Master's thesis

2020

Tobias Bergkvist

Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Engineering Cybernetics

Tobias Bergkvist

Power Distribution Systems

Assessing Grid Health With Computer Simulation and Interactive Visualization

June 2020

NTNU
Norwegian University of
Science and Technology

Power Distribution Systems

Assessing Grid Health With Computer Simulation and Interactive Visualization

Tobias Bergkvist

Engineering Cybernetics
Submission date: June 2020
Supervisor: Ole Morten Aamo

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Abstract

In the years to come, distribution system operators will face challenges if they don't adapt. The transport industry is becoming electrified, and production is becoming less reliable through an increase in reneweable energy sources like solar and wind.

Processing times for finding out if the power grid can handle the installation of an electric car charger can be long and tedious. Capacity problems, voltage problems, short circuit currents, and stability problems are all things that can be calculated through computer simulation. What if we could automate the processing of these applications?

We demonstrate the process of translating XML-files (Common Information Model) of imperfect data quality into a converging simulation model in the Python library pandapower. We extract the geometry from the CIM-data, combining it with simulation results to visualize large amounts of assets in the web browser. The visualization is color coded according to simulation results, and powered by the WebGL2 and React framework deck.gl.

The results are two protoype applications: "Grid Overview": Visualize and run powerflows on large grid models and "Subgrid Health": Provide a user-interface for case-workers to quickly approve applications for electric car chargers.

Sammendrag

I årene som kommer vil distribusjonsnettoperatørene møte på utfordringer dersom de ikke tilpasser seg. Transportindustrien går mot å bli elektrifisert. Produksjon blir mindre forutsigbar grunnet et økning av fornybare energikilder som vind og solkraft.

Prosesseringstiden for å finne ut om strømnettet kan takle at du installerer en elbillader i garasjen kan være lange og slitsomme. Kapasitetsproblemer, spenningsproblemer, kortslutningsproblemer og stabilitetsproblemer er alle ting som kan beregnes gjennom simulering. Ville det vært mulig å automatiserte prosessering av disse søknadene?

Vi demonstrerer prosessen av å oversette XML-filer (Common Information Model) med ufullkommen datakvalitet til konvergerende simuleringsmodeller i Python-biblioteket pandapower. Vi henter ut geometrien fra CIM-dataen, og kombinerer denne med simuleringsresultater for å visualisere et stort antall komponenter i nettleser. Visualiseringen fargekodes basert på simuleringsresultater og er drevet av det WebGL2 og React-baserte rammeverket deck.gl.

Resultatet er to prototype-applikasjoner: Grid Overview": Visualisering og lastflytanalyse av store nettverksmodeller og Subgrid Health": Et brukergrensesnitt for saksbehandlere til å raskt kunne godkjenne søknader om elbillader.

Contents

1 Structure

This report does not intend on being a perfect record of everything that has been done. Instead, the goal is to create an intuition-focused journey through some interesting insights and bigger ideas. Several details will therefore not be included.

It is recommended for the reader to be familiar with linear algebra and electrical engineering on a basic level. Having been exposed to JavaScript/TypeScript/React and Python is also an advantage.

There are three sections of background material, to provide context and a general overview of ideas, technologies and data - to substantiate the choices of technologies used in the project. The method is also divided into three sections, which describes the journey of the grid data from CIM files to visualizations on a screen.

The result sections provides screenshots of the resulting application prototypes, and a compact overview of features. At last, we will discuss the results, and this can be further worked on and improved.

2 Introduction

This is a cross-disciplinary, and quite wide master thesis, focusing on electrical engineering, modeling and simulation, backend web development, UI and frontend development, data science/visualization, geographic information systems as well as a little bit on Norwegian laws/regulations.

2.1 Kongsberg Digital

Kongsberg Digital was founded in 2015, as part of the Kongsberg Group. and is a provider of next generation software and digital solutions to customers within maritime, oil & gas and utilities. ("Kongsberg Digital" 2015).

This master thesis is written for the department of electrical utilities in Kongsberg Digital. In particular, it is written as part of the KogniTwinGrid project ("KogniTwinGrid" 2019).

2.2 Task Description

2.2.1 General goals

1. Ability to create a converging powerflow model from CIM files in spite of imperfect data quality.
2. Simulation of voltage, capacity, and short-circuit currents
3. Geospatial visualization of grid assets and simulation results
   - Color coding of grid assets based on simulation results
   - The ability to search for and find grid assets in a map application

2.2.2 Subgrid Health

In Norway, right now - if you want to get an electrical car charger, or install solar panels - you have to go through a tedious application process. An engineer will have to look at the grid, and perform manual calculations to see if the grid can handle it. This process can take several weeks. What if we could make a system for automating this work - reducing costs and drastically speeding up processing times?

Creating a prototype application for case workers to assess the health of subgrids will be our second goal.

2.3 Scope

This project will focus on:

- How to build a converging powerflow model from CIM files.
- Interactive visualization of grid assets and simulation results.
- Application prototypes that could improve insight, save money and resources, and reduce processing times for grid companies.

In order to limit the scope of this project, some things that are NOT focused on or considered are:

- Realistic load profiles for customer power consumption. *Instead, for the sake of simplicity, a flat usage throughout the entire year based on annual consumption will be assumed.*
- Verification and tuning of model against measurements from the real power grid.
- Authentication, authorization, multi-tenancy and an automated onboarding process.
- Real-time or historical sensor data
- Economics of grid management: Cost of delivering power or building out infrastructure

3 Background: Power grids and numerical methods

“A power grid is a network of power lines and associated equipment used to transmit and distribute electricity over a geographic area” (Collins Dictionary n.d.).

3.1 History

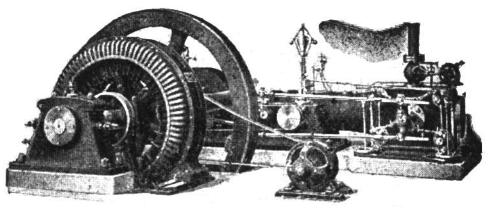


Figure 1: Alternating current generator (Hawkins 1917b)

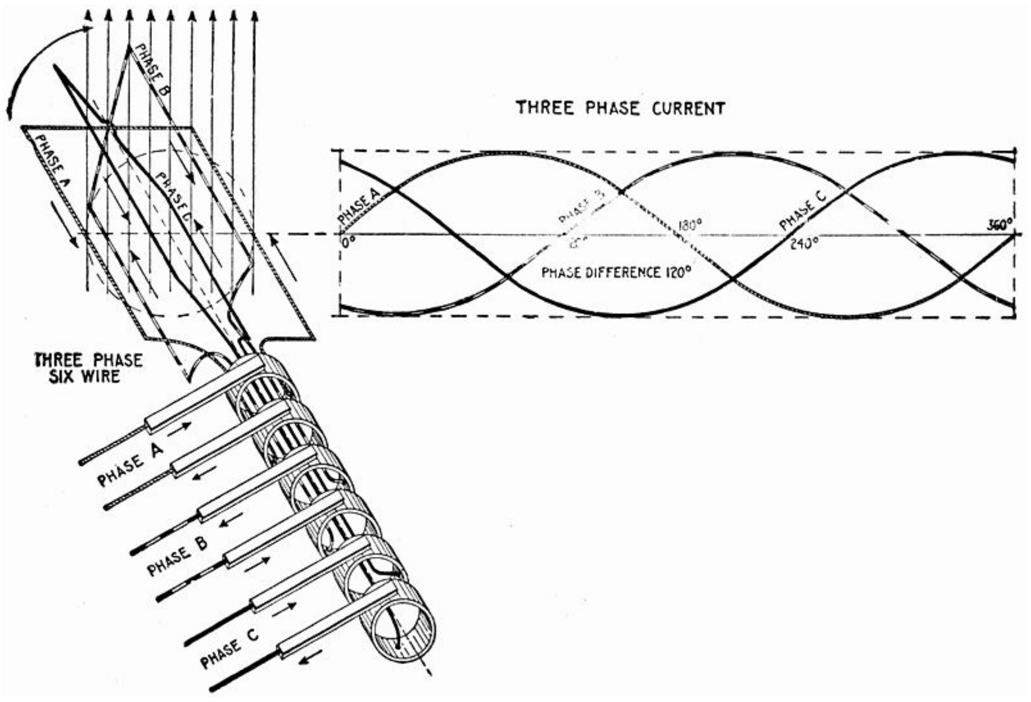


Figure 2: Illustration of 3-phase alternating current (Hawkins 1917a)

You might be familiar with the rivalry of Edison, Tesla and Westinghouse, and their race to electrify the world (Jonnes 2004). With this race came the fight between direct current (DC) and alternating current (AC) for usage in power grids.

In the 1870s and 1880s, DC power systems were dominating (led by Thomas Edison, Charles Brushm, and Werner von Siemens). At this point, only power grids stretching over small geographic areas were available. And 95% of residents in the US did not have access to the power grid. (Edison Tech Center n.d.)

The interest in AC power was sparked because it could be transformed up and down in voltage using power transformers. With a higher voltage, the percentage of wasted energy through transmission losses goes down, meaning it can be carried over longer distances, and transformed back down before being used. *By doubling the voltage, you reduce losses to a quarter.*

The development of 3-phase AC power in the late 1880s ultimately proved the effectiveness of the system, which paved the road for the electrification of entire cities and regions in the 1890s and onwards.

High-voltage direct current (HVDC) is used today for transmitting power over very long distances, since it is more efficient to transfer over similar infrastructure compared to AC. Especially in underwater cables. Converting to and from AC is the expensive part of this solution, however - requiring large specialized plants. This is the reason it is only economical to use for very long distances.

In this project, we won't be looking at HVDC transmission, but focus on AC distribution networks owned by individual grid companies operating at lower voltages (up to 132kV).

3.2 Future

NVE estimates that a full electrification of the transport industry might happen within 20-30 years, potentially creating challenges for today's distribution network. (NVE 2018)

As home owners install solar panels on their roofs, and electric car chargers in their garages - consumption is increased, and production is decentralized. This is something the distribution grids will need to adapt to.

At the same time, with the global interest in renewable energy like solar and wind - generation of power will become less reliable, and more prone to fluctuate. Solar panels are in general a bad idea to use for large scale generation in Norway. Why is that?

