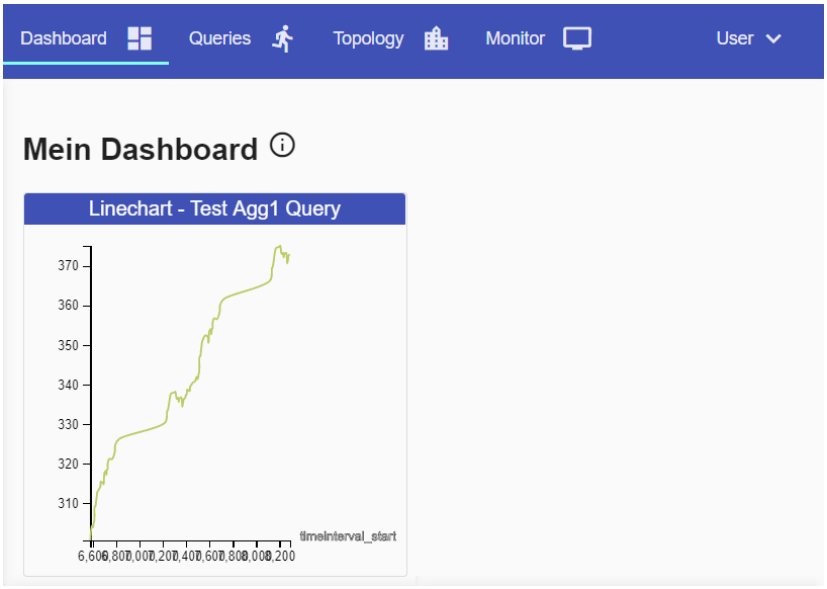


Abbildung 10.3: *Erste Einstellung des Dashboards mit einer Linechart-Visualisierung*

Abbildung 10.4: *Vierte Einstellung des Dashboards mit vier Linechart-Visualisierungen*

In Abbildung 10.5 sind die gemessenen Bildraten dargestellt. Dabei ist zu sehen, dass die durchschnittliche Bildrate bei fast der maximalen Bildrate von 60fps liegt (da es in allen Messungen mindestens einen Zeitpunkt gab, wo die Ansicht sich nicht aktualisiert hat, haben alle Messungen 60fps als höchsten Wert und deshalb wurde dieser in der Abbildung weggelassen). Allerdings ist auch ein Trend nach unten bei steigender Anzahl der Visualisierungen zu erkennen. Zu beachten ist auch, dass die minimale Bildrate bei steigender Anzahl an Visualisierungen sinkt. Jedoch sinkt sie nicht unterhalb von 2fps, so dass das Arbeiten ohne starke Beeinträchtigung für den Benutzer weiterhin möglich ist. Da diese Bildrateneinbrüche nur beim Aktualisieren der Visualisierung auftreten können, sind sie ebenfalls nur von kurzer Dauer, so dass es auch für den Benutzer kaum merkbar sein sollte, selbst wenn kurzzeitig eine sehr geringe Bildrate vorliegt.

Neben der Performance mit mehreren Visualisierungen im Dashboard wurde ebenfalls, wie oben beschrieben, die Topologie-Ansicht auf die Performance evaluiert. Innerhalb dieses Testszenarios wurde die Topologie ausgeblendet, so dass nur das Root-Element zu sehen ist. Anschließend wurde in die Filter-Komponente der Ansicht ein Filterkriterium eingegeben, so dass je nach Filter nur bestimmte Elemente der Topologie angezeigt werden. Da die Strukturierung der Topologie dynamisch ist, muss die Topologie sich nach dem Anzeigen der Elemente neu strukturieren. Deshalb wurden drei verschiedene Suchbegriffe verwendet, um die Anzahl der dargestellten Elemente zu beeinflussen und so die Last auf das Frontend mit steigender Anzahl an dargestellten Elementen zu erhöhen. Die Suchbegriffe waren:

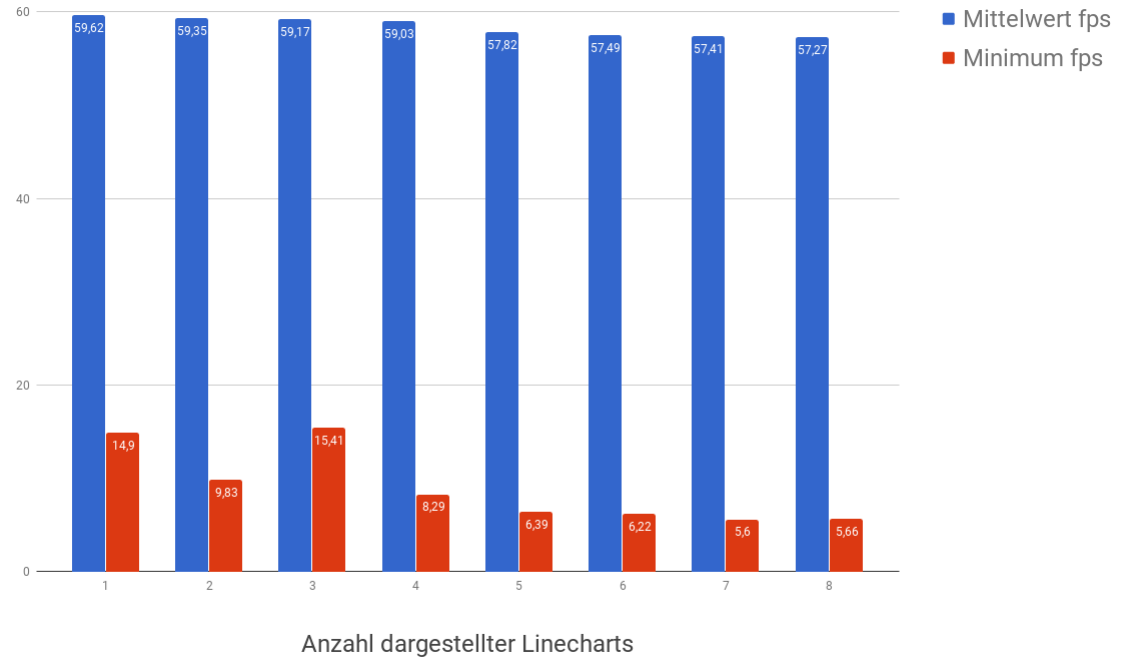


Abbildung 10.5: *Bildrate gemessen in fps aufgeteilt nach Anzahl der Linechart-Visualisierungen*

- **Transformator** - Bei dieser Filtereinstellung werden alle Transformatoren mit ihren Kindelementen (Node) dargestellt, wie in Abbildung 10.6 dargestellt. Es waren insgesamt 79 Elemente in der Ansicht sichtbar.
- **Node** - Bei dieser Filtereinstellung werden alle Transformatoren, Nodes und die an den Nodes liegenden SMGWs dargestellt, wie in Abbildung 10.7 zu sehen ist. In dieser Einstellung wurden insgesamt 159 Elemente gezeigt.
- **Smgw** - Bei dieser Filtereingabe waren alle Elemente der Topologie dargestellt. Das ist in Abbildung 10.8 zu sehen. Insgesamt wurden 338 Elemente angezeigt.

Bei der Durchführung der Evaluation war deutlich zu erkennen, dass je mehr Elemente in der Topologie-Ansicht dargestellt werden, desto mehr sinkt auch die Bildrate. Eine Übersicht ist in Abbildung 10.9 dargestellt.

Bei 79 angezeigten Elementen ist die Bildrate durchschnittlich bei 34,04fps und sinkt auf ein Minimum von 3,90fps. Damit ist die Ansicht in den 9,75 Sekunden noch ausreichend reaktiv, damit der Benutzer sie weiterhin benutzen kann. Bei der Filterung nach *Node* sinkt der Mittelwert der Bildrate unter 30fps und liegt bei 22,43. Werden alle 338 Elemente der benutzten Topologie angezeigt, liegt der Mittelwert der Bildrate während sich die Topologie ausrichtet bei 15,78fps. Zu beachten ist, dass

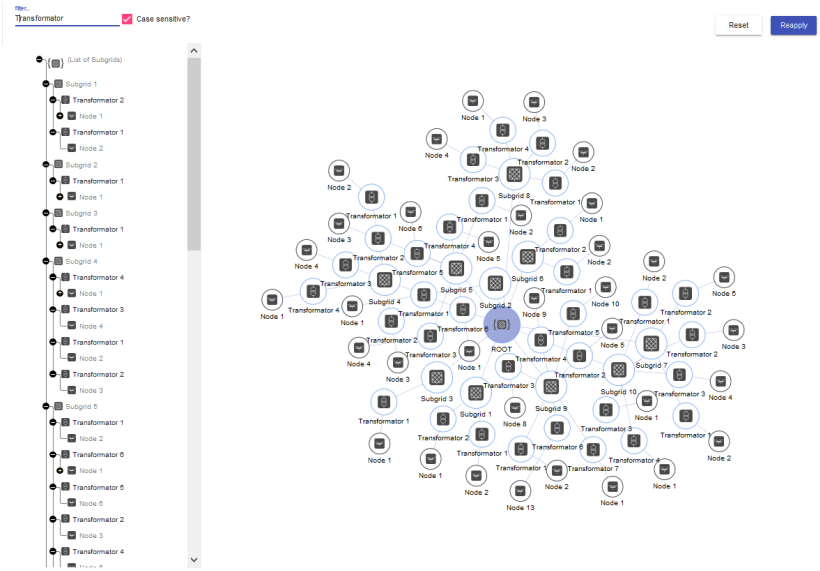


Abbildung 10.6: *Ansicht der Topologie gefiltert nach Transformatoren. Insgesamt werden 79 Elemente dargestellt.*

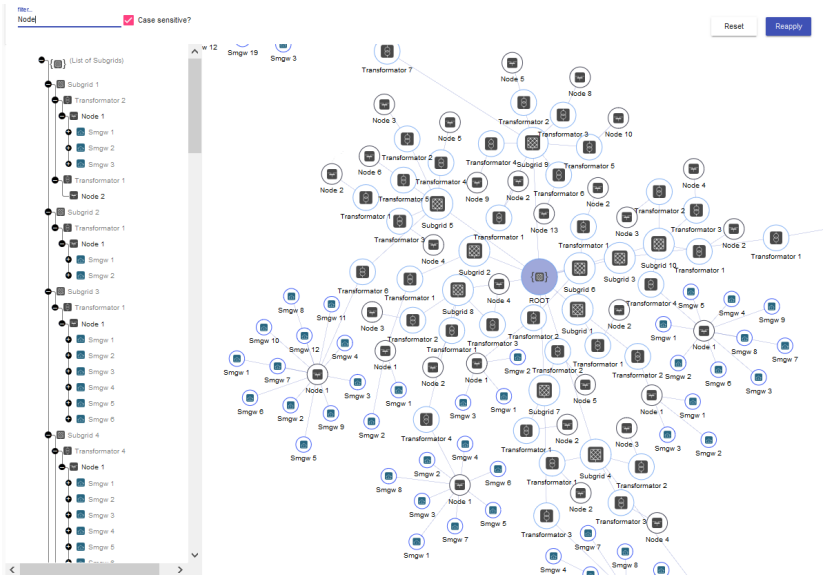


Abbildung 10.7: *Ansicht der Topologie gefiltert nach Nodes. Insgesamt werden 159 Elemente dargestellt.*

die Minimum Werte der Bildrate bei den Filtereinstellungen für *Node* und *Smgw* beide unter 2fps liegen und somit eine deutliche Beeinträchtigung der Ausführung von Aufgaben auftreten kann, wie in [194] beschrieben ist. Allerdings ist es fraglich wie realistisch es ist, dass ein Nutzer alle Elemente gleichzeitig anzeigen lässt, da durch die Anzahl nicht alle Elemente in der Ansicht dargestellt werden

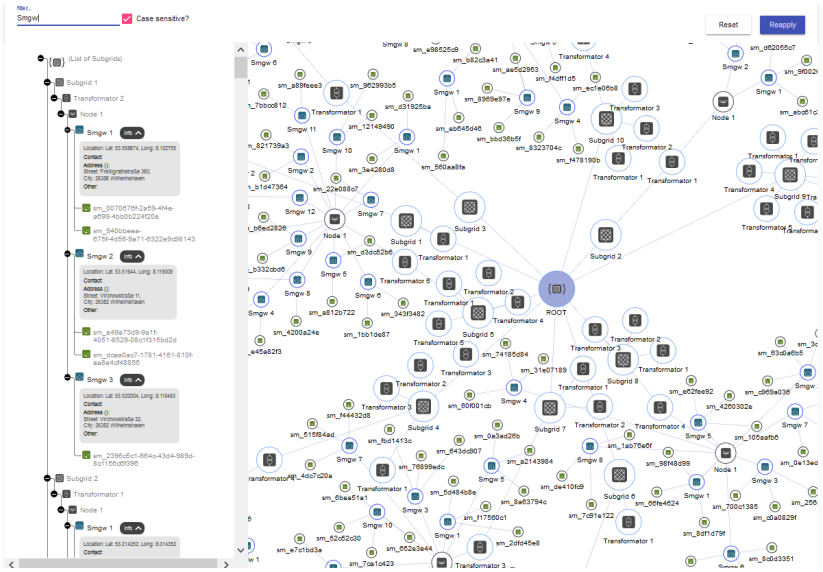


Abbildung 10.8: *Ansicht der Topologie gefiltert nach SMGW. Bei dieser Filtereinstellung werden alle 338 Elemente der benutzten Topologie dargestellt.*

können und somit ein effektives Arbeiten ebenfalls nicht stattfinden kann. Ebenfalls ist zu erkennen, dass je nach Anzahl der angezeigten Elemente die Dauer für die Neuausrichtung der Topologie steigt.

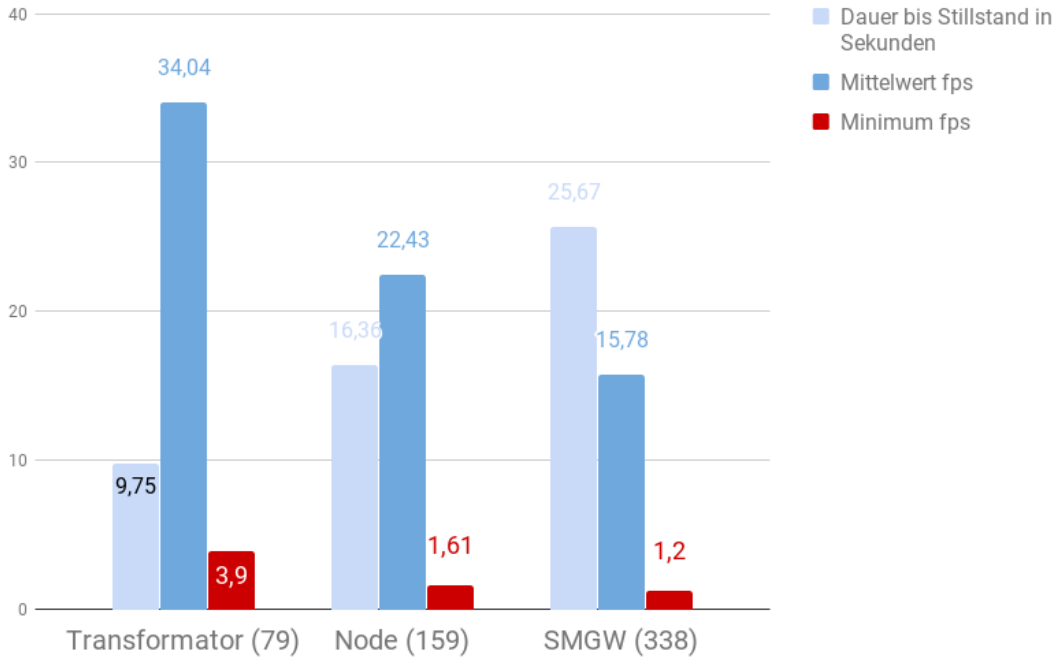


Abbildung 10.9: *Ergebnisse für die Bildrate in fps während die Topologie-Ansicht sich neu berechnet unterteilt nach den Filtereinstellungen mit angezeigten Elementen pro Filtereinstellung.*

11 Ausblick

11.1 Sicherheit des Systems

11.1.1 Kafka Authentifizierung

In Unterunterabschnitt 6.3.3.1 wurden bereits verschiedene Möglichkeiten erläutert Authentifizierungsmechanismen in Apache Kafka umzusetzen. Während der Entwicklung des E-Stream-Systems wurde hierauf jedoch verzichtet, um keine zusätzlichen Hindernisse beim Aufbau der Infrastruktur zu schaffen. Zudem wurde die Priorität als niedrig eingestuft, da es sich hierbei in erster Linie um die reine Konfiguration von Apache Kafka handelt, welches nicht zu den Kern-Komponenten des von uns entwickelten E-Stream-Systems zählt.

