

Bachelorarbeit

Entwicklung eines webbasierten Informationssystems zur energetischen Abbildung von Stadtquartieren anhand des Anwendungsbeispiels des Forschungszentrums Jülich

Building a web-based information system for district energy mapping
based on the application example Research Center Jülich

Univ.-Prof. Dr. Torsten Wolfgang Kuhlen

Virtual Reality Group
Fakultät für Mathematik, Informatik und Naturwissenschaften
RWTH Aachen University

Univ.-Prof. Dr.-Ing. habil. Christoph van Treeck

Lehrstuhl für Energieeffizientes Bauen E3D
Fakultät für Bauingenieurwesen
RWTH Aachen University

Vorgelegt von:
Felix Kirchmann
331290

Betreuer:
Eric Spinnräker, M. Sc.
Daniel Koschwitz, M. Sc.

Abgegeben am:
14. August 2017

Eidesstattliche Versicherung

Name, Vorname

Matrikelnummer (freiwillige Angabe)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Ort, Datum

Unterschrift

*Nichtzutreffendes bitte streichen

Belehrung:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

Ort, Datum

Unterschrift

Aufgabenstellung

Für die nachhaltige Energieversorgung von Gebäuden und Stadtquartieren mit Wärme, Kälte und Strom bedarf es ganzheitlicher Energiekonzepte, die städtebauliche und technische Aspekte im Einklang mit den Bedürfnissen der Nutzer berücksichtigen. Aufgrund auslaufender Verträge mit den örtlichen Energieversorgern muss sich das Forschungszentrum Jülich dahingehend neu ausrichten. Zusammen mit einem hohen Sanierungsbedarf der Bestandsgebäude, voraussichtlich notwendiger Neubauten und dem Ziel vorwiegend regenerative und emissionsarme Energien in die Versorgungsinfrastruktur zu integrieren, bietet sich somit eine im hohen Maße geeignete Plattform zur Entwicklung neuer Energiekonzepte. An diesem Punkt knüpft das Projekt „EnEff-Campus Living Roadmap“ an.

Um den Mitarbeiter/-innen des Forschungszentrums relevante Gebäude- und Netzparameter unter Einbindung des Nutzerverhaltens zugänglich zu machen, soll ein webbasiertes Informationssystem zur Verfügung gestellt werden, die neben der Visualisierung des Forschungszentrums als virtuelle Liegenschaft auch weitergehende Informationen zu Energiesystemen bereitstellt.

Im Rahmen dieser Bachelorarbeit soll auf Grundlage einer bestehenden objektorientierten Messdatenbank mit GIS-Anbindung eine Weboberfläche mit Schnittstellen zu vorhandenen Auswertungstools erstellt werden. Die Weboberfläche soll zum einen das Forschungszentrum Jülich visualisieren und einen Zugang zu relevanten Gebäude- und Netzinformationen ermöglichen. Hierbei soll insbesondere untersucht werden, welche Daten in welchem Umfang und in welcher Art dargestellt werden, um die wesentlichen Aussagen den Mitarbeitern/-innen entsprechend zugänglich zu machen. Mit Hilfe von zu definierenden Schnittstellen soll weiterhin der Aufruf von bestehenden Auswertungstools möglich sein, wodurch weitergehende Optimierungsrechnungen durchgeführt werden können und die wesentlichen Ergebnisse dargestellt werden können.

Der Arbeit ist eine einseitige Zusammenfassung der Ergebnisse in deutscher und englischer Sprache voran zu stellen. Nach Abgabe der Arbeit sind die Ergebnisse in einem 15minütigen Vortrag zu präsentieren.

Zusammenfassung

Um das Einsparpotenzial von Gebäudegruppen wie Stadtquartieren oder Liegenschaften auszuschöpfen bietet es sich an, nicht jedes Bauwerk einzeln, sondern alle Gebäude als ein zusammenhängendes System zu betrachten. Da die Verträge des Forschungszentrums Jülich mit den lokalen Energieversorgern innerhalb der nächsten Jahre auslaufen, wurde das Forschungsprojekt „EnEff-Campus Living Roadmap“ gestartet. In dessen Rahmen wird ein solches ganzheitliches Energiekonzept für das Forschungszentrum entwickelt, welches die gesamte Liegenschaft inklusive der gebäudeübergreifenden Versorgungsleitungen umfasst. Durch die Erfassung von relevanten Gebäudedaten und technischen Einrichtungen und der Einbindung von Simulationsmodellen können beispielsweise Energieverbrauchs- werte prognostiziert werden. In den Gebäuden sind zudem Sensoren installiert, wodurch die Simulationsmodelle anhand von Messdaten evaluiert werden können.

Dazu wird im Rahmen dieser Arbeit eine Webanwendung entwickelt, welche das virtuelle Abbild des Forschungszentrums visualisiert und Sensormesswerte graphisch darstellt. Nach einer kurzen Darstellung der Projekthintergründe beginnt die Ausarbeitung mit einer Erläuterung der relevanten theoretischen und softwaretechnischen Grundlagen. Darauf aufbauend folgt eine Beschreibung der verwendeten Datenquellen sowie die Definition der Anforderungen, die die Anwendung erfüllen soll. Das folgende Kapitel stellt dar, wie diese Anforderungen softwaretechnisch umgesetzt sind. In der Auswertung wird schließlich anhand einiger Messungen darauf eingegangen, wie die durch die Anwendung benötigte Rechenleistung für die Darstellung der Daten des Forschungszentrums reduziert werden kann. Die Arbeit schließt mit einer Perspektive darauf, in welche Richtungen die Anwendung im zukünftigen Verlauf des Forschungsprojekts „EnEff-Campus Living Roadmap“ weiterentwickelt werden kann.

Abstract

To fully utilize energy savings potentials in groups of buildings (such as districts or compounds), observing them as a connected system may yield better results than examining each building individually. Since the Jülich Research Center's contracts with local energy suppliers are due to expire within the next years, the research project "EnEff Campus Living Roadmap" was started. Its objective is to plan the energy production, transmission and consumption within the research center using methods that simultaneously observe its buildings as well as the supply lines between them. To achieve this, a virtual representation of the compound, including its supply systems, was created, thus enabling energy consumption values to be estimated using a simulation. Additionally, the buildings are augmented with sensors continuously collecting information on energy usage and environmental parameters - this data can be compared with simulation results to evaluate the quality of the simulation model.

The Web Application developed as a part of this thesis lays the foundation for this evaluation. It visualizes the research center's virtual representation and presents graphs of collected sensor data. After a short introduction of the research project "EnEff Campus Living Roadmap", some of the theoretical knowledge relevant to this thesis is explained. This is followed by a description of the database from which the application sources the displayed data and a definition of the requirements it implements. The next chapter discusses the technical details of this implementation and how they relate to the requirements. In the final chapter, measurements from the application are used to illustrate how the computational resources required to display the project's data can be lowered. The thesis concludes with a perspective on how the application may be further developed.

Inhaltsverzeichnis

Abkürzungsverzeichnis	XIII
Abbildungsverzeichnis	XV
1 Einleitung	1
2 Theoretische Grundlagen	3
2.1 Baualtersklassen	3
2.2 Spezifischer Energieverbrauch	4
2.3 Douglas-Peucker Algorithmus	4
2.4 Geoinformationssysteme	5
2.4.1 Keyhole Markup Language	5
2.4.2 CityGML	6
2.4.3 PostGIS	7
2.4.4 3DCityDB	7
2.4.5 OpenStreetMap	7
2.4.6 OpenLayers	7
2.5 Model-View-Controller	8
2.6 Webentwicklung in Java	9
2.6.1 JAX-RS	9
2.6.2 Jersey	10
2.6.3 FreeMarker	10
2.6.4 jOOQ	11
3 Ausgangssituation	13
3.1 Aktueller Stand	13
3.2 Anforderungen	15
3.2.1 Funktionen	15
3.2.2 Kompatibilität	16
3.2.3 Langfristige Wartbarkeit	16

4 Realisierung des Informationssystems	19
4.1 Modularisierung	19
4.2 Auswahl der Bibliotheken	19
4.2.1 Jersey und FreeMarker	20
4.2.2 OpenLayers	22
4.3 EnergyData-API	22
4.3.1 API der Datenbank-Abstraktionsschicht	24
4.3.2 Aggregationen und Kardinalitäten	26
4.3.3 Weitere Datenbanktabellen	27
4.3.4 Paralleler Abruf von Zeitreihen	27
4.4 EnergyData-Web	30
4.4.1 Implementierung der Gebäudeübersicht	30
4.4.2 Darstellung der Messwertgraphen	32
5 Auswertung im Anwendungsfall Campus Jülich	35
5.1 Vereinfachung von Gebäudegeometrie	35
5.1.1 Entfernen von Gebäuden unterhalb der Toleranz	36
5.1.2 Auswirkungen auf die Kartendarstellung	36
5.1.3 Punkteanzahl in Abhängigkeit von der Toleranz	38
5.1.4 Auswahl der Konfigurierten Toleranz	38
5.1.5 Auswirkung auf Mobile Endgeräte	38
5.2 Parallelismus beim Abruf von Zeitreihen	41
5.3 Messfehler	42
5.3.1 Falsche Werte am Anfang von Messreihen	43
5.3.2 Einzelne Fehler in Messreihen	43
5.3.3 Temporäres Aussetzen der Messungen	43
6 Fazit	45
7 Ausblick	47
7.1 Kartendarstellung	47
7.2 Messfehlerbereinigung	48
7.3 Simulationsanbindung	48
Literaturverzeichnis	49

Abkürzungsverzeichnis

Abkürzung	Bedeutung
FZJ	Forschungszentrum Jülich
GIS	Geoinformationssystem
NRF	Netto-Raumfläche
KML	Keyhole Markup Language
CityGML	City Geography Markup Language
OGC	Open Geospatial Consortium
LoD	Level of Detail
ADE	Application Domain Extension
API	Application Programming Interface
MVC	Model-View-Controller
Java SE	Java Platform, Standard Edition
JAX-RS	Java API for RESTful Web Services
JAR	Java Archive
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
jOOQ	Java Object Oriented Querying
SQL	Structured Query Language
PS	Primärer Schlüssel
URL	Uniform Resource Locator
FPS	Frames pro Sekunde

Index	Bedeutung
m^2	Quadratmeter
kWh	Kilowattstunden
$\frac{kWh}{a}$	Kilowattstunden pro Jahr
$\frac{kWh}{m^2 \cdot a}$	Spezifischer Energieverbrauch in Kilowattstunden pro m^2 und Jahr
ms	Millisekunden

Abbildungsverzeichnis

1.1	Karte des Forschungszentrums Jülich. Aus [11]	1
2.1	Detailstufen in CityGML. Aus [22]	6
2.2	Beispielaufruf von Freemarker	10
2.3	Funktionsweise von jOOQ	11
3.1	Von der Anwendung verwendeter Teil des Datenbankschemas. Die Primären Schlüssel der jeweiligen Tabelle sind mit PS markiert.	14
4.1	Komponenten der Anwendung <i>EnergyData</i> , die Pfeile stellen Abhängigkeiten dar. PostGIS-Logo aus [37]	20
4.2	API der Datenbank-Abstraktionsschicht	23
4.3	Beispielhafter Zeitverlauf bei der Ausführung von <code>ParallelQueryExecutor</code> mit $n = 2$ Threads	29
4.4	Datenfluss beim Abruf der Karte, Pfeile in Flussrichtung. Logos aus [34], [35], [37]	30
5.1	Gebäudegeometrie nach Vereinfachung mit verschiedenen Toleranzen ε in Metern	37
5.2	Anzahl der Punkte in der vereinfachten Gebäudegeometrie (bei Betrachtung aller Gebäude), in Abhängigkeit von der Toleranz.	39
5.3	Frames pro Sekunde (FPS) beim Scrollen über die Kartendarstellung auf einem mobilen Endgerät.	40
5.4	Abfragedauer von Sensormesswerten, unter Verwendung verschiedener Abfragemethoden.	41
5.5	Fehlerhafte Daten am Anfang von Messreihen.	42
5.6	Fehlerhafte Messwerte in Messreihen, durch welche auch die Skalierung beeinflusst wird.	43
5.7	Temporäres Aussetzen der Messungen.	44

1 Einleitung



Abbildung 1.1: Karte des Forschungszentrums Jülich. Aus [11]

Um bei Gebäuden möglichst große Energieeinsparungen mit geringem Kostenaufwand zu erreichen, bieten sich zunehmend ganzheitliche Energiekonzepte an. Diese zeichnen sich dadurch aus, dass sie kein Bauwerk einzeln betrachten, sondern das Einsparpotenzial einer gesamten Liegenschaft oder eines Stadtquartiers möglichst maximieren. Dazu werden neben den energetischen Eigenschaften der Gebäude an sich auch die Versorgungsnetze (zum Beispiel für Strom, Wärme und Kälte) betrachtet.

In dieser Bachelorarbeit wird diesbezüglich dazu das in Abbildung 1.1 dargestellte Forschungszentrum Jülich (FZJ) betrachtet, welches auf einer Fläche von 2200 m^2 circa 5600 Mitarbeiter beschäftigt (Stand 2015) [8]. Mehrere Faktoren erfordern für diese Liegenschaft ein neues ganzheitliches Energiekonzept. So haben viele der Bestandsgebäude inzwischen ein Alter von über 40 Jahren erreicht und entsprechen deswegen meist dem damaligen Stand der Bautechnik. Um ihren Energiebedarf zu senken, sind sie nach den

1 Einleitung

heutigen energetischen Anforderungen zu sanieren. Weiterhin laufen die Verträge mit den lokalen Energieversorgern, insbesondere die aktuelle Fernwärmeversorgung des Forschungszentrums durch das Braunkohlekraftwerk Weisweiler in wenigen Jahren aus. Dies erfordert tiefgreifendere Änderungen der Versorgungssysteme, insbesondere im Bereich der Wärmeversorgung. Das Forschungszentrum setzt sich diesbezüglich das Ziel, die zukünftige Energieversorgung möglichst umweltfreundlich zu gestalten und möchte daher auf vorwiegend emissionsarme und erneuerbare Energiequellen umsteigen [27].

Aus diesen Gründen wurde das Forschungsprojekt „EnEff Campus Living Roadmap“ gestartet, welches das Ziel hat, ein ganzheitliches Energiekonzept zu entwickeln. Im Rahmen dieses Projekts wird ein virtuelles Abbild des Forschungszentrums erstellt, mithilfe dessen die Energieerzeugung, -verarbeitung und der Verbrauch aller Gebäude und Versorgungseinrichtungen simuliert und somit prognostiziert werden können. Das Forschungszentrum eignet sich für diese Aufgabenstellung sehr gut, da für ein Großteil der Gebäude ein Energiemonitoringsystem existiert. Die vorhandenen Sensoren zeichnen unter anderem den Strom-, Wärme- und Kälteverbrauch der Gebäude auf. Weiterhin werden laufend Messwerte hinsichtlich der Vor- und Rücklauftemperaturen im Nahwärmenetz aufgenommen. Sensoren im Außenbereich zeichnen zusätzlich Umwelteinflüsse wie die Außentemperatur, Niederschlagsmenge, Windrichtung und -stärke sowie den Luftdruck auf. Diese Messwerte können laufend mit den Simulationsergebnissen abgeglichen werden, um bei Abweichungen das virtuelle Abbild des FZJ verbessern zu können [27]. An dieses Projekt knüpft die im Rahmen dieser Arbeit entwickelte Software an. Das Ziel ist die Erstellung einer Webanwendung, welche das virtuelle Abbild des Forschungszentrums visualisiert und den Abruf sowie die Darstellung der gemessenen Sensorwerte ermöglicht. Die Anwendung wird dabei so entwickelt, dass sie später um eine Anbindung an die bestehende Simulationsinfrastruktur ergänzt werden kann. Damit soll letztendlich eine Plattform geschaffen werden, mithilfe derer die laufenden Energiedaten des FZJ beobachtet und das Simulationsmodell des Projekts „Living Roadmap“ evaluiert werden können. Durch die Verwendung standardisierter Schnittstellen kann diese Plattform im Idealfall auch auf die Sensor- und Versorgungsnetze anderer Liegenschaften übertragen werden.

2 Theoretische Grundlagen

In diesem Kapitel werden einige Konzepte vorgestellt, welche für die im Rahmen dieser Arbeit entwickelte Software verwendet werden. Dazu gehören zunächst zwei Eigenschaften, die für die Einschätzung der energetischen Merkmale eines Gebäudes relevant sind: die Baualtersklasse und der spezifische Energieverbrauch. Danach wird der Douglas-Peucker Algorithmus vorgestellt, mit dem beliebige Kantenzüge (also beispielsweise auch Umrisse auf einer Karte) vereinfacht werden können. Darauf folgt eine Beschreibung der verschiedenen Geoinformationssysteme (GIS) sowie des *Model-View-Controller*-Prinzips, auf denen die Anwendung basiert. Der letzte Abschnitt behandelt dann einige Komponenten, die für die Implementierung dieser Software als Java-Webanwendung relevant sind.