In Norway, we consume the most energy when it is cold, and use very little energy when it is warm and sunny outside. Solar panels produce energy when we least need it in Norway. Not only that - but battery technology is expensive, meaning it usually isn't economical to store the energy.

Solar panels likely makes for a much better investment in countries where an increase in the amount of sun leads to an increase in power usage. This is typical in warmer countries, where a lot of the energy is spent on cooling down the inside of buildings, rather than heating them up.

There are a range of potential problems that can occur in a power grid. Some of these are:

- Capacity problems
- Short circuit problems
- Voltage problems
- Stability problems (frequency and voltage collapse)

The best way of discovering these problems before they occur is through simulation, hence arises the need for a digital twin. A digital twin could also be very helpful for planning what to build next.

3.3 Grid frequency

The grids in Norway operate at a frequency of 50Hz (300rPM), in contrast to The United States, where a frequency of 60Hz (360rPM) is used (Grøn 2019). The grid frequency reflects how quickly the generators are spinning.

Changes in frequency can tell us something about the balance of consumption and production.

- If generation exceeds consumption, frequency will rise.
- If consumption exceeds the generation, frequency will fall.

To help visualize why this is the case, consider that you are sitting in a car, and push the accelerator exactly half way down, acheiving some constant speed. Your car engine will now have a constant rPM/frequency.

- As you push the accelerator further down (increasing production), your engine will start spinning more quickly (the frequency increases).
- If you let the accelerator go, your engine will spin more slowly.
- If you start going downhill (decreasing load on the engine), your engine will start spinning more quickly.
- If you start going uphill (increasing engine load), the engine will start spinning more slowly.

Grid companies around the world are usually monopolies, regulated by national or international laws regarding the quality of service they deliver. In Norway, there are laws (Norwegian oil and energy department 2004), as well as a nordic agreement ("Nordic System Operation Agreement" 2019) for what are considered acceptable deviations in frequency.

For technical reasons, the generators are limited to a certain operational frequency range. Safety mechanisms will disconnect generators from the grid if their frequency deviate outside this range. This means that a large unexpected load on the network can cause the network frequency to drop below the safety threshold. This causes generators to disconnect from the network, further reducing network frequency - propagating a chain reaction that can turn into a large-scale blackout.

According to the Nordic System Operation Agreement, the frequency in the transmission network must not exceed ± 0.1 Hz for more than 15,000 minutes per year. The Transmission System Operators (TSOs) are responsible for ensuring this. A frequency deviation of more than ± 1 Hz for any amount of time is considered an emergency.

3.4 Phasors as a tool for analyzing AC power grids

Phasors are used in steady-state AC analysis because they can turn differential equations into algebraic equations for well behaved sinusoidal signals. This is a big deal, because algebraic equations are a lot easier to solve than differential equations. A phasor describes the magnitude and phase of a the sinusoidal signal using a complex number, usually written on exponential form.

Algebraic phasor equations for AC circuits look and feel a lot like the algebraic equations used in DC analysis. This means that knowledge and intuition around DC analysis largely can be carried over to AC analysis, by simply substituting in complex numbers.

3.4.1 Complex number notation

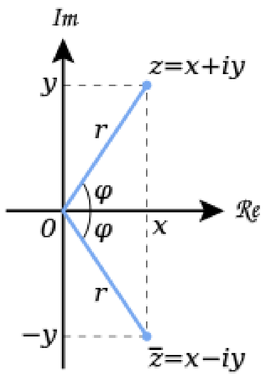


Figure 3: Complex conjugate illustration (Alexandrov 2007)

As is customary in electrical engineering, j will be used as the imaginary unit instead of i.

$$j^2 = -1$$

Given a complex number z, we can define it as the sum of its real (x) and imaginary part (y)

$$z = x + jy$$

We can also represent a complex number on polar form with size $r = |z|$ and angle $\phi = \arg z$:

$$z = r \cdot e^{j\phi} = r\angle\phi$$

The real and imaginary parts of a complex number on polar form can be extracted using Eulers formula:

$$r \cdot e^{j\phi} = r \cdot (\cos\phi + j\sin\phi)$$

A complex conjugate (represented by either star or a bar) flips the sign of the imaginary part:

$$z^* = \bar{z} = x - jy = r\angle(-\phi)$$

3.4.2 Voltage (Volt [V])

Voltage measures the difference in electric potential between two points. It corresponds to the amount of work needed to move a unit of charge between the points.

In an AC circuit, the sinusoidal voltage signal can be described with a phasor.

$$V = |V|\angle\arg V$$

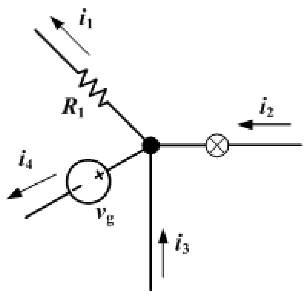


Figure 4: Illustration of Kirchoffs law $[-i_1 + i_2 + i_3 - i_4 = 0]$ (Omegatron 2006)

3.4.3 Current (Ampere [A])

Current measures the flow of charge through a circuit.

The sum of currents moving into a single branch is 0 (Kirchhoff 1845).

$$\sum_{k=1}^{n} I_k = 0$$

Kirchhoffs current law forms the basis for nodal network analysis of electrical grids, which is often used in computer simulation tools. In an AC circuit, the sinusoidal current can be described with a phasor.

$$I = |I| \angle \arg I$$

3.4.4 Impedance (Ohm [Ω])

Impedance is a generalization of resistors, capacitors and inductors. (Kennelly 1893)

$$Z = R + jX = |Z| \angle \arg Z$$

The impedance (Z) is a measure for how current (I) is opposed in an electric circuit when a voltage (V) is applied. This relationship is described through a generalization of Ohm's law:

$$I = \frac{V}{Z}$$

3.4.5 AC Power (Volt-Ampere [VA])

The power phasor S is defined given the voltage (V) and current (I) phasors. Notice that we are using the complex conjugate of the current.

$$S = |S| \angle \phi = VI^*$$

The effect of this is that we find the phase difference instead of the sum.

$$|S| = |V| \cdot |I|$$

$$\phi = \arg V - \arg I$$

Active power (P) is defined as the real part of the power (S), and corresponds to power that can be consumed (to light up your lightbulb or similar).

$$P = |S| \cos \phi$$

Reactive power (Q) is defined as the imaginary part of the power (S). It can be visualized as only being "borrowed", before quickly being given back right afterwards. (due to the angle between voltage and current) This is power the network has to be able to carry around, but which in practice is not usable for performing work.

$$Q = |S| \sin \phi$$

Dealing with reactive power and synchronizing phase shifts in power grids are some of the biggest challenges with using alternating current (AC) for distribution. Connecting consumers and generators to the grid will affect voltage and current angles. This means that some consumers/generators might increase the amount of reactive power in the grid, while others might decrease it.

3.5 Newton-Raphson

The Newton-Raphson method can solve simultanous sets of equations numerically. This means that by setting up equations for Kirchoffs current law at every branch, expressed in voltages - we would be able to find the voltages at every branch, which would allow us to calculate currents and power.

3.5.1 Finding the roots of a single function

Let's say we want to solve the following equation for the scalar variable x:

$$f(x) = 0$$

For this we can use Newton's method, given some initial guess x_0:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

And stop iteration when $|x_{n+1} - x_n|$ falls below a predefined threshold. If this requires more than for example 10 iterations, we could give up - and report this as an error to the user. Some things to be aware of when using Newton's method:

- If $f'(x_n)$ gets close to (or equal to) 0, this can cause divergence - even though a valid solution might exist.
- Convergence is only guaranteed if your initial guess is in the "neighborhood" of the solution. The more unlinear the function, the closer to the solution you need to start.
- If there is more than one solution, at most one of these can be found for each initial guess x_0

3.5.2 Set of equations

$$\mathbf{f}(\mathbf{x}) = \begin{pmatrix} f_1(\mathbf{x}) \\ f_2(\mathbf{x}) \\ \vdots \\ f_k(\mathbf{x}) \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} = \mathbf{0}$$

A rule of thumb is that you need the same number of equations as the number of unknowns you are trying to solve for. These equations also need to be linearly independent of each other.

The Jacobi matrix $\mathbf{J}(\mathbf{x})$ is the matrix equivalent of the derivative in the single-equation version of Newton's method. Notice that $\mathbf{J}(\mathbf{x})$ is square ($k \times k$) for a set of k equations and k variables

$$\mathbf{J}(\mathbf{x}) = \frac{\partial \mathbf{f}}{\partial \mathbf{x}} = \begin{pmatrix} \partial f_1/\partial \mathbf{x} \\ \partial f_2/\partial \mathbf{x} \\ \vdots \\ \partial f_k/\partial \mathbf{x} \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_k} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_k} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_k}{\partial x_1} & \frac{\partial f_k}{\partial x_2} & \cdots & \frac{\partial f_k}{\partial x_k} \end{pmatrix}$$

The version of the Newton-Raphson method for a set of equations, rather than a single equation then becomes:

$$\mathbf{x_{n+1}} = \mathbf{x_n} - \mathbf{J}(\mathbf{x_n})^{-1}\mathbf{f}(\mathbf{x_n})$$

Notice that if $\mathbf{J}(\mathbf{x_n})$ ever becomes singular (corresponding to $f'(x_n) = 0$ in the 1-dimensional case), or if its determinant approaches zero - this can cause divergence.

4 Background: The modern web and programming paradigms

Now we will jump into a very different topic. We are going to create a web application as the client-facing part of our digital twin.

But why use the web, instead of a traditional desktop application that you download and install? Some significant advantages of building a web application are:

- Easy onboarding: The customer won't need to download any software

- Accessibility: The customer can access the application easily from multiple machines
- Seamless updates: No effort is required from the customer to update to the latest version, since they just need to refresh the website.

This part of the background material will take a deep dive into programming paradigms and web technology to substantiate the further choices of libraries and technologies used for the project.

4.1 Imperative vs Declarative code

Imperative and declarative programming are paradigms in computer programming. Imperative code tells the computer "what to do", whilst declarative code tells the computer "what it wants". As a rule of thumb, below any declarative code, you will generally find some kind of imperative implementation. Declarative code in practice is nothing more than a (good) abstraction over imperative code.