Bei einer Weiterentwicklung des Systems ist auf eine entsprechende Absicherung der Kommunikation zwischen Kafka-Broker und Kafka-Client jedoch zu achten. Insbesondere, da der Datenaustausch mit dem Datenarchiv eine Kommunikation über das Internet erfordert, muss sichergestellt sein, dass der Kafka-Server über das Internet erreichbar ist, gleichzeitig jedoch Unbefugte keinen Zugriff auf die Daten erhalten können.

Dabei ist nicht nur der reine Zugriffsschutz, sondern zudem ein Rechtemanagement für die Verwendung von Kafka notwendig. Hierdurch kann festgelegt werden, welcher Client in welcher Form auf einzelne Topics zugreifen darf. So ist beispielsweise ein Anwendungsfall, dass die Projektgruppe *DAvE* die Smartmeter-Rohdaten speichern soll. Sie muss daher die Daten eines entsprechenden Kafka Topics abfragen können, soll an diesem jedoch keine Änderungen vornehmen dürfen. Gleichzeitig soll die Projektgruppe PMML-Daten über ein anderes Topic von der Projektgruppe *DAvE* empfangen können, auf welchem ebenfalls sonst keine Änderungen vorgenommen werden sollen. Wenn Kafka Rechte für jeden Client über ein Rechtemanagement verwaltet, können nicht zugelassene Zugriffe auf Topics vermieden und so eventuellen Datenverlusten oder Dateninkonsistenzen in den Topics vorgebeugt werden.

11.2 Datamining Schnittstelle

In diesem Abschnitt wird dargestellt, welche Use Cases in Kooperation mit einem Langzeitdatenarchiv weiterentwickelt werden könnten und welche Zwecke dies erfülle. Die Schnittstellen für alle Use Cases wurden dabei in beide Richtungen umgesetzt, jedoch wurden nur die obligatorischen Use Cases *Senden von Rohdaten zur Speicherung* (siehe Tabelle 14.24), *Erstellen von Mining Model Aufträgen* (siehe Tabelle 14.23), *Abfrage von Mining Modellen* (siehe Tabelle 14.22) wie bereits in Abschnitt 14.1 aufgezeigt, in ihrer Grundfunktionalität implementiert.

11.2.0.1 Senden von Rohdaten zur Speicherung

Derzeit ist es einem Langzeitdatenarchiv somit nur gestattet, wie in der Anforderungsanalyse erhoben, die Smart Meter Rohdaten über das „subscriben“ auf einen bereitgestellten Kafka Topic zu erhalten. Es wäre jedoch auch denkbar über ein erweitertes Rollen und Rechtemanagement (siehe Unterabschnitt 11.4.2) verschiedenen Langzeitdatenarchive und ihrem dazugehörigen Account diverse aufbereitete/auf dem Datenstrom geminte Daten über weitere Topics bereitzustellen. Hierfür müsste

der REST Endpunkt „/topic/“ erweitert werden. Erstens um eine Resource zur Abfrage von Topics die dem jeweiligen Langzeitdatenarchiv die ihm erlaubten, derzeit zur Verfügung stehenden Datenanalysen auflistet und eine weitere Resource zum Erhalt eines passenden Kafka Topics zu einer aufbereiteten Datenquelle durch Angabe des Datenanalysetyps.

11.2.0.2 Abfrage von historischen Smart Meter Daten

Die Abfrage von historischen Smart Meter Daten von einem Langzeitdatenarchiv wurde in der Schnittstellendefinition festgelegt, jedoch war im Rahmen dieser Arbeit keine Zeit mehr vorhanden das Abfragen von historischen Daten sinnvoll zu implementieren. Dabei wäre es nun noch möglich zum Beispiel im Falle einer Störungserkennung (siehe Tabelle 14.20) einem Nutzer des Systems die Möglichkeit zu bieten, einen Zeitraum und den betreffenden Stromnetzbereich anzugeben und für diesen, historischen Daten abzufragen, um damit auf diesen Daten weitere bzw. noch nicht angewandte Analysen durchführen zu können.

11.2.0.3 Abfrage von historischen Wetterdaten

Für die Abfrage von historischen Wetterdaten, gilt dasselbe wie für die Abfrage von historischen Smart Meter Daten, denn der Anwendungsfall ist der gleiche. Es wäre durch die Abfrage von historischen und Smart Meter Daten in Kombination mit historischen Wetterdaten durch die Ablage auf einen Kafka Topic auch möglich erweiterte und differenzierte Analaysen auf diesen kombinierten Datenströmen durchzuführen.

11.2.0.4 Erstellen von Mining Model Aufträgen

Der Anwendungsfall des Erstellens von Mining Model Aufträgen wurde umgesetzt. Dabei ist es derzeit durch das Langzeitdatenarchiv (PG DAvE) jedoch nur möglich ganz bestimmte Mining Anfragen durch senden fest definierter PMML Schematas auf den neuesten, dem Langzeitdatenarchiv bekannten Daten, neu/erneut zu trainieren. Das Problem besteht einerseits darin, dass keine Möglichkeit besteht Datenschemata bzw. Bestand der Daten des derzeitigen Datenarchives zu erfragen und andererseits, dass offengelegt werden müsste, welches Framework bzw. welche Algorithmen/Analysemöglichkeiten bestehen. Hier wäre es denkbar, dass ein kooperierendes Langzeitdatenarchiv versucht durch Mining Frameworks eine hohe Abdeckung an Analysemöglichkeiten zu erreichen und es müsste anderen durch einen gesicherten REST Endpunkt gestattet werden, Strukturen, Beschreibungen und Beispieldaten des Datenarchivs erfragen zu können, um anhand dessen passende PMML Schemata definieren zu können. Diese können dann wiederrum über die „/api/dataming/submissions/“ Ressource in Auftrag gegeben werden.

Außerdem ist es durch die Änderung der Schnittstelle durch Kompromissverhandlungen zwischen den Projektgruppen im jetzigen Zustand nicht mehr möglich, einen Zeitraum oder ähnliche zusätzliche Informationen, neben der PMML Spezifikation, bei dem Auftrag zum Trainieren des Modelles anzugeben. Somit wird das Training der Modelle auf den passenden, jedoch kompletten Daten des Datenarchives durchgeführt, was unter Umständen kontraproduktiv sein kann. Deshalb sollte bei einer späteren Verbesserung des Systems der RequestBody der Ressource „/api/datamining/submissions“ um passende Parameter wie „from, to“, welche in der initialen Schnittstellendefinition auch zu finden waren, erweitert werden.

11.2.0.5 Abfrage von Mining Modellen (PMML)

Das Abfragen von Mining Modellen wurde in der Grundfunktionalität umgesetzt. Durch die Problematik, welche im vorherigen Paragraph beschrieben wurde, können durch die Implementierung des Langzeitdatenarchivs jedoch nur eine bestimmte Menge von trainierten Modellen abgefragt werden.

11.3 Frontend-Erweiterungen

Im Frontend ist die Datenverarbeitung vollständig implementiert und erweiterbar. Daher sind im Frontend andere, insbesondere administrative Funktionen, in Zukunft hinzuzufügen. Dazu zählt eine Konfigurierbarkeit des Sicherheitskonzepts aus Unterabschnitt 8.4.2 im Frontend. Dafür ist durch das Frontend eine Übersicht über alle Nutzer aufzustellen. Diese Nutzer sollen ebenfalls bearbeitet werden können, sodass die Nutzernamen und Rollen angepasst werden können. Außerdem sollten nicht benutzte Konten durch Administratoren im Frontend gelöscht werden können, da dies die Sicherheit des Gesamtsystems erhöht. Dies liegt daran, dass alte beziehungsweise ungenutzte Konten leichter angegriffen werden können, denn Passwörter oder andere Sicherheitsmechaniken werden nicht regelmäßig durch einen Nutzer überprüft. Neben den Standardrollen sollte es im Frontend auch möglich sein die Rollenhierarchie zu erweitern, indem neue Rollen erstellt und in die bestehende Hierarchie eingegliedert werden können. Sollten diese Rollen zum Beispiel durch eine Restrukturierung nicht mehr benötigt werden.

Eine Erweiterung des Anfragemanagements ist ebenfalls notwendig, da bisher neue Snippets und Templates manuell in der Datenbank eintragen werden müssen. In Unterunterabschnitt 8.5.1.2 sind Konzepte für die Erweiterung berücksichtigt worden. Zu den bislang nicht umgesetzten Konzepten gehört insbesondere die Anzeige von Snippets und Templates, um diese direkt im Frontend ändern zu können. Dabei sollten die Snippets im Frontend zu Templates verbunden werden können, um die Administration zu vereinfachen. Dabei sieht die bisherige Implementierung der Konzepte in Unterunterabschnitt 8.5.1.2 die entsprechende Erweiterbarkeit vor.

11.3.1 Monitoring

In einem solchen System ist außerdem der Status wichtig. Daher sollten Funktionen implementiert werden, die den Systemzustand überwachen. Dazu gehören Auslastungsmetriken, wie die CPU-Belastungen der beteiligten Systeme (zum Beispiel für jeden Knoten, auf dem eine Kafka-Instanz liegt). Das Gleiche gilt für andere Hardwareeigenschaften, wie Hauptspeicher- und Festplattenauslastung. Außerdem ist die Netzwerkauslastung ebenfalls wichtig, da die Netzwerkauslastung zentral für ein Echtzeitsystem ist.

Neben den Hardware- und Netzwerküberwachungen sollte darüber hinaus auch der Zustand der Docker-Container überwacht werden können, da diese eigene Metriken haben. Ein Ansatz für diese Umsetzung findet sich im *Docker Swarm Visualizer*. Dabei sollte der aktuelle Status der Maschinen und der Verteilung der Container auf die Maschinen angezeigt werden.

Zusätzlich sollte ein Health-Monitor eingeführt werden. Dieser Monitor sollte mindestens die Laufzeit der Komponenten beinhalten, da eine lange Laufzeit ein Softwaresystem in der Regel anfälliger macht. Schließlich wäre hier auch eine Übersicht über Systemfehler (zum Beispiel `Exceptions` in Java) für die einzelnen Instanzen wichtig, da dies einen Aufschluss über die Stabilität der Instanz gibt.

11.3.2 Andere Erweiterungen

Es kann eine weitere Visualisierung hinzugefügt werden: Die Stromnetztopologie kann auf einer geographischen Karte dargestellt werden. Die Karte ist bereits im Frontend eingebunden und die Topologie ist ebenfalls abrufbar, sodass hier die Verbindung der Karte mit der Topologie vorgenommen werden kann. Auf der Karte kann dann ebenfalls eine Ausfallanzeige stattfinden, die einem Techniker zur Verfügung gestellt werden kann. Dadurch kann dieser von der Komponente zum ausgefallenen Teilsystem navigiert werden.

Das System ist zwar internationalisiert, aber nur für Englisch lokalisiert. Weitere Lokalisierungen (wie zum Beispiel Deutsch) können dem System hinzugefügt werden, sodass mehr Nutzer angesprochen werden können. Außerdem sollte die Speicherung von Einstellungen vom Local Storage des Browsers in den Server verlagert werden, denn sonst müssen auf jedem Gerät Einstellungen mehrfach vorgenommen werden. Die Usability des Dashboards kann weiterhin ebenfalls verbessert werden, wenn die bisher implementierte `GridList` des Material Designs durch ein Dashboard Grid System wie *Angular2 Gridster* ausgetauscht wird.

11.4 Benutzerverwaltung im Frontend

In Unterabschnitt 8.4.2 ist das Konzept für die Benutzerverwaltung beschrieben. Dieses wurde allerdings nur im Backend umgesetzt. Im Frontend existiert bis jetzt nur ein Admin Tab, in dem aber noch keine Funktion implementiert ist. Dieser kann so erweitert werden, dass Nutzer mit Admin-Rechten Nutzer verwalten können. Die Seitennavigation des Admin Tabs kann dafür zwei Unterpunkte beinhalten, die Nutzerverwaltung und die Rollenverwaltung. Im Folgenden wird beschrieben welche Aktionen in den beiden Unterpunkten durchgeführt werden können.

11.4.1 Nutzerverwaltung

Hier soll sich ein Administrator alle Nutzer anzeigen lassen können oder nach einem bestimmten Nutzer suchen. Nutzer sollten mindestens über den Benutzernamen, die E-Mail oder über die Rollen des Nutzers gefunden werden können. Gefundene Nutzer werden daraufhin in einer Liste angezeigt. In der Liste wird der Benutzername, der Zeitstempel des letzten Einloggens und die Rollen des jeweiligen Nutzers angezeigt. Außerdem soll es ein Feld mit Aktionen geben, die der Administrator durchführen kann. Zu den Aktionen gehören das Löschen und das Bearbeiten des Nutzers. Das Löschen könnte der Administrator über ein Dialog Fenster bestätigen. Beim Klicken auf das Bearbeiten des Nutzers soll das Profil des Nutzers geöffnet werden.

Über das Profil können die Daten des Nutzers bearbeitet werden. Dazu gehört die E-Mail, der Vorname, der Nachname und die Rollen des Nutzers. Alle Rollen müssen vorher erstellt worden sein. Es dürfen dabei also keine neuen Rollen eingeführt werden. Das Passwort kann nicht von dem Administrator verändert werden, da die Nutzer ihre Passwörter aus Sicherheitsgründen selbst ändern sollten. Ergänzend kann eine Funktion zum Verändern des Passworts implementiert werden. Diese könnte genutzt werden, wenn ein Nutzer das Passwort vergessen hat. Sollte der Nutzer keinen Zugriff mehr auf die E-Mail haben, kann ein Administrator die E-Mail des Nutzers ändern. So hat der Nutzer immer eine Möglichkeit auf sein Konto zuzugreifen. Auch der Benutzername kann nicht mehr

verändert werden. Das Löschen des Nutzers sollte auch über das Profil stattfinden, muss aber auch hier von dem Administrator vorher nochmal bestätigt werden.

11.4.2 Rollenverwaltung

Hier soll ein Administrator neue Rollen erstellen können, nicht mehr benötigte Rollen löschen und die Rollenhierarchie verändern. Beim Löschen soll dem Administrator eine Warnung angezeigt werden, falls die Rolle in der Rollenhierarchie noch verwendet wird. Wenn möglich soll die Rollenhierarchie beim Bestätigen des Löschens durch den Administrator automatisch angepasst werden. Falls dies nicht möglich ist, soll die Aktion abgebrochen werden. Dann muss der Administrator die Rollenhierarchie zuerst so anpassen, dass die zu löschende Rolle nicht mehr in der Rollenhierarchie verwendet wird.

12 Fazit

Die Aufgabenstellung der Projektgruppe war die Entwicklung eines prototypischen Softwaresystems, welches die Echtzeitdatenverarbeitung von Smart Meter Daten ermöglicht und ein konfigurierbares Web-Frontend für die Visualisierung von Analyseergebnissen integriert. Es sollte ein System entwickelt werden, welches als Teillösung in ein umfangreiches Smart Grid System eingebettet werden kann. Dadurch soll das System einen Beitrag für die Problemstellung leisten, welche darin besteht, dass in Zukunft die Komplexität bei der Lastverteilung im Stromnetz steigen wird und durch die immer weiter voranschreitende Dezentralisierung der Energieerzeugung entsteht.

Das Ergebnis der Projektgruppe ist ein prototypisches System, welches die Anforderungen für die Echtzeitdatenanalyse und -visualisierung von Smart Metern Daten erfüllt. Die Datenverarbeitung sowie die Prognose von Verbrauch und Erzeugung von Haushalten wurden durch die Einbettung eines leistungsstarken DSMS realisiert, welches durch das entwickelte Template-System eine dynamische Anfragesteuerung sowie -generierung erlaubt und über das Web-Frontend gesteuert werden kann. Dazu können weitere Datenverarbeitungsverfahren über die Erweiterungsschnittstellen des DSMS hinzugefügt werden. Die Datenübertragung wurde durch das moderne Publish-Subscribe-System Apache Kafka realisiert, welches automatisch durch das Backend-System bei der Anfrageverarbeitung gesteuert und verwaltet wird. Das Backend-System wurde in einen Docker-Container eingebettet, sodass für die Datenverarbeitung eine hohe Skalierbarkeit und generelle eine hohe System-Ausfallsicherheit realisiert werden kann.

Das integrierte Web-Frontend stellt ein konfigurierbares Dashboard-System dar, in dem verschiedene Komponenten für die Darstellung arrangiert werden können, sodass eine gehaltvolle und strukturierte Überwachung eines Niederspannungsnetzes durchgeführt werden kann. Die Kommunikation zwischen Frontend und Backend ist durch eine JWT-Access-Token-Authentifizierung gesichert, um eine Benutzer- sowie Rollenverwaltung durch Claims verifizieren zu können. Die Performance-Evaluation des System hat gezeigt, dass einerseits die Anfrageverarbeitung stabile Latenzen sowie Datenraten liefert und andererseits die Framerate bei der Darstellung von Analyseergebnissen im Mittel konstant bleibt. Daher kann grundsätzlich von einer performanten Datenverarbeitung gesprochen werden und in Anbetracht der hohen Skalierbarkeit kann argumentiert werden, dass das System im Big Data Kontext eingesetzt werden kann. Die intervallgetriebene Datenverarbeitung der Smart Meter Daten kann jedoch zu momentanen Performanceverlusten bei der Graphenvisualisierung führen. Allerdings hätte die Evaluation des Produktes noch umfangreicher sein können, was aber durch die straffe Zeitplanung gegen Ende des Projekts unrealistisch wurde.