2.1 Baualtersklassen

Das Bundesministerium für Wirtschaft und Energie sowie das Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit haben im Jahr 2015 eine Gebäudetopologie veröffentlicht, welche die in Deutschland stehenden Nichtwohngebäude abhängig von ihrem Baujahr in Baualtersklassen einordnet (siehe Tabelle 2.1). Diese sollen es vor allem ermöglichen, anhand der Baualtersklasse eine grobe Einschätzung bezüglich der thermischen Qualität der Gebäudehülle vornehmen zu können. Dafür orientieren sie sich unter anderem an den Daten technischer und baurechtlicher Neuerungen, die einen maßgeblichen Einfluss auf den Heizwärmebedarf der ab diesem Zeitpunkt konstruierten Gebäude haben [5][18].

Tabelle 2.1: Baualtersklassen, entnommen aus [5], S. 9. Buchstabenkennzeichnung aus [18], S. 1

Baualtersklasse	von Jahr	bis Jahr
A	-	1918
B	1919	1948
C	1949	1957
D	1958	1968
E	1969	1978
F	1979	1983
G	1984	1994
H	1995	2001
I	2002	-

2.2 Spezifischer Energieverbrauch

Der spezifische Energieverbrauch (auch *Energieverbrauchskennwert* genannt) bildet die Grundlage für eine Einschätzung der Energieeffizienz eines Gebäudes. Dazu wird der Energieaufwand für das Heizen oder Kühlen des Gebäudes (in Kilowattstunden, kWh) über den Zeitraum eines Jahres in der Einheit $\frac{kWh}{a}$ herangezogen. Dieser wird dann durch die Gesamtgrundfläche aller nutzbaren Räume des Gebäudes (die *Netto-Raumfläche*, NRF) geteilt. Der daraus resultierende spezifische Energieverbrauch (Einheit: $\frac{kWh}{m^2 \cdot a}$) gibt dann die jährliche Energiemenge an, die durchschnittlich für das Beheizen oder Kühlen jedes nutzbaren Quadratmeters aufgewendet wird [41][9].

2.3 Douglas-Peucker Algorithmus

Der von Douglas und Peucker 1973 veröffentlichte Algorithmus dient dazu, einen beliebigen Kantenzug zu approximieren, wobei das Resultat bei kleinerer oder gleicher Punktezahl dem Original möglichst ähnlich sehen soll. Hierzu wird ein maximaler Abstand (die Toleranz ε) angegeben, den die Punkte der ausgegebenen Approximation zum eingegebenen Kantenzug haben dürfen. Der Algorithmus verarbeitet dann rekursiv die Eingabe und gibt eine Teilmenge der eingegebenen Punkte aus, die diese Eigenschaft erfüllen. Dabei wird wie folgt vorgegangen:

Sei

$$K = (P_1, \dots, P_n), n \in \mathbb{N}$$

der eingegebene Kantenzug und $\varepsilon \in \mathbb{R}, \varepsilon > 0$ die Toleranz. Wenn der Kantenzug aus maximal zwei Punkten besteht (also $n \leq 2$ gilt), wird er unverändert wieder ausgegeben. Ansonsten wird nun eine Linie $\overline{P_1 P_n}$ von P_1 nach P_n gezogen und ein Punkt P_x bestimmt, der maximalen Abstand d_{max} zu dieser Linie hat:

$$d_{max} = \max_{i \in \{2, \dots, n-1\}} d(P_i, \overline{P_1 P_n})$$

$$P_x \in \{P_1, \dots, P_n\}, d(P_x, \overline{P_1 P_n}) = d_{max}$$

Wenn dieser maximale Abstand nun innerhalb der Toleranz liegt ($d_{max} \leq \varepsilon$), so ist keiner der inneren Punkte P_2, \dots, P_{n-1} weiter als ε von dieser Linie entfernt. Also gibt der Algorithmus den Kantenzug (P_1, P_n) aus. Liegt der maximale Abstand jedoch nicht innerhalb der Toleranz, so wird der Algorithmus rekursiv jeweils für die Kantenzüge $K_1 = (P_1, \dots, P_x)$ und $K_2 = (P_x, \dots, P_n)$ ausgeführt. Sei dann $K'_1 = (P_1, P'_2, \dots, P'_{x-1}, P_x)$ und $K'_2 = (P_x, P'_{x+1}, \dots, P'_{n-1}, P_n)$ jeweils das Ergebnis der Ausführung für die Eingabe K_1 und K_2 , so wird am Ende der Kantenzug

$$(P_1, P'_2, \dots, P'_{x-1}, P_x, P'_{x+1}, \dots, P'_{n-1}, P_n)$$

ausgegeben [10].

2.4 Geoinformationssysteme

Geoinformationssysteme (GIS) dienen der Erfassung, Weiterverarbeitung und Anzeige geografischer Informationen [7]. Im Folgenden werden die in dieser Arbeit verwendeten GIS kurz vorgestellt.

2.4.1 Keyhole Markup Language

Die Keyhole Markup Language ist ein XML-basiertes Dateiformat, welches das Speichern von einfachen geografischen Objekten und ortsgebundenen Metadaten ermöglicht. Anfangs noch kommerziell entwickelt [28], wurde es 2008 durch das *Open Geospatial Consortium* (OGC) offen standardisiert und kann somit frei verwendet werden [31]. Anwendungsbeispiele für KML reichen vom Speichern der Koordinaten von Sehenswürdig-

keiten (*Points of Interest*, POI) und Routen [28] bis hin zur Darstellung ganzer Gebäude [30].

2.4.2 CityGML

Die City Geography Markup Language (CityGML), welche ebenfalls auf XML basiert und 2008 vom OGC standardisiert wurde, dient im Kontrast zu KML dazu, komplexere Abbilder von Bauwerken und ganzen Städten zu erfassen. Sie definiert fünf Detailstufen (*Level of Detail*, LoD), für die jeweils Geometriedaten abgespeichert werden können. Wie in Abbildung 2.1 dargestellt, reicht die Genauigkeit dabei von groben Straßen- und Gebäudeumrissen (LoD0) bis hin zu sehr detaillierten Modellen von Fassaden (LoD3) und Innenräumen (LoD4) [16].

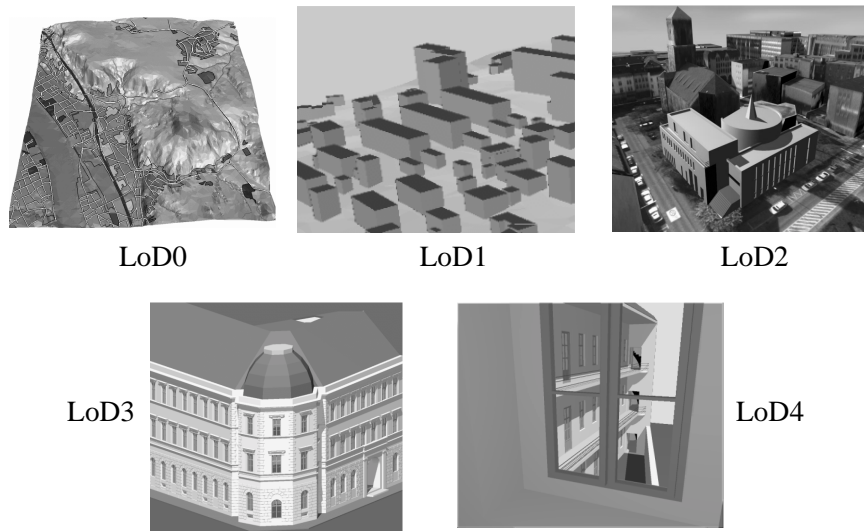


Abbildung 2.1: Detailstufen in CityGML. Aus [22]

Mittels sogenannter *Application Domain Extensions* (ADEs) kann CityGML zudem erweitert werden, um nicht durch den Standard abgedeckte Daten abzuspeichern [16]. Ein Beispiel hierfür ist die *UtilityNetworkADE*, welche gebäudeübergreifende Versorgungsnetze unter Anderem für Wasser, Gas und Strom modelliert [4]. Darauf baut wiederum die *EnergyADE* auf, mit der die in Gebäuden installierten Energieerzeuger und -verbraucher sowie ihre Verbindung mit den Versorgungsnetzen abgebildet werden können. Zusätzlich erlaubt sie auch die Speicherung der thermischen Eigenschaften eines Bauwerks. Dadurch wird mit hinreichend präzisen Daten eine Simulation der Gebäudeenergetik ermöglicht [29].

2.4.3 PostGIS

Bei *PostGIS* handelt es sich um eine Erweiterung der Datenbank *PostgreSQL* um Funktionen für die Speicherung und Verarbeitung geografischer Informationen. So stellt sie unter Anderem spezielle Datentypen bereit, mit denen sich Umrisse von Gebäuden oder anderen Strukturen als Vektordaten erfassen lassen. Diese Vektordaten können mittels des in PostGIS implementierten Douglas-Peucker Algorithmus vereinfacht werden, was eine weniger rechenintensive Darstellung auf Kosten der Detailgenauigkeit erlaubt. Auch ermöglicht die Erweiterung den Export der Daten in verschiedene Standardformate, unter anderem KML. [30].

2.4.4 3DCityDB

PostGIS stellt selber nur Funktionen und Datentypen bereit, es definiert jedoch nicht, wie diese im Rahmen eines relationalen Datenbankschemas verwendet werden sollen. So wäre es möglich, basierend auf PostGIS für jeden Anwendungsfall (zum Beispiel den Campus Jülich) ein spezielles Datenbankschema zu entwickeln. Aufgrund des damit verbundenen Aufwands und dem einhergehenden Mangel an Interoperabilität ist es jedoch oft wünschenswert, ein standardisiertes Schema zu verwenden. Ein solches Schema ist die von der TU München entwickelte 3DCityDB. Aufbauend auf CityGML definiert dieses Schema, wie beliebige CityGML-Daten konvertiert werden können, um sie in einer relationalen räumlichen Datenbank wie PostGIS abzulegen [38]. Hierzu werden vom Projekt auch Programme bereitgestellt, die CityGML-Daten in eine 3DCityDB (und umgekehrt) umwandeln können [1].

2.4.5 OpenStreetMap

OpenStreetMap (OSM) ist ein 2004 gegründetes Projekt, welches das Ziel verfolgt, ein offen zugängliches und durch jeden verbesserbares Abbild der Erde zu schaffen [17]. Bereits 2011 wurde dabei in bewohnten Gebieten Deutschlands eine Vollständigkeit von 79,8% relativ zum kommerziellen Kartenanbieter Navteq erreicht [24].

2.4.6 OpenLayers

Bis vor kurzem waren die webbasierten Kartenbetrachtungsprogramme für Datenbanken wie OpenStreetMap fast vollständig serverseitig implementiert: um einen anderen Teil der Karte zu sehen oder die Zoomstufe zu verändern, musste der gewünschte Ausschnitt

an den Webserver gesendet, dort gerendert und dann an den Client zurückgeschickt werden [26]. Dies führt zu einer nicht unerheblichen Eingabeverzögerung, was die Bedienbarkeit der damals verwendeten Anzeigeprogramme stark einschränkt. Mit der Veröffentlichung von Google Maps im Jahr 2005 und OpenLayers im Jahr 2006 hat sich dies jedoch geändert. Beide Projekte stellen JavaScript-Bibliotheken bereit, die einen Teil der Renderaufgaben in den Webbrowser verlagern, um ein flüssiges Scrollen und Zoomen durch Kartenbereiche zu ermöglichen. Der Server stellt dazu fertig gerenderte Kartenausschnitte für verschiedene Zoomstufen in Kacheln unterteilt bereit. Bewegt der Nutzer nun die Karte im Webbrowser, so werden die fehlenden Kacheln asynchron im Hintergrund nachgeladen und automatisch angezeigt, ohne dass dabei die Scroll- oder Zoominteraktion unterbrochen wird. Dafür muss der Server nur einmal alle Kacheln der Karte in den gewünschten Zoomstufen rendern; diese können dann als statische Bilddateien mittels eines einfachen Webserverns an die Clients verteilt werden [32]. Der Vorteil dabei ist, dass der Server nicht für jede Clientanfrage einen separaten Renderprozess starten muss, wodurch sich die serverseitige Rechenlast deutlich verringert.

Google Maps und OpenLayers basieren dabei zwar auf dem selben Prinzip, weisen jedoch im Detail erhebliche Unterschiede auf. Als kommerzielles Projekt werden bei Google Maps je nach Art der Nutzung Gebühren erhoben [14]. Zudem ist dabei der clientseitige Code vollständig proprietär, kann also vom Nutzer nur mithilfe der momentan von Google bereitgestellten Programmierschnittstelle (*Application Programming Interface*, API) erweitert werden [15]. Im Kontrast dazu wird OpenLayers quelloffen entwickelt. Die Lizenz erlaubt es jedem, den Code für eigene Zwecke zu verwenden und anzupassen ohne dafür Nutzungsentgelte zu zahlen [33]. Beide Projekte unterstützen dabei die Einbindung von KML-Dateien, womit Strukturen mittels ihrer Vektordaten auf der Karte dargestellt und um interaktive Elemente erweitert werden können [36].

2.5 Model-View-Controller

Das Model-View-Controller (MVC) Prinzip beschreibt ein Architekturmuster für Anwendungen, das diese in die drei Komponenten *Modell* (Model), *Präsentation* (View) und *Steuerung* (Controller) unterteilt. Hierbei beschreibt das Modell die Daten sowie die Logik, auf der das Programm operiert. So wären beispielsweise in einem Buchhaltungsprogramm die Liste aller Kunden sowie die Regel, dass nach einer bestimmten Zahlungsfrist automatisch eine Mahnung versendet wird, Teil des Modells. Die Steuerungs- und Präsentationskomponenten definieren das Benutzerinterface. Ruft der Benutzer die

ses auf, bereitet die Steuerungskomponente die Daten für die Anzeige vor. Das eigentliche Aussehen der angezeigten Oberfläche (also zum Beispiel die Position und Größe der Ein- und Ausgabeelemente) wird dann durch die Präsentationskomponente bestimmt. Nach abgeschlossener Interaktion nimmt die Steuerungskomponente die getätigten Eingaben entgegen und gibt sie an das Modell weiter, wofür sie die Eingaben (wenn nötig) überprüft und konvertiert [23].

Durch diese Aufspaltung der Anwendung entsteht auch die Möglichkeit, mehrere unterschiedliche Benutzerschnittstellen mit jeweils eigenen Steuerungs- und Präsentationskomponenten zu schaffen, ohne dass dafür das Modell angepasst werden muss. Weiterhin können so Personen, die ausschließlich mit der Präsentationskomponente beschäftigt sind (beispielsweise Designer), daran arbeiten ohne dass sie zwingend Kenntnis von der Geschäftslogik haben oder darauf achten müssen, diese nicht zu modifizieren.

2.6 Webentwicklung in Java

Die Standardbibliothek der Programmiersprache Java (*Java Platform, Standard Edition* / Java SE) enthält zwar viele Funktionen für die Erstellung verschiedener Arten von Bibliotheken und Anwendungen, jedoch keine, die speziell für die Entwicklung von Webanwendungen ausgelegt sind [39]. Deswegen werden im Folgenden die für diese Arbeit verwendeten zusätzlichen Bibliotheken kurz vorgestellt.

2.6.1 JAX-RS

Die *Java API for RESTful Web Services* (JAX-RS) ist ein 2008 veröffentlichter Standard, welcher es ermöglicht, Webdienste und -seiten in Java zu implementieren. Die API besteht zu einem wesentlichen Teil aus Annotationen, welche zu Klassen und Methoden hinzugefügt werden, um diese über das dem Web zugrunde liegende *Hypertext Transfer Protocol* (HTTP) zugänglich zu machen. Auf diese Art zugänglich gemachte Methoden werden auch als *Endpunkte* einer Webanwendung bezeichnet. Die Annotationen deklarieren für jeden Endpunkt auch, welche Ein- und Ausgaben er verarbeitet. Die verwendete JAX-RS Implementierung konvertiert dann automatisch die Parameter aus dem HTTP-Request in die deklarierten Java-Objekte. Nachdem der Request abgearbeitet wurde, wird dann wiederum das vom Endpunkt zurückgegebene Objekt in das durch die Annotation deklarierte Ausgabeformat umgewandelt [6].