Example: You write a function in C that returns the sum of two numbers. The compiler will then translate this into imperative instructions (assembly language), that your computer understands and is able to execute.

Why is it that a program written entirely in assembly language is hard to understand? After all, the possible instructions are few and conceptually very easy to understand independently. The emergence that makes an imperative program scale poorly, is that the amount of things you need to keep track of in your brain at the same time, increase rapidly with the size of the program.

By this logic, small units of imperative code should be fine as long as they are properly isolated from each other. A good functional or declarative abstraction removes the need to keep track of how something is achieved. The programmer can get away with describing what he or she wants, without having to understand or even think about how to achieve it.

Another concept used heavily in functional/declarative programming is immutability. Once a variable has been assigned a value, it can't be changed. This significantly reduces mental overhead - since you know a value can't have changed between its definition and a place it is being used. You won't need to check the lines of code in between.

The example below is probably not something you'd ever see in real code (unless you are working on a kitchen robot), but it is shown to the reader because the steps are easy to visualize for a human being.

```
// IMPERATIVE example (detail oriented)
// Notice the use of verbs here. Describing the procedural creation of the hamburger.
function createHamburger({ upperBun, lowerBun, onions, patty, salad }) {
  let result = createEmptyIngredientStack()

  lowerBun.heat()
  result.place(lowerBun)

  patty.heat()
  result.place(patty)

  onions.chop()
  result.place(onions)

  salad.chop()
  result.place(salad)

  upperBun.heat()
  result.place(upperBun)

  return result
}

// DECLARATIVE example (big picture oriented)
// Notice the lack of verbs. We describe the desired result instead of the actions.
function hamburger({ upperBun, lowerBun, onions, patty, salad }) {
  return stackedIngredients([
    heated(upperBun),
    chopped(salad),
    chopped(onions),
    heated(patty),
    heated(lowerBun),
  ])
}
```

4.2 What about side effects?

Side effects generally describe interactions between your program and the outside world. Essentially, these are imperative actions, that cannot reasonably be isolated within your program. An example of this, is controlling a robot arm, writing to a database, or printing something to the screen.

It might be tempting to bury the side effects below abstractions and hide them deep inside your functions, but this is actually a trap. Since side effects are by definition coupled to the outside world, burying them in “isolated modules” is not going to change that. They are still coupled, and so is all the code you now used to bury it with.

There will be no way of running the code, without also causing the buried side effects, which is not desirable. It makes the code harder to reason about and harder to test. It is better to keep your side effects at the “top” of your program, rather than deep within functions. “If you want to get rid of someone, make them your manager” (Rhodes 2014)

But what about our hamburger example? Haven’t we buried side effects here? A very simple question you could ask is: “Is it possible to run this function without causing side effects?”. If the answer is yes, then that means the side effects are decoupled from our logic (which is what we want).

4.3 The unnatural split of HTML, CSS and JavaScript

HTML and CSS are both declarative languages, that describe a result, rather than how to achieve it. HTML and CSS are essentially domain-specific, non-turing-complete languages. If you want to do something that HTML and CSS does not support, you have to turn to JavaScript to achieve it. JavaScript is turing-complete, and is allowed to imperatively modify the HTML tree and CSS rules (the DOM).

Although HTML and CSS are great declarative languages for their specific domains, they also present a somewhat unnatural split in your code base. Essentially, if you want to create a new button, you would have to spread its definition between HTML, CSS and JavaScript files. Reusing this exact button elsewhere is not exactly straight forward when its code is split between at least 3 files.

Of course, you could manipulate the HTML tree/CSS rules dynamically from JavaScript to create a button, and then reuse these instructions when you want to create another button of the same type. The problem here is that you sacrifice the overview, big picture and robustness that declarative code is able to give you. (burying side-effects) The code base becomes harder to follow, and you might even need to draw a state diagram to debug your website. Is there a way we could avoid turning everything into an imperative mess?

Imagine if we created our own declarative data structure for describing a tree of custom components, where each component can describe its HTML, CSS and JavaScript in a single file/definition. We would also need to make sure the actual HTML and CSS are in sync with our tree of custom components (the code that does this should not be buried).

Turns out, Facebook has already done this, by creating the libraries React, and ReactDOM. React takes care of building/updating this custom tree structure (known as the virtual DOM). ReactDOM ensures that the HTML tree and the CSS rules are in sync with the virtual DOM (performing all the necessary modifications/side-effects to ensure this).

4.4 React and ReactDOM

```
// index.jsx
import React from 'react'
import ReactDOM from 'react-dom'

// This creates/"mounts" a virtual DOM into an already existing HTML element.
// ReactDOM ensures that that the HTML and CSS are in sync with the virtual DOM.
ReactDOM.render(<App />, document.getElementById('app'))

// Defines the <App /> component. Notice that it contains two button components
// and it has a background color set to blue.
```

```
function App() {
  return (
    <div style={{ backgroundColor: 'blue' }}>
      <Button color='red' />
      <Button color='black' />
    </div>
  ) // color as passed to <Button /> is called a "prop".
}

// Our reusable <Button color=? /> component that keeps track of
// how many times it has been clicked
function Button({ color }) {
  const [timesClicked, setTimesClicked] = React.useState(0)
  return (
    <button style={{ color }} onClick={() => setTimesClicked(timesClicked + 1)}>
      I have been clicked {timesClicked} times!
    </button>
  )
}
```

When working with React and ReactDOM, you might come across two file extensions you haven't seen before. These are `.jsx` and `.tsx` (TypeScript equivalent of jsx). The jsx and tsx extensions provide some syntactic sugar to make building the component tree a bit more familiar and aesthetically pleasing.

The "compilation step" will strip away the syntactic sugar, and replace it with JavaScript as in the example below:

```
// jsx
const app = (
  <App x={1} y='test'>
    <button>Click me {someVariable}</button>
  </App>
)

// translated to regular javascript
const app = React.createElement(
  App,
  { x: 1, y: 'test' },
  React.createElement('button', null, `Click me ${someVariable}`)
)
```

What if we want more than just one component to access the number of times a button has been clicked? Let's say we want the buttons to share the number of clicks, such that clicking any of the buttons will increment the number displayed on both. State in React only flows "downwards". Our button cannot pass data to our App component, but our App component can pass state (or functions) to our Button components through the use of props.

To get both buttons to display the same number, we can then "lift" the timesClicked-state up from the buttons, and into our App component. Notice that a downside of this, is that our App-component now has to care about the state of its buttons.

```
function App() {
  const [timesClicked, setTimesClicked] = React.useState(0)
  return (
    <div style={{ backgroundColor: 'blue' }}>
      <Button
        color='red'
        timesClicked={timesClicked}
        setTimesClicked={setTimesClicked}
      />
      <Button
        color='black'
        timesClicked={timesClicked}
        setTimesClicked={setTimesClicked}
      />
    </div>
  )
}

function Button({ color, timesClicked, setTimesClicked }) {
  return (
    <button style={{ color }} onClick={() => setTimesClicked(timesClicked + 1)}>
      I have been clicked {timesClicked} times!
    </button>
  )
}
```

If we want all our application state to be made accessible to all our components, we could lift all the state up to a wrapper component around our App. One problem with this is that a lot of props still has to be spread downwards, and usually we don't want intermediate components to be concerned with the state of its children.

Is there a way we could pass props through a component to its child, without this component in the middle knowing anything about it? This is known as prop drilling. (Dodds 2018)

4.5 React Context and React-Redux

React Context specifically aims at fixing the prop drilling problem, where you can pass data to a component not directly below you, without touching the components in between.

react-redux (Abramov 2015) takes this a step further, and embraces the idea of lifting all state to the top of the application. This means that components are stateless, with the state simply "flowing down" into all the components below.

An advantage of this is that a component can be removed, and then later readded to the virtual DOM, without any data being lost. Furthermore, you could persist the entire state of your application using the `localStorage` API (Mozilla 2020) in your browser, so that the next time you open the website, it will look exactly the way you left it. If you keep track of how the state has changed, you could also "travel back in time" to an earlier state in your application. (Abramov 2020)

Essentially, Redux allows you to isolate and centralize the state that components depend on. A component will only be responsible for displaying this state, or for issuing (dispatching) commands (known as actions) to the Redux store. The dispatched action will then be combined with the current state of the application to produce a new state.

A function that takes in the current state, an action, and produces a new state is called a reducer in React-Redux. It is important to emphasise that a reducer never modifies the current state, but instead creates a new modified copy. This allows us to track exactly how the state is changing, and what is causing the changes - keeping us reasonably sane while working with a huge centralized store. It is also safe to take a shallow copy of the state, without worrying it will change below your feet.

Once your application state grows above a certain size, it might be desirable to split it into "slices" that manage smaller parts of the entire state. The library redux-toolkit (Erikson 2018) allows for doing this, along with providing other useful tools for working with React-Redux.

4.5.1 Immer

Immer - "Create the next immutable state by mutating the current one" (Weststrate 2017) is a JavaScript library, which won "Breakthrough of the year" (OS Awards React 2019) and "Most impactful contribution" (OS Awards JavaScript 2019) despite being a very simple library. It exposes one function:

```
import produce from 'immer'
// both nextState and currentState are immutable (cannot be changed)
const nextState = produce(currentState, (draftState) => {
  // draftState is mutable, and can be operated on imperatively within this function
})
```

Immer is used in redux-toolkit to allow using seemingly imperative modifications to the state, while still keeping the state immutable.