Die Projektarbeit hat gezeigt, dass sich das E-Stream Team durch eine besonders starke Zusammenarbeit und Motivation innerhalb der Gruppe bei der Bearbeitung der Aufgabenstellung profilieren konnte, wodurch eine ehrgeizige Agenda im Verlauf des Projekts erkennbar wurde. Im Allgemeinen lässt sich das durch die sehr aufwändige und komplexe Architektur des Softwaresystems nachvollziehen, die mehrere Teilsysteme zu einem Gesamtsystem integriert. Die Arbeitsweise der Projektgruppe führte zu einer Vielzahl an Konzepten und Entwürfen für verschiedene zusätzliche Funktionalitäten des Systems, die zunächst angedacht waren, jedoch aus unterschiedlichen Gründen nicht alle umgesetzt werden konnten. Beispielsweise stellte sich die angestrebte Kollaboration mit einer weiteren Projektgruppe (DAvE) als problematisch und zeitaufwendiger dar, als zu Beginn des Projekts angenommen. Die Zusammenarbeit bestand darin, eine gemeinsame Schnittstellen-Beschreibung zu entwickeln, durch die ein Austausch von Daten und trainierten Data Mining Modellen zwischen den beiden entwickelten

Systemen ermöglicht werden sollte. Die Schnittstellen-Beschreibung musste mehrfach überarbeitet und geändert werden, weil die unterschiedlichen Entwicklungsstadien der beiden Projektgruppen zu einem asymmetrischen Problemverständnis und zu Missverständnissen führte. Teilweise wurde die Zusammenarbeit noch durch fehlende Kommunikation oder durch das nicht Einhalten von Vereinbarungen seitens DAvE erschwert. Nichtsdestotrotz wurde eine Schnittstelle umgesetzt und ein funktionierender Platzhalter-Service vom E-Stream Team implementiert, um die Funktionalitäten der Schnittstelle simulieren zu können. Im Gegensatz dazu funktionierte die Kommunikation und die Zusammenarbeit mit dem Forschungsprojekt NetzDatenStrom einwandfrei, sodass die Integration der *mosaik* Simulation in das Projektsystem trotz laufender Änderungen problemlos durchgeführt wurden konnte.

Eine weitere Herausforderung entstand durch einen Organisations-Overhead, der dazu führte, dass nicht immer genügend Arbeitsstunden zur Verfügung standen um Aufgaben umzusetzen. Das lag zum einen daran, dass die Projektgruppe die Domäne der Energieinformatik nicht vollständig durchdrungen hat, wodurch domänen-spezifische Aufgaben schwieriger zu bearbeiten und zu planen waren. Insbesondere konnten hierdurch Teilziele weniger genau definiert und konkretisiert werden. Zum anderen können interne wie auch externe Projektbedingungen verantwortlich gemacht werden. Beispielsweise wurde eine strikte Stundenerfüllung pro Woche verlangt und die Spezialisierung in bestimmte Fachbereiche untersagt. Die Stundenerfüllung hat dazu geführt, dass weniger die Fertigstellung von Aufgaben im Vordergrund stand, sondern eher darauf geachtet wurde, ob schon genügend Stunden geleistet worden sind. Dadurch wurde teilweise ineffektiver gearbeitet. Die geforderte Generalisierung, also die Einarbeitung in mehrere Fachbereiche, sorgte ebenfalls für eine ineffektivere Arbeitsweise, da jede Einarbeitung in einen neuen Fachbereich mit voranschreitendem Projektverlauf immer aufwendiger wurde. Insgesamt lässt sich das Projekt aber als voller Erfolg verbuchen, da ein erweiterbares, flexibles und leistungsstarkes Softwaresystem entstanden ist, das einen Lösungsansatz für die gestellte Aufgabenstellung darstellt.

13 Liste aller Anforderungen

14 Anhang

14.1 Use Case Tabellen

ID	UC-PGESTREAM-01
Name	Topologie bereitstellen
Akteur	System Netzdaten, E-Stream
Trigger	System E-Stream Start
Kurzbeschreibung	Nachdem das System hochgefahren wurde, sollen Topologiedaten eingelesen und ggf. transformiert werden
Vorbedingung	1. Topologiedaten müssen vorhanden sein 2. NetzDatenStrom ist aktiv
Essenzielle Schritte	1. Anfragen der Datenverarbeitung verwalten 2. Topologiedaten transformieren
Ausnahmefälle	
Nachbedingung	Die letzten Topologiedaten sind verfügbar
Zeitverhalten	Sofort

Tabelle 14.1: *UC-PGESTREAM-01*

ID	UC-PGESTREAM-02
Name	Smart Meter Daten erhalten
Akteur	NetzDatenStrom, E-Stream
Trigger	System E-Stream ist gestartet
Kurzbeschreibung	Nachdem das System hochgefahren wurde, sollen Smart Meter Daten kontinuierlich empfangen werden
Vorbedingung	1. Smart Meter Daten müssen vorhanden sein 2. NetzDatenStrom ist aktiv
Essenzielle Schritte	Smart Meter Daten empfangen
Ausnahmefälle	
Nachbedingung	Die Smart Meter Daten des letzten Zeit-/Elementfensters sind verfügbar
Zeitverhalten	Kontinuierlich (Standard 15 Minuten)

Tabelle 14.2: *UC-PGESTREAM-02*

ID	UC-PGESTREAM-03
Name	Verbrauch und Erzeugung prognostizieren
Akteur	Nutzer, E-Stream , Odysseus, DAvE
Trigger	Erhalten von (historischen) Smart Meter- und Wetterdaten über Verbrauch und Erzeugung
Kurzbeschreibung	Eine Regressionsanalyse kann auf erhaltenen (historischen) Smart Meter Daten über den Verbrauch/die Erzeugung und Wetter Daten durchgeführt werden
Vorbedingung	1. Ein Nutzer des Systems besitzt die Berechtigung eine Anfrage (zur Prognose) zu erstellen 2. (historische) Smart Meter Daten und historische Wetterdaten sind vorhanden
Essenzielle Schritte	1. Regressionsanalyse auf statistischen Werten 2. Je nach Anfrage werden verschiedene Daten benötigt a) bei Prognosen mit Daten auf kurzem Zeitraum (letzten Zeitraum/Window) mit DSMS (Odysseus) b) bei Prognosen mit Daten auf langen Zeitraum (Archiv) durch Model von Big Data Archive c) Vergleich der Abweichungen der vorherigen Optionen
Ausnahmefälle	(historische) Smart Meter Daten und historische Wetterdaten nicht vorhanden
Nachbedingung	Die letzten Ergebnisse der Regressionsanalyse sind verfügbar
Zeitverhalten	Sofort

Tabelle 14.3: *UC-PGESTREAM-03*

ID	UC-PGESTREAM-04
Name	Verbrauch und Erzeugung anzeigen
Akteur	Nutzer, E-Stream
Trigger	Erhalten von (historischen) Smart Meter- und Wetterdaten über Verbrauch und Erzeugung
Kurzbeschreibung	Die erhaltenen Smart Meter Daten werden statistisch ausgewertet und in Form von verschiedenen Chart Typen angezeigt
Verwendete Anwendungsfälle	UC-PGESTREAM-04-1, UC-PGESTREAM-04-2
Vorbedingung	1. Ein Nutzer des Systems besitzt die Berechtigung eine Anfrage (zur Prognose) zu erstellen 2. (historische) Smart Meter Daten und historische Wetterdaten sind vorhanden a) alle verfügbaren Daten wurden aus dem DSMS (Odysseus) geladen b) Daten, die nicht mehr im DSMS (Odysseus) vorhanden waren, wurden aus dem Archiv (Big Data Archive) geladen
Essenzielle Schritte	1. Berechnung statistischer Werte (Durchschnitt, Maximum, Minimum, Median, Standardabweichung, Varianz) 2. Die Ergebnisse werden einer passenden Darstellungsart zugeordnet 3. Anzeigen dieser Werte in Form von Tabellen (generische Tabelle), Line Charts, Bar Charts, als Topologie (geografisch, Netz)
Ausnahmefälle	(historische) Smart Meter Daten und historische Wetterdaten nicht vorhanden Dann: Anzeige (der zuletzt gespeicherten Werte und) einer entsprechenden Fehlermeldung
Nachbedingung	Die letzten Werte aus der statistischen Berechnung sind verfügbar
Zeitverhalten	Sofort

Tabelle 14.4: *UC-PGESTREAM-04*

ID	UC-PGESTREAM-04-1
Name	Historischen Verbrauch und Erzeugung anzeigen
Akteur	Nutzer, E-Stream
Trigger	Öffnen der BarChart Komponenten durch den Nutzer
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden akkumuliert in einem Balkendiagramm dargestellt
Vorbedingung	
Essenzielle Schritte	1. System-Defaultwert ist beim Öffnen vorauswählt: z. B. „Verbrauch“ und Zeitfenster „1 Tag“ 2. Anzeige der Daten in Form eines Balkendiagramms a) Auswahl der Daten (Wetter, Verbrauch oder Erzeugung) b) Definieren des Zeitfensters
Ausnahmefälle	(historische) Smart Meter Daten und historische Wetterdaten nicht vorhanden → Dann: Anzeige (der zuletzt gespeicherten Werte und) einer entsprechenden Fehlermeldung
Nachbedingung	Letzte Nutzerauswahl wird mindestens bis zum Sitzungsende gespeichert, sodass seine letzte Auswahl beim nächsten Öffnen direkt angezeigt wird

Tabelle 14.5: *UC-PGESTREAM-04-1*

ID	UC-PGESTREAM-04-2
Name	Echtzeit Verbrauch und Erzeugung anzeigen
Akteur	Nutzer, E-Stream
Trigger	Öffnen der LineChart Komponente durch den Nutzer
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden akkumuliert in einem Liniendiagramm dargestellt
Vorbedingung	
Essenzielle Schritte	1. System-Defaultwert ist beim Öffnen vorausgewählt: z. B. „Verbrauch“ und Zeitfenster „1 Tag“ 2. Anzeige der Daten in Form eines Liniendiagramms a) Auswahl der Daten (Verbrauch oder Erzeugung, Prognose, Vergleich/Varianz,...) b) Definieren des Zeitfensters
Ausnahmefälle	(historische) Smart Meter Daten und historische Wetterdaten nicht vorhanden → Dann: Anzeige (der zuletzt gespeicherten Werte und) einer entsprechenden Fehlermeldung
Nachbedingung	Letzte Nutzerauswahl wird mindestens bis zum Sitzungsende gespeichert, sodass seine letzte Auswahl beim nächsten Öffnen direkt angezeigt wird
Zeitverhalten	Kleinste kontinuierliche Schritte (Standard 15 Minuten, diskret)

Tabelle 14.6: *UC-PGESTREAM-04-2*

ID	UC-PGESTREAM-05
Name	Topologie visualisieren
Akteur	Nutzer, E-Stream
Trigger	Nutzer öffnet eine Topologieansicht
Kurzbeschreibung	Nachdem ein Nutzer eine Topologieansicht öffnet soll die passende Topologiedarstellung visualisiert werden
Verwendete Anwendungsfälle	UC-PGESTREAM-05-1, UC-PGESTREAM-05-2, UC-PGESTREAM-05-3
Vorbedingung	1. Der Nutzer des Systems ist berechtigt sich die Topologie anzeigen zu lassen 2. Topologiedaten sind vorhanden
Essenzielle Schritte	1. Topologieansicht wird geöffnet 2. Topologiedaten werden visualisiert
Ausnahmefälle	Topologiedaten werden nicht erhalten → Dann: Anzeige (der zuletzt gespeicherten Werte und) einer entsprechenden Fehlermeldung
Nachbedingung	Die letzten Topologiedaten sind durch die ausgewählte Ansicht visualisiert
Zeitverhalten	Sofort

Tabelle 14.7: *UC-PGESTREAM-05*

ID	UC-PGESTREAM-05-1
Name	Topologie abstrakt anzeigen
Akteur	Nutzer, E-Stream
Trigger	Öffnen des abstrakte Topologie Fensters durch den Nutzer
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden akkumuliert in einer abstrakten Topologie dargestellt
Vorbedingung	
Essenzielle Schritte	1. System-Defaultwert ist beim Öffnen vorausgewählt, z. B.: Anzeige aller Smart Meter Daten 2. Anzeige der Daten in Form einer abstrakten Topologie a) Auswahl der Daten (alle Smart Meter Daten, Region, ...) b) (optional/Idee: Auswahl des Zeitfensters)
Ausnahmefälle	(optional/Idee: Keine Daten vorhanden für ausgewählten Zeitfenster → Dann: Fehler-/Hinweismeldung an den Nutzer)
Nachbedingung	Letzte Nutzerauswahl wird mindestens bis zum Sitzungsende gespeichert, sodass seine letzte Auswahl beim nächsten Öffnen direkt angezeigt wird
Zeitverhalten	Kleinsten kontinuierlichen Schritte (Standard 15 Min, diskret)

Tabelle 14.8: *UC-PGESTREAM-05-1*

ID	UC-PGESTREAM-05-2
Name	Topologie geografisch anzeigen
Akteur	Nutzer, E-Stream
Trigger	Öffnen des geografische Topologie Fensters durch den Nutzer
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden akkumuliert in einer geografischen Topologie dargestellt
Vorbedingung	
Essenzielle Schritte	1. System-Defaultwert ist beim Öffnen vorausgewählt, z. B.: Anzeige einer bestimmten Region (Oldenburg) 2. Anzeige der Daten in Form einer geografischen Topologie a) Auswahl der Daten (alle Smart Meter Daten, Region, ...) b) (optional/Idee: Auswahl eines Zeitfensters)
Ausnahmefälle	(optional/Idee: Keine Daten vorhanden für ausgewählten Zeitfenster→ Dann: Fehler-/Hinweismeldung an den Nutzer)
Nachbedingung	Letzte Nutzerauswahl wird mindestens bis zum Sitzungsende gespeichert, sodass seine letzte Auswahl beim nächsten Öffnen direkt angezeigt wird
Zeitverhalten	Kleinsten kontinuierlichen Schritte (Standard 15 Min, diskret)

Tabelle 14.9: *UC-PGESTREAM-05-2*

ID	UC-PGESTREAM-05-3
Name	Topologie als Navigationtree anzeigen
Akteur	Nutzer, E-Stream
Trigger	Öffnen eines Topologie Fensters durch den Nutzer
Kurzbeschreibung	Die erzeugte Topologie wird in einem Navigationtree dargestellt
Vorbedingung	Die abstrakte oder geografische Topologie wurden erfolgreich geladen
Essenzielle Schritte	Markieren von Knotenpunkten im Navigationtree führt zur gleichzeitigen Markierung in der Topologie
Ausnahmefälle	
Zeitverhalten	Sofort

Tabelle 14.10: *UC-PGESTREAM-05-3*

ID	UC-PGESTREAM-06
Name	Topologie visualisieren
Akteur	Nutzer, E-Stream , DAvE
Trigger	Erhalten von Smart Meter Daten und ggf. Abweichung dieser von gewissen Werten
Kurzbeschreibung	Störungen (+Kaskaden) und Unregelmäßigkeiten werden erkannt. Dies wird dem Nutzer global im Dashboard signalisiert. Außerdem kann die erkannte Unregelmäßigkeit/Störung lokalisiert werden
Verwendete Anwendungsfälle	UC-PGESTREAM-06-1, UC-PGESTREAM-06-2, UC-PGESTREAM-06-3
Vorbedingung	1. Der Nutzer hat die Berechtigung diese Daten einzusehen und ist eingeloggt 2. Smart Meter Daten werden erhalten 3. Es muss ein Trainingsset zur Klassifizierung oder eine Klassifikation durch Big Data Archive (PMML) bereitgestellt worden sein 4. Um eine Störung/Unregelmäßigkeit zu lokalisieren, muss eine solche aufgetreten sein
Essenzielle Schritte	1. Störungen (+Kaskaden)/ Unregelmäßigkeiten erkennen 2. Störungen/Unregelmäßigkeiten lokalisieren a) Durch die bekannte Topologie und erkannte Störungen werden diese lokalisiert und angezeigt b) Im Bezug auf UC-PGESTREAM-07 eingrenzen
Ausnahmefälle	Smart Meter Daten werden nicht erhalten oder die Verfahren können nicht durchgeführt werden → Dann: Anzeige (des zuletzt gespeicherten Wertes und) einer entsprechenden Fehlermeldung
Nachbedingung	Die letzten Störungs-/ Unregelmäßigkeitenerkennungen und gegebenenfalls deren Lokalisierung sind verfügbar
Zeitverhalten	Sofort