2.6.2 Jersey

Die Referenzimplementierung von JAX-RS, Jersey [19], bietet über den Standard hinaus noch einige Zusatzfunktionen. Eine davon sind Helferklassen, die das Einbetten eines Webservers in eine mit JAX-RS geschriebene Applikation vereinfachen. Eine so erstellte Webanwendung lässt sich zu einem einzigen *Java Archive File* (JAR) kompilieren, welches dann sehr einfach zu installieren ist: sie muss nur auf einen Server kopiert und über eine Java-Laufzeitumgebung gestartet werden. Als weitere Funktion bietet Jersey Annotationen an, die eine kompakte Integration von *Template Engines* in eine Webanwendung ermöglichen. Dazu müssen Endpunkte lediglich mit dem Pfad des dazugehörigen *Templates* annotiert werden, Jersey übernimmt dann den Aufruf der Template Engine und leitet das Ergebnis an den Client weiter [20]. Im Folgenden wird eine solche Template Engine kurz vorgestellt und ihre Funktionsweise erläutert.

2.6.3 FreeMarker

Freemarker ist eine vollständig in Java implementierte Template Engine. Sie wird als quelloffene, freie Software seit 1999 aktiv entwickelt und steht seit 2015 unter der Betreuung der Apache Software Foundation [3]. Wie in Abbildung 2.2 dargestellt besteht ihre Hauptaufgabe darin, Daten sowie eine Vorlagendatei (das Template) entgegenzunehmen, um dann Platzhalter in der Vorlage durch diese Daten (oder Teile davon) zu ersetzen.

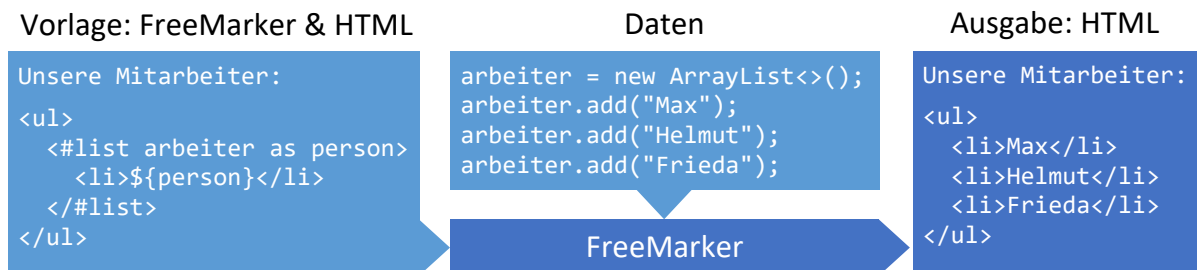


Abbildung 2.2: Beispielaufnahme von FreeMarker

Die Vorlage kann dabei in einem beliebigen textbasierten Format wie zum Beispiel XML, KML oder der *Hypertext Markup Language* (HTML) verfasst sein. Innerhalb der Vorlage dürfen durch FreeMarker spezifizierte Platzhalter und Befehle verwendet werden, durch die dann beim Ausführen von FreeMarker die eingegebenen Daten weiterverarbeitet oder ausgegeben werden. Dabei unterstützt die Template Engine auch komplexere

Operationen wie if-else-Abfragen, Variablen und Schleifen [2].

2.6.4 jOOQ

Bei *Java Object Oriented Querying* (jOOQ) handelt es sich um eine Java-Bibliothek für den Zugriff auf Datenbanken, welche die Structured Query Language (SQL) unterstützen. Seit dem ersten Release 2010 wird sie im Wesentlichen unter einer offenen Lizenz von der Data Geekery GmbH entwickelt [40]. Die Firma bietet gegen Bezahlung zusätzlichen Support sowie proprietäre Erweiterungen für kommerzielle Datenbanken an [21]. Basierend auf der in Java SE enthaltenen Java Database Connectivity (JDBC) bietet jOOQ eine zusätzliche Abstraktionsschicht, mit der in Java typsichere Datenbankabfragen formuliert werden können, welche dann automatisch in SQL-Code für die jeweilige Datenbank übersetzt werden. Ebenso wird das von der Datenbank zurückgegebene Ergebnis durch jOOQ wieder zurück in Java-Objekte umgewandelt [25].

Um dabei zu garantieren, dass die Typen innerhalb der Datenbankabfragen und der daraus resultierenden Java-Objekte zu den im SQL-Schema definierten Typen passen, verwendet jOOQ Codegenerierung (siehe Abbildung 2.3).

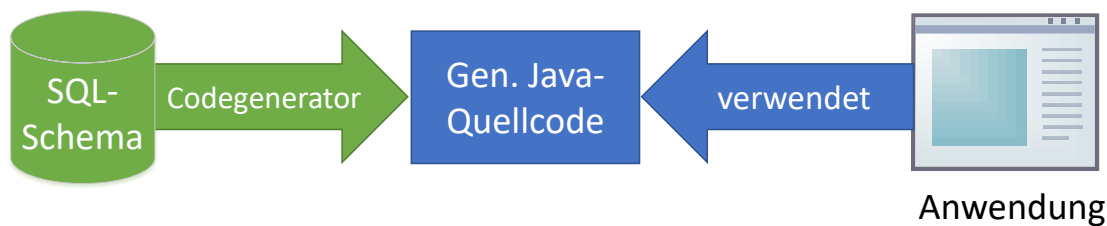


Abbildung 2.3: Funktionsweise von jOOQ

Hierzu verbindet sich der Generator mit der verwendeten Datenbank, liest das existierende Schema aus und generiert darauf basierend Java-Quellcode, welcher unter Anderem die existierenden Tabellen mit ihren Feldern, Datentypen und Relationen beschreibt. Dann reicht es aus, die jOOQ-Bibliothek und den generierten Code in ein Projekt einzubinden, das von der Datenbank anbindung Gebrauch macht. Falls sich das Datenbankschema ändert, kann der Codegenerator erneut ausgeführt werden. Hierdurch wird der generierte Java-Code aktualisiert. Wenn sich das Schema dabei so geändert hat, dass es nicht mehr zur Anwendung kompatibel ist, so passt auch der neue generierte Code nicht mehr zum Anwendungscode. Wird dann versucht die Anwendung mit dem neuen Code zu kompilieren, meldet bereits der Java-Compiler die Stellen, an denen die Datenbank zum Programm inkompatibel geworden ist [25]. Damit kann in vielen Fällen verhindert

2 Theoretische Grundlagen

werden, dass bei einer Modifikation des Datenbankschemas die verursachten Fehler erst bei laufender Anwendung auftreten.

3 Ausgangssituation

Nachdem die relevanten Grundlagen erklärt wurden, wird in diesem Kapitel die Ausgangssituation für den Entwicklungsprozess des Informationssystems beschrieben. Dazu werden zunächst die vor der Entwicklung bereits existierenden Systeme vorgestellt, auf denen die Anwendung aufbaut. Darauf folgt eine Definition der Anforderungen, welche das System nach Fertigstellung dieser Arbeit erfüllen soll.

3.1 Aktueller Stand

Als zentraler Datenspeicher für das Projekt Living Roadmap wird eine PostgreSQL Datenbank zusammen mit der PostGIS-Erweiterung verwendet. Darauf ist das Forschungszentrum Jülich nach dem 3DCityDB-Schema abgebildet, wobei die vorhandene Gebäudegeometrie aus 553 Bauwerken mit der Detailstufe LoD2 besteht. Weiterhin sind in der Datenbank bereits Versorgungsnetze sowie thermische Eigenschaften von Gebäuden mithilfe der UtilityNetworkADE und der EnergyADE abgebildet. Lediglich die Tabellen mit Sensor- und erweiterten Gebäudeinformationen verwenden ein projektspezifisches Schema.

Der von dieser Anwendung verwendete Teil des Datenbankschemas ist in Abbildung 3.1 skizziert. Davon entsprechen die Tabellen *building* und *surface_geometry* dem 3DCityDB-Standard, der verbleibende Teil des dargestellten Schemas ist proprietär. Die in der Datenbank vorhandenen Gebäude sind mittels der Tabellen *campus* und *building_campus* in verschiedene Campusse partitioniert. Weiterhin sieht die Tabelle *building_campus* vor, dass jedes Gebäude mittels der Spalte *campus_building_parent_id* auch Teil eines anderen Gebäudes sein kann - damit können dann komplexe Bauwerke modelliert werden, die aus mehreren Untergebäuden bestehen. Somit lässt sich die Menge aller Gebäude auch als Baumstruktur betrachten. Neben den Gebäuden beinhaltet der proprietäre Teil des Schemas auch Informationen über die vorhandenen Sensoren, welche in der Tabelle *sensor* beschrieben sind. Zu jedem Sensor wird darin festgehalten, welchen Messwerttyp er aufnimmt (zum Beispiel den thermischen Energieverbrauch) und in wel-

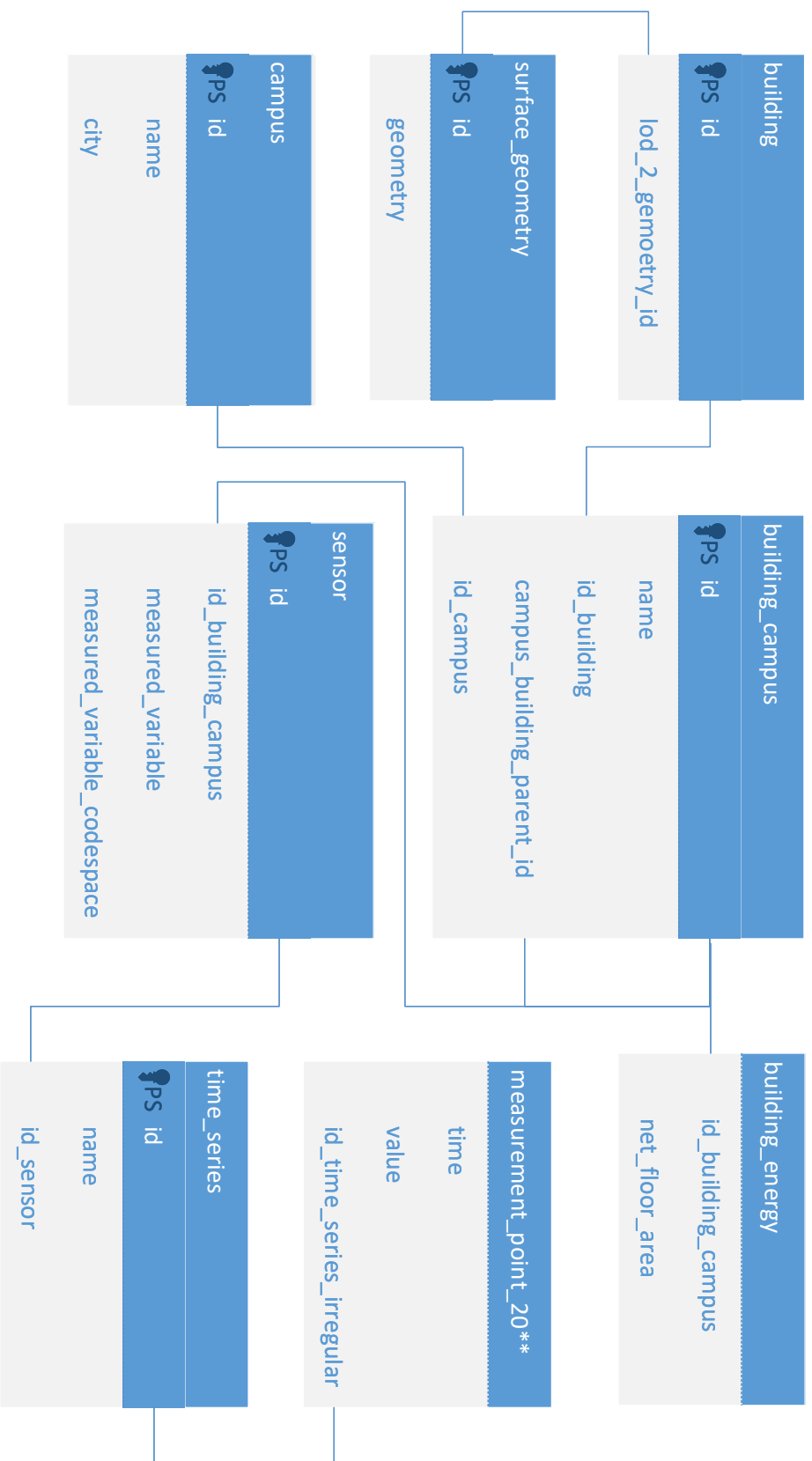


Abbildung 3.1: Von der Anwendung verwendeter Teil des Datenbankschemas. Die Primären Schlüssel der jeweiligen Tabelle sind mit PS markiert.

cher Einheit die Messwerte aufgenommen werden (für den thermischen Energieverbrauch sind dies Kilowattstunden, also kWh). Ein Sensor kann dann eine beliebige Anzahl an Messreihen generieren, über die Informationen (wie zum Beispiel der Name jeder Messreihe) in der Tabelle *time_series* festgehalten werden. Die eigentlichen Messdaten dazu befinden sich nach Jahren unterteilt in den Tabellen *measurement_point_2009*, *measurement_point_2010*, usw.

3.2 Anforderungen

Basierend auf den gegebenen Informationssystemen und dem in der Aufgabenstellung formulierten Ziel werden in diesem Abschnitt die Anforderungen begründet und herausgearbeitet, welche die Anwendung erfüllen soll.

3.2.1 Funktionen

Zunächst bringt das Ziel, eine Weboberfläche zur Visualisierung von Gebäuden und Sensorwerten zu schreiben einige funktionale Anforderungen mit sich.

Interaktive Gebäudevisualisierung

Die Gebäudevisualisierung dient dabei dem Zweck, dem Nutzer in Form einer Karte eine einfache Übersicht über alle Bauwerke zu geben, deren Energiedaten abgerufen werden können. Der Vorteil gegenüber einer einfachen Auflistung aller Gebäude in Text- oder Tabellenform besteht einerseits darin, dass er so Gebäude einfacher wiedererkennen kann, wenn er den geografischen Aufbau des Forschungszentrums bereits kennt. Andererseits können so die Gebäude auch nach verschiedenen energetischen Merkmalen eingefärbt werden. Konkret vorgesehen ist hier eine Einfärbung nach Baualtersklassen und spezifischem Energieverbrauch.

Ein weiterer Nutzen der geografischen Darstellung besteht darin, die Gebäudeumrisse rudimentär auf ihre Genauigkeit überprüfen zu können. So würde es bereits bei einer einfachen zweidimensionalen Karte auffallen, wenn sich Gebäude gegenseitig oder mit Umgebungsobjekten wie zum Beispiel Straßen überschneiden.

Darstellung von Sensorwerten

Eine übersichtliche Darstellung der Messwerte ist unerlässlich, um dem Benutzer einen Einblick in die energetischen Eigenschaften der Gebäude sowie später einen Vergleich mit

den Simulationsergebnissen zu ermöglichen. Dafür sollen die Daten in einem zweidimensionalen Diagramm aufgezeichnet werden, wobei sowohl eine Betrachtung des gesamten Messzeitraums als auch von kleineren, benutzerdefinierten Zeiträumen (zum Beispiel einzelnen Tagen oder Wochen) möglich sein soll.

3.2.2 Kompatibilität

Neben den Funktionen an sich gibt es auch einige Anforderungen, wie diese Funktionen umzusetzen sind; diese werden als *nichtfunktionale Anforderungen* bezeichnet. Eine davon ist Kompatibilität - die Webanwendung soll auf möglichst vielen Webclients lauffähig sein. Genauer wurde hierbei ausdrücklich gefordert, dass sie auf den aktuellen Desktopversionen der Browser Google Chrome und Mozilla Firefox lauffähig ist, da dies die im Rahmen des Projektumfelds gängigen Browser sind. Um eine langfristige Funktionsfähigkeit zu sichern, sollen hierbei ausschließlich im Webbrowser selbst eingebaute und von dessen Hersteller unterstützte Technologien verwendet werden, was eine Verwendung von Plugins ausschließt. Eine zusätzliche Unterstützung der mobilen Versionen von Chrome und Firefox wurde als hilfreich, jedoch nicht obligatorisch eingestuft.

3.2.3 Langfristige Wartbarkeit

Eine weitere wesentliche nichtfunktionale Anforderung besteht darin, die Anwendung so zu programmieren, dass sie auch noch innerhalb der nächsten Jahre gewartet und erweitert werden kann. Dies ist wichtig, da in der Aufgabenstellung explizit die Möglichkeit einer zukünftigen Einbindung von Simulationssystemen vorgesehen ist. Zudem können vor allem im proprietären Teil der Datenbank (beschrieben in Abschnitt 3.1) noch Änderungen des Schemas auftreten, welche eine Anpassung der Anwendung erfordern. Deswegen werden im Folgenden die Konsequenzen betrachtet, die aus der Forderung nach langfristiger Wartbarkeit entstehen und selber wiederum Anforderungen darstellen.