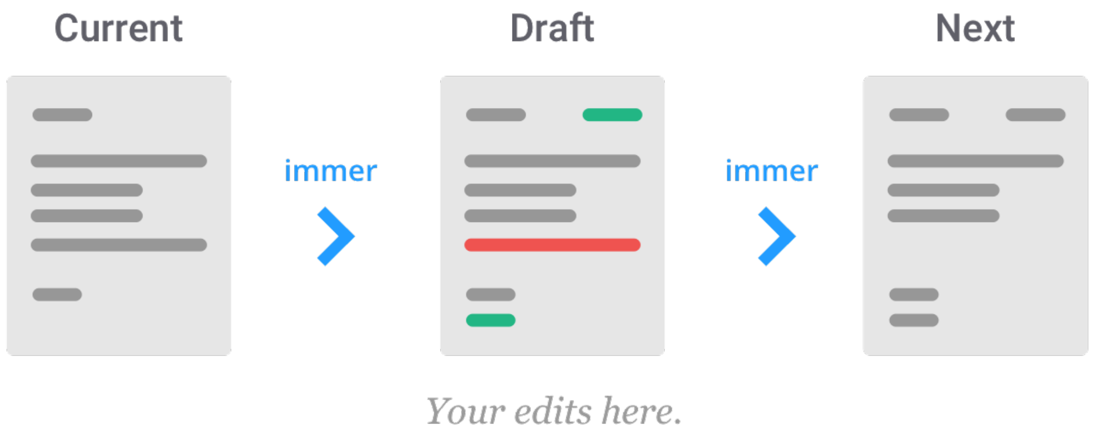


Figure 5: Illustration of immer produce function (Weststrate 2017)

4.6 BlueprintJS and React Mosaic

One of the biggest advantages of React making it easy to reuse custom components, is that it is also easy to reuse components created by other people. As a consequence there has been a rise of component libraries for React, like Material UI (Hai Nguyen 2014), Semantic UI React (Jack Lukic 2013), BlueprintJS (Palantir 2016), Ant-design components (afc163 2015) and many more.

Modifying styling might be one of the most time-consuming parts of creating a custom component - one which you will no longer need to think about when using one of these component libraries. Productivity is massively increased - as you can just throw some already good looking components together to get what you need. And importantly, you can do this without losing any of the flexibility you'd get compared to making it from scratch.

BlueprintJS is optimized for building complex, data-dense web interfaces for desktop applications. This is perfect for our use case (data visualization). Another potentially time-consuming part of developing a user interface, is figuring out how components should be placed around the screen.

To further relieve mental effort, and in turn increase productivity - we can use React Mosaic (Verdieck 2017), which is a tiling window manager for React within the browser. This allows us to put groups of components together, without needing to think about the bigger picture of how to entire application should look. This tiling window manager is also already using the BlueprintJS styles, meaning the two libraries will look seamless together.

5 Background: CIM format and Geospatial data

In this chapter, we will delve into some older systems and concepts - related to how power grid data is represented as data, and how traditional as well as more modern geographic information systems work.

5.1 The CIM export data

In Norway, there are a lot of different grid companies. In general, they all use something known as CIM (Common Information Model) to keep track of the assets in their grid - including their properties and location.

Since these assets are often older than the digital systems, and physical records were used to keep track of them, they have at some point been manually entered into the grid management software, which is an error-prone process. The grid data might also not be optimized for running powerflow calculations.

5.1.1 CIM XML example

Powel NETBAS has been market leading in the grid management industry for more than 30 years (Powel n.d.). This is also what practically every grid company in Norway is using to keep track of their power grid. NETBAS can export a network on a CIM format ("Common Information Model" n.d.).

This is a fairly verbose and not exactly compact data format. In the example below, you can see some information (with scrambled names and coordinates) about a single substation.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF xmlns:cim="http://iec.ch/TC57/2010/CIM-schema-cim15#"
         xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <cim:Substation rdf:ID="_8f701089-1e66-4eca-b0d8-817cf983bdbe">
    <cim:IdentifiedObject.mRID>8f701089-1e66-4eca-b0d8-817cf983bdbe</cim:IdentifiedObject.mRID>
    <cim:PowerSystemResource.Location rdf:resource="#_b8f068fb-2d33-40aa-8c9f-f941f9dd380c" />
    <cim:PowerSystemResource.PSRType rdf:resource="#_8859169a-3779-47b1-b9d4-b41a1b82ef9f" />
    <cim:IdentifiedObject.name>qeaxscdrsk</cim:IdentifiedObject.name>
    <cim:IdentifiedObject.alias>toekdqoavx</cim:IdentifiedObject.alias>
    <cim:IdentifiedObject.objectNumber>11099814</cim:IdentifiedObject.objectNumber>
  </cim:Substation>
  <cim:Location rdf:ID="_b8f068fb-2d33-40aa-8c9f-f941f9dd380c">
    <cim:IdentifiedObject.mRID>b8f068fb-2d33-40aa-8c9f-f941f9dd380c</cim:IdentifiedObject.mRID>
    <cim:Location.CoordinateSystem rdf:resource="#_24b5e8cf-17bf-4e77-acf9-c91096612d02" />
    <cim:Location.mainAddress />
  </cim:Location>
  <cim:PositionPoint rdf:ID="_9c8223c3-62d0-4c70-8606-e1dd7088b9da">
    <cim:PositionPoint.Location rdf:resource="#_b8f068fb-2d33-40aa-8c9f-f941f9dd380c" />
    <cim:PositionPoint.xPosition>375097</cim:PositionPoint.xPosition>
    <cim:PositionPoint.yPosition>691907</cim:PositionPoint.yPosition>
    <cim:PositionPoint.sequenceNumber>1</cim:PositionPoint.sequenceNumber>
  </cim:PositionPoint>
  <cim:PSRType rdf:ID="_8859169a-3779-47b1-b9d4-b41a1b82ef9f">
    <cim:IdentifiedObject.mRID>8859169a-3779-47b1-b9d4-b41a1b82ef9f</cim:IdentifiedObject.mRID>
    <cim:IdentifiedObject.name>emmmdsrfek</cim:IdentifiedObject.name>
  </cim:PSRType>
    <cim:CoordinateSystem rdf:ID="_24b5e8cf-17bf-4e77-acf9-c91096612d02">
    <cim:IdentifiedObject.mRID>24b5e8cf-17bf-4e77-acf9-c91096612d02</cim:IdentifiedObject.mRID>
    <cim:CoordinateSystem.crsUrn>urn:ogc:def:crs:EPSG::32632</cim:CoordinateSystem.crsUrn>
  </cim:CoordinateSystem>
  <cim:VoltageLevel rdf:ID="_73a13e03-b240-4952-9d6b-93951dacc888">
    <cim:IdentifiedObject.mRID>73a13e03-b240-4952-9d6b-93951dacc888</cim:IdentifiedObject.mRID>
    <cim:VoltageLevel.Substation rdf:resource="#_8f701089-1e66-4eca-b0d8-817cf983bdbe" />
    <cim:VoltageLevel.BaseVoltage rdf:resource="#_c49f941a-33fa-4fc1-a28b-b3e269f27cab" />
    <cim:IdentifiedObject.name>230 V</cim:IdentifiedObject.name>
  </cim:VoltageLevel>
  <cim:BaseVoltage rdf:ID="_c49f941a-33fa-4fc1-a28b-b3e269f27cab">
    <cim:IdentifiedObject.mRID>c49f941a-33fa-4fc1-a28b-b3e269f27cab</cim:IdentifiedObject.mRID>
    <cim:BaseVoltage.nominalVoltage>230</cim:BaseVoltage.nominalVoltage>
  </cim:BaseVoltage>
</rdf:RDF>
```

You might have noticed that the CIM format heavily uses references to describe relationships in data (to a quite excessive degree). This is also used for describing connectivity. In this particular example, there are no conducting equipment types - so this data cannot be used for any powerflow calculations.

Too see all the different CIM classes, and which properties they have - check out the Appendix section at the very end of this report. Is there a way we could represent the CIM data more compactly, and with less references?

5.1.2 The same data represented as GeoJSON

GeoJSON is commonly used to represent geospatial information. (Howard Butler 2016) Since the CIM data contains positions and coordinates for all the assets, this seems like an appropriate choice. We are also able to shed a lot of references in the process.

```json
{
  "type": "FeatureCollection",
  "crs": {
    "type": "EPSG",
    "properties": {
      "code": 32632
    }
  },
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "Point",
        "coordinates": [375097, 691907]
      },
      "properties": {
        "id": "8f701089-1e66-4eca-b0d8-817cf983bdbe",
        "type": "Substation",
        "name": "qeaxscdrsk",
        "alias": "toekdqoavx",
        "objectNumber": 11099814,
        "PSRType": "emmmdsrfek",
        "baseVoltage": 230
      }
    }
  ]
}
```

5.1.3 An even more compact alternative (using CSV and EWKT)

Substations.csv