Tabelle 14.11: *UC-PGESTREAM-06*

ID	UC-PGESTREAM-06-1
Name	Störungen erkennen
Akteur	Nutzer, E-Stream
Trigger	Starten der Plausibilitätsprüfung
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden analysiert und auf Störungen überprüft
Vorbedingung	
Essenzielle Schritte	1. Schwellwert/Threshold Verfahren a) Ausfall eines Netzes/Teilnetzes/Smart Meter? 2. Klassifizierung (Supervised) a) Durch eigenes bereitgestelltes Trainingsset b) Durch Big Data Archive bereitgestellte Klassifikation (PMML)
Ausnahmefälle	
Nachbedingung	Erkannte Störungen werden gespeichert
Zeitverhalten	Sofort

Tabelle 14.12: *UC-PGESTREAM-06-1*

ID	UC-PGESTREAM-06-2
Name	Störungen erkennen
Akteur	Nutzer, E-Stream
Trigger	Starten der Plausibilitätsprüfung
Kurzbeschreibung	Alle erhaltenen Smart Meter Daten werden analysiert und auf Unregelmäßigkeiten überprüft
Vorbedingung	
Essenzielle Schritte	1. Schwellwert/Threshold Verfahren a) Einspeisung von Strom bei Nacht? b) Einspeisung von zu viel Strom für aktuelles Wetter? (Falls möglich) 2. Ausreißererkennung durch Nachbarschaftsanalyse (KNN) (Am besten unsupervised direkt auf dem Datenstrom)
Ausnahmefälle	
Nachbedingung	Erkannte Störungen werden gespeichert
Zeitverhalten	Sofort

Tabelle 14.13: *UC-PGESTREAM-06-2*

ID	UC-PGESTREAM-06-3
Name	Störungen erkennen
Akteur	Nutzer, E-Stream
Trigger	Es wurden Störungen/Unregelmäßigkeiten erkannt
Kurzbeschreibung	Alle erkannten Störungen/Unregelmäßigkeiten werden lokalisiert und in einer Topologie angezeigt
Vorbedingung	
Essenzielle Schritte	1. Die erkannten Störungen/Unregelmäßigkeiten werden lokalisiert a) Im Bezug auf UC-PGESTREAM-07 eingrenzen 2. Benachrichtigung des Nutzers über gefundene Störungen/Unregelmäßigkeiten 3. Die lokalisierten Störungen/Unregelmäßigkeiten werden in einer Topologie angezeigt
Ausnahmefälle	
Nachbedingung	Die zuletzt lokalisierten Störungen/Unregelmäßigkeiten werden angezeigt
Zeitverhalten	Sofort

Tabelle 14.14: *UC-PGESTREAM-06-3*

ID	UC-PGESTREAM-07
Name	Geografische/topologische Filterung von Smart Meter Daten
Akteur	Nutzer, E-Stream
Trigger	Smart Meter Daten werden erhalten
Kurzbeschreibung	Nachdem Smart Meter Daten abgefragt und erhalten wurden, kann eine geografische/topologische Filterung durchgeführt werden, sodass sie in Form verschiedener Charts angezeigt werden kann
Vorbedingung	1. Der Nutzer des Systems ist berechtigt sich die Charts anzeigen zu lassen, 2. Smart Meter Daten sind vorhanden
Essenzielle Schritte	1. Geografische/topologische Filterung von Smart Meter Daten in Bezug auf Scope 2. Anzeige aller Smart Meter Daten, einer Region, von Transformatoren (bevorzugt Netzbetreiber), von Smart Meter Gateways oder einzelner/zusammengefasster Smart Meter in Form verschiedener Chart Typen
Ausnahmefälle	Smart Meter Daten werden nicht erhalten $\rightarrow$ Dann: Anzeige (des zuletzt gespeicherten Wertes und) einer entsprechenden Fehlermeldung
Nachbedingung	Die Daten der letzten Filterung sind verfügbar
Zeitverhalten	Sofort

Tabelle 14.15: *UC-PGESTREAM-07*

ID	UC-PGESTREAM-08
Name	Dashboard konfigurieren
Akteur	Nutzer, E-Stream
Trigger	Öffnen des „Dashboard konfigurieren“-Menüs durch den Benutzer
Kurzbeschreibung	In diesem Menü kann der Benutzer auswählen, welche Informationen er auf seinem Dashboard (Startseite) direkt angezeigt bekommen möchte. Also konkretes Auswählen einer Darstellungsform (Topologische Karte, BarChart, LineChart), welche Daten darin angezeigt werden sollen, an welcher Position diese Informationen stehen sollen und wie groß die Informationen dargestellt werden sollen
Vorbedingung	
Essenzielle Schritte	1. Auswahl der Darstellungsform der Informationen 2. Auswahl welche Daten angezeigt werden sollen 3. Auswahl bzw. Positionierung der Darstellungsform auf dem Dashboard 4. Anpassen der Größe der Darstellungsform
Ausnahmefälle	
Nachbedingung	Sofortige Anzeige des neu konfigurierten Dashboards. Speichern der Konfiguration des Benutzers, sodass beim nächsten Login diese Konfiguration verwendet wird
Zeitverhalten	Sofort

Tabelle 14.16: *UC-PGESTREAM-08*

ID	UC-PGESTREAM-09
Name	Anfragen der Datenverarbeitung verwalten
Akteur	Administrator, E-Stream
Trigger	Öffnen des „Anfragen verwalten“-Fensters durch den Administrator
Kurzbeschreibung	Der Administrator kann verschiedene Anfragen aus Vorlagen erstellen oder selbst neue Vorlagen anlegen. Bei diesen Vorlagen kann eine Datenart angegeben werden. Außerdem kann angegeben werden, ob und wie die Daten verarbeitet werden sollen
Vorbedingung	Ein Nutzer hat Administrator Berechtigungen
Essenzielle Schritte	1. Auswählen von vordefinierten Anfragen oder Auswahl einer Datenart 2. Ändern der vordefinierten Anfrage oder Auswahl der Datenverarbeitung 3. Die Änderungen bzw. Zusammenstellung der Anfrage können gespeichert oder verworfen werden
Ausnahmefälle	
Nachbedingung	
Zeitverhalten	Sofort

Tabelle 14.17: *UC-PGESTREAM-09*

ID	UC-PGESTREAM-10
Name	Rohdaten in Tabelle anzeigen
Akteur	Administrator, E-Stream
Trigger	Öffnen der (Rohdaten-) Tabellen Komponenten durch den Administrator
Kurzbeschreibung	In der Tabelle können Rohdaten direkt angezeigt werden. Hierbei werden die Daten unverarbeitet angezeigt. Der Benutzer kann auswählen von welcher Datenart (Quelle) er die Rohdaten angezeigt bekommen möchte
Vorbedingung	Ein Nutzer hat Administrator Berechtigungen
Essenzielle Schritte	Auswählen welche Datenart angezeigt werden soll
Ausnahmefälle	
Nachbedingung	Letzte Nutzerauswahl wird mindestens bis zum Sitzungsende gespeichert, sodass seine letzte Auswahl beim nächsten Öffnen direkt angezeigt wird
Zeitverhalten	Sofort

Tabelle 14.18: *UC-PGESTREAM-10*

ID	UC-PGESTREAM-11
Name	Anfragen der Datenverarbeitung verwalten
Akteur	Administrator, E-Stream
Trigger	Öffnen des „Nutzer verwalten“-Fensters durch den Administrator
Kurzbeschreibung	In der Nutzerverwaltung können Administratoren neue Benutzer anlegen und Ihnen verschiedene Rechte geben. Bestehenden Nutzern können neue Rechte vergeben bzw. entzogen oder vollständig gelöscht werden
Vorbedingung	Ein Nutzer hat Administrator Berechtigungen
Essenzielle Schritte	1. Auswahl eines bestehenden Nutzers oder neuen Nutzer erstellen 2. Dem ausgewählten Nutzer Rechte geben bzw. entziehen oder den Benutzer löschen
Ausnahmefälle	
Nachbedingung	
Zeitverhalten	Sofort

Tabelle 14.19: *UC-PGESTREAM-11*

14.2 Use Case Tabellen in Kooperation mit PG DAvE

ID	UC-PGESTREAM-PGDAVE-01
Name	Abfrage historischer Smart Meter Daten
Akteur	System E-Stream , System DAvE
Trigger	
Kurzbeschreibung	E-Stream stellt über eine HTTP-REST-Schnittstelle eine Anfrage mit Parametern. Dazu zählt eine ID (z. B. Hash-Wert), Datenformat (JSON oder XML) und (Key-Value-Paare) / oder es wird eine fertige CQL-Anfrage übergeben. DAvE überprüft die Anfrage und gibt Rückmeldung. Bei valider Anfrage werden die Daten von DAvE für die Übergabe verarbeitet. DAvE erstellt ein Kafka-Topic mit der übergebenen ID (z. B. Hash-Wert) als Topicbezeichnung. Die Daten werden über das Kafka Topic bereitgestellt. E-Stream ruft die Daten über Kafka Connect ab.
Vorbedingung	• Historische Smart Meter Daten liegen im Archiv bereit • Tabellennamen werden durch die Bezeichnungen der Datenquellen bestimmt. • ID für den Topic-Namen muss vorhanden sein • Kafka Broker sind bekannt.
Essenzielle Schritte	1. E-Stream stellt eine Anfrage über HTTP-REST-Schnittstelle mit Angabe des erwarteten Formats und der Topic-Bezeichnung. 2. Anfrage überprüfen (DAvE) 3. Daten werden im angegeben Format in einem Kafka-Topic hinterlegt. 4. E-Stream ruft die Daten über Kafka Connect ab
Ausnahmefälle	Für angefragten Zeitraum existieren keine Smart Meter Daten → Response von Http Anfrage enthält diese Info. Kafka-Topic ist nicht erreichbar über die ID.
Nachbedingung	PG E-Stream kann auf angefragte Daten per bereitgestelltem Topic zugreifen.
Zeitverhalten	Asynchron und sofort verfügbar.

Tabelle 14.20: *Abfrage historischer Smart Meter Daten*

ID	UC-PGESTREAM-PGDAVE-02
Name	Abfrage historischer Wetterdaten (DWD)
Akteur	System E-Stream , System DAvE
Trigger	
Kurzbeschreibung	E-Stream stellt über eine HTTP-REST-Schnittstelle eine Anfrage mit Parametern. Dazu zählt eine ID (z. B. Hash-Wert), Datenformat (JSON oder XML) und (Key-Value-Paare) / oder es wird eine fertige CQL-Anfrage übergeben. DAvE überprüft die Anfrage und gibt Rückmeldung über eine HTTP-Response. Bei valider Anfrage werden die Daten von DAvE für die Übergabe verarbeitet. DAvE erstellt ein Kafka-Topic mit der übergebenen ID (z. B. Hash-Wert) als Topicbezeichnung. Die Daten werden über das Kafka Topic bereitgestellt. E-Stream ruft die Daten über Kafka Connect ab.
Vorbedingung	• Historische Wetterdaten liegen im Archiv bereit • Tabellenschema muss bekannt sein • Kafka Broker sind bekannt.
Essenzielle Schritte	1. E-Stream stellt eine Anfrage über HTTP-REST-Schnittstelle mit Angabe des erwarteten Formats und der Topic-Bezeichnung. 2. Anfrage überprüfen (DAvE) 3. Daten werden in XML Format in einem Kafka-Topic hinterlegt. 4. E-Stream ruft die Daten über Kafka Connect ab
Ausnahmefälle	Für angefragten Zeitraum existieren keine Wetterdaten → Fehlermeldung in Response. Kafka-Topic ist nicht erreichbar über die ID.
Nachbedingung	PG E-Stream kann auf angefragte Daten per bereitgestelltem Topic zugreifen.
Zeitverhalten	Asynchron und sofort verfügbar.

Tabelle 14.21: *Abfrage historischer Wetterdaten*

ID	UC-PGESTREAM-PGDAVE-03
Name	Abfrage von Mining Modellen (PMML)
Akteur	System E-Stream , System DAvE
Trigger	System PG E-Stream wird hochgefahren
Kurzbeschreibung	Bereits bestehende PMML Modelle können über eine Http Anfrage abgefragt werden. Die Http Antwort enthält bei erfolgreicher Anfrage und dem Vorhandensein mindestens einer passenden PMML Spezifikation im Big Data Archive, das zeitlich letzte passende PMML Modell. Falls noch kein Model mit dieser Spezifikation erstellt wurde, wird diese Information als Http Antwort verschickt.
Vorbedingung	• Falls die HTTP-Schnittstelle per Authentifizierung geschützt ist, existiert ein Nutzer des Systems E-Stream mit ausreichenden Rechten um auf die benötigten Anfragen zugreifen zu können. • Standard (default) PMML Modelle sind vorhanden und müssen nicht extra in Auftrag gegeben werden. • Es muss eine Anfrage bereitstehen, die es ermöglicht, alle verfügbaren Verfahren/Modelle abzufragen.
Essenzielle Schritte	1. PMML Modell zu passender PMML Spezifikation wird angefragt. 2. Auf Basis der PMML Spezifikation, wird ein Hash berechnet und mit diesem im Big Data Archive nach dem letzten zeitlich passenden PMML Modell gesucht. 3. Falls die Anfrage erfolgreich war und mindestens ein PMML Modell zu der angegebenen PMML Spezifikation passt, wird dies als Antwort zurückgesandt.
Ausnahmefälle	Modell nicht vorhanden, da es sich bei der Anfrage um kein ausgemachtes Standard PMML Modell handelt: 1. Fall, das angefragte PMML Modell nicht beziehungsweise nicht mehr unterstützt wird, dann soll eine passende Fehlermeldung zurückgegeben werden. 2. Falls, das angefragte PMML Modell untersützt wird, dann soll eine passende Information zurückgegeben werden, dass derzeit kein Modell existiert, jedoch eines in Auftrag gegeben werden kann.
Nachbedingung	Letztes PMML Modell in HTTP-Antwort verfügbar.
Zeitverhalten	Sofort

Tabelle 14.22: *Abfrage von Mining Modellen (PMML)*

ID	UC-PGESTREAM-PGDAVE-04
Name	Erstellen von Mining Model Aufträgen
Akteur	System E-Stream , System DAvE
Trigger	
Kurzbeschreibung	Das System E-Stream stellt einen Mining Model Auftrag über eine HTTP-REST-Schnittstelle. Dieser Auftrag muss folgende Parameter beinhalten: • Use Case: Power Forecast oder Load Forecast • Dimensionseingrenzung: Eingrenzung der einzubeziehenden Daten • gewünschtes Model (z. B. Regeln, Cluster, Klassifikationen o.ä.) • Hash zum bestehenden Modell mit übergeben Das System DAvE prüft daraufhin den Auftrag und gibt eine Rückmeldung als HTTP-Response. Bei valider Anfrage wird das Model erstellt und bei erfolgreicher Bearbeitung wird dies auf den dafür vorgesehen Topic des Systems DAvE gesandt.
Vorbedingung	Topic für PMML Modelle muss bestehen.
Essenzielle Schritte	1. Auftrag mit notwendigen Parametern (Hash...) stellen 2. Erstellen des Modells
Ausnahmefälle	• Falls Modell/Anfrage doch nicht unterstützt wird → Fehlermeldung in Response. • Topic existiert nicht mehr. • Modell konnte nicht erstellt werden.
Nachbedingung	Model steht nach einiger Zeit im Topic bereit.
Zeitverhalten	Asynchron und nach Bearbeitungszeit (eventuell länger)

Tabelle 14.23: *Erstellen von Mining Model Aufträgen*

ID	UC-PGESTREAM-PGDAVE-05
Name	Senden von Rohdaten zur Speicherung
Akteur	System Netzdaten, System E-Stream , System DAvE
Trigger	System Netzdaten sendet neue Smart Meter Daten
Kurzbeschreibung	Die neuen Daten vom Netzdatensystem werden auf dem passenden Topic einer Kafka Instanz des E-Stream Systems gesandt und können so durch das System DAvE empfangen, aufbereitet und archiviert werden.
Vorbedingung	• Kafka-Topic für Smart Meter Rohdaten vorhanden. • Das System DAvE hat auf dem Rohdatentopic eine Subscription. • Es stehen neue Smart Meter Rohdaten bereit.
Essenzielle Schritte	1. Das Netzdatensystem sendet neue Daten auf den passenden Topic einer Kafka Instanz des E-Stream Systems. 2. Die Kafka Instanz kümmert sich um die intelligente Weiterleitung der eingegangenen/vorhandenen Daten an die Subscriber. 3. Das System DAvE erhält die Daten und bereitet diese mit passenden Metadaten auf. 4. Das System DAvE archiviert die aufbereiteten Daten und stellt diese nun für die Abfrage bereit.
Ausnahmefälle	Das Topic ist während Bearbeitung nicht mehr verfügbar bzw. erreichbar
Nachbedingung	• Das System DAvE kann die Daten vom bereitgestellten Kafka Topic abfragen. • Die Daten sind im Big Data Archive des Systems DAvE persistiert.
Zeitverhalten	Kontinuierlich (standardmäßig vorgesehen im 15 Minutentakt. Zu wissenschaftlichen Zwecken und zur Evaluation auch im Sekunden/Millisekundenbereich)

Tabelle 14.24: *Senden von Rohdaten zur Speicherung*

Glossar

Application Lifecycle Management ∼ ist der Produktlebenszyklus(Governance, Aufsetzen und Warten) von Anwendungen. .