Anforderungen an verwendete Bibliotheken

Als Teil des Programmcodes müssen die verwendeten Bibliotheken ebenfalls einige Anforderungen erfüllen, damit das Programm wartbar bleibt. Dazu gehört in diesem Fall, dass ihr Quellcode frei verfügbar und modifizierbar sein muss. Dies ermöglicht es, im Fall eines Fehlers im Quellcode einer Bibliothek diesen zu lokalisieren und selber zu beheben. Positiv ist auch, dass bei der Verwendung freier Software keine Lizenzgebühren

anfallen, die sich auf Dauer akkumulieren könnten. Zusätzlich sollte für die verwendeten Bibliotheken absehbar sein, dass sie in Zukunft gewartet werden. Dies erhöht die Wahrscheinlichkeit, dass gefundene Fehler und insbesondere Sicherheitslücken schnell beseitigt und neue oder sich ändernde Standards implementiert werden. Weiterhin wird davon ausgegangen, dass mit der Anzahl der Mitarbeiter und der Anzahl an Jahren, über die das Projekt bisher fortgeführt wurde die Wahrscheinlichkeit steigt, dass es auch in naher Zukunft noch gewartet wird. Also liegt der Fokus bei der Auswahl der Bibliotheken auf solchen, an denen viele Personen mitarbeiten und die bereits seit einigen Jahren aktiv entwickelt werden.

Flexibilität der Datenanbindung

Aufgrund der Möglichkeit von zukünftigen Änderungen im Datenbankschema soll die Anwendung so aufgebaut sein, dass eine Anpassung an ein neues Schema möglichst reibungslos gelingen kann. Dazu soll der für den Zugriff auf die Datenbank verantwortliche Teil des Programms soweit abgespalten werden, dass er auch ohne Kenntnis des restlichen Codes gewartet werden kann.

4 Realisierung des Informationssystems

Aufbauend auf den Anforderungen beschreibt dieses Kapitel, wie das Informationssystem realisiert wurde. Dazu gehört zunächst eine Darstellung der Anwendungsarchitektur sowie eine Erläuterung, welche Bibliotheken verwendet werden und wie sie zu den gestellten Anforderungen passen. Weiterhin wird am Ende genauer darauf eingegangen, wie die beiden Module der Anwendung implementiert wurden.

4.1 Modularisierung

Um die Anforderung aus Abschnitt 3.2.3 zu erfüllen, dass die Datenbankbindung unabhängig vom Rest der Anwendung sein soll, wird das Programm wie in Abbildung 4.1 dargestellt in die zwei Teile *EnergyData-API* und *EnergyData-Web* aufgeteilt. *EnergyData-API* ist dabei eine Abstraktionsschicht für die PostGIS-Datenbank, auf der die Anwendung basiert. Sie enthält den gesamten datenbankspezifischen Code und stellt selber eine Java-API bereit, mit der andere Komponenten auf die Gebäude, Sensoren und Messwerte in der Datenbank zugreifen können. Wenn sich das Schema der Datenbank ändert, muss lediglich die Implementierung von *EnergyData-API* daran angepasst werden; die nach außen bereitgestellte API kann dabei unverändert bleiben. Somit können darauf aufbauende Komponenten wie *EnergyData-Web*, welche die Weboberfläche bereitstellt, unabhängig entwickelt werden.

4.2 Auswahl der Bibliotheken

Bevor der genauere Aufbau der beiden Module erläutert wird, folgt zunächst eine kurze Erläuterung, welche Bibliotheken wegen welcher Anforderungen verwendet werden. Innerhalb des Moduls *EnergyData-API* wäre es zwar möglich, die in der Standardbibliothek enthaltene JDBC-Schnittstelle zum Zugriff auf die Datenbank zu benutzen. Der Anforderung der guten Wartbarkeit des Moduls wird jedoch durch die hier verwendete Schnittstelle *jOOQ* deutlich besser entsprochen: wenn sich das Modul oder das Daten-

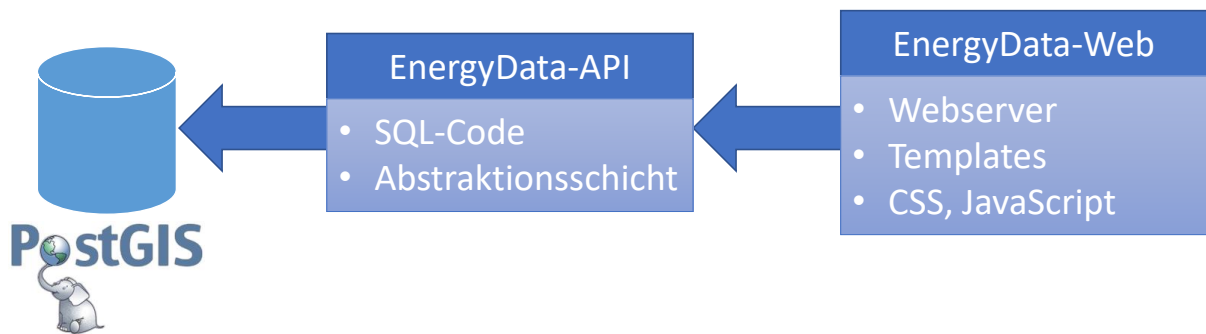


Abbildung 4.1: Komponenten der Anwendung *EnergyData*, die Pfeile stellen Abhängigkeiten dar. PostGIS-Logo aus [37]

bankschema ändert und diese dadurch zueinander inkompatibel werden, verursacht dies bei jOOQ meistens einen Compilerfehler. Hierdurch kann deutlich früher eingeschätzt werden, ob eine Datenbankänderung einen Fehler hervorruft, als wenn dieser Fehler erst zur Laufzeit sichtbar wird.

Das Modul EnergyData-Web benutzt deutlich mehr Bibliotheken, da es neben Java-Code auch noch HTML, JavaScript und Cascading Style Sheets (CSS) für die Darstellung der Weboberfläche beinhaltet. Deswegen beinhaltet der dazugehörige Abschnitt nur eine Auswahl der in dieser Komponente verwendeten Technologien; dazu gehören Jersey, FreeMarker sowie OpenLayers.

4.2.1 Jersey und FreeMarker

Jersey wurde vor allem deswegen ausgewählt, weil es als Implementierung des JAX-RS Standards und mittels seiner zusätzlichen Einbindung von FreeMarker erlaubt, mit wenigen Zeilen Code Endpunkte von Webanwendungen zu realisieren. Um dies zu illustrieren folgt hier ein kurzes Beispiel für einen Endpunkt, der mit 10 Zeilen Java- und 4 Zeilen FreeMarker-Code eine Liste von Campusgeländen ausgibt:

```

1 public class CampusEndpoint {
2     @GET
3     @Path("/campusList")
4     @Template(name = "/campus.ftl")
5     public Map<String, Object> get() {
6         Map<String, Object> model = new HashMap<>();
7         model.put("campusList", Service.listCampus());
8         return model;
9     }
10 }

```

Hierbei wird mittels `@GET` und `@Path("/{}/campusList/{})` festgelegt, dass der Endpunkt über die HTTP GET-Methode unter dem Pfad `/campusList` aufrufbar ist. Er greift dann auf das folgende Template zu, das unter dem Namen `campus.ftl` gespeichert ist:

```

1 Bekannte Campusgelände:
2 <#list campusList as campus>
3 - ${campus.name} in ${campus.city}
4 </#list>

```

Innerhalb des Templates wird auf die `campusList` zugegriffen, die eine Liste von Campusgeländen beinhaltet, welche in Zeile 7 vom Java-Code aus der Datenbank abgerufen werden. FreeMarker ersetzt dabei `\${campus.name}` und `\${campus.city}` automatisch durch das Resultat der Methodenaufrufe `campus.getName()` und `campus.getCity()`.

Wird dann ein lokaler Webserver mit diesem Endpunkt ausgeführt, so ist es möglich, diesen unter dem *Uniform Resource Locator* (URL) `http://localhost/campusList` aufzurufen, was dann zum Beispiel zu folgender Ausgabe führen kann:

```

Bekannte Campusgelände:
- Campus Mitte in Aachen

```

- Campus Melaten in Aachen
 - Forschungszentrum Jülich in Jülich
-

Diese in diesem Beispiel illustrierte kompakte Syntax für Webendpunkte ermöglicht es, die Menge an Quelltext im Programm möglichst gering zu halten. Das erhöht wiederum die Wartbarkeit, da der Einarbeitungs- und Wartungsaufwand in der Regel mit der Anzahl an Codezeilen zunimmt.

4.2.2 OpenLayers

Ebenfalls in EnergyData-Web verwendet wird die in Kapitel 2.4.6 vorgestellte Bibliothek OpenLayers, die mittels JavaScript Karten im Webbrowser darstellen kann. Einer der Hauptgründe für ihre Verwendung liegt darin, dass sie die von der PostGIS-Datenbank zurückgegebenen KML-Gebäudedaten auf einer OpenStreetMap-Karte darstellen kann, ohne dass dafür zusätzlicher Code für das Parsen oder Rendern dieser Daten geschrieben werden muss. Positiv ist auch, dass die Darstellung der Vektordaten über JavaScript frei konfigurierbar ist [36]: Anstatt mehrere KML-Dateien mit jeweils verschiedenen Gebäudeeinfärbungen zu generieren muss so nur eine einzige KML-Datei bereitgestellt werden. Das Farbschema jedes einzelnen Gebäudes kann dann clientseitig beliebig verändert werden, ohne dass dafür Daten vom Server nachgeladen werden müssen. Dies ermöglicht es, die in Abschnitt 3.2.1 formulierten Anforderungen nach einer Gebäudeeinfärbung nach Baualtersklasse und spezifischem Energieverbrauch effizient umzusetzen. Wie die anderen in dieser Anwendung verwendeten Bibliotheken wird auch das OpenLayers-Projekt seit Jahren von mehreren Personen weiterentwickelt und bietet seinen Quellcode unter einer freien Lizenz an [32][13][33], womit es auch die Anforderungen aus Abschnitt 3.2.3 erfüllt.

4.3 EnergyData-API

Basierend auf diesen Anforderungen und den eingesetzten Bibliotheken werden nun einige Details der Implementierung der beiden Anwendungskomponenten beschrieben. Wie in Abschnitt 4 beschrieben stellt die Komponente EnergyData-API eine Abstraktionsschicht dar, die die Logik für den Zugriff auf die Datenbank vom Rest der Anwendung entkoppelt.

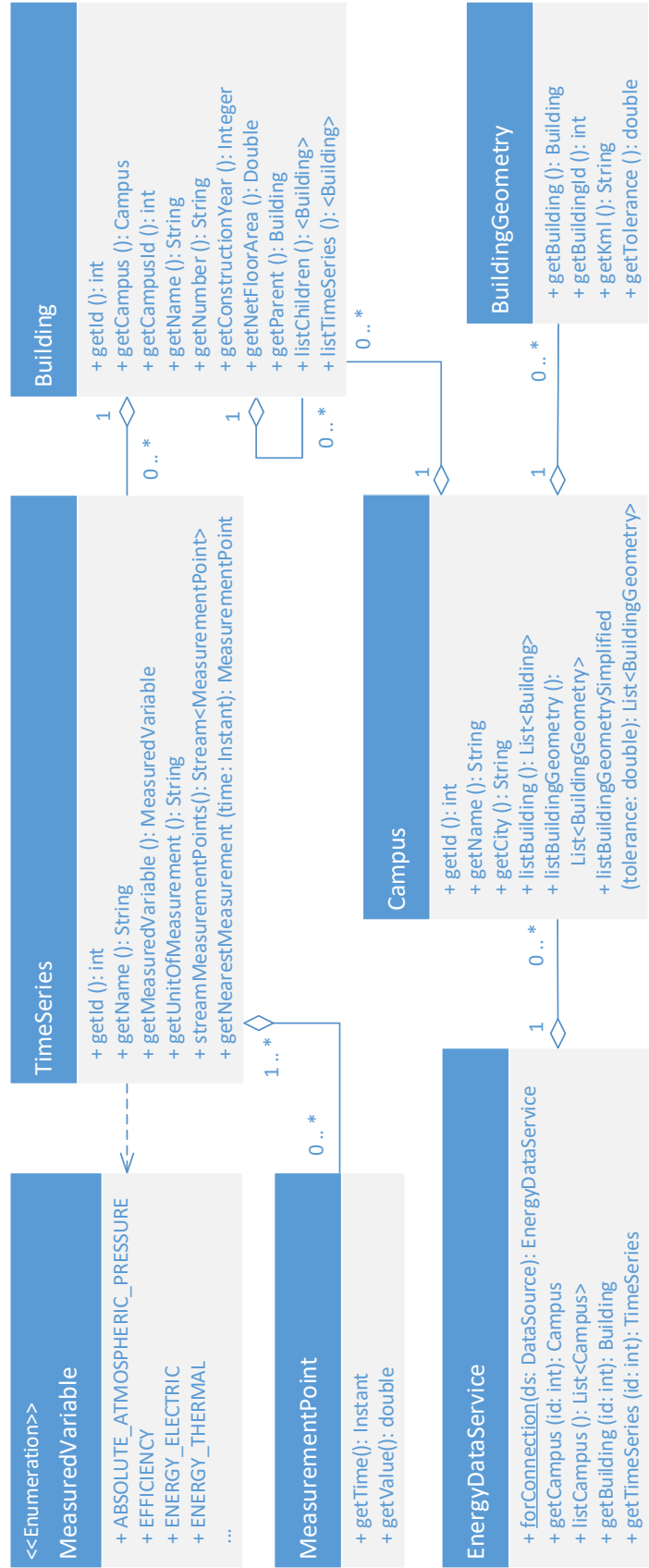


Abbildung 4.2: API der Datenbank-Abstraktionsschicht

4.3.1 API der Datenbank-Abstraktionsschicht

Diese API, welche in Abbildung 4.2 als Klassendiagramm der *Unified Markup Language* (UML) dargestellt ist und nun beschrieben wird, nimmt als Eingabe eine JDBC **Data Source** entgegen. Die **DataSource** beinhaltet die Informationen, die zum Zugriff auf die Datenbank notwendig sind, also unter anderem Benutzername und Kennwort sowie die Adresse des Datenbankservers. Sie wird intern an jOOQ weitergegeben und ermöglicht es der Bibliothek, eine Verbindung zur Datenbank herzustellen. Wird diese **DataSource** an die statische Methode **EnergyDataService.forConnection(DataSource)** übergeben, gibt diese Methode eine Instanz von **EnergyDataService** zurück, die den Zugriff auf die Objekte in der Datenbank ermöglicht. Dazu bietet sie die Methoden **getCampus**, **listCampus**, **getBuilding** und **getTimeSeries** an, mit der Instanzen der Klassen **Campus**, **Building** (Gebäude) oder **TimeSeries** (Zeitreihe von Sensormesswerten) aus der Datenbank abgerufen werden können. Zusätzlich besteht die Möglichkeit, über **listCampus** die Liste aller in der Datenbank gespeicherten Campusgelände abzurufen.

Ein durch den **EnergyDataService** konstruiertes Objekt der Klasse **Campus** repräsentiert dann genau eine Zeile in der Datenbanktabelle **campus** und gibt über **getName** und **getCity** den Namen des Campus sowie die Stadt aus, in der er sich befindet. Mittels **listBuilding** und **listBuildingGeometry** bietet sie zudem Listen aller Gebäude und Gebäudegeometrien an, die sich auf dem Campus befinden. Auch kann über **listBuildingGeometrySimplified** eine vereinfachte Gebäudegeometrie generiert werden, die maximal genauso viele Punkte wie die von **listBuildingGeometry** hat. Der in Metern angegebene Parameter **tolerance** definiert dabei, wie weit die Punkte der vereinfachten Geometrie von der ursprünglichen abweichen dürfen. Intern greift die API dazu auf die PostGIS-Funktion **ST_Simplify** zurück, welche den in Abschnitt 2.3 beschriebenen Douglas-Peucker Algorithmus implementiert.

In einer Instanz der Klasse **BuildingGeometry** sind nur der Umriss eines Bauwerks sowie gegebenenfalls seine Gebäude-ID (**Building.getId()**) gespeichert. Letztere ist nicht immer vorhanden, da es in der Datenbank auch Geometrieinformationen über einfachere Bauwerke geben kann, die nicht in den Tabellen **building**, **building_campus** oder **building_energy** abgebildet wurden. Der Umriss kann über **getKml** als KML-Polygon abgerufen werden. Wurde er zuvor vereinfacht, so kann die dabei verwendete Toleranz mittels der Methode **getTolerance** abgerufen werden. Sie gibt 0.0 zurück, wenn der Umriss nicht vereinfacht wurde.

Die für Gebäudemetadaten zuständige Klasse **Building** abstrahiert in ihrer Implementierung die drei Tabellen **building**, **building_campus** und **building_energy**. Aus diesen Tabellen bietet sie Informationen über Gebäudenamen und -nummer (**getName**, **getNumber**), das Baujahr (**getConstructionYear**) sowie mittels **getNetFloorArea** die Netto-Raumfläche in Quadratmetern an. Der in Abschnitt 3.1 beschriebene Mechanismus, dass ein Gebäude auch innerhalb eines anderen Gebäudes liegen kann, wird hier ebenfalls zugänglich gemacht: **getParent** gibt das übergeordnete Gebäude zurück. Wenn das Gebäude selbst Untergebäude besitzt, können diese mittels **listChildren** abgerufen werden. Zudem können über **listTimeSeries** alle Zeitreihen aufgelistet werden, die von Sensoren in diesem Gebäude (nicht in Untergebäuden) gemessen werden.