```
id,name,alias,objectNumber,PSRType,baseVoltage,geometry
8f701089-1e66-4eca-b0d8-817cf983bdbe,qeaxscdrsk,toekdqoavx,11099814,emmmdsrfek,230,\
    SRID=32632;POINT(375097,691907)
```

Notice that csv-files do not support escaping newlines. Since not everything would fit onto the page otherwise, a backslash ("\") along with indent on the following line is used to signify that this is a continuation of the above line. This would not be valid syntax in practice

---

A consequence of this approach is that we need a separate file for every type of asset. CSV is short for Comma Separated Values. We can represent the geometry using the Extended Well-Known Text (EWKT) format. (Santilli 2005)

Some further optimizations we could make on this:

- Store numbers as binary data instead of text
- Use EWKB (the binary equivalent of EKWT) instead of EWKT to represent the geometry even more compactly.
- Both of the improvements above could be done automatically using something like PostgreSQL with the PostGIS extension.
- Another option, is to use the columnar storage format Apache Parquet (Apache 2013) for compact representation.
- For a more inflated format - which would be able to load faster into memory, we have Apache Arrow. (Apache 2016)

5.2 The different CIM classes and their properties

You have seen a few CIM classes already. For a complete overview of the different CIM classes and their properties, look at the Appendix section at the end of the report.

5.3 Reprojecting to longitude and latitude

Every geospatial point must be specified according some kind of coordinate system. The most common one might be EPSG:4326, also known as “longitude-latitude”. These coordinate systems differ in projection and valid regions. Using a coordinate system that is only valid within a specific time zone, one could in theory store coordinates with the same precision a bit more compactly. (Or achieve greater precision with the same storage).

Reprojecting a lot of points from one coordinate system to another can be computationally expensive. Potentially being very bad for performance when done in “real-time” as someone navigates around a map. Because of this, it is generally better to reproject in advance - storing the points in the same EPSG-projection they will be displayed with.

Since we will be displaying in EPSG:4326, and our CIM-files are specifying points in EPSG:32632, we will want to perform this reprojection as part of our data preparation process. The data should already have been reprojected before a client sends a data request to the server.

5.4 Transfer formats for map data

When you want a server to communicate map data to a client, it is an advantage if there is some kind of underlying protocol for geospatial data that both the server and client agree on. Now we will look at some of these protocols.

5.4.1 WMS (Web Map Service)

A WMS service receives the map viewport of a client, and responds with data corresponding to everything within this viewport. Typically this can be an image of the background map. This is not typically used, since dynamically generating images for every request quickly becomes expensive.

5.4.2 WMTS (Web Map Tile Service)

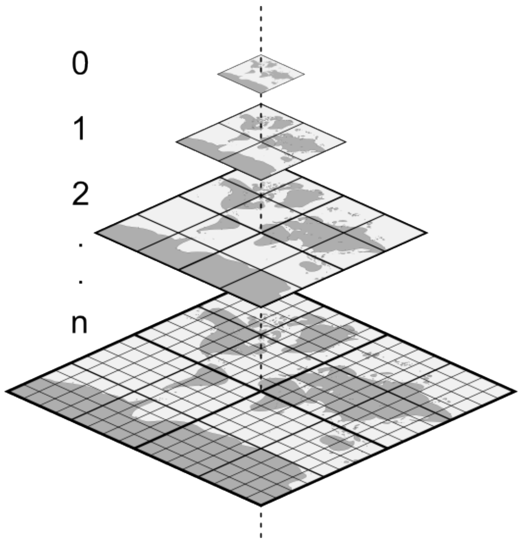


Figure 6: Illustration of Web Map Tile Service (Nedkov 2015)

Since serving static images is a lot faster than generating images on the fly, WMTS was created. Instead of mapping one viewport to one image, WMTS divides a map into pregenerated images/“tiles” at different zoom levels/coordinates. These will typically be 256x256 pixels. In contrast to WMS, this allows the tiles to be prepared beforehand instead of generated on the fly. The tiles can also be cached for reuse in the client.

Based on the viewport, the client will know which tiles to ask for. The server doesn’t need to know anything about the viewport in the case of WMTS. This is the “de-facto” standard used by most online map APIs.

5.4.3 WFS (Web Feature Service)

WMS and WMTS are generally used to serve “raster” data. (Typically jpeg or png images). What if you want to serve vector features? With WFS you can request vector features and display them on top of the map. Here you can for example request all the vector features within your viewport, or a specific feature based on its ID.

5.4.4 MVT (Mapbox Vector Tile)

Bandwidth is usually the bottleneck of any modern web application, meaning it is important to minimize the amount of data transfer between server and client. Mapbox has created a format “Mapbox Vector Tile” which delivers a binary/compressed alternative to WFS with the purpose of minimizing data transfer. (Mapbox 2014)

MVT is based on the same principles as WMTS, allowing tiles to be pregenerated as well as cached in the client. The transfer format is specified using Google’s protocol buffers (Google 2009).

6 Method: Selecting a software for modelling the grid

6.1 Motivation

In order to avoid a lot of work, and to minimize our surface of where things can go wrong, we will be using an already existing software solution for modelling our grids. This means a couple of things:

- We won't have to implement Newton-Raphson and can take advantage of optimizations in the software.
- We won't have to set up the system equations, as this is done automatically by the software solution.

6.2 Qualities to look for in a software solution

- Has an engaged community and is being actively maintained
- Is easy to use, extend and automate
- Is well documented
- Is free to use
- Is thoroughly tested to make sure the powerflow results are correct

6.3 Goals

What is the point of running a power flow simulation? What can this tell us about the grid that we don't already know?

6.3.1 Stability

Simulations can predict stability of grid. If the grid approaches stability margins, we can get something known as voltage collapse. Voltage collapse has been the cause of several large-scale blackouts. (John W. Simpson-Porco 2016) A voltage collapse typically coincides with divergence of the Newton-Rhapson method.

6.3.2 Capacity (current/power)

Another type of problem is overheating of lines or cables/power transformers due to high current. Typically, lines/cables will have a maximum rated current, specifying how much current the equipment can handle without overheating. By running simulations, we can predict which lines/cables/power transformers might get overloaded and when it happens.

6.3.3 Voltage

According to the "Regulation on Supply Quality in the Power System" (Norwegian oil and energy department 2004), the slow-changing component of the voltage should not deviate by more than 10%. We can use simulations to predict when this might happen.

6.3.4 Short circuit

The big idea of a short circuit current is that it should be big enough to cause the circuit breaker switches to flip, but small enough not to destroy them. In general, the short circuit current should be above 800 Amperes.

6.4 pandapower

pandapower is "An easy to use open source tool for power system modeling, analysis and optimization with a high degree of automation." (Thurner et al. 2018). More specifically, it is a Python-library with a permissive license - actively developed by PhD students in Kassel, Germany, as well as other third party contributors.

The name "pandapower" is inspired by the library pandas, which is a foundational Python library for data analysis and statistics (McKinney 2010),(McKinney 2011). It is used to represent all the grid components, and calculation results in pandapower.

pandapower will be our software of choice for "forgetting about the phasor equations and Newton-Raphson". Since this is a Python library - the model is constructed by writing Python-code. We will look at a concrete example now.

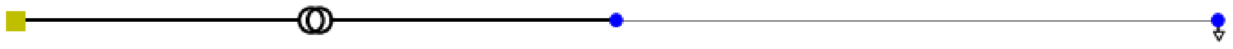
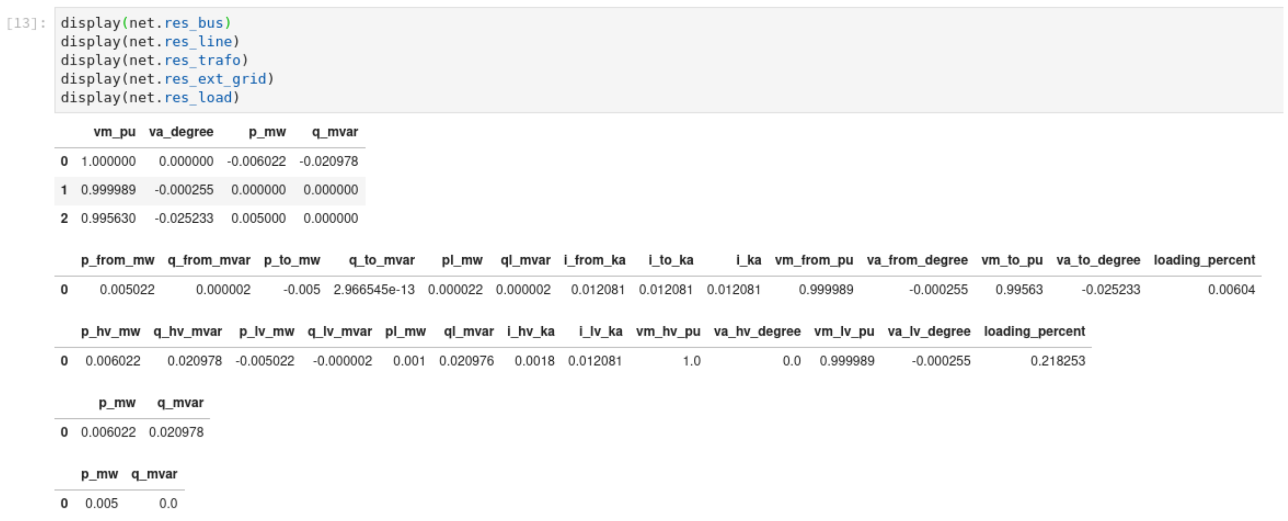


Figure 7: `pandapower.plotting.simple_plot(net, plot_loads=True)`

6.5 pandapower example

```python
import pandapower as pp
net = pp.create_empty_network()

# Buses serve as the skeleton of the network - through which everything is connected.
b0 = pp.create_bus(net, vn_kv=7.00)
b1 = pp.create_bus(net, vn_kv=0.24)
b2 = pp.create_bus(net, vn_kv=0.24)

# The ext_grid/slack bus is our AC signal generator.
# This is often used to represent the connection to the transmission grid.
pp.create_ext_grid(net, bus=b0)

# A basic power transformer
pp.create_transformer_from_parameters(net, hv_bus=b0, lv_bus=b1, sn_mva=10,
  vn_hv_kv=7.00, vn_lv_kv=0.24, vkr_percent=0.1, vk_percent=1, pfe_kw=1, i0_percent=0.21)

# An AC transmission line (using the "pi-model")
pp.create_line_from_parameters(net, from_bus=b1, to_bus=b2, length_km=0.1,
  r_ohm_per_km=0.5, x_ohm_per_km=0.05, c_nf_per_km=15, max_i_ka=200)

# A customer with a 5kW consumption
pp.create_load(net, bus=b2, p_mw=5e-3)

print(net) """
This pandapower network includes the following parameter tables:
   - bus (3 elements)
   - load (1 element)
   - ext_grid (1 element)
   - line (1 element)
   - trafo (1 element)
"""

# Running a powerflow calculation (newton-raphson) is very simple:
pp.runpp(net)

print(net) """
This pandapower network includes the following parameter tables:
   - bus (3 elements)
   - load (1 element)
   - ext_grid (1 element)
   - line (1 element)
   - trafo (1 element)
 and the following results tables:
   - res_bus (3 elements)
   - res_line (1 element)
   - res_trafo (1 element)
   - res_ext_grid (1 element)
   - res_load (1 element)
"""
```

By now, you might be a bit blown back by all the different parameters and names. Based on the pandapower documentation, we will now go through what the input parameters mean. In the next section, we will look at how to turn our CIM files into a valid pandapower model. This turns out to be more challenging than you might