Apriori Algorithmus zum Finden von ↑Frequent Pattern in einer Datenmenge .

Apriori-Eigenschaft Eigenschaft von frequent item sets, die besagt, dass Teilmengen einer häufigen Menge selbst häufig sein müssen und Erweiterungen einer nicht häufigen Menge nie häufig sein können .

BoilerplateCode Als ∼ werden Codefragmente bezeichnet, die an vielen Stellen fast ohne Änderung hinzugefügt werden müssen. .

Change Detection Erkennen statistisch signifikanter Änderungen der Daten im Datenstrom .

Cluster Feature Vector Der ∼ (CFV) is eine Datenstruktur, die ein Cluster in Kenngrößen, wie z.B. Anzahl der Punkte, lineare und quadratische Summe zusammenfasst .

Clustering Strukturierung von Daten in Gruppen bezüglich ihrer Ähnlichkeit. Ansätze werden in partitionsbasierte, hierarchische, dichtebasierte und gitterbasierte Ansätze aufgeteilt .

Cognitive Walkthrough Beim ∼ versetzt sich ein Usability-Experte in einen hypothetischen Benutzer und analysiert konkrete vorgegebene Handlungsabläufe. .

Concept Drift Graduelle Änderungen der Daten in einem Datenstrom .

Concept Shift Abrupte Änderungen der Daten in einem Datenstrom .

Confidence Kenngröße beim FPM, die angibt, wie wahrscheinlich ein Datenelement in der Transaktion ist, wenn ein anderes Datenelement bereits gegeben ist .

Data Mining ∼ ist die gezielte Extraktion von Wissen aus einer Datenmenge mit Hilfe von ↑Machine Learning .

Datenkonzentrator Verarbeitet und sammelt Daten, welche von einem ↑Smart Metering Gateway erfasst werden und übermittelt die verarbeiteten Daten an zentrale Server im ↑Smart Grid .

Datenstrom Ein ∼ ist eine zeitlich totalgeordnete Sequenz von potentiell unendlichen Daten. .

Datenstrommanagementsystem Ein ∼ verarbeitet ↑Datenströme und bietet die Möglichkeit Anfragen analog zu Datenbankmanagementsystem auf Datenströmen auszuführen. .

Datentransferobjekt ∼ sind Objekte, die zum Austausch zwischen System benutzt werden. Ihr Ziel ist dabei die Größe der zu übertragenenden Nachricht zu minimieren und nur die relevanten Informationen zu übertragen. .

Distributed Energy Resources ∼ bezeichnet unterschiedliche Arten von Energiequellen, wie lokale Photovolatikanlagen, Blockheizkraftwerke, oder Biogasanlagen .

Feature Eigenschaft eines Datenelements beim ↑Machine Learning .

Fluktuation Als ∼ wird eine Schwankung von Gegebenheiten oder Zuständen bezeichnet. .

FP-Baum Baumartige Datenstruktur zum Speichern von frequent patterns .

Framework Eine Grundstruktur, auf Basis dessen ein Programmierer eine Anwendung erstellen kann. .

Frequent Algorithmus Algorithmus zum Finden häufiger 1-itemsets aus einem Datenstrom .

Frequent Pattern Mining $\sim$, auch (FPM) bezeichnet das Erkennen von häufigen Mustern in einer Datenmenge. Analyse der Zusammenhänge von Datenelementen. .

Gateway Das $\sim$ ist eine Komponente welche zwischen zwei Systemen eine Verbindung herstellt, um Kommunikation zwischen beiden Systemen herzustellen. Ein $\sim$ ist im Allgemeinen für die Protokollumsetzung verantwortlich .

Git Flow $\sim$ ist ein Modell nach dem für die Bearbeitung von Aufgaben bestimmte Branch-Präfixe verwendet werden. .

High-Level API In einer $\sim$ werden Methoden zusammengefasst, in denen Methoden komplexe Vorgänge vornehmen. .

Hoeffding Bound Statistische Kenngröße, die eine obere Schranke der Abweichung des Stichprobenmittles zum tatsächlichen Mittelwert unter Verwendung eines Vetrauensniveaus angibt .

Hoeffding-Tree-Algorithmus Algorithmus zum Erstellen eines Entscheidungsbaums als Lösung eines Klassifizierungsproblems .

Integration Test $\sim$ testen die Kombination und das Zusammenwirken von verschiedenen Programmmodulen oder des Gesamtsystems. .

Intelligentes Messsystem Das $\sim$ bezeichnet ein System im Bereich des ↑Smart Grid, welches für das Erfassen und Übertragen von Smart Meter Daten eingesetzt wird. Es setzt aus drei Komponenten zusammen: ↑Smart Meter, ↑Smart Metering Gateway und einem Sicherheitsmodul. .

Kernel-Trick Im $\sim$ werden Berechnungen implizit in einem höher-dimensionalen Raum ausgeführt. In der Regel wird ein linearer Klassifikator auf nicht linear klassifizierbare Daten angewendet. .

Label Klasse die einem Datenelement bei einer Klassifikation zugeordnet wird .

Latenz Die Zeit, die benötigt wird auf eine Eingabe eine Ausgabe zu produzieren, wird als $\sim$ bezeichnet. .

Lightweight Eine Bibliothek oder ein ↑Framework ist $\sim$, wenn es eine geringe kompilierte Größe durch einen begrenzten Funktionsumfang mit schnellen Ladezeiten vorweisen kann. .

Low-Level API In einer $\sim$ werden Methoden zusammengefasst, die einfache Vorgänge beschreiben. Diese wird normalerweise durch eine ↑High-Level API zusammengefasst. .

Machine Learning Oberbegriff für die künstliche Generierung von Wissen aus Erfahrung .

Metadatum Ein $\sim$ ist ein Datensatz mit Zusatzinformationen. In den meisten Fällen werden ein Zeitstempel oder eine Ordnungszahl angefügt, sodass Daten über die Daten verfügbar werden. .

Minification $\sim$ ist der Prozess für den Interpreter unwichtige Zeichen aus dem Code zu entfernen und so die Größe der Datei zu minimieren. Dies wird typischerweise für JavaScript-Bibliotheken durchgeführt. .

Model-View-ViewModel-Pattern $\sim$ trennt im Gegensatz zum Model-View-Controller-Pattern das Modell vollständig von der View. Dabei wird durch Datenbindung das Modell an das ViewModell gekoppelt und so die View aktualisiert. .

Netztopologie Die Struktur und der Aufbau eines Netzes wird durch die $\sim$ beschrieben. .

Object Identification System Das $\sim$ ist ein Standard (IEC 62056-61) und definiert ein Datenformat für den Austausch zur eindeutigen Indentifkation von Messwerten und wird im Bereich des ↑Smart Metering eingesetzt, um ↑Smart Meter zu beschreiben .

Online-Algorithmen Algorithmen, die inkrementell Ergebnisse liefern. Bei ihrem Start müssen nicht alle zu verarbeitenden Daten vorhanden sein. Beim Eintreffen eines neuen Datenelements wird ein neuer Output generiert .

Overfitting Zu starkes anpassen des Vorhersagemodells an die Trainingsdaten .

Partielle Aggregate $\sim$ ist eine Klasse von Aggregaten, die inkrementell berechnet wird. Weitere Daten zur Berechnung des Aggregats erfordern keine vollständige Neuberechnung, sondern aktualisieren nur Metadaten und das Ergebnis. .

Precision Anteil der wirklich zu diesem ↑Label gehörenden Daten an der Gesamtmenge der Daten, die diesem Label zugeordnet worden sind .

Recall Anteil der Daten, die zu dem Label gehören, jedoch einem anderen ↑Label zugeordnet worden sind an der Gesamtmenge der zu diesem Label gehörenden Daten .

Recommender System System, das basierend auf vergangenen Transaktionen Vorschläge für zukünftige Transaktionen macht .

RFID-Technologie $\sim$ besteht aus einem Empfänger und einem Sender. Dabei kommunizieren beide über Radio-Frequenzen miteinander, wodurch Empfänger Sender erkennen können, ohne das zusätzlicher Aufwand nötig ist. .

Single-Page-Webanwendung $\sim$ ist eine Anwendung, bei der nur eine Seite geladen wird. Änderungen und Anfragen von Ressourcen werden durch die Anwendungen selber übernommen. Dafür existiert in einer $\sim$ normalerweise ein Routing-System, welches eine mehrseitige Anwendung simuliert. .

Smart Grid Ein $\sim$ ist ein Energienetzwerk, das das Verbrauchts- und Einspeiseverhalten aller Marktteilnehmer, die mit ihm verbunden sind, integriert. Es sichert ein ökonomisch, effizientes, nachhaltiges Versorgungssystem mit niedrigen Verlusten und hoher Verfügbarkeit.

Smart Home Das $\sim$ bezeichnet die intelligente Überwachung und Steuerung eines Haushaltes .

Smart Meter Ein $\sim$ bezeichnet einen "intelligenten" Verbrauchsmesser im Kontext des ↑Smart Grid .

Smart Metering Gateway Das ∼ bildet die zentrale Kommunikationseinheit eines Smart Grid und verbindet Komponenten, wie ↑Smart Meter, ↑Smart Home und Benuterschnittstellen, ↑Datenkonzentratoren und zentrale Server des ↑Smart Grids .

spatiotemporale Anfragen ∼ sind Anfragen, die mit raum-zeitlichen Daten durchgeführt werden. .

Supervised Learning Ein System lernt aus Beispielen und kann diese nach der Lernphase verallgemeinern und damit Eigenschaften vorhersagen. Werden diskrete Vorhersagen getroffen, nennt man das Vorhersagemodell Klassifikator, bei kontinuierlichen Daten Regressor. .

Support Kenngröße beim FPM, die angibt, wie häufig zwei Datenelemente gemeinsam in der Gesamtmenge der Transaktionen vorkommen .

TypeScript ∼ ist eine von Microsoft entwickelte Sprache, die ein Superset von JavaScript ist und den ECMA-Script-6 Standard erfüllt. Durch einen Compiler kann TypeScript auf einen gängigen JavaScript-Standard wie ES2015 reduziert werden. Dadurch wird das polyfill-Problem gelöst. .

Unsupervised Learning Ein System betreibt ↑Machine Learning ohne Beispieldaten. ↑Clustering ist eine Form von Unsupervised Learning .

Usability ∼ bezeichnet den Ausmaß, in dem ein Produkt, System oder Dienst durch bestimmte Benutzer in einem bestimmten Anwendungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen. .

Web-Visualisierung Eine ∼ ist eine Webseite zur Darstellung von Ergebnissen einer Anwendungssoftware. .

Wrapper Eine Programm, welches ein anderes Programm umgibt und es so abkapselt. .

Akronyme

ACL Access Control List.

ACSI Abstract Communication Service Interface.

ADWIN Adaptive Window.

ANN Artificial Neural Networks.

AOP Aspektorientierte Programmierung.

API Application Programming Interface.

AWT Abstract Window Toolkit.

BIB Bruttoinlandprodukt.

BSI Bundesamt für Sicherheit in der Informationstechnik.

CDC Common Data Class.

CDPSM Common Distribution Power System Model.

CFV Cluster Feature Vector.

CIM Common Information Model.

CIS Common Interface Specification.

CLI Command-Line-Interface.

CLS Controllable Local Systems.

COSEM Companion Specification for Energy Metering.

COSMO Consortium for Small-scale Modeling.

CQL Continuous Query Language.

CRUD Create, read, update and delete.

CSS Cascading-Style-Sheet.

CSV Comma-separated values.

DBMS Datenbankmanagementsysteme.

DER Distributed Energy Resources.

DLMS Device Language Message Specification.

DNS Desoxyribonukleinsäure.

DOM Document Object Model.

DS Datenstrom.

DSMS Datenstrommanagementsystem.

DWD Deutscher Wetterdienst.

EA Evolutionärer Algorithmus.

EEG Erneuerbare-Energie-Gesetz.

EJB Enterprise JavaBeans.

EMS Energiemanagementsystem.

ENTSO-e European Network of Transmission System Operators for Electricity.

ERPI Electric Power Research Institute.

ESB Enterprise Service Bus.

FPM Frequent Pattern Mining.

GIS Generic Interface Defintion.

GPL General Public License.

GPS Global Positioning System.

GSSAPI Generic Security Service Application Program Interface.

GUI Graphical User Interface.

GWT Google Web Toolkit.

HTML Hypertext-Markup-Language.

HTTP Hypertext-Transfer-Protocol.

ICON Icosahedral Nonhydrostatic.

IEC International Electrotechnical Commission.

JAAS Java Authentication and Authorization Service.

Java EE Java Platform, Enterprise Edition.

JCP Java Community Process.

JFS Java Server Faces.

JPA Java Persistence API.

JSON JavaScript Object Notation.

JSR Java Specification Requests.

KDM Knowledge Discovery Metamodel.

KWKG Kraft-Wärme-Kopplungsgesetz.

LAM limited area model.

LTLF long term load forecasting.

MAE Mean absolute error.

MBE Mean bias error.

MEP Mathematical Expression Parser.

MMS Manufacturing Message Specification Protokoll.

MOP Master Optimization Process.

MOS Model Output Statistic.

MSE Mean square error.

MTLF medium term load forecasting.

MVC Model View Controller.

NWA Nutzwertanalyse.

NWP Numerical weather prediction.

OBIS Object Identification System.

ORM Objektrelationales Mapping.

OSGi Open Service Gateway initiative.

PFA Portable Format for Analytics.

PG Projektgruppe.

PMML Predictive Model Markup Language.

POP Post-Office-Protocol.

PQL Procedural Query Language.

PV Photovoltaik.

RDF Resource Description Framework.

RDFS Resource Description Framework Schema.

REST Representational State Transfer.

RMSE Root mean square error.

RPC Remote Procedure Call.

SASL Simple Authentication and Security Layer.

SCADA Supervisory Control and Data Acquisition.

SDE Standard deviation.

SMART spezifisch, messbar, akzeptiert, realistisch und terminiert.

SMGW Smart Meter Gateway.

SOAP Simple Object Access Protocol.

SPA Single-Page-Webanwendung.

SQL Structured Query Language.

STLF short term load forecasting.

SWT Standard Widget Toolkit.

TCP Transmission Control Protocol.

TCP/IP Transmission Control Protocol/Internet Protocol.

TDD Test-Driven-Development.

TSV Tab-separated values.

UCD User-Centered-Design.

UDP User Datagram Protocol.

UI User-Interface.

VM Virtuelle Maschine.

XML Extensible Markup Language.

XSD XML-Schema-Definition.

Abbildungen

Literatur

[1] Vladimir Agafonkin und CloudMade. *BSD 2-clause SSimplified"License*. 2016. URL: https://github.com/Leaflet/Leaflet/blob/master/LICENSE (besucht am 02. 06. 2017).

[2] Charu C. Aggarwal. *Data Mining*. 2015.

[3] Jonas Bonér et al. *The Reactive Manifesto*. URL: https://www.reactivemanifesto.org/ (besucht am 18. 03. 2018).

[4] Shirin Mohammadi et al. „Adaptive Data Stream Management System Using Learning Automata“. In: *CoRR* abs/1110.1700 (2011). arXiv: 1110.1700. URL: http://arxiv.org/abs/1110.1700.

[5] Albano, Michele; et al. „Convergence of Smart Grid ICT Architectures for the Last Mile“. In: *IEEE* 11.1 (2015), S. 187–197. DOI: 10.1109/TII.2014.2379436. URL: http://dx.doi.org/10.1109/TII.2014.2379436.

[6] Susanne Albers. *Online Algorithms*. University of Freiburg.

[7] Ben Alman u. a. *Grunt*. 2012. URL: https://gruntjs.com (besucht am 20. 05. 2017).

[8] Tim Ambler, Nicholas Cloud und Robin Andrew Hawkes. *JavaScript frameworks for modern web dev*. Springer, 2015.

[9] Angular. *Angular Architecture*. URL: https://angular.io/guide/architecture (besucht am 20. 12. 2017).

[10] Angular. *Angular NgModule*. URL: https://angular.io/guide/ngmodule (besucht am 20. 12. 2017).

[11] Angular. *Angular NgModule FAQ*. URL: https://angular.io/guide/ngmodule-faq (besucht am 20. 12. 2017).