Informationen über die aufgelisteten Zeitreihen sind in Instanzen der Klasse **TimeSeries** enthalten, welche ihre Daten aus den Datenbanktabellen **sensor** und **time_series** sowie **measurement_point_2010**, **measurement_point_2011**, **measurement_point_2012**, ... bezieht. Jede Zeitreihe hat dabei einen mit **getName** abrufbaren Namen. Die Methoden **getMeasuredVariable** und **getUnitOfMeasurement** beschreiben, welcher Wert in welcher Einheit innerhalb der Zeitreihe gespeichert wird. Ein Beispiel hierfür wäre der Volumenstrom (**VOLUME_FLOW**) einer Wasserleitung in Kubikmetern pro Stunde (**m3/h**). Zum Abruf der Messwerte kann **streamMeasurementPoints** aufgerufen werden, es gibt dann einen **Stream** von endlich vielen Instanzen von **MeasurementPoint** zurück, was eine Iteration über alle Messwerte ermöglicht. Im Gegensatz zu einer Liste muss bei einem Stream nur das momentan behandelte Element in den Arbeitsspeicher geladen sein, was besonders bei einer hohen Anzahl an Messwerten die serverseitigen Hardwareanforderungen deutlich reduziert. Zusätzlich liefert die Methode **getNearestMeasurementPoint** einen Messpunkt, der der im Parameter **time** spezifizierten Zeit am nächsten ist. Dies ist zum Beispiel nützlich, um den jährlichen Stromverbrauch eines Gebäudes zu berechnen, wofür man den Zählerstand am Anfang und am Ende des Jahres benötigt. Anstatt über alle Zählerstände seit Messbeginn zu iterieren und die gewünschten Zeitpunkte heraus zu filtern reicht es aus, **getNearestMeasurementPoint** jeweils mit den Jahresanfangs- und Jahresendzeitpunkten aufzurufen.

Innerhalb einer Messreihe wird jeder gemessene Punkt durch einen **MeasurementPoint** repräsentiert. **getTime** und **getValue** geben den Zeitpunkt der Messung sowie den gemessenen Wert aus. In dieser Klasse wurde bewusst darauf verzichtet, zusätzliche Infor-

mationen (wie zum Beispiel eine Referenz auf die Zeitserie, zu der die Messung gehört) zu integrieren. Dies hält den Speicherverbrauch jedes Messpunkts gering, was sich bei der Verarbeitung großer Mengen an Messdaten positiv auf die Performance auswirken kann.

4.3.2 Aggregationen und Kardinalitäten

Das UML-Klassendiagramm in Abbildung 4.2 beinhaltet neben den Klassen auch *Aggregationen*, welche als Linie von einer Klasse zu einer anderen (oder der selben) dargestellt werden, wobei ein Ende der Linie mit einem diamantförmigen Viereck gekennzeichnet ist. Eine Aggregation von Klasse *A* nach Klasse *B* bedeutet in UML, dass Objekte von *A* Objekte von *B* *aggregieren* (also enthalten), wodurch Instanzen von *B* Teil einer Instanz von *A* werden. Hierbei wird keine Abhängigkeit impliziert, Objekte von *A* können also auch ohne Unterinstanzen von *B* existieren. Zusätzlich ist es möglich, eine Aggregation um *Kardinalitäten* zu erweitern, die ihre Multiplizität (also wie viele Objekte an der Aggregation teilnehmen können) definieren. Dazu wird bei der Linie, die die Aggregation darstellt, an das diamantförmige Ende annotiert wie viele Objekte der Klasse *A* gemeinsam Objekte der Klasse *B* enthalten. Am anderen Ende der Linie steht dann, wie viele Objekte der Klasse *B* durch diese Aggregation in *A* enthalten sind. Wenn zum Beispiel in UML modelliert werden soll, dass ein Auto 0 bis 4 Personen beinhalten kann, so kann dies als eine Linie zwischen Auto und Person dargestellt werden, bei der die Diamantform auf der Seite des Autos steht. Als Kardinalität wird dann auf der Seite des Autos eine 1 und auf Seite der Person 0 .. 4 angegeben. Ansonsten kann ein Sternsymbol (*) in einer Kardinalität dazu benutzt werden, um anzudeuten, dass eine Aggregation unbeschränkt viele Objekte enthalten kann [12].

Nach dieser Definition wird nun auf die Aggregationen in Abbildung 4.2 eingegangen. Ein **EnergyDataService** kann beliebig viele Objekte vom Typ **Campus** enthalten, da er die gesamte Datenbank mit mehreren Campusgeländen abstrahiert. Jeder Campus wiederum besteht aus Gebäuden (**Building**) sowie Geometriedaten, die Bauwerke auf dem Campus beschreiben (**BuildingGeometry**). Hierbei aggregiert ein **Building** bewusst keine **BuildingGeometry**, da es nicht nur Gebäude ohne Geometrie in der Datenbank geben kann, sondern auch Bauwerksgeometrien, die zu keinem Gebäude gehören (beispielsweise Brücken oder Sportplätze). Weiterhin kann ein Gebäude beliebig viele Sensoren beinhalten, die Zeitreihen messen, es aggregiert also unbeschränkt viele **TimeSeries**. Zusätzlich enthält ein Gebäude auch unbeschränkt viele Untergebäude, was hier als Aggregation der Klasse **Building** zu sich selbst modelliert wurde. Eine Besonderheit stellt die Ag-

gregation zwischen Zeitreihen (**TimeSeries**) und Messwerten (**MeasurementPoint**) dar. Intuitiv könnte man davon ausgehen, dass eine Zeitreihe beliebig viele Datenpunkte enthält. Hier wird jedoch auch die Möglichkeit in Betracht gezogen, dass ein Datenpunkt Teil mehrerer Zeitreihen ist. Das tritt genau dann auf, wenn in zwei Zeitreihen zum gleichen Zeitstempel der gleiche Wert gemessen wird. Es ist nur deshalb möglich, weil ein Datenpunkt sich nicht auf eine Zeitreihe bezieht, in der er gemessen wurde - der Grund hierfür wird in Unterabschnitt 4.3.1 erläutert.

4.3.3 Weitere Datenbanktabellen

In der vorhandenen Datenbank sind noch weitaus mehr Informationen abrufbar, von deren Visualisierung das Projekt Living Roadmap profitieren würde. Beispielsweise ist bereits jetzt geplant, die Simulationsdaten später in der Weboberfläche anzuzeigen (siehe Abschnitt 1). Es lässt sich also die Frage stellen, warum nicht bereits jetzt schon Funktionen in die EnergyData-API zum Zugriff auf diese Daten implementiert werden. Das wurde aus drei Gründen entschieden: einerseits reicht die momentan implementierte Schnittstelle aus, damit EnergyData-Web darauf aufbauend die in der Aufgabenstellung geforderten Funktionen anbieten kann. Weiterhin enthält die Datenbank neben den bereits verwendeten Tabellen noch über 30 weitere nicht-leere Tabellen; diese ebenfalls zu abstrahieren hätte den Zeitrahmen dieser Arbeit überschritten. Vor allem aber besteht die Möglichkeit, dass sich beispielsweise das Schema der Simulationsdaten noch ändert, bevor die dazugehörige Visualisierung in EnergyData-Web implementiert wird. In diesem Fall würde es zusätzlichen Wartungsaufwand verursachen, eine jetzt implementierte Schnittstelle bis zu ihrer Verwendung aktuell zu halten.

4.3.4 Paralleler Abruf von Zeitreihen

Nachdem der Aufbau der gesamten API beschrieben wurde, erläutert dieser Abschnitt die Implementation der Methode **TimeSeries.streamMeasurementPoint**. Weil die Messwerte in der Datenbank nicht in einer zentralen Tabelle gespeichert werden, sondern in mehrere Tabellen nach Jahr unterteilt sind (siehe Abschnitt 3.1), ergeben sich zwei mögliche Vorgehensweisen für den Abruf sämtlicher Messwerte einer Zeitreihe.

Eine Möglichkeit besteht darin, eine SQL-Abfrage zu schreiben, die mittels **UNION ALL** die Inhalte aller Messwertetabellen zusammenfügt und ausgibt. Eine solche Abfrage könnte zum Beispiel wie folgt aussehen, um die Messwerte der Zeitreihe mit der ID 2420 abzurufen:

```
1      SELECT time, value FROM measurement_point_2013
2          WHERE id_time_series_irregular = 2420
3  UNION ALL
4      SELECT time, value FROM measurement_point_2014
5          WHERE id_time_series_irregular = 2420
6  UNION ALL
7      SELECT time, value FROM measurement_point_2015
8          WHERE id_time_series_irregular = 2420
9  UNION ALL
10 ...
```

Die zweite Möglichkeit besteht darin, jede Messwertetabelle mit einer separaten SQL SELECT Abfrage auszulesen und die Ergebnisse in der Anwendung zusammenzufügen. In obigem Beispiel würden dann die Zeilen 1 - 2, 4 - 5 und 7 - 8 jeweils als eigenständige Abfrage an die Datenbank gesendet.

Den Abruf von Messwerten in mehrere SQL-Abfragen zu unterteilen eröffnet die Möglichkeit, die Ausführung dieser Abfragen zu parallelisieren. Dies wurde innerhalb der EnergyData-API in der Klasse `ParallelQueryExecutor` implementiert, welche von `TimeSeries.streamMeasurementPoint` verwendet wird. Die statische Methode `ParallelQueryExecutor.beginExecution` nimmt eine Datenbankanzbindung (in Form eines `jOOQ DSLContext`), eine Liste an SQL-Abfragen sowie eine Zahl $n \in \mathbb{N}$ entgegen, die die Anzahl an Threads spezifiziert. Diese Klasse führt dann im Hintergrund die gegebenen SQL-Abfragen parallel mit n Threads aus. Die Liste der Abfragen wird hierbei wie eine Warteschlange ausgeführt: wenn ein Thread momentan mit keiner SQL-Abfrage beschäftigt ist, entfernt er die erste Abfrage aus der Liste und führt diese aus. Über `getResultStream` wird dann ein `Stream` zurückgegeben, mit dem sequenziell über die Ergebnisse der Abfragen iteriert werden kann. Die Sortierung bleibt dabei unverändert, `getResultStream` beinhaltet also die gleichen Ergebnisse in der gleichen Reihenfolge, wie wenn die Abfragen aus der Liste sequenziell ausgeführt würden.

Um die Funktionsweise von `ParallelQueryExecutor` zu verdeutlichen, wird nun anhand eines Beispiels beschrieben, wie dieser die übergebenen SQL-Anfragen parallelisiert. In diesem Beispiel, welches in Abbildung 4.3 visualisiert ist, führt der Executor SQL-Abfragen für die Messwerte der Jahre 2013 bis 2017 aus. Die blau eingefärbten Blö-

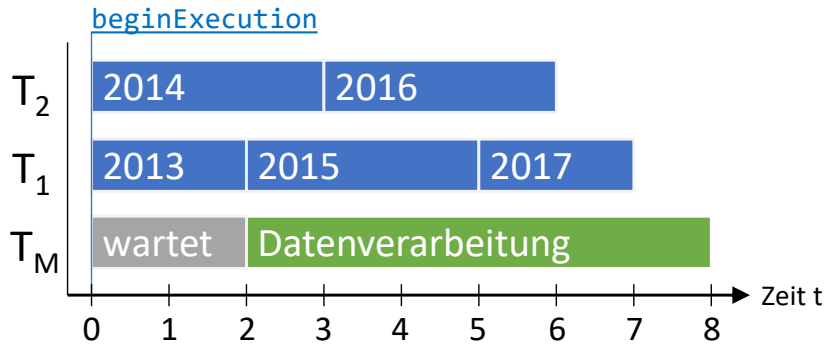


Abbildung 4.3: Beispielhafter Zeitverlauf bei der Ausführung von ParallelQuery Executor mit $n = 2$ Threads

cke in der Abbildung signalisieren, dass der jeweilige Thread innerhalb des markierten Zeitraums die Messwerte des im Block spezifizierten Jahres aus der Datenbank ausliest. Die Aktivitäten des Hauptthreads T_M werden durch die grauen und grünen Blöcke dargestellt.

Zum Anfangszeitpunkt $t = 0$ wird vom Hauptthread T_M die Funktion `beginExecution` mit $n = 2$ Threads und einer Liste von SQL-Abfragen ausgeführt, bei der jede Abfrage genau eine Tabelle mit Messwerten eines Jahres ausliest. Die Elemente dieser Liste sind dabei aufsteigend nach Jahr sortiert. Der Executor erstellt dann zwei neue Threads T_1 und T_2 , welche die SQL-Anfragen ausführen. Aufgrund der Sortierung führt T_1 bei $t = 0$ die Abfrage für das Jahr 2013 aus, welche in diesem Beispiel zwei Zeiteinheiten benötigt. Die simultan von T_2 gestartete Abfrage für das Jahr 2014 terminiert erst nach drei Zeiteinheiten. Also wartet der Hauptthread bis $t = 2$, da erst zu diesem Zeitpunkt die Daten der ersten Abfrage (also für das Jahr 2013) verfügbar sind. Bereits ab $t = 2$ kann der Hauptthread diese Messdaten weiterverarbeiten. Während dessen laden die Hintergrundthreads T_1 und T_2 die Messdaten der Jahre 2014 bis 2017 aus der Datenbank, was in diesem Beispiel sechs Zeiteinheiten, also bis $t = 8$ dauert. Zu diesem Zeitpunkt $t = 8$ terminieren T_1 und T_2 , da alle SQL-Abfragen abgearbeitet sind. In diesem Beispiel wird davon ausgegangen, dass die Datenverarbeitung wesentlich länger dauert als das Abrufen der Daten. Wenn dies nicht der Fall ist kann es auch vorkommen, dass der Hauptthread nach dem Verarbeiten der ersten Ergebnisse nochmals pausiert, um auf das Resultat der restlichen Abfragen zu warten.

4.4 EnergyData-Web

Die zweite Komponente EnergyData-Web visualisiert die von der EnergyData-API bereitgestellten Daten in Form eines Webinterfaces. Dazu werden in diesem Abschnitt die beiden Hauptfunktionen des Webinterfaces, also die geografische Gebäudeübersicht sowie die Messwertgraphen, vorgestellt.

4.4.1 Implementierung der Gebäudeübersicht

Die Gebäudeübersicht ist in EnergyData-Web mithilfe der in Abschnitt 2.4.6 vorgestellten OpenLayers-Bibliothek realisiert. Sie wird hier verwendet, um eine interaktive Karte bereitzustellen, die im Vordergrund die Gebäudedaten des FZJ und im Hintergrund eine Straßenkarte zeigt. Letztere wird in dieser Arbeit vom OpenStreetMap-Projekt bereitgestellt (siehe Abschnitt 2.4.5). Um die Funktionsweise der Karte zu veranschaulichen, zeigt Abbildung 4.4 den Datenfluss beim Abruf von Kartendaten aus beiden Quellen. Die dargestellte Einbindung der OpenStreetMap-Straßenkarte wurde durch Aktivieren der entsprechenden Option in OpenLayers durchgeführt. Die Bibliothek verbindet sich dadurch automatisch mit den OpenStreetMap-Servern, lädt die benötigten Kartenabschnitte herunter und zeigt sie dem Benutzer an.