```
[13]: display(net.res_bus)
      display(net.res_line)
      display(net.res_trafo)
      display(net.res_ext_grid)
      display(net.res_load)
```

	vm_pu	va_degree	p_mw	q_mvar
0	1.000000	0.000000	-0.006022	-0.020978
1	0.999989	-0.000255	0.000000	0.000000
2	0.995630	-0.025233	0.005000	0.000000

	p_from_mw	q_from_mvar	p_to_mw	q_to_mvar	pl_mw	ql_mvar	i_from_ka	i_to_ka	i_ka	vm_from_pu	va_from_degree	vm_to_pu	va_to_degree	loading_percent
0	0.005022	0.000002	-0.005	2.966545e-13	0.000022	0.000002	0.012081	0.012081	0.012081	0.999989	-0.000255	0.99563	-0.025233	0.00604

	p_hv_mw	q_hv_mvar	p_lv_mw	q_lv_mvar	pl_mw	ql_mvar	i_hv_ka	i_lv_ka	vm_hv_pu	va_hv_degree	vm_lv_pu	va_lv_degree	loading_percent
0	0.006022	0.020978	-0.005022	-0.000002	0.001	0.020976	0.0018	0.012081	1.0	0.0	0.999989	-0.000255	0.218253

	p_mw	q_mvar
0	0.006022	0.020978

	p_mw	q_mvar
0	0.005	0.0

Figure 8: Displaying the result tables in jupyter lab

think, so this is something to look forward to.

6.5.1 bus

Acts as the skeleton of the network. Two lines that want to connect together must do so through a bus. The same is true for all other electrical equipment. The reference voltage level is also defined at the bus level.

input	description
vn_kv	Reference voltage level (kV)

6.5.2 line

Represents a conductor (a line or cable). It has the following parameters in pandapower:

input	description
from_bus	id of bus to connect to
to_bus	id of a second bus to connect to
length_km	length of line in km
r_ohm_per_km	Resistance per km in Ohms
x_ohm_per_km	Reactive impedance per km in Ohms
c_nf_per_km	Line capacitance in nanofarad per km
max_i_ka	Rated current in kA

6.5.3 load

Represents a customer on the grid (consuming power)

input	description
bus	id of bus to connect to
p_mw	Active power consumption
q_mvar	Reactive power consumption

6.5.4 trafo

Represents a 2-winding power transformer on the grid. Note that you can specify voltages here that differ from the connected bus voltage. This allows for outputting voltage that is above or below reference voltage to compensate for consumption or production.

input	description
hv_bus	id of bus on HV-side to connect to
lv_bus	id of bus on LV-side to connect to
sn_mva	Rated apparent power
vn_hv_kv	Reference voltage on HV-side
vn_lv_kv	Reference voltage on LV-side
vkr_percent	Real part of relative short-ciruit voltage
vk_percent	Absolute value of relative short-circuit voltage
pfe_kw	Iron losses (in kW)
i0_percent	Open loop losses in percent of rated current

6.5.5 trafo3w

Represents a 3-winding power transformer on the grid.

input	description
hv_bus	id of bus on HV-side to connect to
mv_bus	id of bus on MV-side to connect to
lv_bus	id of bus on LV-side to connect to
vn_hv_kv	Reference voltage on HV-side
vn_mv_kv	Reference voltage on MV-side
vn_lv_kv	Reference voltage on LV-side
sn_hv_mva	Rated apparent power
sn_mv_mva	Rated apparent power
sn_lv_mva	Rated apparent power
vk_hv_percent	Absolute value of relative short-circuit voltage
vk_mv_percent	Absolute value of relative short-circuit voltage
vk_lv_percent	Absolute value of relative short-circuit voltage
vkr_hv_percent	Real part of relative short-circuit voltage
vkr_mv_percent	Real part of relative short-circuit voltage
vkr_lv_percent	Real part of relative short-circuit voltage
pfe_kw	Iron losses (in kW)
i0_percent	Open loop losses in percent of rated current

6.5.6 switch

Represents an ideal switch with no impedance.

input	description
bus	The id of a bus to connect to
et	Element type of the element to connect to (b=bus, l=line, t=trafo)
element	The id of the element to connect to (usually a bus)
closed	True if the switch is closed, False if the switch is open.

6.5.7 ext_grid

Represents the slack bus/external grid. This is simply a mathematical construct used when running powerflow computations. It's purpose is to provide or absorb active and reactive power from the electrical grid.

input	description
bus	The id of a bus to connect to
vm_pu	The provided voltage in "per unit" compared to reference voltage

7 Method: From CIM to a (converging) pandapower model

This section is the longest one. It is also where a lot of the interesting challenges and insights are hiding. If we are going to turn the CIM files into a pandapower-model, we will first need to somehow read them into memory.

7.1 Parsing/Reading the CIM-data into computer memory

7.1.1 Python ElementTree API vs lxml

In order to parse the CIM-files, we can use an the ElementTree API in Python (Python 2005). However, since parsing large XML-files is a particularly slow activity in Python, this can be sped up by instead using lxml, which is a Python binding for the C-library libxml2 - providing around twice the parsing speed, and a nearly identical API to the the Python ElementTree API. (Faassen 2005)

7.1.2 CimPandas

For our in-memory representation, it is natural to use pandas DataFrames, since this is how the data will eventually be stored in pandapower anyways. For each CIM-class, we will create one dataframe, and organize all the data frames in a Python dictionary. We can create a wrapper class around the data structure with some convenience methods, and the ability to access the dictionary keys as properties.

This class will be named CimPandas. It will be responsible for storing the in-memory representation of dataframes, as well as building it from one or more XML files. For very large amounts of XML data, the parsing process can take more than one hour to complete.

To avoid having to do this every time, we can use Apache Parquet (Apache 2013) as an intermediate storage format to store each dataframe in one file. The result is that it takes 30 seconds to read everything into memory instead of one hour - and it uses around 10% of the storage space of the XML files. In other words, this is a significant improvement.

The CimPandas class provides the following interface for reading XML and parquet files, as well as writing parquet files:

```
# Read a single CIM file into memory
dfs = CimPandas.from_xml('./some-folder/some-file.xml')

# Read multiple CIM files into memory
dfs = CimPandas.from_xmls('./some-folder/')

# Read parquet files into memory
dfs = CimPandas.from_parquet('./parquet-files-folder/')

# Write to disk using the Apache Parquet format
dfs.to_parquet('./parquet-files-folder')

# If you try printing out dfs, you will see something like the following
# Note that the numbers/counts below are made up (but reasonable) to demonstrate
# the amount of different component types you might find
print(dfs) """
"some-cim" contains the following DataFrames:
 - ACLineSegment (150,000)
 - BaseVoltage (30)
 - Bay (140,000)
 - Breaker (2000)
 - BusbarSection (20,000)
 - ConnectivityNode (300,000)
 - CoordinateSystem (1)
 - Disconnector (3000)
 - EnergyConsumer (80,000)
 - Fuse (60,000)
 - GroundDisconnector (1000)
 - Jumper (90,000)
 - LoadBreakSwitch (9000)
 - Location (350,000)
 - PositionPoint (1,500,000)
 - PowerTransformer (4000)
 - PowerTransformerEnd (8100)
 - PSRType (7)
 - RatioTapChanger (3500)
 - Substation (20,000)
 - Terminal (750,000)
 - UsagePoint (150,000)
 - VoltageLevel (22,000)
[time loaded: 2020-06-01 00:00:00.000000]
"""