[12] Angular. *Angular Style Guide*. URL: https://angular.io/guide/styleguide (besucht am 20. 12. 2017).

[13] Angular. *E2E Testing*. URL: https://docs.angularjs.org/guide/e2e-testing (besucht am 28. 02. 2018).

[14] H Appelrath u. a. *Odysseus: a highly customizable framework for creating efficient event stream management systems*. ACM, 2012.

[15] Arvind Arasu, Shivnath Babu und Jennifer Widom. „The CQL continuous query language: semantic foundations and query execution“. In: *The VLDB Journal - The International Journal on Very Large Data Bases* 15.2 (2006), S. 121–142.

[16] Atlassian. *Engineering higher quality through agile testing practices*. URL: https://www.atlassian.com/agile/testing (besucht am 14. 06. 2017).

[17] Atlassian. *GitFlow Workflow*. URL: https://www.atlassian.com/git/tutorials/comparing-workflows#gitflow-workflow (besucht am 08. 06. 2017).

[18] Atlassian. *Interactive rebase in SourceTree*. URL: https://www.atlassian.com/blog/sourcetree/interactive-rebase-sourcetree (besucht am 12. 06. 2017).

[19] Atlassian. *Merging vs. Rebasing*. URL: https://www.atlassian.com/git/tutorials/merging-vs-rebasing (besucht am 12. 06. 2017).

[20] atombender. *Slow performance with many markers*. Nov. 2013. URL: https://github.com/Leaflet/Leaflet.markercluster/issues/278 (besucht am 06. 06. 2017).

[21] Zeyar Aung u. a. „Towards accurate electricity load forecasting in smart grids“. In: *The Fourth International Conference on Advances in Databases, Knowledge, and Data Applications, DBKDA*. 2012.

[22] Austin. *MIT License*. 2016. URL: https://github.com/swimlane/ngx-charts/blob/master/docs/license.md (besucht am 02. 06. 2017).

[23] Baigent, D.; et al. „IEC 61850 Communication Networks and Systems In Substations: An Overview for Users“. In: *Proceedings of the VIII SIPSEP, Monterey, Mexico*. 2005.

[24] Helmut Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Heidelberg: Spektrum Akademischer Verlag: Heidelberg, 2011.

[25] Jens Bastian u. a. „Master for Co-Simulation Using FMI“. In: *Proceedings of the 8th International Modelica Conference; March 20th-22nd; Technical Univeristy; Dresden; Germany*. 63. Linköping University Electronic Press; Linköpings universitet, 2011, S. 115–120.

[26] Kent Beck u. a. *Agile Manifesto*. URL: https://www.agilealliance.org/agile101/12-principles-behind-the-agile-manifesto/ (besucht am 29. 05. 2017).

[27] Joseph Bergner, Hrsg. *Untersuchungen zu prognosebasierten Betriebsstrategien für PV-Speichersysteme*. Hochschule für Technik und Wirtschaft (HTW) Berlin: Bergner, 2014.

[28] Bermbach, Rainer. *Kommunikation im Smart Grid - Kommunikationsbedarf eines MUC Controllers*. Wolfenbüttel: Ostfalia Hoschule für angewandte Wissenschaften, 2013.

[29] Hans Georg Beyer u. a., Hrsg. *Vorhersagen der Stromerzeugung aus Sonnen- und Windenergie*. FH Magdeburg, Carl von Ossietzky Universität Oldenburg: Beyer, 2000.

[30] Albert Bifet. „Adaptive Learning and Mining for Data Streams and Frequent Patterns“. Universitat Politecnica de Catalunya, 2009.

[31] Christopher M Bishop. *Pattern recognition*. Springer Science+Business Media: New York, 2006. ISBN: ISBN 978-0387-31073-2.

[32] think geo blog. *Leaflet vs. OpenLayers 3: Which is the better client-side JavaScript mapping library?* Sep. 2015. URL: http://blog.thinkgeo.com/2015/09/08/leaflet-vs-openlayers-3-which-is-the-better-client-side-javascript-mapping-library/ (besucht am 06. 06. 2017).

[33] Barry W. Boehm u. a. „Developing Multimedia Applications with the WinWin Spiral Model“. In: *Software Engineering - ESEC/FSE '97, 6th European Software Engineering Conference Held Jointly with the 5th ACM SIGSOFT Symposium on Foundations of Software Engineering, Zurich, Switzerland, September 22-25, 1997, Proceedings*. Hrsg. von Mehdi Jazayeri und Helmut Schauer. Bd. 1301. Lecture Notes in Computer Science. Springer, 1997, S. 20–39. DOI: 10.1007/3-540-63531-9_3. URL: https://doi.org/10.1007/3-540-63531-9_3.

[34] Andre Bolles u. a. „Odysseus: Ein Framework für maßgeschneiderte Datenstrommanagementsysteme.“ In: *GI Jahrestagung*. 2009, S. 2000–2014.

[35] J. et al. Boner. 2014. URL: http://www.reactivemanifesto.org/ (besucht am 22. 05. 2017).

[36] Jonas Bonér. *Reactive Microservices Architecture*. O'Reilly Media, Incorporated, 2016.

[37] Jonas Bonér. *Reactive Microsystems - The Evolution Of Microservices At Scale*. O'Reilly Media, Incorporated, 2016.

[38] Mike Bostock. *BSD 3-clause "Newör "Revised"License*. 2017. URL: https://github.com/d3/d3/blob/master/LICENSE (besucht am 02. 06. 2017).

[39] Mike Bostock. *Visualization: GeoChart*. 2017. URL: https://developers.google.com/chart/interactive/docs/gallery/geochart (besucht am 02. 06. 2017).

[40] Fredrik Boström. *Slow performance in OpenLayers 3 when panning with 500 features*. Sep. 2016. URL: https://stackoverflow.com/questions/39125958/slow-performance-in-openlayers-3-when-panning-with-500-features (besucht am 06. 06. 2017).

[41] Bower. *Bower: A package manager for the web*. URL: https://bower.io (besucht am 29. 05. 2017).

[42] Gert Brettlecker. „Efficient and reliable data stream management". Diss. University of Basel, 2008.

[43] BSI. *Technische Richtlinie BSI TR-03109*. Bonn: Bundesamt für Sicherheit in der Informationstechnik, 2013.

[44] Umwelt Bundesamt. *Erneuerbare Energien in Zahlen*. März 2017. URL: http://www.umweltbundesamt.de/themen/klima-energie/erneuerbare-energien/erneuerbare-energien-in-zahlen (besucht am 07. 06. 2017).

[45] Bundesnetzagentur. *Gesetz zur Digitalisierung der Energiewende*. Feb. 2017. URL: https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/Unternehmen_Institutionen/NetzzugangundMesswesen/Mess-undZaehlwesen/Smart_Metering/Smart_Metering_node.html (besucht am 29. 04. 2017).

[46] Prof. Dr. Bruno Burger. *Stromerzeugung aus Solar- und Windenergie im Jahr 2015*. Jan. 2016. URL: https://www.ise.fraunhofer.de/content/dam/ise/de/documents/news/folien-stromerzeugung-aus-solar-und-windenergie-im-jahr-2015.pdf (besucht am 22. 04. 2017).

[47] Michael Cammert u. a. „Anfrageverarbeitung auf Datenströmen". In: *Datenbank-Spektrum* 4.11 (2004), S. 5–13. URL: http://dbs.mathematik.uni-marburg.de/publications/myPapers/2004/CHKS04.pdf.

[48] CasparWaara. *forEachFeatureAtPixel freezes/super slow performance on ff and ie*. Okt. 2015. URL: https://github.com/openlayers/openlayers/issues/4232 (besucht am 06. 06. 2017).

[49] CasparWaara. *high-performance-markers-on-leaflet-map*. Feb. 2010. URL: https://gis.stackexchange.com/questions/227868/high-performance-markers-on-leaflet-map (besucht am 06. 06. 2017).

[50] Google Charts. *Google Charts Frequently Asked Questions*. Feb. 2017. URL: https://developers.google.com/chart/interactive/faq (besucht am 06. 06. 2017).

[51] Mike Clark. *Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Applications*. Pragmatic Bookshelf, 2004. ISBN: 9780974514031.

[52] Chart.js community. *Issus*. URL: https://github.com/chartjs/Chart.js/issues (besucht am 02. 06. 2017).

[53] D3 community. *D3.js community*. URL: https://groups.google.com/forum/#!forum/d3-js (besucht am 02. 06. 2017).

[54] Google Chart API community. *Google Chart API*. URL: https://groups.google.com/forum/#!forum/google-chart-api (besucht am 02. 06. 2017).

[55] Leaflet community. *Official Leaflet support community*. URL: https://groups.google.com/forum/#!forum/leaflet-js (besucht am 02. 06. 2017).

[56] ngx-charts community. *Issus*. URL: https://github.com/swimlane/ngx-charts/issues (besucht am 02. 06. 2017).

[57] OpenLayers community. *Issus*. URL: https://github.com/openlayers/openlayers/issues (besucht am 02. 06. 2017).

[58] Smoothie Charts community. *Discussion about Smoothie Charts JavaScript library*. URL: https://groups.google.com/forum/#!forum/smoothie-charts (besucht am 02. 06. 2017).

[59] OpenLayers Contributors. *BSD 2-clause SSimplified"License*. 2011. URL: http://svn.openlayers.org/trunk/openlayers/license.txt.

[60] iMatix Corporation. *Distributed Messaging - zeromq*. URL: http://zeromq.org/ (besucht am 12. 03. 2018).

[61] Oracle Corporation. 2017. URL: https://docs.oracle.com/javaee/7/tutorial/overview.htm (besucht am 28. 05. 2017).

[62] Edward Crookshanks. *Practical enterprise software development techniques : tools and techniques for large scale solutions*. [Berkeley, Calif.]: [Berkeley, Calif.] : Apress, 2015. ISBN: 978-1-4842-0620-1.

[63] datenbanken-verstehen.de. *Oracle Database - Editionen*. URL: http://www.datenbanken-verstehen.de/datenbankarten/oracle/ (besucht am 12. 06. 2017).

[64] Estelle DeBlois. *Build Better Desktop Apps With Ember*. URL: https://speakerdeck.com/brzpegasus/build-better-desktop-apps-with-ember (besucht am 29. 05. 2017).

[65] Google Developers. *Minify Resources (HTML, CSS, and JavaScript)*. URL: https://developers.google.com/speed/docs/insights/MinifyResources (besucht am 29. 05. 2017).

[66] Deveo. *GitVsSvn*. URL: https://deveo.com/git-vs-svn/ (besucht am 08. 06. 2017).

[67] *DMG, Data Mining Group*. 1997. URL: http://dmg.org/pmml/v4-3/GeneralStructure.html,%20Stand:%2022.04.2017.

[68] Pedro Domingos und Geoff Hulten. *Mining High-Speed Data Streams*. University of Washington, 2000.

[69] Nick Downie. *The MIT License*. 2017. URL: https://github.com/chartjs/Chart.js/blob/master/LICENSE.md (besucht am 02. 06. 2017).

[70] Vincent Driessen. *A successful Git branching model*. URL: http://nvie.com/posts/a-successful-git-branching-model/ (besucht am 08. 06. 2017).

[71] Vincent Driessen. *GitFlow CLI*. URL: https://github.com/nvie/gitflow (besucht am 08. 06. 2017).

[72] Joseph S. Dumas und Janice Redish. *A Practical Guide to Usability Testing*. Intellect Books, 1999. ISBN: 9781841500201.

[73] *DWD, Deutscher Wetterdienst*. www.dwd.de, Stand: November 2016. 2016. URL: http://www.dwd.de/DE/forschung/wettervorhersage/num_modellierung/06_num_wettervorhersage_notfallsystem/num_wettervorhersage_notfallsystem_node.html.

[74] Arbeitsgemeinschaft Energiebilanzen e.V. *Bruttostromerzeugung in Deutschland von 1990 bis 2013 nach Energieträgern*. Dez. 2013. URL: https://web.archive.org/web/20140201140922/http://www.ag-energiebilanzen.de/index.php?article_id=29&fileName=20131220_brd_stromerzeugung1990-2013.pdf (besucht am 22. 04. 2017).

[75] Arbeitsgemeinschaft Energiebilanzen e.V. *Entwicklung des Erdgasverbrauchs*. Feb. 2017. URL: https://www.bdew.de/internet.nsf/id/9F0B3AEB36061207C1257AFA0051FB27/%24file/Erdgasverbrauch%20Entwicklung%2010%20Jahre_o_jaehrlich_Ki_online_02032017.pdf (besucht am 22. 04. 2017).

[76] George Ellis. *Project management in product development : leadership skills and management techniques to deliver great products*. Amsterdam: Amsterdam : Elsevier Ltd, 2016.

[77] Hicham G. Elmongui. „Query optimization for spatio-temporal data stream management systems“. In: *SIGSPATIAL Special* 1.1 (2009), S. 21–26. DOI: 10.1145/1517463.1517465. URL: http://doi.acm.org/10.1145/1517463.1517465.

[78] Ember. *Component*. URL: https://guides.emberjs.com/v2.13.0/components/defining-a-component/ (besucht am 29. 05. 2017).

[79] Ember. *Ember Core Team*. URL: https://emberjs.com/team/ (besucht am 29. 05. 2017).

[80] Ember. *Models*. URL: https://guides.emberjs.com/v2.13.0/models/ (besucht am 29. 05. 2017).

[81] Ember. *SEE WHO'S USING EMBER.JS*. URL: https://emberjs.com/ember-users/ (besucht am 29. 05. 2017).

[82] Ember. *Templates*. URL: https://guides.emberjs.com/v2.13.0/templates/handlebars-basics/ (besucht am 29. 05. 2017).

[83] Ralf S. Engelschall. *ECMAScript 6 - New Features: Overview & Comparison*. 1. Juni 2015. URL: http://es6-features.org/#Constants.

[84] *Enterprise Architect 13*. URL: https://www.sparxsystems.de/uml/neweditions/ (besucht am 17. 04. 2017).

[85] *ENTSO-E Standards*. URL: https://www.entsoe.eu/major-projects/common-information-model-cim/cim-for-grid-models-exchange/standards/Pages/default.aspx (besucht am 08. 04. 2017).

[86] M. Ester und J. Sander, Hrsg. *Knowledge Discovery in Databases. Techniken und Anwendungen*. Berlin: Format-Verlag, 2000.

[87] Facebook. *React*. URL: https://github.com/facebook/react (besucht am 23. 05. 2017).

[88] Facebook. *React*. URL: https://facebook.github.io/react/ (besucht am 23. 05. 2017).

[89] X. Fang u. a. „Smart grid – the new and improved power grid: A survey.“ In: *IEEE Communications Surveys Tutorials* 14.4 (2012), S. 944–980.

[90] Fredrik Farnstrom, James Lewis und Charles Elkan. *Scalability for Clustering Algorithms Revisited*. University of California, Lund Institute of Technology, 2000.

[91] Stefan Feuerhahn u. a. „Comparison of the communication protocols DLMS/COSEM, SML and IEC 61850 for smart metering applications“. In: *SmartGridComm*. IEEE, 2011, S. 410–415.

[92] Apache Software Foundation. *Feature Summary*. URL: https://maven.apache.org/maven-features.html (besucht am 29. 05. 2017).

[93] Apache Software Foundation. *Guide to Working with Multiple Modules*. URL: https://maven.apache.org/guides/mini/guide-multiple-modules.html (besucht am 29. 05. 2017).

[94] Apache Software Foundation. *Introduction to Repositories*. URL: https://maven.apache.org/guides/introduction/introduction-to-repositories.html (besucht am 29. 05. 2017).

[95] Apache Software Foundation. *Introduction to the Build Lifecycle*. URL: https://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html (besucht am 29. 05. 2017).

[96] Apache Software Foundation. *Introduction to the POM*. URL: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html (besucht am 29. 05. 2017).

[97] Apache Software Foundation. *Ivy The Agile Dependency Manager*. 2007. URL: http://ant.apache.org/ivy/ (besucht am 19. 05. 2017).

[98] Apache Software Foundation. *The Apache Ant Project*. 2000. URL: https://ant.apache.org (besucht am 19. 05. 2017).

[99] MariaDB Foundation. *MariaDB.org*. URL: https://mariadb.org/ (besucht am 12. 06. 2017).

[100] Node.js Foundation. 2017. URL: https://github.com/nodejs/node (besucht am 18. 05. 2017).

[101] The Apache Software Foundation. *Apache CouchDB*. URL: http://couchdb.apache.org/ (besucht am 12. 06. 2017).

[102] Martin Fowler. *SelfTestingCode*. URL: https://martinfowler.com/bliki/SelfTestingCode.html (besucht am 16. 06. 2017).

[103] John Gantz und David Reinsel. *The Digital Universe in 2020*. study. EMC Corporation, 2012.