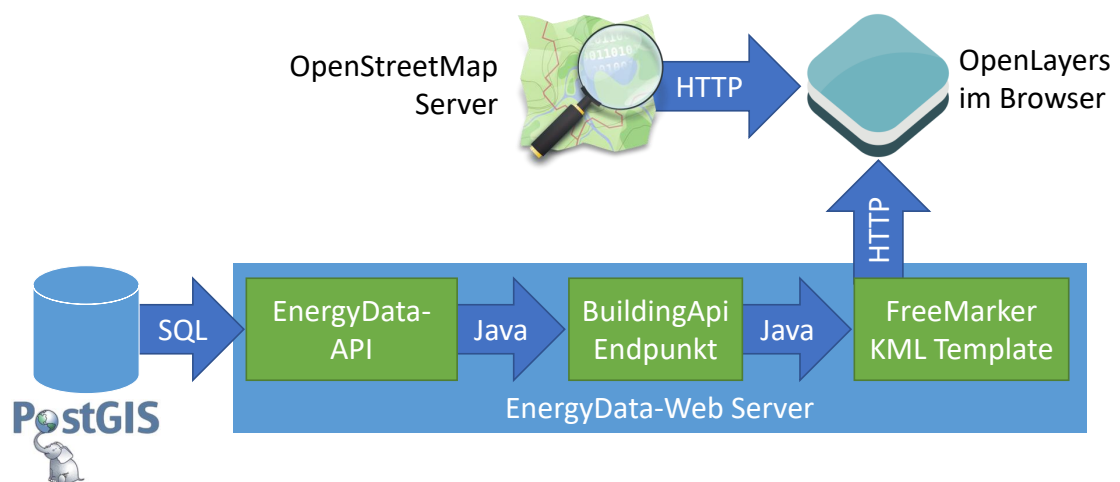


Abbildung 4.4: Datenfluss beim Abruf der Karte, Pfeile in Flussrichtung. Logos aus [34], [35], [37]

Etwas komplizierter ist der Datenfluss für die Darstellung der Gebäudegeometrien aus der PostGIS-Datenbank, welcher im unteren Teil von Abbildung 4.4 dargestellt ist. Zum Zugriff auf die Datenbank bindet EnergyData-Web die EnergyData-API als Bibliothek

ein. EnergyData-API fragt dann mittels SQL die Gebäudedaten und -geografie ab, wobei die PostGIS-Datenbank bereits die Gebäudeumrisse in KML-Polygone konvertiert. Falls gewünscht, kann PostGIS die Umriss dabei zusätzlich mit dem Douglas-Peucker-Algorithmus vereinfachen (siehe Abschnitte 2.3 und 2.4.3). Sobald die Abfrage abgeschlossen ist, gibt EnergyData-API diese Daten als Java-Objekte an den Endpunkt (`BuildingApi`) weiter. Dieser sammelt daraus die für die Karte relevanten Informationen, wozu zu jedem Gebäude neben dem Umriss auch das Baujahr, der spezifische Energieverbrauch sowie die installierten Sensortypen gehören. Anschließend fasst er die Daten für jedes Gebäude in eine separate *Key-Value-Map* zusammen und gibt diese an das FreeMarker Template `building_list.kml.ftl` weiter, welches die übergebenen Daten in ein KML-Dokument integriert. Zur Veranschaulichung folgt hier ein Auszug aus diesem Template:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <#list model as building>
      <Placemark>
        <name>${building.name}</name>
        <ExtendedData>
          <Data name="buildingId">
            <value>${building.id?c}</value>
          </Data>
          <#if building.constructionYear??>
            <Data name="constructionYear">
              <value>${building.constructionYear?c}</value>
            </Data>
          </#if>
        </ExtendedData>
        ${building.geometryKml}
      </Placemark>
    </#list>
  </Document>
</kml>
```

Wird dieses Template ausgeführt, so kann dadurch zum Beispiel das folgende KML-Dokument erzeugt werden (in diesem Beispiel ist nur ein Gebäude in der Datenbank eingetragen):

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.2">
  <Document>
    <Placemark>
      <name>0415_U</name>
      <ExtendedData>
        <Data name="buildingId">
          <value>11618</value>
        </Data>
        <Data name="constructionYear">
          <value>1985</value>
        </Data>
      </ExtendedData>
      <Polygon> ... </Polygon>
    </Placemark>
  </Document>
</kml>
```

Die so erzeugte KML-Datei wird dann über HTTP durch OpenLayers abgerufen, welches die darin enthaltenen Gebäude als Overlay über der Straßenkarte darstellt. Neben der Karte sind auf der Webseite Schaltflächen vorhanden, die es dem Nutzer ermöglichen, die vorhandenen Gebäude zum Beispiel nach Baualtersklasse einzufärben. Wird eine dieser Schaltflächen aktiviert, so ruft der in die Webseite integrierte JavaScript-Code die OpenLayers-API auf, um die Einfärbung (in diesem Beispiel nach dem in der KML-Datei vorhandenen Baujahr) für jedes Gebäude entsprechend anzupassen.

4.4.2 Darstellung der Messwertgraphen

Ähnlich wie bei der Kartenansicht sind die Messwertgraphen ebenfalls mittels JavaScript im Browser implementiert. Um die Messwerte abzurufen, lädt eine JavaScript-Bibliothek

diese asynchron im Hintergrund als CSV-Datei vom Server. Diese CSV-Datei wird vom Endpunkt **TimeseriesApi** bereitgestellt. Im Unterschied zum **BuildingApi**-Endpunkt ist die Logik für die Umwandlung der Daten ins Zielformat (hier: CSV) nicht in einem FreeMarker-Template ausgelagert, sondern als Java-Code in den Endpunkt integriert. Damit soll der zusätzliche Rechenaufwand für die Ausführung eines Templates vermieden werden. Es wird davon ausgegangen, dass sich hierdurch die Verarbeitungsdauer signifikant reduzieren lässt, da Zeitreihen in dieser Datenbank bereits jetzt über 200.000 Einträge enthalten.

5 Auswertung im Anwendungsfall

Campus Jülich

Nachdem die Anforderungen an diese Anwendung und verschiedene Aspekte ihrer Umsetzung betrachtet wurden, widmet sich dieses Kapitel der Auswertung, wie gut die Umsetzung die gestellten Anforderungen erfüllt. Da die funktionalen Anforderungen vollständig umgesetzt sind, konzentriert sich die Auswertung auf messbare Aspekte dieser Umsetzung. Am Ende des Kapitels steht eine Beschreibung einiger Messfehler in den Sensordaten des FZJ, die sich auch auf die Benutzbarkeit auswirken.

5.1 Vereinfachung von Gebäudegeometrie

Einer der messbaren Aspekte der Anwendung ist die Effizienz der Kartendarstellung hinsichtlich der dazu benötigten clientseitigen Rechenleistung. Je geringer die Rechenleistung ausfällt, die das Endgerät eines Nutzers für die Kartendarstellung benötigt, desto höher ist die Wahrscheinlichkeit, dass die Karte auch auf mobilen Browsern benutzbar ist. Dies liegt daran, dass bei modernen Browsern die zur Verfügung stehende JavaScript-Rechenleistung auf Mobilgeräten eine Größenordnung kleiner ist als auf Desktoprechnern [42]. Weiterhin ist es möglich, dass eine Webanwendung mit sehr hohem Bedarf an Rechenleistung auch auf Desktoprechnern nicht verzögerungsfrei dargestellt wird. Um also die Anforderungen aus Abschnitt 3.2.2 bestmöglich zu erfüllen ist es wünschenswert, die für die Kartenansicht benötigte Rechenleistung möglichst niedrig zu halten. Dazu kann beigetragen werden, in dem die Komplexität der auf der Karte dargestellten Gebäude reduziert wird. Wie in Abschnitt 4.3.1 ausgeführt, wird hierzu der Douglas-Peucker Algorithmus eingesetzt. Dieser nimmt als zusätzlichen Parameter die Toleranz ε entgegen, womit sich der Grad der Vereinfachung der Gebäudegeometrie einstellen lässt. Deswegen betrachten die folgenden Abschnitte, wie sich unterschiedliche Werte für ε bei der Vereinfachung der Gebäudedaten des Campus Jülich auswirken.

Tabelle 5.1: Anzahl der von `ST_Simplify` zurückgegebenen Gebäude, in Abhängigkeit von der Toleranz.

Toleranz	0 m	1 m	1,25 m	1,5 m	1,75 m	2 m	2.5 m	3.0 m	3.5 m	4 m
Gebäudeanzahl	334	334	333	333	331	327	304	293	286	280

5.1.1 Entfernen von Gebäuden unterhalb der Toleranz

Bei der Auswertung der Auswirkungen verschiedener Toleranzen fällt auf, dass mit steigender Toleranz die Anzahl an zurückgegebenen Gebäudepolygonen abnimmt (siehe Tabelle 5.1). Dies liegt daran, dass die anfangs verwendete PostGIS-Funktion `ST_Simplify` (welche den Douglas-Peucker-Algorithmus implementiert) einen Gebäudeumriss vollständig entfernt, wenn keine zwei Punkte des Umrisses einen Abstand zueinander haben, der die Toleranz erreicht oder überschreitet. PostGIS kennt jedoch zusätzlich auch die Funktion `ST_SimplifyPreserveTopology`, welche Geometrien nicht entfernt und auch nicht weiter vereinfacht, wenn sie bereits auf ein Viereck reduziert wurden [30]. Da mit dem Entfernen von Gebäuden die Anforderung nach einer vollständigen Kartendarstellung nicht erfüllt werden würde, verwendet die Anwendung `ST_SimplifyPreserveTopology`.

5.1.2 Auswirkungen auf die Kartendarstellung

Dieser Abschnitt beschreibt, wie sich verschiedene Toleranzwerte bei der Vereinfachung der Gebäudegeometrien auf ihre Darstellung auswirken. Dazu wurde das Gebäude mit der Nummer `0140_X` herangezogen, weil es unter den Gebäuden in der Datenbank eine vergleichsweise komplexe Form besitzt, bei der Vereinfachungen eher auffallen. In Abbildung 5.1 ist die Geometrie des Gebäudes nach Vereinfachung mit verschiedenen Toleranzen dargestellt. Hier ist positiv zu beobachten, dass die Vereinfachungen bis zu einer Toleranz von $\varepsilon = 1\text{ m}$ nur bei sehr genauer Betrachtung auffallen. Bei $\varepsilon = 1,5\text{ m}$ und $\varepsilon = 2\text{ m}$ fällt zwar auf, dass die Gebäude vereinfacht wurden (so wird beispielsweise das runde Bauwerk auf ein Oktagon reduziert), die grobe Form der Gebäude wird aber beibehalten. Ab $\varepsilon = 4\text{ m}$ ist dies nicht mehr gegeben, bei dieser Toleranz wird zum Beispiel das L-förmige Gebäude am oberen Rand des gezeigten Kartenabschnitts zu einem Viereck vereinfacht.

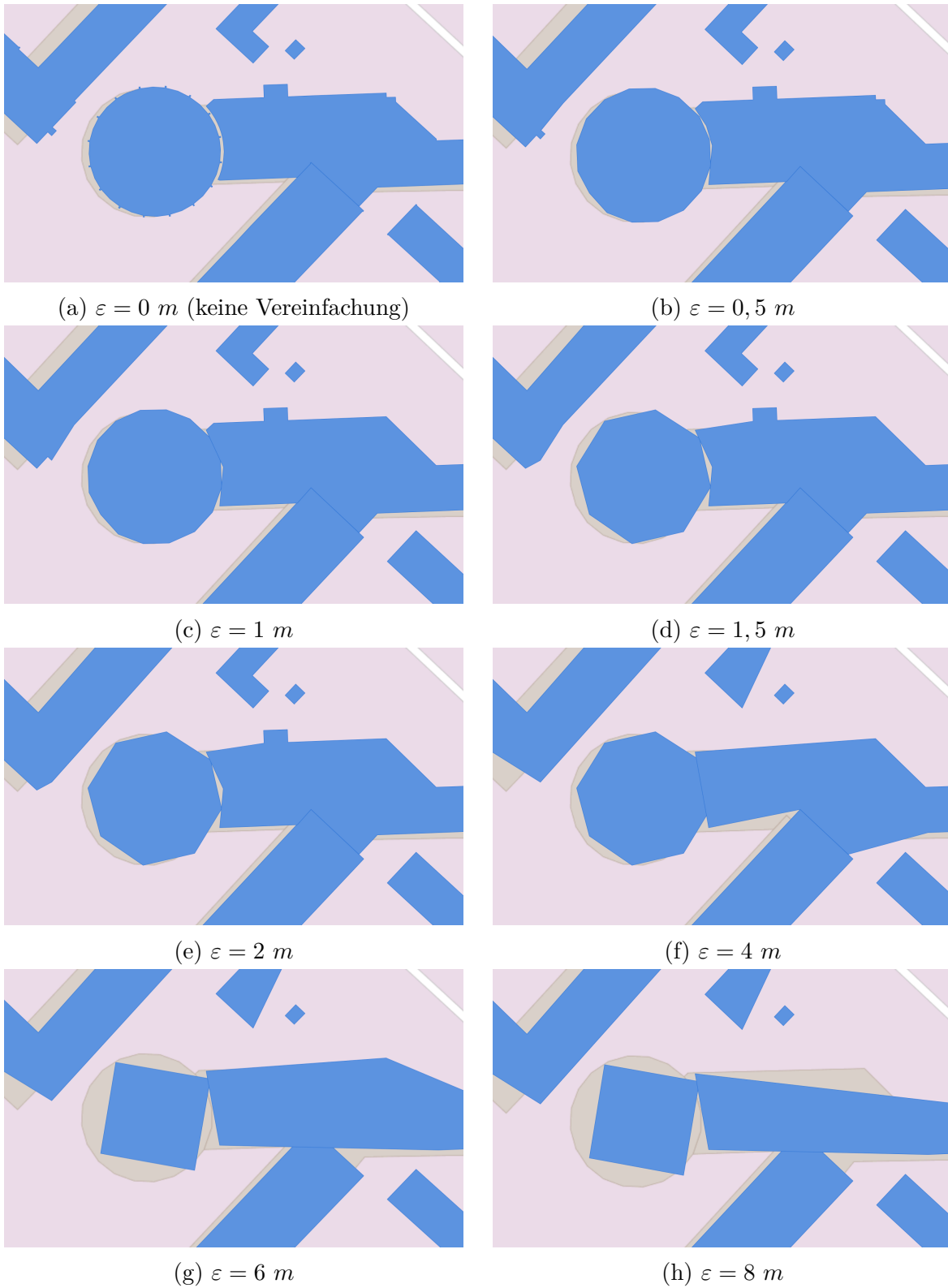


Abbildung 5.1: Gebäudegeometrie nach Vereinfachung mit verschiedenen Toleranzen ε in Metern

5.1.3 Punkteanzahl in Abhängigkeit von der Toleranz

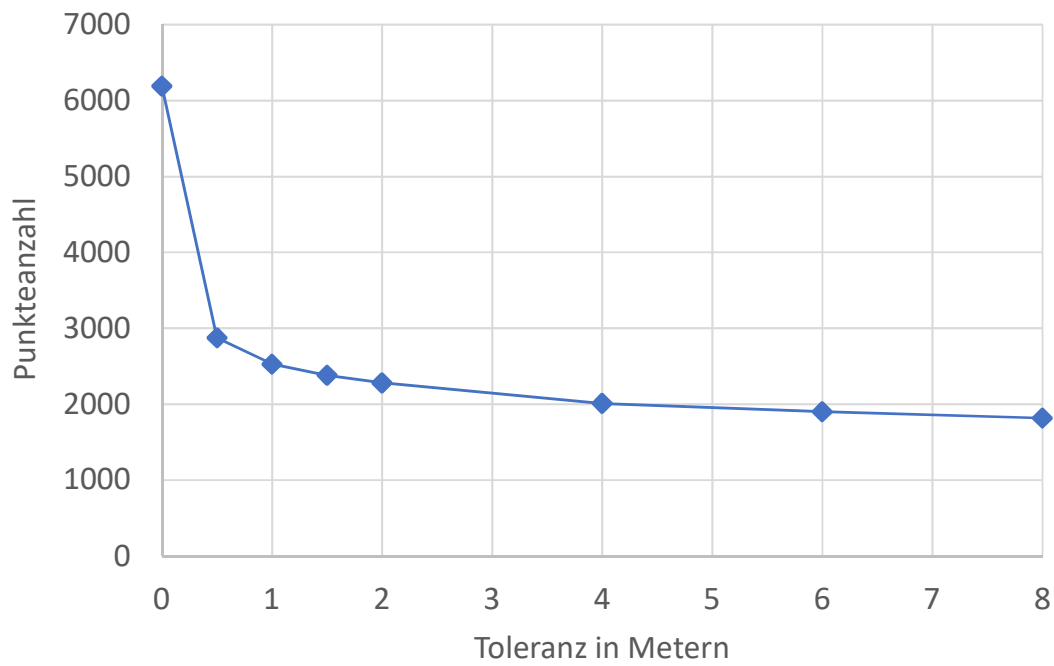
Wird dabei betrachtet, wie viele Geometriepunkte bei der Vereinfachung mit unterschiedlichen Toleranzen beibehalten werden werden, so fällt folgendes auf (vgl. Abbildung 5.1): Bei einer Vereinfachung mit $\varepsilon = 0,5 \text{ m}$ werden im Vergleich zum Original bereits ca. 54% der Punkte entfernt. Daraus lässt sich schließen, dass der Großteil der Geometriepunkte in der Datenbank relativ kleine Gebäudedetails darstellt. Wird die Toleranz auf $\varepsilon = 1 \text{ m}$ erhöht, so werden dabei im Vergleich zu $\varepsilon = 0,5 \text{ m}$ jedoch nur 5% mehr Punkte (relativ zur Gesamtpunktzahl) eingespart. Auch bei einer weiteren Erhöhung der Toleranz in Schritten von 0,5 bis 2 m bleibt die Einsparung in jedem Schritt (im Vergleich zum letzten) im einstelligen Prozentbereich.