# Access a specific DataFrame
dfs.BaseVoltage

# Get a mapping between class id and a property
dfs.BaseVoltage.nominalVoltage
```

7.2 A mapping between CIM classes and pandapower elements

If we are going to build a pandapower model from this, it would be useful to have some idea of which CIM classes correspond to which pandapower elements. A minimal overview can be seen in the table below.

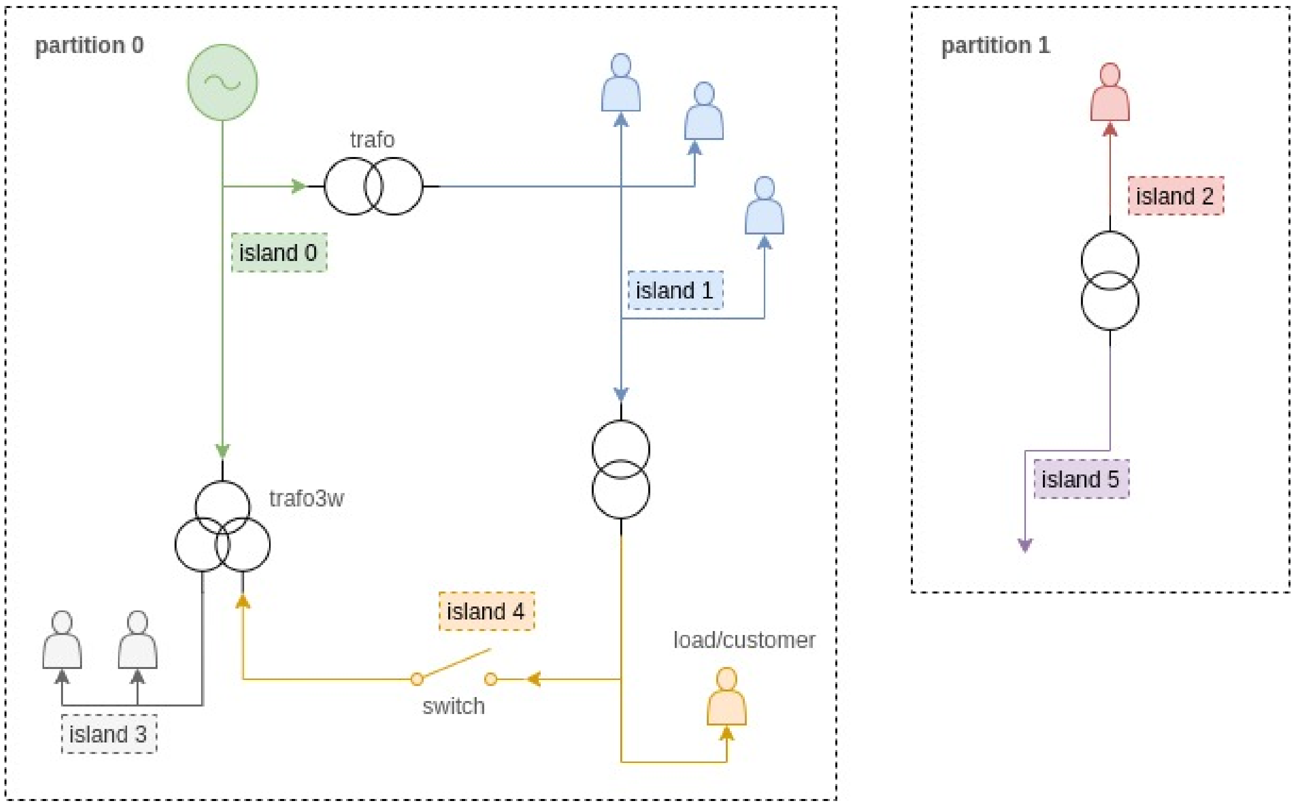
pandapower element	CIM class
bus	ConnectivityNode
line	ACLineSegment
switch	GroundDisconnector, Fuse, Disconnector, LoadBreakSwitch, Breaker, Jumper
trafo	PowerTransformer, PowerTransformerEnd
trafo3w	PowerTransformer, PowerTransformerEnd
load	EnergyConsumer, UsagePoint
ext_grid	-

As you might be able to notice, the CIM format doesn't tell us anything about where the ext_grid should be placed. CIM also doesn't have a direct relationship between for example ConnectivityNode and ACLineSegment. Instead, there are Terminals in the middle.

Not having a direct one-to-one relationship between pandapower and CIM makes this a somewhat challenging process.

7.3 Building the pandapower model

In pandapower, voltage is specified per bus. The buses have a one-to-one relationship with ConnectivityNode in the CIM data. It is important that the voltage level is the same for all buses within an island. We will now define two new concepts: (topological) islands and partitions.



We define an **island** to be:

- A subset of the grid that shares the same reference voltage
- Directly interconnected (excluding power transformers)
- Unaffected by switches opening or closing

Purpose: define regions of shared reference voltage

We define a **partition** to be:

- A subset of the grid that is interconnected (including power transformers).
- Unaffected by switches opening or closing.
- A superset of one or more islands.

Purpose: detect disconnected grid data

As you can see from the image, islands are separated by power transformers. Partitions are separated by grids being electrically independent/not connected in any way. An island can only belong to one partition, but a partition can contain many islands. An open switch does not split up an island or a partition.

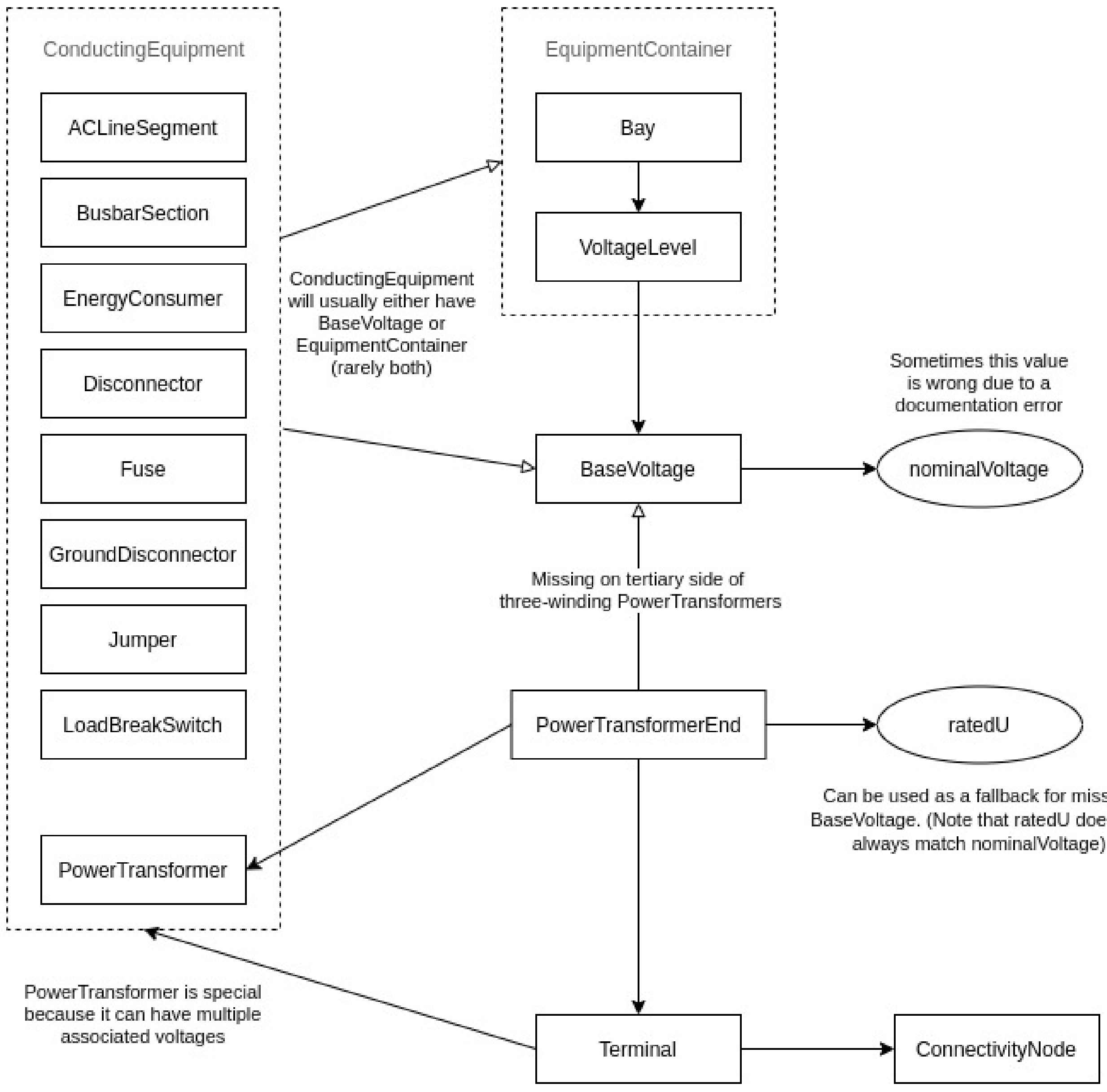
7.3.1 Finding the reference voltage for a voltage island

Turns out that the exported grid on the CIM format can sometimes disagree on voltage. To work around this, we will essentially let everything within an island “vote” on what the reference voltage is - and then the median of these votes will be used.

To be able to “collect” these votes, we need to follow a lot of references in the CIM data. We will also need to perform topological analysis to find out which ConnectivityNodes/buses are within the same island. To perform this topological analysis, we can use the builtin NetworkX support in pandapower.

Let’s take one step at a time, and start looking at how we can find a mapping between ConnectivityNodes/buses and voltage.

7.3.2 The challenge of mapping a voltage to each bus/ConnectivityNode



ConductingEquipment
ACLineSegment
BusbarSection
EnergyConsumer
Disconnector
Fuse
GroundDisconnector
Jumper
LoadBreakSwitch
PowerTransformer
EquipmentContainer
Bay
VoltageLevel
ConductingEquipment will usually either have BaseVoltage or EquipmentContainer (rarely both)
Sometimes this value is wrong due to a documentation error
BaseVoltage
nominalVoltage
Missing on tertiary side of three-winding PowerTransformers
PowerTransformerEnd
ratedU
Can be used as a fallback for missing BaseVoltage. (Note that ratedU does not always match nominalVoltage)
PowerTransformer is special because it can have multiple associated voltages
Terminal
ConnectivityNode

In the image above, an arrow with a hollow head represents a reference that is not guaranteed to be present. Dotted rectangles represent a hidden parent class that the classes inside inherit from. The ellipsoids represent values (instead of references).

Mapping voltages to the ConnectivityNodes is a stepping stone in the process of assigning a single voltage to each island. A ConnectivityNode might not have any Terminals pointing at it. Even if it does, it might not always be possible to assign a voltage to it. A single ConnectivityNode could also have multiple voltages associated with it - which in practice might not match (due to errors in the data). Voltage values of 0 will be considered as “missing” values, and not counted in the voting process.

How can we express these relationships declaratively with code, to assign one or more possible voltages to as many of the buses/ConnectivityNodes as possible? Let’s look at some more programming concepts that might be useful to us.

7.3.3 Infix operators and reductions in Python

What is an infix operator and why would we care? Examples of infix operators in most programming languages are the mathematical operators (+, -, *, /). In contrast to a prefix operator, they appear between the arguments instead of in front of them. One advantage of infix operators is that they are a bit more intuitive to humans, likely due to how they are heavily used in mathematics. It also allows for cleaner syntax when chaining multiple values.

```
def add(x, y):
  return x + y

# addition as a prefix operation
add(add(add(1,2), 3), 4)

# `+` is a infix operator because it appears between the arguments
1 + 2 + 3 + 4
```

Python allows for creating custom classes that overload the mathematical operators. In languages like haskell, any function can be used as an infix operator by surrounding it with backticks.

```
# Haskell
add x y = x + y
1 `add` 2 `add` 3 `add` 4
```

What if we could do something similar in Python? It turns out we can define our own infix functions using a little trick (based on operator overloading):

```
class Infix
  def __init__(self, function):
      self.function = function
  def __rlshift__(self, other):
    return Infix(lambda x, self=self, other=other: self.function(other, x))
  def __rshift__(self, other):
    return self.function(other)
  def __call__(self, value1, value2):
    return self.function(value1, value2)

@Infix
def add(x, y):
  return x + y

1 <<add>> 2 <<add>> 3 <<add>> 4
```

Notice that the chaining looks a bit cleaner and easier to understand for a human than the prefix alternative. This type of chaining has a more formal name in programming: `reduce`. It is actually this chaining we are interested in, rather than the infix operations themselves. We have only looked at infix functions for the intuition they provide in understanding how a reduction works. A reduction in Python can be done as follows:

```
from functools.partial import reduce
reduce(add, [1, 2, 3, 4]) == 1 <<add>> 2 <<add>> 3 <<add>> 4
```

We can extend our Infix-class to include a reduce-property for convenience:

```
from functools.partial import reduce
class Infix
  def __init__(self, function):
      self.function = function
  def __rlshift__(self, other):
    return Infix(lambda x, self=self, other=other: self.function(other, x))
  def __rshift__(self, other):
    return self.function(other)
  def __call__(self, value1, value2):
    return self.function(value1, value2)
  # Adds a reduce-property to our function
  def reduce(self, values):
    return reduce(self.function, values)

@Infix
def add(x, y):
  return x + y

add.reduce([ 1, 2, 3, 4 ]) == 1 <<add>> 2 <<add>> 3 <<add>> 4
```

7.3.4 Manage relationships in data declaratively

We can define a “map” to be a mapping between two data columns `A` and `B` (which we represent using a pandas DataFrame/Series in Python). It is possible to have duplicate entries in both columns.