[104] Sandra Geisler. „Data Stream Management Systems“. In: *Data Exchange, Integration, and Streams*. 2013, S. 275–304. DOI: 10.4230/DFU.Vol5.10452.275. URL: http://dx.doi.org/10.4230/DFU.Vol5.10452.275.

[105] Sandra Geisler. „Data Stream Management Systems“. In: *Data Exchange, Integration, and Streams*. Hrsg. von Phokion G. Kolaitis, Maurizio Lenzerini und Nicole Schweikardt. Bd. 5. Dagstuhl Follow-Ups. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2013, S. 275–304. ISBN: 978-3-939897-61-3. DOI: 10.4230/DFU.Vol5.10452.275. URL: http://drops.dagstuhl.de/opus/volltexte/2013/4297.

[106] Chris Giannella u. a. *Mining Frequent Patterns in Data Streams at Multiple Time Granularities*. Indiana University, 2003.

[107] Gregor Giebel u. a. *The state-of-the-art in short-term prediction of wind power: A literature overview*. Techn. Ber. ANEMOS. plus, 2011.

[108] Christian Münch GmbH. *Photovoltaikanlage*. URL: http://www.photovoltaik.org/photovoltaikanlagen (besucht am 07. 06. 2017).

[109] Rheinwerk Verlag GmbH. *Das MVVM-Pattern*. 1. Juli 2013. URL: http://openbook.rheinwerk-verlag.de/visual_csharp_2012/1997_28_005.html.

[110] Lukasz Golab und M Tamer Özsu. „Issues in data stream management“. In: *ACM Sigmod Record* 32.2 (2003), S. 5–14.

[111] Lukasz Golab und M. Tamer Özsu. *Data Stream Management*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. DOI: 10.2200/S00284ED1V01Y201006DTM005. URL: http://dx.doi.org/10.2200/S00284ED1V01Y201006DTM005.

[112] Lukasz Golab und M. Tamer Özsu. „Issues in data stream management“. In: *SIGMOD Record* 32.2 (2003), S. 5–14. DOI: 10.1145/776985.776986. URL: http://doi.acm.org/10.1145/776985.776986.

[113] Lukasz Golab und Tamer Özsu. *Data Stream Management*. Online access: IEEE (Institute of Electrical and Electronics Engineers) IEEE Morgan & Claypool Synthesis eBooks Library. Morgan & Claypool Publishers, 2010. ISBN: 9781608452729.

[114] Joachim Goll. *Mit Scrum zum gewünschten System*. Wiesbaden: Wiesbaden : Springer Vieweg, 2015.

[115] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *DeviationAnomalyDetection Operator*. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/DeviationAnomalyDetection+operator (besucht am 31. 03. 2018).

[116] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *Odysseus Data Stream Management and Complex Event Processing*. 2007. URL: http://odysseus.informatik.uni-oldenburg.de/ (besucht am 28. 04. 2017).

[117] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *Odysseus Home*. 2007. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Odysseus+Home (besucht am 28. 04. 2017).

[118] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *Odysseus Home - Aggregation Operator*. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Aggregation+operator (besucht am 31. 03. 2018).

[119] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *Odysseus Home - Enrichment Operator*. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Enrich+operator (besucht am 03. 04. 2018).

[120] Marco Grawunder und Abteilung Informationssysteme – Department für Informatik – Universität Oldenburg. *Odysseus Home - TimeWindow*. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/TimeWindow (besucht am 31. 03. 2018).

[121] gwtproject.org. 2017. URL: http://www.gwtproject.org/overview.html (besucht am 28. 05. 2017).

[122] Peter J Haas und Joseph M Hellerstein. „Ripple joins for online aggregation“. In: *ACM SIGMOD Record* 28.2 (1999), S. 287–298.

[123] E. M. Hahn. *Express in Action*. Manning Publications Company, 2016.

[124] Al-Hamadi, HM und SA Soliman. „Long-term/mid-term electric load forecasting based on short-term correlation and annual growth“. In: *Electric power systems research* 74.3 (2005), S. 353–361.

[125] Annette Hammer u. a., Hrsg. *Short-Term Forecasting of Solar Ratdiation: A statistical approach using satellite Data*. Department of Energy und Semiconductor Research, Faculty of Physics, University of Oldenburg: Hammer, 2000.

[126] Jiawei Han, Micheline Kamber und Jian Pei. *Data Mining Concepts and Techniques*. 2011. ISBN: 978-0-12-381479-1.

[127] Erik Hanchett. *Ember. js cookbook*. Packt Publishing Ltd, 2016.

[128] Eckhart Hanser. *Agile Prozesse: Von XP über Scrum bis MAP*. Berlin, Heidelberg: Berlin, Heidelberg : Springer-Verlag Berlin Heidelberg, 2010.

[129] HashiCorp. *Vagrant Documentation*. URL: https://www.vagrantup.com/docs/ (besucht am 18. 02. 2018).

[130] Wilhelm Hasselbring. *Softwarearchitektur an der Schnittstelle zwischen Entwicklung und Betrieb*. GI Fachtagung Architekturen 2015. 2015. URL: http://eprints.uni-kiel.de/29215/1/2015-07-10Architekturen.pdf (besucht am 24. 04. 2017).

[131] Trevor Hastie, Robert Tibshirani und Jerome Friedman. *The Elements of Statistical Learning*. 2. Aufl. Springer New York, 2009. DOI: 10.1007/978-0-387-84858-7. URL: https://doi.org/10.1007%2F978-0-387-84858-7.

[132] Justin Heinermann und Oliver Kramer. „Short-Term Wind Power Prediction with Combination of Speed and Power Time Series“. In: *Joint German/Austrian Conference on Artificial Intelligence (Künstliche Intelligenz)*. Springer. 2015, S. 100–110.

[133] Pat Helland. „Life Beyond Distributed Transactions“. In: *Queue* 14.5 (Okt. 2016), 70:69–70:98. ISSN: 1542-7730. DOI: 10.1145/3012426.3025012. URL: http://doi.acm.org/10.1145/3012426.3025012.

[134] Klaus Heuck, Klaus-Dieter Dettmann und Detlef Schulz. *Elektrische Energieversorgung: Erzeugung, Übertragung und Verteilung elektrischer Energie für Studium und Praxis*. 9. Aufl. Springer Vieweg, Sep. 2013. ISBN: 9783834816993.

[135] Kazuo Hiekata u. a. „Design of Software Development Architecture Comparison of Waterfall and Agile Using Reliability Growth Model“. In: *Transdisciplinary Engineering: Crossing Boundaries - Proceedings of the 23rd ISPE Inc. International Conference on Transdisciplinary Engineering, Curitiba, Parana, Brazil, October 3-7, 2016*. Hrsg. von Milton Borsato u. a. Bd. 4. Advances in Transdisciplinary Engineering. IOS Press, 2016, S. 471–480. DOI: 10.3233/978-1-61499-703-0-471. URL: https://doi.org/10.3233/978-1-61499-703-0-471.

[136] Christoph Höft. *Bewertung von Verfahren zur Prognose der elektrischen Last–eine empirische Analyse*. 2004.

[137] Bill Holtsnider. *Agile development & goals : the six week solution*. Burlington, MA: Burlington, MA : Morgan Kaufmann Publishers/Elsevier, 2010.

[138] Yan Huang. „Toward a data-type-based real time geospatial data stream management system“. Diss. University of North Texas, 2011.

[139] Kai Hufendiek und Martin Kaltschmitt. „Einsatz künstlicher neuronaler Netze bei der kurzfristigen Lastprognose“. In: (1998).

[140] Geoff Hulten, Laurie Spencer und Pedro Domingos. *Mining Time-Changing Data Streams*. University of Washington, Innovation Next, 2001.

[141] IEC 62056-47. *COSEM Transport Layers for IPv4 Networks*. Geneva: International Electrotechnical Commission, 2006.

[142] Red Hat Inc. *What's a Linux container?* URL: https://www.redhat.com/en/topics/containers/whats-a-linux-container (besucht am 20. 02. 2018).

[143] Google Inc. *Angular JS*. 1. Sep. 2009. URL: https://angular.io/features.html.

[144] Google Inc. *ECMAScript 6 - New Features: Overview & Comparison*. 1. Aug. 2016. URL: https://docs.angularjs.org/guide/di.

[145] Google Inc. *Terms of Service*. 2013. URL: https://developers.google.com/chart/terms (besucht am 02. 06. 2017).

[146] Gradle Inc. *Building Java Libraries*. URL: https://guides.gradle.org/building-java-libraries/ (besucht am 28. 05. 2017).

[147] Gradle Inc. *Dependency Management Basics*. URL: https://docs.gradle.org/current/userguide/artifact_dependencies_tutorial.html (besucht am 29. 05. 2017).

[148] Gradle Inc. *Gradle Build Tool*. URL: https://gradle.org (besucht am 17. 05. 2017).

[149] Gradle Inc. *Maven vs Gradle: Feature Comparison Chart*. URL: https://gradle.org/maven-vs-gradle (besucht am 17. 05. 2017).

[150] Open Container Initiative. *Open Container Initiative*. URL: https://www.opencontainers.org/ (besucht am 21. 02. 2018).

[151] solid IT GmbH. *DB-Engines Ranking - Methode*. URL: https://db-engines.com/de/ranking_definition (besucht am 23. 05. 2017).

[152] solid IT GmbH. *DB-Engines Ranking - Rangliste*. URL: https://db-engines.com/de/ranking (besucht am 12. 06. 2017).

[153] Niklaas Ites. *Ermittlung von ertragsrelevanten Fehlern bei Photovoltaik Parks mittels verschiedener Mess- und Prüfmethoden*. URL: http://edoc.sub.uni-hamburg.de/haw/volltexte/2015/2829/pdf/Ites_Niklaas_140929.pdf (besucht am 31. 03. 2018).

[154] Ja.Sruthi und R.L.Helen Catherine. „A Review On Electrical Load Forecasting In Energy Management“. In: *International Journal of Innovative Science Engineering and Technology* 3.3 (2015), S. 670–676.

[155] Pankaj Jalote. *Fault Tolerance in Distributed Systems*. PTR Prentice Hall, 1994.

[156] Yegor Jbanov und Tobias Bosch. *Angular 2 Rendering Architecture*. 1. Aug. 2016. URL: https://docs.google.com/document/d/1M9FmT05Q6qpsjgvH1XvCm840yn2eWEg0PMskSQz7edit.

[157] Thomas Joos und Nico Litzel. *Realtime Analytics mit Apache Kafka*. URL: http://www.bigdata-insider.de/so-analysieren-sie-logdateien-mit-open-source-software-a-498565/ (besucht am 30. 07. 2017).

[158] Patrick W. Jordan u. a. *Usability Evaluation In Industry*. CRC Press, 1996. ISBN: 9780748404605.

[159] Željko Jovanović. „Data stream management system for moving sensor object data“. In: *Serbian Journal of Electrical Engineering* 12.1 (2015), S. 117–127.

[160] Ismael Juma. *Apache Kafka Security 101*. URL: https://www.confluent.io/blog/apache-kafka-security-authorization-authentication-encryption/ (besucht am 30. 07. 2017).

[161] Richard M. Karp und Scott Shenker. *A Simple Algorithm for Finding Frequent Elements in Streams and Bags*. International Computer Science Institute und University of California, 2003.

[162] Meggin Kearney und Matt Gaunt. *Set Up Your Build Tools*. URL: https://developers.google.com/web/tools/setup/setup-buildtools (besucht am 29. 05. 2017).

[163] Fabian Kern, Hrsg. *Wechselrichter-Kommunikation und Solarleistungsprognose*. Karlsruher Institut für Technologie: Kern, 2013.

[164] Paul Kitawa. *Fehler an PV-Anlagen mit thermografie detektieren*. URL: http://www.kitawa.de/thermografie-pv-anlagen (besucht am 31. 03. 2018).

[165] Knockout. *Knockout*. URL: http://knockoutjs.com/documentation/introduction.html (besucht am 29. 05. 2017).

[166] *Knowledge Discovery Metamodel (KDM)*. 2008. URL: http://www.omg.org/spec/KDM/ (besucht am 21. 04. 2017).

[167] Stefan Kolek und Oezkan Kirmaci. *Web Mining Clickstream Analysis*. University of Fribourg, 2006.

[168] Tobias Koppers u. a. *webpack*. 2012. URL: https://github.com/webpack/webpack (besucht am 20. 05. 2017).

[169] Tobias Koppers u. a. *webpack Getting Started*. 2012. URL: https://webpack.js.org/guides/get-started/ (besucht am 20. 05. 2017).

[170] Eugeniya Korotya. *Angular - Dependency Injection*. 19. Jan. 2017. URL: https://hackernoon.com/5-best-javascript-frameworks-in-2017-7a63b3870282.

[171] Jürgen Krämer. „Continuous queries over data streams-semantics and implementation“. Diss. 2007.

[172] Jürgen Krämer und Bernhard Seeger. *A temporal foundation for continuous queries over data streams*. Univ., 2004.

[173] Jürgen Krämer u. a. „Dynamic plan migration for snapshot-equivalent continuous queries in data stream systems“. In: *Current Trends in Database Technology–EDBT 2006* (2006), S. 497–516.

[174] Oliver Kramer. *Machine Learning in Evolution Strategies*. Bd. 20. Springer, 2016.

[175] Bernd Kratz, Hrsg. *Präzise Wind- und Photovoltaik-Vorhersagen über das Internet*. Kassel,FORUM GEOÖKOL. 25 (1): Kratz, 2014.

[176] Patrick Kroniga u. a. *ELBE: Validierung und Verbesserung von Lastprognosen (Projektphase 1)*. Bern (CH): Bundesamt für Energie BFE: Eidgenössisches Departement für Umwelt, Verkehr, Energie und Kommunikation UVEK, 2009.

[177] Rudolf Kruse u. a. *Computational Intelligence*. Springer Fachmedien Wiesbaden, 2015. DOI: 10.1007/978-3-658-10904-2. URL: https://doi.org/10.1007%2F978-3-658-10904-2.

[178] Rudolf Kruse u. a. *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze (German Edition)*. 2. Aufl. Springer Vieweg, Okt. 2015. ISBN: 9783658109035. URL: http://amazon.com/o/ASIN/3658109033/.

[179] E. et al. Lambert. *Common Distribution Power System Model (towards 61968-13 Ed 2.0)*. URL: http://cimug.ucaiug.org/Meetings/Europe2015/Documents/UCACIMug%5C_CDPSM%5C_v1.4.pdf (besucht am 08. 04. 2017).

[180] Bernhard Lange u. a., Hrsg. *Prognosen der zeitlich-räumlichen Variabilität von Erneuerbaren*. Fraunhofer IWES, ZSW, FVEE-Themen: Lange, 2011.

[181] N. Leroux und S. de Kaper. *Play for Java: Covers Play 2*. Manning Publications, 2014. ISBN: 9781617290909.

[182] Libscore. *Libscore Ember*. URL: http://libscore.com/#Ember (besucht am 29. 05. 2017).

[183] Libscore. *Libscore React*. URL: http://libscore.com/#React (besucht am 23. 05. 2017).

[184] Vaadin Ltd. 2017. URL: https://vaadin.com/docs/-/part/framework/introduction/intro-overview.html (besucht am 28. 05. 2017).

[185] Vaadin Ltd. 2017. URL: https://vaadin.com/pricing (besucht am 28. 05. 2017).

[186] Stefan Macke. 2016. URL: https://www.heise.de/developer/artikel/Moderne-Webentwicklung-mit-Java-EE-7-Ein-Experiment-3234862.html (besucht am 28. 05. 2017).

[187] magic890 u. a. *json.md*. URL: https://github.com/bower/spec/blob/master/json.md (besucht am 29. 05. 2017).

[188] Gurpreet Matharu u. a. „Empirical Study of Agile Software Development Methodologies: A Comparative Analysis“. In: *ACM SIGSOFT Software Engineering Notes, 06 February 2015, Vol.40(1), pp.1-6* 40.1 (2015), S. 1.

[189] Benjamin Matthiß, Jann Binder und Benjamin Schott, Hrsg. *Reduzierte Netzbelastung und optimierter Eigenverbrauch von dezentralen PV-Speichersystemen durch modellprädiktive Betriebsführung von Speichern*. OTTI-Konferenz ?Zukünftige Stromnetze für Erneuerbare Energien?,Berlin , Zentrum für Sonnenenergie- und Wasserstoff-Forschung (ZSW)Industriestraße 6, D-70565 Stuttgart: Matthiß, 2014.

[190] Dominik Maximini. *Scrum - Einführung in der Unternehmenspraxis : von starren Strukturen zu agilen Kulturen*. Berlin Heidelberg: Berlin Heidelberg : Springer, 2013.

[191] Dominik Maximini. *The Scrum Culture*. Cham: Springer International Publishing, Cham, 2015.

[192] mburger81. *Performance problems comparing to other chart libs*. Feb. 2017. URL: https://github.com/swimlane/ngx-charts/issues/248 (besucht am 06. 06. 2017).