5.1.4 Auswahl der Konfigurierten Toleranz

Basierend auf den Erkenntnissen der letzten beiden Unterabschnitte wird nun untersucht, bei welchen Toleranzwerten die visuelle Ähnlichkeit der vereinfachten Geometrie in gutem Verhältnis zur Anzahl der entfernten Punkte steht. Hierbei muss auch beachtet werden, dass der in den Anforderungen (Abschnitt 3.2.1) formulierte Zweck der Karte unter Anderem darin besteht, den einzelnen Gebäuden einen Wiedererkennungswert zu verschaffen. Die auf der Karte angezeigten Geometrien müssen dafür die Gebäude nicht exakt darstellen, sie sollen allerdings die grobe Form jedes Bauwerks widerspiegeln. Da sie dieses Kriterium nicht erfüllen, kommen Vereinfachungen mit Toleranzen von $\varepsilon > 2 \text{ m}$ nicht in Frage. Auf der anderen Seite erscheint eine Vereinfachung mit einer Toleranz von $\varepsilon = 0,5 \text{ m}$ lohnenswert, da hierbei der Großteil der Punkte eingespart wird und die Unterschiede zu den Originaldaten nur bei sehr genauem Hinsehen erkennbar sind. Bei der Betrachtung der damit in Frage kommenden Werte für ε im Intervall $[0,5 \text{ m}, 2 \text{ m}]$ fällt auf, dass der Unterschied bei der Anzahl eingesparter Punkte zwischen $\varepsilon = 2 \text{ m}$ und $\varepsilon = 0,5 \text{ m}$ weniger als 10 % beträgt. Weil zudem die Vereinfachungen ab $\varepsilon = 1 \text{ m}$ stärker auffallen als mit $\varepsilon = 0,5 \text{ m}$, verwendet die Anwendung für die Darstellung des Campus Jülich eine Toleranz von $\varepsilon = 0,5 \text{ m}$.

5.1.5 Auswirkung auf Mobile Endgeräte

Um eine sinnvolle Aussage darüber treffen zu können, ob eine Vereinfachung der Karten-geometrie mit $\varepsilon = 0,5 \text{ m}$ letztlich auch die Benutzbarkeit verbessert, wurde die Karte jeweils mit und ohne Vereinfachung auf einem mobilen Endgerät getestet. Dazu wurde die mobile Version des Browsers Google Chrome auf einem Smartphone der Serie „X“



Toleranz	0 m	0,5 m	1 m	1,5 m	2 m	4 m	6 m	8 m
Punkteanzahl	6189	2876	2528	2382	2283	2011	1904	1818
Anteil entfernter Punkte	0 %	54 %	59 %	62 %	63 %	68 %	69 %	71 %

Abbildung 5.2: Anzahl der Punkte in der vereinfachten Gebäudegeometrie (bei Betrachtung aller Gebäude), in Abhängigkeit von der Toleranz.

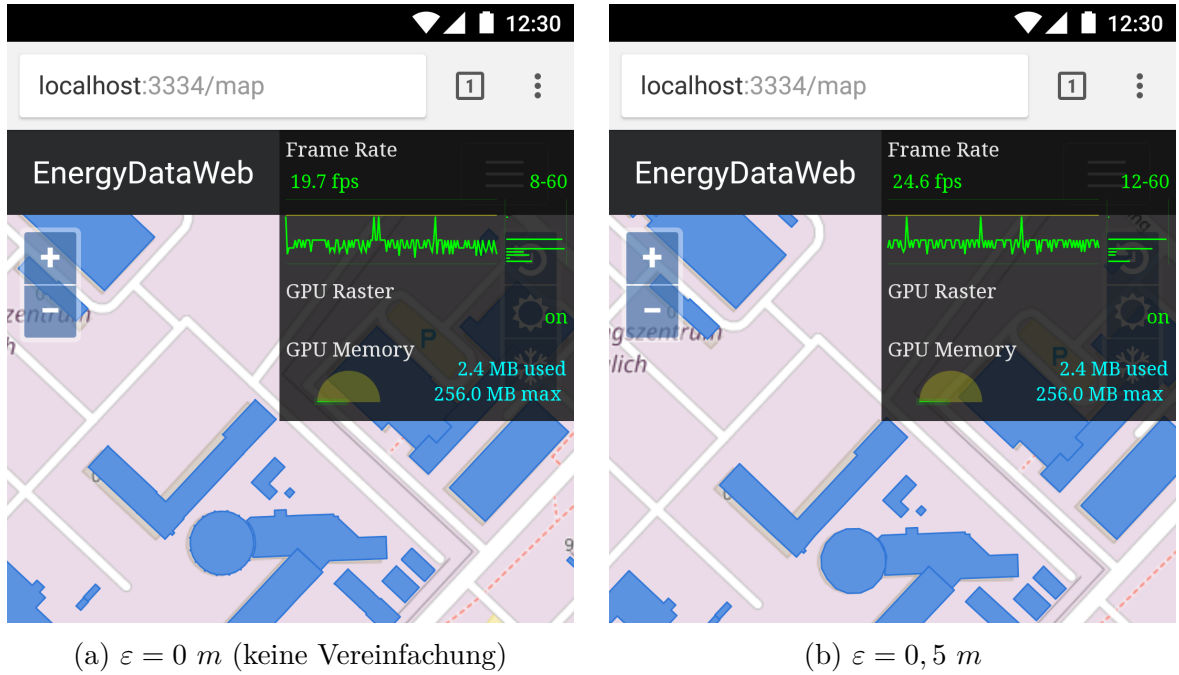


Abbildung 5.3: Frames pro Sekunde (FPS) beim Scrollen über die Kartendarstellung auf einem mobilen Endgerät.

des Herstellers *OnePlus* ausgeführt. Diese Mobilversion kann dabei über die Desktopversion von Chrome ferngesteuert werden, in dem Smartphone und Computer über ein Kabel verbunden und die entsprechenden Entwickleroptionen aktiviert werden. Diese Fernsteuerung ermöglicht es auch, auf dem Mobilgerät einen Graph der *Frame Rate* anzeigen zu lassen, welcher darstellt, wie viele Bilder pro Sekunde (*Frames pro Sekunde*, FPS) vom Mobilbrowser berechnet werden. Je höher diese Zahl ausfällt, desto geringer ist dabei im Allgemeinen die Eingabeverzögerung.

Unter den beschriebenen Versuchsbedingungen hat sich dabei gezeigt, dass die Vereinfachung mit $\varepsilon = 0.5 \text{ m}$ die Frame Rate beim Scollen über die Karte um ca. 4 FPS erhöht (siehe Abbildung 5.3). Dabei ist zu beachten, dass dieses Messergebnis zunächst nicht verallgemeinert werden kann, da es unter Anderem abhängig von der Zoomstufe, dem Browser, dem eingesetzten Mobilgerät sowie den dargestellten Gebäudedaten ist. Dennoch zeigt dieser Fall, dass eine Vereinfachung der Kartendaten einen messbaren positiven Einfluss auf die Frame Rate und somit die Benutzbarkeit haben kann.

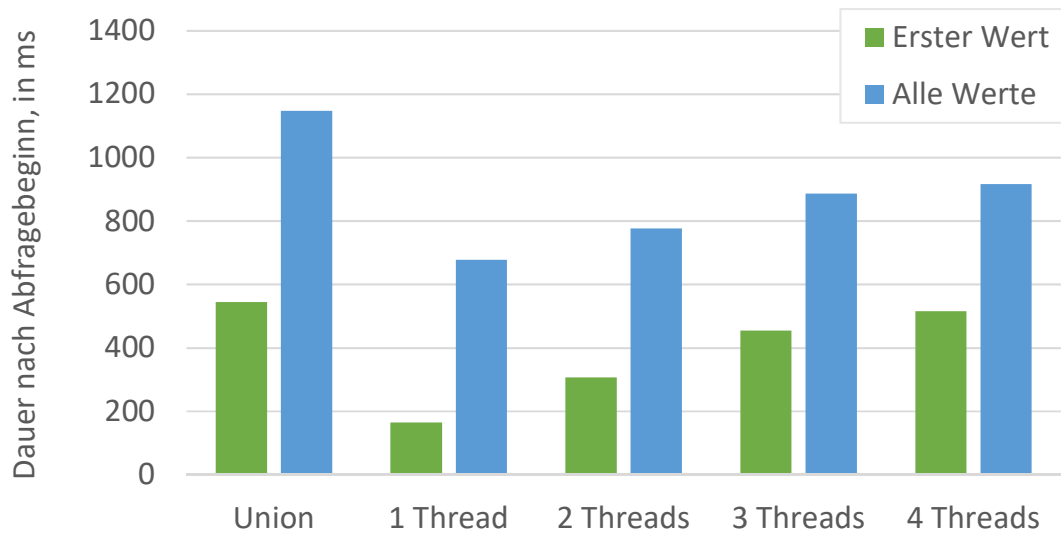


Abbildung 5.4: Abfragedauer von Sensormesswerten, unter Verwendung verschiedener Abfragemethoden.

5.2 Parallelismus beim Abruf von Zeitreihen

Neben der Kartendarstellung besteht eine weitere Hauptfunktion der Anwendung darin, Sensormesswerte auf einem Graphen anzuzeigen. Da einige der Sensoren im Forschungszentrum bereits seit mehreren Jahren in 5-Minuten-Intervallen Messwerte erzeugen, kann bereits die Abfrage der Daten eines einzigen Sensors hunderttausende Messwerte zurückliefern. Somit ist es durchaus denkbar, dass die für die Abfrage benötigte Zeitspanne vom Anwender als Verzögerung wahrgenommen wird. Wie in Abschnitt 4.3.4 beschrieben existieren dabei mehrere Möglichkeiten, wie diese Abfrage durchgeführt werden kann. Deswegen wird nun untersucht, ob die Auswahl einer bestimmten Methode die Abfragedauer reduziert.

Hierzu wurden versuchsweise die Daten eines Sensors mit verschiedenen Methoden abgerufen. Zu jeder Methode wurde ab Abfragebeginn gemessen, wie viel Zeit bis zur Rückgabe des ersten Messwerts benötigt wird und wie lange es dauert, bis alle Messwerte abgerufen wurden. Dabei wurde einerseits der Fall betrachtet, dass in einer einzigen SQL-Abfrage mittels `UNION ALL` die Daten durch den Datenbankserver zusammengefügt werden (Methode: *Union*). Andererseits wurde auch der Fall ausgewertet, dass die Anwendung mehrere SQL-Abfragen parallel startet und die Daten von der Anwendung zusammengefügt werden, wobei maximal $n \in \mathbb{N}_0$ Abfragen parallel laufen (Methode: *n Threads*). Der Sonderfall $n = 1$ Threads beschreibt hierbei das Abfrageverhalten mit der Methode, dass die Anwendung die Abfragen sequenziell ausführt und zusammenfügt,

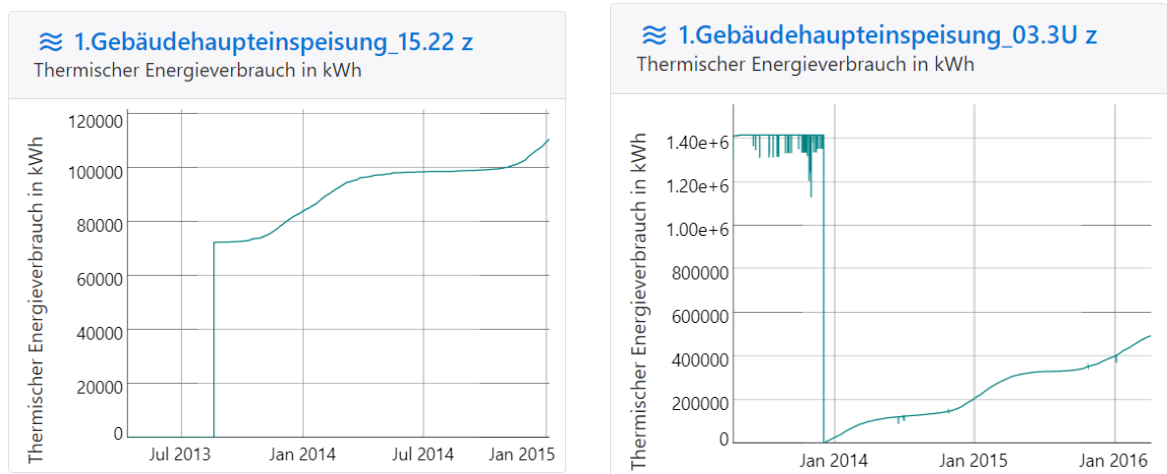


Abbildung 5.5: Fehlerhafte Daten am Anfang von Messreihen.

also ohne Parallelisierung.

Wie in Abbildung 5.4 dargestellt hat der Versuch unter diesen Bedingungen ergeben, dass das datenbankseitige Zusammenfügen der Daten (die Union-Methode) in diesem Fall langsamer ist, als wenn die Daten durch die Anwendung mit $n \in 1, \dots, 4$ Threads abgerufen und zusammengefügt werden. In letzterem Fall fällt zudem auf, dass die Abfragedauer mit steigender Threadanzahl n ebenfalls steigt - die kürzeste Abfragedauer wurde also in diesem Versuch mit $n = 1$ Threads beobachtet. Zumindest bei der momentan in diesem Projekt verwendeten Konfiguration erscheint es also sinnvoll, die Messwerte sequenziell abzufragen und innerhalb der Anwendung zusammenzufügen. Ansonsten ist es auch durchaus denkbar, dass durch eine Änderung der Datenbankkonfiguration oder der in der Anwendung eingesetzten SQL-Bibliotheken andere Abfragemethoden begünstigt werden.

5.3 Messfehler

Bei der Betrachtung der durch die Anwendung visualisierten Sensordaten fällt auf, dass bei einem Großteil der vorhandenen Sensoren zu mehreren Zeitpunkten Messfehler auftreten. Diese können nicht nur die Benutzbarkeit der Messwertgraphen einschränken, sondern auch einen Vergleich zwischen Messwerten und Simulationsergebnissen erschweren. Aus diesen Gründen werden nun beispielhaft verschiedene Typen von Messfehlern beschrieben, die in der verwendeten Datenbank vorkommen.

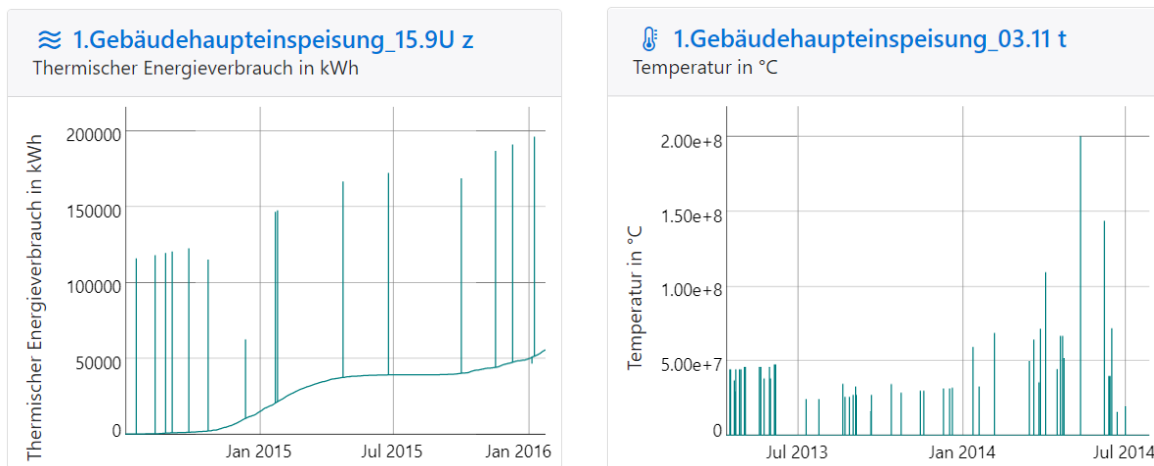


Abbildung 5.6: Fehlerhafte Messwerte in Messreihen, durch welche auch die Skalierung beeinflusst wird.

5.3.1 Falsche Werte am Anfang von Messreihen

Ein häufig vorkommender Messfehler besteht darin, dass am Anfang einer Messreihe über einen längeren Zeitraum kontinuierlich Nullwerte oder fehlerhafte Werte gespeichert sind, bevor die ersten sinnvollen Werte festgehalten werden (siehe Abbildung 5.5).

5.3.2 Einzelne Fehler in Messreihen

Weiterhin sind in vielen Messreihen einzelne Messwerte zu beobachten, die offensichtlich fehlerhaft sind. Wie in Abbildung 5.6 dargestellt können diese fehlerhaften Messwerte auch um mehrere Größenordnungen außerhalb des zu erwartenden Messwertebereichs liegen. Dazu muss beachtet werden, dass die in der Anwendung verwendeten Graphen so implementiert sind, dass sie die Y-Achse automatisch anhand des größten Messwerts der betrachteten Messreihe skalieren. So kann bereits ein einziger fehlerhafter Messwert, der deutlich außerhalb des gültigen Messwertebereichs liegt, die Skalierung so beeinflussen, dass die gültigen Messwerte nur verkleinert oder gar nicht dargestellt werden.