[193] A.W McMorran. *An Introduction to IEC 61970-301 & 61968-11: The Common Information Model*. 2007.

[194] Swartz Merryanna, Daniel Wallace und Sharon Tkacz. „The Influence of Frame Rate and Resolution Reduction on Human Performance“. In: *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 36.18 (1992), S. 1440–1444. DOI: 10.1177/154193129203601817. URL: https://doi.org/10.1177/154193129203601817.

[195] Microsoft. *TypeScript - JavaScript that scales*. 1. Jan. 2012. URL: http://www.typescriptlang.org/.

[196] F. Milano. *Power System Modelling and Scripting*. Power Systems. Springer Berlin Heidelberg, 2010. ISBN: 9783642136696. URL: https://books.google.de/books?id=MQu7IqoLrfYC.

[197] Hoda Mohamed. *Data Stream Mining*. Article. Ain Shams University, Okt. 2010. URL: https://www.researchgate.net/publication/259647558_Data_Stream_Mining (besucht am 11. 04. 2017).

[198] James Momoh. *Smart Grid: Fundamentals of Design and Analysis*. Wiley, 2012.

[199] Inc. MongoDB. *MongoDB*. URL: https://www.mongodb.com/ (besucht am 12. 06. 2017).

[200] Janina Moshövel, Hrsg. *Projektvorstellung: PV-Speicher im Haushalt*. Institut für Hochspannungstechnik, Aachen: Moshövel, 2015.

[201] Gabriela Motroc. 2016. URL: https://jaxenter.com/how-much-harm-can-11-javascript-lines-of-code-can-do-npm-kik-azer-koculu-saga-125063.html (besucht am 18. 05. 2017).

[202] Mustache. *Logic-less templates.* URL: https://mustache.github.io/ (besucht am 30. 03. 2018).

[203] Detlef Neuhaus. *Deutschland steht vor der zweiten Energierevolution*. Okt. 2016. URL: http://www.focus.de/immobilien/experten/erneuerbare-energien-deutschland-steht-vor-der-zweiten-energierevolution_id_6056871.html (besucht am 07. 06. 2017).

[204] Alexander Neumann. 2016. URL: https://www.heise.de/developer/meldung/Analyse-Gibt-es-einen-Ausweg-aus-dem-Dilemma-um-Java-EE-3255563.html (besucht am 28. 05. 2017).

[205] ngx-charts. *ngx-charts Swimlange Community*. 2017. URL: https://swimlane.gitbooks.io/ngx-charts/content/contributing/community.html (besucht am 06. 06. 2017).

[206] Jakob Nielsen. *How Many Test Users in a Usability Study?* URL: https://www.nngroup.com/articles/how-many-test-users/ (besucht am 16. 01. 2018).

[207] Inc. npm. *Using a package.json*. URL: https://docs.npmjs.com/getting-started/using-a-package.json (besucht am 29. 05. 2017).

[208] Inc. npm. *What is npm?* URL: https://docs.npmjs.com/getting-started/what-is-npm (besucht am 29. 05. 2017).

[209] Das Odysseus Team. *Access framework*. URL: https://wiki.odysseus.informatik.uni-oldenburg.de/display/ODYSSEUS/Access+framework (besucht am 04. 09. 2017).

[210] Frank Olken und Le Gruenwald. „Data Stream Management: Aggregation, Classification, Modeling, and Operator Placement“. In: *IEEE Internet Computing* 12.6 (2008), S. 9–12. DOI: 10.1109/MIC.2008.121. URL: http://dx.doi.org/10.1109/MIC.2008.121.

[211] openKONSEQUENZ. *Projekt NetzDatenStrom (NDS)*. URL: https://www.openkonsequenz.de/projekt-netzdatenstrom (besucht am 12. 03. 2018).

[212] OpenLayers. *Introduction*. URL: https://openlayers.org/en/latest/doc/tutorials/introduction.html (besucht am 02. 06. 2017).

[213] OpenLayers. *Open Layers 2*. 2008. URL: http://docs.openlayers.org/ (besucht am 06. 06. 2017).

[214] Oracle. *MySQL*. URL: https://www.mysql.com/ (besucht am 12. 06. 2017).

[215] Oracle. *Oracle Database Express Edition*. URL: http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html (besucht am 12. 06. 2017).

[216] OSGi Alliance. *Benefits of Using OSGi*. 2017. URL: https://www.osgi.org/developer/benefits-of-using-osgi/ (besucht am 28. 04. 2017).

[217] Helmuth Partsch. *Requirements-Engineering systematisch - Modellbildung für softwaregestützte Systeme*. Springer, 1998. ISBN: 978-3-540-64391-3.

[218] Vanessa Peinhart u. a., Hrsg. *Der Photoelektrische Effekt*. Institut für Physik, Graz, Österreich: Peinhart, 2004.

[219] Sophie Pelland u. a., Hrsg. *Photovoltaic and Solar Forecasting: State of the Art*. Internationale Energy Agency Photovoltaic Power Systems Programme: Pelland, 2013.

[220] Stefan Peter. „Modellierung einer vollständig auf erneuerbaren Energien basierenden Stromerzeugung im Jahr 2050 in autarken, dezentralen Strukturen“. In: (2013).

[221] *PMML Clustering Example Code - KMeans cluster model*. 2015. URL: http://dmg.org/pmml/pmml%5C_examples/rattle%5C_pmml%5C_examples/AuditKMeans.xml (besucht am 27. 04. 2017).

[222] *PMML Powered*. 2017. URL: http://dmg.org/pmml/products.html (besucht am 21. 04. 2017).

[223] Protractor. *Protractor - End to end testing for Angular*. URL: http://www.protractortest.org/#/ (besucht am 28. 02. 2018).

[224] Quora. *How does D3.js compare with Google Charts API*. Juli 2015. URL: https://www.quora.com/How-does-D3-js-compare-with-Google-Charts-API (besucht am 06. 06. 2017).

[225] Alexander Neumann Rainald Menge-Sonnentag. *JavaScript-Bibliothek Vue.js: React-Konkurrent erscheint in Version 2.0*. URL: https://www.heise.de/developer/meldung/JavaScript-Bibliothek-Vue-js-React-Konkurrent-erscheint-in-Version-2-0-3342484.html (besucht am 29. 05. 2017).

[226] RapidMiner. *Data Science Platform - RapidMiner*. URL: https://rapidminer.com/ (besucht am 31. 03. 2018).

[227] *RDF 1.1 Concepts and Abstract Syntax*. URL: https://www.w3.org/TR/rdf11-concepts/ (besucht am 22. 04. 2017).

[228] *RDF 1.1 XML Syntax*. URL: https://www.w3.org/TR/rdf-syntax-grammar/ (besucht am 22. 04. 2017).

[229] Rechnerphotovoltaik.de. *Photovoltaik*. URL: http://www.rechnerphotovoltaik.de/pv/photovoltaik/ (besucht am 07. 06. 2017).

[230] redislabs. *Redis*. URL: https://redis.io/ (besucht am 12. 06. 2017).

[231] regenerative-energie24. *Photovoltaik Eigenverbrauch: So funktioniert es*. URL: http://www.regenerative-energie24.de/photovoltaik/photovoltaik-eigenverbrauch (besucht am 07. 06. 2017).

[232] Lior Rokach. „Ensemble-based classifiers“. In: *Artificial Intelligence Review* 33.1 (2010), S. 1–39.

[233] Serafin von Roon, Hrsg. *Empirische Analyse über die Kosten des Ausgleichs von Prognosefehlern der Wind- und PV-Stromerzeugung*. Internationale Energiewirtschaftstagung an der TU Wien,Forschungsstelle für Energiewirtschaft e.V., Am Blütenanger 71, 80995 München: Roon, 2011.

[234] Jeff Rubin und Dana Chisnel. *Handbook of Usability Testing*. Wiley Publishing, Inc., 2008. ISBN: 978-0-470-18548-3.

[235] *Run your analytic anywhere*. 2015. URL: http://dmg.org/pfa/ (besucht am 21. 04. 2017).

[236] Muhammad Sarwar und Bilal Asad. „A Review on Future Power Systems; Technologies and Research for Smart Grids“. In: (2016).

[237] Jacob Schatz. *Why We Chose Vue.js*. URL: https://about.gitlab.com/2016/10/20/why-we-chose-vue/ (besucht am 29. 05. 2017).

[238] Felix Schnorr und Heinrich Hinze. *Erstellung von Lastprognosen für den elektrischen Strombedarf von Einfamilienhäusern*. 2014.

[239] Gerda Schubert, Hrsg. *Modellierung der stündlichen Photovoltaik- und Windstromeinspeisung in Europa*. Fraunhofer Institut für System- und Innovationsforschung,Karlsruhe, 12. Symposium Energieinnovation, Graz/Austria: Schubert, 2012.

[240] Steffen Schütte. „Simulation Model Composition for the Large-Scale Analysis of Smart Grid Control Mechanisms“. Diss. Carl von Ossietzky Universität Oldenburg, Nov. 2013.

[241] Jan Sören Schwarz. „Adaptive State Estimation in Odysseus“. Magisterarb. Carl von Ossietzky Universität Oldenburg, Feb. 2015.

[242] Ben Shneiderman. *User interface design : [effektive Interaktion zwischen Mensch und Maschine; Leitfaden für intelligentes Schnittstellendesign; was Programmierer und Designer über den Anwender wissen müssen]. Designing the user interface <dt.>* Bonn : mitp-Verl. 2002.

[243] Yogesh Simmhan u. a. „Adaptive rate stream processing for smart grid applications on clouds“. In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM. 2011, S. 33–38.

[244] Yogesh Simmhan u. a. „Adaptive rate stream processing for smart grid applications on clouds“. In: *Proceedings of the 2nd international workshop on Scientific cloud computing*. ACM. 2011, S. 33–38.

[245] J. Simmins. *Common Information Model Primer: Thrid Edition*. Techn. Ber. 3002006001. Palo Alto, CA: EPRI, 2015.

[246] Smartgrids-net. *Die Netzwerkidee*. URL: http://www.smartgrids-net.de/ (besucht am 07. 06. 2017).

[247] Fabian Sobiech. *Abbildung von Synergiepotenzialen zwischen IT-Anforderungen in Scrum*. Bd. 95. Wiesbaden: Springer Fachmedien Wiesbaden, Wiesbaden, 2016.

[248] Softizy. *MariaDB 10.1 vs MySQL 5.7*. URL: https://www.softizy.com/blog/mariadb-10-1-mysql-5-7-performances-ibm-power-8/ (besucht am 12. 06. 2017).

[249] Vijay K Sood u. a. „Developing a communication infrastructure for the smart grid“. In: *Electrical Power & Energy Conference (EPEC), 2009 IEEE*. IEEE. 2009, S. 1–7.

[250] Spektrum. *Photoeffekt*. URL: http://www.spektrum.de/lexikon/physik/photoeffekt/11182 (besucht am 07. 06. 2017).

[251] International Organization for Standardization. *Ergonomics of human-system interaction*. ISO 9241-110:2006. Apr. 2006.

[252] tannerlinsley. *Possible to further improve animation performance?* März 2015. URL: https://github.com/chartjs/Chart.js/issues/1013 (besucht am 06. 06. 2017).

[253] Jörn Trefke u. a. „Smart Grid Architecture Model use case management in a large European Smart Grid project“. In: *ISGT Europe*. IEEE, 2013, S. 1–5.

[254] M. Uslar u. a. *The Common Information Model CIM: IEC 61968/61970 and 62325 - A practical introduction to the CIM*. Power Systems. Springer Berlin Heidelberg, 2012. ISBN: 9783642252150. URL: https://books.google.de/books?id=cdw6gtzwc-QC.

[255] OFFIS e. V. *How mosaik communicates with a simulator*. URL: http://mosaik.readthedocs.io/en/latest/mosaik-api/overview.html (besucht am 01. 04. 2017).

[256] OFFIS e. V. *Integrating a Model in Java*. URL: http://mosaik.readthedocs.io/en/latest/tutorials/tutorial_api-java.html (besucht am 01. 04. 2017).

[257] OFFIS e. V. *Odysseus*. URL: http://mosaik.readthedocs.io/en/latest/ecosystem/odysseus.html (besucht am 01. 04. 2017).

[258] OFFIS e. V. *Overview: Mosaik's main components*. URL: http://mosaik.readthedocs.io/en/latest/overview.html%5C#mosaik-s-main-components (besucht am 01. 04. 2017).

[259] OFFIS e. V. *Overview: What's mosaik supposed to do?* URL: http://mosaik.readthedocs.io/en/latest/overview.html%5C#what-s-mosaik-supposed-to-do (besucht am 01. 04. 2017).

[260] OFFIS e. V. *Scenario definition*. URL: http://mosaik.readthedocs.io/en/latest/scenario-definition.html (besucht am 01. 04. 2017).

[261] OFFIS e. V. *Scheduling and simulation execution*. URL: http://mosaik.readthedocs.io/en/latest/scheduler.html (besucht am 01. 04. 2017).

[262] OFFIS e. V. *The high-level API*. URL: http://mosaik.readthedocs.io/en/latest/mosaik-api/high-level.html (besucht am 01. 04. 2017).

[263] OFFIS e. V. *The mosaik API*. URL: http://mosaik.readthedocs.io/en/latest/mosaik-api/index.html (besucht am 01. 04. 2017).

[264] OFFIS e. V. *The simulator manager*. URL: http://mosaik.readthedocs.io/en/latest/simmanager.html (besucht am 01. 04. 2017).

[265] vagusae. *Drawing line graph, slow performances when a lot of points to draw*. März 2016. URL: https://github.com/jtblin/angular-chart.js/issues/324 (besucht am 06. 06. 2017).

[266] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, 2000. DOI: 10.1007/978-1-4757-3264-1. URL: https://doi.org/10.1007%2F978-1-4757-3264-1.

[267] Vue. *Comparison with Other Frameworks*. URL: https://vuejs.org/v2/guide/comparison.html (besucht am 29. 05. 2017).

[268] Vue. *Components*. URL: https://vuejs.org/v2/guide/components.html (besucht am 29. 05. 2017).

[269] Vue. *Reactivity in Depth*. URL: https://vuejs.org/v2/guide/reactivity.html (besucht am 29. 05. 2017).

[270] Vue. *Transition Effects*. URL: https://vuejs.org/v2/guide/transitions.html (besucht am 29. 05. 2017).

[271] Mike Wacker. *Just Say No to More End-to-End Tests*. URL: https://testing.googleblog.com/2015/04/just-say-no-to-more-end-to-end-tests.html (besucht am 16. 06. 2017).

[272] Joe Walnes. *Introducing SmoothiCharts*. 2010. URL: https://joewalnes.com/2010/08/10/introducing-smoothie-charts/ (besucht am 06. 06. 2017).

[273] Joe Walnes und Drew Noakes. *MIT License*. 2014. URL: http://smoothiecharts.org/LICENSE.txt (besucht am 02. 06. 2017).

[274] Dirk Weil. 2009. URL: https://jaxenter.de/java-ee-6-auf-einen-blick-8960 (besucht am 28. 05. 2017).

[275] Johannes Weniger u. a., Hrsg. *Bedeutung von prognosebasierten Betriebsstrategien für die Netzintegration von PV-Speichersystemen*. Hochschule für Technik und Wirtschaft (HTW) Berlin, Symposium Photovoltaische Solarenergie, Kloster Banz, Bad Staffelstein: Weniger, 2014.

[276] *What is big data?* 2012. URL: https://www.oreilly.com/ideas/what-is-big-data (besucht am 21. 04. 2017).

[277] Wikipedia. *Comparison of JavaScript charting frameworks*. Apr. 2017. URL: https://en.wikipedia.org/wiki/Comparison_of_JavaScript_charting_frameworks (besucht am 06. 06. 2017).

[278] Wikipedia. *Single Responsibility Principle*. URL: https://en.wikipedia.org/wiki/Single_responsibility_principle (besucht am 20. 12. 2017).

[279] Andrea Windolph. 2015. URL: https://projekte-leicht-gemacht.de/blog/pm-methoden-erklaert/nutzwertanalyse/.

[280] Alain Filbois Xavier Cavin Christophe Mion. „COTS cluster-based sort-last rendering: performance evaluation and pipelined implementation“. In: *VIS 05. IEEE Visualization, 2005.* Okt. 2005, S. 111–118. DOI: 10.1109/VISUAL.2005.1532785.

[281] Evan You. *Vue*. URL: https://github.com/vuejs/vue (besucht am 29. 05. 2017).

[282] A. Young und M. Harter. *Node.Js in Practice*. Manning Publications Company, 2015.

[283] Christof Zangemeister. *Nutzwertanalyse in der Systemtechnik*. 1976. ISBN: ISBN 3-923264-00-3.

[284] ZVEI/BDE. *Smart Grids in Deutschland - Handlungsfelder für Verteilnetzbetreiber auf dem Weg zu intelligenten Netzen, Frankfurt/Berlin, 2012*.

Index

Q

R

S

T

U

V

W