5.3.3 Temporäres Aussetzen der Messungen

Bei einigen Sensoren ist zu beobachten, dass die Messungen auch über längere Zeiträume aussetzen (siehe Abbildung 5.7). In der Messreihe sind für den Zeitraum eines solchen Ausfalls zwar ebenfalls im normalen Intervall Messwerte gespeichert, diese sind jedoch konstant. Bei der Konstante handelt es sich dann entweder um den letzten gemessenen

5 Auswertung im Anwendungsfall Campus Jülich

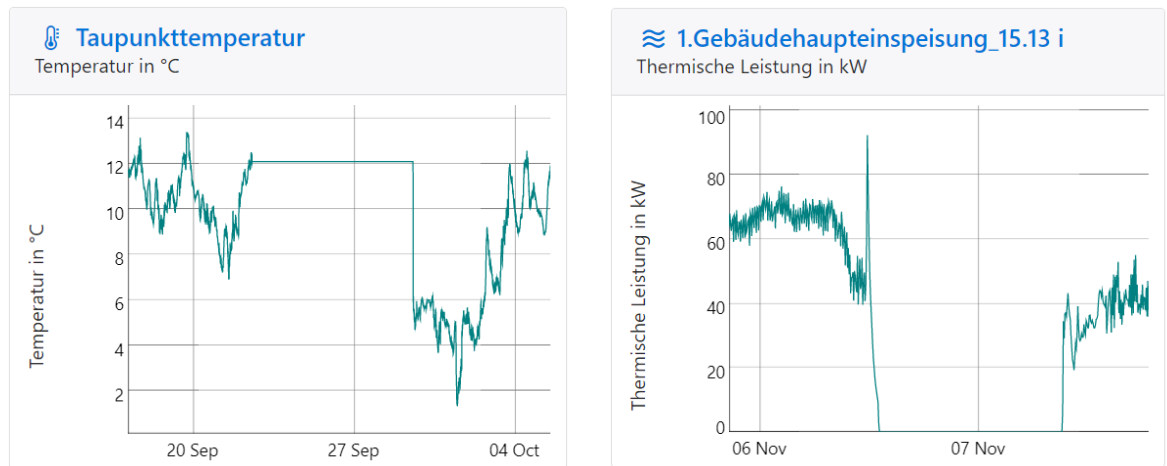


Abbildung 5.7: Temporäres Aussetzen der Messungen.

Wert oder um einen Nullwert.

6 Fazit

In dieser Arbeit wurde die Entwicklung einer Webanwendung für das Forschungsprojekt „EnEff-Campus Living Roadmap“ erläutert. Dazu sind zunächst die Projektziele, die relevanten theoretischen Grundlagen sowie die vorhandene technische Infrastruktur beschrieben worden. Darauf aufbauend folgte eine Definition der Anforderungen, welche die Anwendung erfüllen soll. Weiterhin wurde erklärt, wie diese Anforderungen technisch umgesetzt sind. Zuletzt ist die so entwickelte Anwendung dahingehend ausgewertet worden, wie gut sie die gestellten Anforderungen erfüllt. Da die funktionalen Anforderungen vollständig umgesetzt sind, lag der Fokus der Auswertung darauf, ob einige der verwendeten Algorithmen die Benutzbarkeit der Anwendung verbessern. Dabei ist positiv aufgefallen, dass mittels einer bestimmten Methode die Abfragedauer für einen Messwertegraphen um über 400 ms reduziert werden konnte. Durch den Einsatz des Douglas-Peucker-Algorithmus kann zudem die für die Kartendarstellung benötigte Rechenleistung etwas gesenkt werden. Es ist jedoch möglich, dass auch dies nicht für eine verzögerungsfreie Darstellung ausreicht, wenn die Anwendung für wesentlich umfangreichere Liegenschaften eingesetzt wird.

7 Ausblick

Aus den Ergebnissen der Auswertung und den Zielen des Projekts Living Roadmap ergeben sich verschiedene Perspektiven für die Weiterentwicklung der Anwendung. Dazu gehören sowohl Verbesserungen bestehender Funktionen wie den Gebäude- und Messdatenvisualisierungen, als auch vollständig neue Features wie zum Beispiel eine Anbindung an Simulationssoftware.

7.1 Kartendarstellung

Wie in Abschnitt 5.1 besprochen kann eine Vereinfachung der Kartendaten (zum Beispiel mittels des Douglas-Peucker Algorithmus) bereits dazu beitragen, die für die Kartendarstellung benötigte Rechenleistung messbar zu verringern. Sollte die Anwendung jedoch dazu verwendet werden, Liegenschaften mit wesentlich mehr Gebäuden zu visualisieren, besteht die Möglichkeit dass diese Maßnahme nicht mehr ausreicht, um eine möglichst verzögerungsfreie Bedienbarkeit der Karte zu gewährleisten. In dem Fall könnten die Gebäudedaten bereits auf der Serverseite gerendert werden. Der Vorteil besteht dabei darin, dass der Webbrowser keine Vektordaten visualisieren, sondern nur noch eine Menge an Bildern als Overlay über die Straßenkarte legen muss. Hierbei stellt sich die Herausforderung, die Interaktivität einer so realisierten Kartendarstellung zu gewährleisten: Momentan sind die Informationen zu jedem Gebäude (zum Beispiel Name, Identifikationsnummer und Baujahr) zusammen mit den Vektordaten in einer KML-Datei vorhanden. Dies hat es ermöglicht, Funktionen wie die Verlinkung von oder Tooltips zu jedem Gebäude mit relativ wenig Aufwand zu implementieren. Da fertig gerenderte Kartenabschnitte als Rasterdaten vorliegen, in denen die einzelnen Gebäude durch den Computer nicht ohne weiteres unterscheidbar sind, müssten dafür die interaktiven Funktionen mit einer anderen Methode realisiert werden.

7.2 Messfehlerbereinigung

Neben der Kartendarstellung besteht auch bei den Messwertgraphen noch Verbesserungspotential: hier ist es wünschenswert, dass möglichst viele der in Abschnitt 5.3 beschriebenen Messfehler automatisch vom Server erkannt und entfernt werden. Die so bereinigten Messreihen sollten getrennt von ihrem Original abgelegt werden, um Datenverlust in dem Fall zu vermeiden, dass gültige Messwerte fälschlicherweise als fehlerhaft eingestuft werden. Weiterhin ist es auch denkbar, fehlerhafte Daten in den bereinigten Messreihen durch Approximationen zu ersetzen, welche auf Basis der existierenden korrekten Messwerte berechnet werden. Als Beispiel hierzu kann angenommen werden, dass drei Sensoren S_1, S_2, S_3 und zwei Gebäude B_1, B_2 existieren, wobei S_1 den Heizenergieverbrauch von B_1 , S_2 den Verbrauch von B_2 und S_3 den gemeinsamen Verbrauch von B_1 und B_2 misst. Dann ist stets einer der Sensoren S_1, \dots, S_3 redundant, sein gemessener Wert kann bei einem Ausfall auch aus den Messwerten der anderen beiden Sensoren approximiert werden.

7.3 Simulationsanbindung

Im Rahmen der Einleitung ist bereits das mittel- bis langfristige Ziel beschrieben worden, die Anwendung um eine Anbindung an Simulationssoftware zu erweitern. Damit soll ein Vergleich zwischen Simulationsergebnissen und Messwerten ermöglicht werden, anhand dessen das Simulationsmodell laufend evaluiert und gegebenenfalls verbessert werden kann. Da bereits jetzt mehrere hundert Messreihen existieren kann es auch vorteilhaft sein, diesen Vergleich zumindest teilweise zu automatisieren. Das kann zum Beispiel geschehen, in dem statistische Größen wie die Standardabweichung zwischen simulierten und gemessenen Daten berechnet werden. Dafür ist es jedoch essentiell, dass die Messreihen vorher von Messfehlern bereinigt werden.

Literaturverzeichnis

- [1] *3DCityDB Key Features and Functionalities*. Aufgerufen am 25.06.2017. URL: <http://www.3dcitydb.net/3dcitydb/features/>.
- [2] *Apache Freemarker Manual*. Aufgerufen am 27.06.2017. URL: <http://freemarker.org/docs/index.html>.
- [3] *Apache Freemarker Project History*. Aufgerufen am 27.06.2017. URL: <http://freemarker.org/history.html>.
- [4] Thomas Becker, Claus Nagel und Thomas H. Kolbe. „Semantic 3D Modeling of Multi-Utility Networks in Cities for Analysis and 3D Visualization“. In: *Progress and New Trends in 3D Geoinformation Sciences*. Hrsg. von Jacynthe Pouliot u. a. Springer Berlin Heidelberg, 2013, S. 41–62. ISBN: 978-3-642-29793-9. DOI: 10.1007/978-3-642-29793-9_3.
- [5] *Bekanntmachung der Regeln zur Datenaufnahme und Datenverwendung im Nichtwohngebäudebestand vom 7. April 2015*. Von: Bundesministerium für Wirtschaft und Energie; Bundesministerium für Umwelt, Naturschutz, Bau und Reaktorsicherheit.
- [6] Bill Burke. *RESTful Java with JAX-RS 2.0: Designing and Developing Distributed Web Services*. O'Reilly Media, Inc., 2013. ISBN: 978-1-449-36134-1.
- [7] U Dapp und M Dirksen-Fischer. „Einsatz eines Geoinformationssystems (GIS) zur Implementierung einer internetgestützten Informationsplattform räumlicher und inhaltlicher Daten von Gesundheitsdienstleistern in Hamburg“. In: *Prävention und Gesundheitsförderung* 1.3 (2006), S. 159–165.
- [8] *Das Forschungszentrum Jülich in Zahlen und Fakten*. Aufgerufen am 27.06.2017. URL: http://www.fz-juelich.de/portal/DE/UeberUns/DatenFakten/_node.html.
- [9] *DIN 277-1:2016-01: Grundflächen und Rauminhalte im Bauwesen*. Norm. 2016.

- [10] David H Douglas und Thomas K Peucker. „Algorithms for the reduction of the number of points required to represent a digitized line or its caricature“. In: *Cartographica: The International Journal for Geographic Information and Geovisualization* 10.2 (1973), S. 112–122.
- [11] *Forschungszentrum Jülich: Lageplan Geschäftsbereich Personal*. Aufgerufen am 27.06.2017. URL: http://www.fz-juelich.de/gp/DE/UeberUns/Anfahrt/anfahrt_node.html.
- [12] Martin Fowler. *UML Distilled, Third Edition: A Brief Guide to the Standard Object Modeling Language*. Addison-Wesley Professional, 2004. ISBN: 0-321-19368-7.
- [13] *GitHub: OpenLayers Contributors*. Aufgerufen am 07.07.2017. URL: <https://github.com/openlayers/openlayers/graphs/contributors>.
- [14] *Google Maps APIs Pricing and Plans*. Aufgerufen am 24.06.2017. URL: <https://developers.google.com/maps/pricing-and-plans/>.
- [15] *Google Maps APIs Terms of Service*. Aufgerufen am 24.06.2017. URL: <https://developers.google.com/maps/terms>.
- [16] Gerhard Gröger und Lutz Plümer. „CityGML – Interoperable semantic 3D city models“. In: *ISPRS Journal of Photogrammetry and Remote Sensing* 71 (2012), S. 12 –33. ISSN: 0924-2716. DOI: 10.1016/j.isprsjprs.2012.04.004.
- [17] Mordechai Haklay und Patrick Weber. „OpenStreetMap: User-generated street maps“. In: *IEEE Pervasive Computing* 7.4 (2008), S. 12–18.
- [18] *Institut für Wohnen und Umwelt: Deutsche Gebäudetypologie*. Aufgerufen am 03.07.2017. URL: http://www.iwu.de/fileadmin/user_upload/dateien/energie/klima_altbau/Gebaeudetypologie_Deutschland.pdf.
- [19] *Jersey - RESTful Web Services in Java*. Aufgerufen am 27.06.2017. URL: <https://jersey.github.io/>.
- [20] *Jersey User Guide, Version 2.25*. Aufgerufen am 27.06.2017. URL: <https://jersey.github.io/documentation/2.25/user-guide.html>.
- [21] *jOOQ: Downloads and Pricing*. Aufgerufen am 27.06.2017. URL: <https://www.jooq.org/download/>.

- [22] Thomas H. Kolbe, Gerhard Gröger und Lutz Plümer. „CityGML: Interoperable Access to 3D City Models“. In: *Geo-information for Disaster Management*. Hrsg. von Peter van Oosterom, Siyka Zlatanova und Elfriede M. Fendel. Springer Berlin Heidelberg, 2005, S. 883–899. ISBN: 978-3-540-27468-1. DOI: 10.1007/3-540-27468-5_63.
- [23] A. Leff und J. T. Rayfield. „Web-application development using the Model/View/Controller design pattern“. In: *Proceedings of Fifth IEEE International Enterprise Distributed Object Computing Conference*. 2001, S. 118–127. DOI: 10.1109/EDOC.2001.950428.
- [24] Ina Ludwig, Angi Voss und Maike Krause-Traudes. „A Comparison of the Street Networks of Navteq and OSM in Germany“. In: *Advancing Geoinformation Science for a Changing World*. Hrsg. von Stan Geertman, Wolfgang Reinhardt und Fred Toppen. Springer Berlin Heidelberg, 2011, S. 65–84. ISBN: 978-3-642-19789-5. DOI: 10.1007/978-3-642-19789-5_4.
- [25] Vlad Mihalcea. *High-Performance Java Persistence*. Leanpub, 2017.
- [26] Tyler Mitchell. *Web Mapping Illustrated: Using Open Source GIS Toolkits*. O’Reilly Media, Inc., 2005. ISBN: 978-0-596-00865-9.
- [27] Prof. Dr.-Ing. Dirk Müller und Prof. Dr.-Ing. habil. Christoph van Treeck. „Förderschwerpunkt EnEff:Stadt Campus- Projekte. Entwicklung eines Energieversorgungskonzeptes mit Modellbasierter prädiktiver Regelung anhand einer Living Roadmap am Beispiel des Forschungszentrums Jülich“. In: (2015). Förderkennzeichen: 03ET1352A.
- [28] Deborah Nolan und Duncan Temple Lang. „Keyhole Markup Language“. In: *XML and Web Technologies for Data Sciences with R*. Springer New York, 2014, S. 581–618. ISBN: 978-1-4614-7900-0. DOI: 10.1007/978-1-4614-7900-0_17.
- [29] Romain Nouvel u. a. „Genesis of the CityGML Energy ADE“. In: *Proceedings of International Conference CISBAT 2015 - Future Buildings and Districts - Sustainability from Nano to Urban Scale*. EPFL-CONF-213436. LESO-PB, EPFL. 2015, S. 931–936.
- [30] Regina O Obe und Leo S Hsu. *PostGIS in action*. Manning Publications Co., 2015.
- [31] *OGC® Approves KML as Open Standard*. Aufgerufen am 24.06.2017. URL: <http://www.opengeospatial.org/pressroom/pressreleases/857>.

- [32] *OpenLayers History*. Aufgerufen am 23.06.2017. URL: <https://web.archive.org/web/20110817120046/http://trac.osgeo.org/openlayers/wiki/History>.
- [33] *OpenLayers License*. Aufgerufen am 23.06.2017. URL: <https://github.com/openlayers/openlayers/blob/master/LICENSE.md>.
- [34] *OpenLayers Logo*. Aufgerufen am 12.07.2017. URL: http://wiki.openstreetmap.org/wiki/File:Public-images-osm_logo.svg.
- [35] *OpenStreetMap Logo*. Aufgerufen am 12.07.2017. URL: http://wiki.openstreetmap.org/wiki/File:Public-images-osm_logo.svg.
- [36] Antonio Santiago Perez. *OpenLayers Cookbook*. Packt Publishing Ltd, 2012.
- [37] *PostGIS Logo*. Aufgerufen am 03.07.2017. URL: <https://postgis.net/images/postgis-logo.png>.
- [38] F. Prandi u. a. „3D web visualization of huge CityGML models“. In: *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 40.3 (2015), S. 601.
- [39] Herbert Schildt. *Java: The Complete Reference, Ninth Edition*. McGraw-Hill, 2014. ISBN: 978-0-07-180855-2.
- [40] *The jOOQ Release Note History*. Aufgerufen am 27.06.2017. URL: <https://www.jooq.org/notes>.
- [41] *VDI 3801 Blatt 1: Verbrauchskennwerte für Gebäude*. Norm. 2013.
- [42] Nicholas C Zakas. „The Evolution of Web Development for Mobile Devices“. In: *Queue* 11.2 (2013), S. 30.