```
# Naming convention for a mapping
<A>_<B>_map = ...

# Example
import pandas as pd
```

```
house_paint_map = pd.DataFrame(data={
  'house': [ 1    ,  2    ,  3     ,  4      ,  5   ],
  'paint': ['red', 'blue', 'green', 'yellow', 'red']
}).set_index('house').paint
```

```
+----------------+
|          paint |
| house          |
| 1          red |
| 2         blue |
| 3        green |
| 4       yellow |
| 5          red |
+----------------+
```

```
paint_price_map = pd.DataFrame(data={
  'paint': [ 'red', 'blue', 'green', 'yellow' ],
  'price': [  1000,  5000 ,  2000  ,  3000    ]
}).set_index('paint').price
```

```
+----------------+
|          price |
| paint          |
| red       1000 |
| blue      5000 |
| green     2000 |
| yellow    3000 |
+----------------+
```

Now that we see how mappings can be created, we would like to be able to manipulate these by defining some useful operations. Notice that we use the infix notation we previously discussed below.

For a pandas DataFrame/Series `df` and function `fn`, we have that `df.pipe(fn) == fn(df)`. The pipe method is useful because it allows us to chain multiple functions in a syntactially pleasing, and chronologically ordered way.

```
# A mapping can be reversed/flipped around:
# house_color.pipe(reverse)      ==   color_house
+----------------+                    +----------------+
|          color |                    |          house |
| house          |                    | color          |
| 1          red |                    | red          1 |
| 2         blue |.pipe(reverse)  == | blue         2 |
| 3        green |                    | green        3 |
| 4       yellow |                    | yellow       4 |
| 5          red |                    | red          5 |
+----------------+                    +----------------+

# <<traverse>> works a bit like an inner join in SQL.
# The column used for joining is dropped in the process.
# house_paint       <<traverse>>    paint_price      ==    house_price
+----------------+                +----------------+      +----------------+
|          paint |                |          price |      |          price |
| house          |                | paint          |      | house          |
| 1          red |                | red       1000 |      | 1         1000 |
| 2         blue |  <<traverse>>  | blue      5000 |  ==  | 5         1000 |
| 3        green |                | green     2000 |      | 2         5000 |
| 4       yellow |                | yellow    3000 |      | 3         2000 |
| 5          red |                +----------------+      | 4         3000 |
+----------------+                                        +----------------+

# <<concat>> can append/concatenate mappings of the same type together
# house_paint       <<concat>>   house_paint        ==   house_paint
+----------------+              +----------------+     +----------------+
|          paint |              |          paint |     |          paint |
| house          |              | house          |     | house          |
| 1          red |  <<concat>>  | 4       yellow |  == | 1          red |
| 2         blue |              | 5          red |     | 2         blue |
| 3        green |              +----------------+     | 3        green |
+----------------+                                     | 4       yellow |
                                                       | 5          red |
                                                       +----------------+
```

7.3.5 Starting with a simplified example

To understand how these operations can be used as building blocks for combining references in the CIM format, we will start with a slightly simplified example. In this cause, Bay, VoltageLevel, PowerTransformer and PowerTransformerEnd are excluded.

`dfs` is our variable that stores the in-memory representation of the CIM data. The access pattern is `dfs.<asset-type>.<property>`. The `.dropna()`-method will exclude missing/invalid values.

```python
# ConnectivityNode -> nominalVoltage
traverse.reduce([

    # ConnectivityNode -> Terminal
    dfs.Terminal.ConnectivityNode.dropna().pipe(reverse),
    # .dropna() drops np.nan-values (Missing values)

    # Terminal -> ConductingEquipment
    dfs.Terminal.ConductingEquipment.dropna(),

    # ConductingEquipment -> BaseVoltage
    concat.reduce([
        dfs.ACLineSegment.BaseVoltage.dropna(),
        dfs.BusbarSection.BaseVoltage.dropna(),
        dfs.EnergyConsumer.BaseVoltage.dropna(),
        dfs.Breaker.BaseVoltage.dropna(),
        dfs.Disconnector.BaseVoltage.dropna(),
        dfs.Fuse.BaseVoltage.dropna(),
        dfs.GroundDisconnector.BaseVoltage.dropna(),
        dfs.Jumper.BaseVoltage.dropna(),
        dfs.LoadBreakSwitch.BaseVoltage.dropna(),
    ]),

    # BaseVoltage -> nominalVoltage
    dfs.BaseVoltage.nominalVoltage
        .replace({ 0: np.nan }).dropna()  # We ignore 0s
])
```

How does this look when we consider Bay, VoltageLevel, PowerTransformer and PowerTransformerEnd as well?

7.3.6 The entire diagram translated into declarative code

```
def connectivitynode_voltage_map(dfs):
    return concat.reduce([
        traverse.reduce([
            dfs.Terminal.ConnectivityNode.dropna().pipe(reverse),
            dfs.Terminal.ConductingEquipment.dropna(),
            concat.reduce([
                dfs.ACLineSegment.BaseVoltage.dropna(),
                dfs.BusbarSection.BaseVoltage.dropna(),
                dfs.EnergyConsumer.BaseVoltage.dropna(),
                dfs.Breaker.BaseVoltage.dropna(),
                dfs.Disconnector.BaseVoltage.dropna(),
                dfs.Fuse.BaseVoltage.dropna(),
                dfs.GroundDisconnector.BaseVoltage.dropna(),
                dfs.Jumper.BaseVoltage.dropna(),
                dfs.LoadBreakSwitch.BaseVoltage.dropna(),
                traverse.reduce([
                    concat.reduce([
                        dfs.ACLineSegment.EquipmentContainer.dropna(),
                        dfs.BusbarSection.EquipmentContainer.dropna(),
                        dfs.EnergyConsumer.EquipmentContainer.dropna(),
                        dfs.Breaker.EquipmentContainer.dropna(),
                        dfs.Disconnector.EquipmentContainer.dropna(),
                        dfs.Fuse.EquipmentContainer.dropna(),
                        dfs.GroundDisconnector.EquipmentContainer.dropna(),
                        dfs.Jumper.EquipmentContainer.dropna(),
                        dfs.LoadBreakSwitch.EquipmentContainer.dropna(),
                    ]),
                    concat.reduce([
                        traverse.reduce([
                            dfs.Bay.VoltageLevel.dropna(),
                            dfs.VoltageLevel.BaseVoltage.dropna()
                        ]),
                        dfs.VoltageLevel.BaseVoltage.dropna()
                    ])
                ])
            ]),
            dfs.BaseVoltage.nominalVoltage
                .replace({0:np.nan}).dropna()
        ]),
        traverse.reduce([
            traverse.reduce([
                dfs.PowerTransformerEnd.Terminal,
                dfs.Terminal.ConnectivityNode
            ]).pipe(reverse),
            traverse.reduce([
                dfs.PowerTransformerEnd.BaseVoltage,
                dfs.BaseVoltage.nominalVoltage
                    .replace({0: np.nan}).dropna()
            ]).combine_first(
                dfs.PowerTransformerEnd.ratedU
                    .replace({0: np.nan}).dropna()
            )
        ])
    ])
```

7.3.7 Finding a mapping between buses/ConnectivityNodes and islands

We can use `pandapower.topology.create_nxgraph` to create a networkx graph. NetworkX is a graph processing library for Python that pandapower has created an integration against. This sounds great, but now it sounds like we need the network to have already been created before we can create a network. How can we get around this?

The trick is to create the network without setting any voltages. We set all the voltages to 0 for the time being, and correct their values later. Let's assume that this network has already been created, with only the voltage levels missing. Then we can define the island (and partition) mappings as follows.

```python
# Now, we can define island and partition, according to our earlier definition.
net.bus['island'] = topological_group(without_trafo(net), respect_switches=False)
net.bus['partition'] = topological_group(net, respect_switches=False)

def topological_group(net, **kwargs):
    return pd.Series([
        list(component_group)
        for component_group in pp.topology.connected_components(
            pp.topology.create_nxgraph(net, **kwargs))
    ]).explode().pipe(reverse).rename_axis('bus').rename('group')

def without_trafo(net):
    # This creates a copy, in order not to modify the passed in network
    n = net.deepcopy()
    n.trafo.in_service = False
    n.trafo3w.in_service = False
    return n
```

7.3.8 A single voltage level for each island

Now we want to find a single voltage for each island. To do this, while being as resilient as possible to errors in the data, we will use a voting process as previously discussed. Essentially, every ConnectivityNode / bus within an island will cast its voltage(s) as votes - and then the median of all the votes for that island will be selected as the island's reference voltage. As long as more than 50% of the voltages for an island agree, this exact voltage will always be picked.

A consequence of using a median, is that sometimes the average of two voltage levels might be chosen. If 0 votes are cast for any specific island, the voltage will simply be set to 0. A reason this might happen is that a grid company might enter grid they plan on building into their systems, without assigning any parameters to it. Regardless, this is something that has been observed in practice.

```python
# We divide by 1000, since pandapower wants the voltage in kV, not V
net.bus.vn_kv = 1e-3 * bus_voltage_map(net, dfs)
net.bus.vn_kv = net.bus.vn_kv.fillna(0)

def bus_voltage_map(net, dfs):
    return traverse.reduce([
        net.bus.island,
        traverse.reduce([
            net.bus.island.pipe(reverse).rename('bus'),
            net.bus.name,
            connectivitynode_voltage_map(dfs)
        ]).groupby(level=0).median()
    ])
```

7.3.9 Building the model

pandapower_extended.py contains several utility functions we have created for more efficiently working with pandapower. This includes the ability to add multiple components at the same time in a performant way.

```
import pandapower_extended as ppe

def create_network(dfs):
    net = pp.create_empty_network()
    pp.create_buses(net, **bus_params(dfs))
    t1, t2, trafoend = terminals(dfs, net.bus.name)

    net.line    = ppe.append_line(net, **line_params(dfs, t1, t2))
    net.switch  = ppe.append_switch(net, **switch_params(dfs, t1, t2))
    net.trafo   = ppe.append_trafo(net, **trafo_params(dfs, trafoend))
    net.trafo3w = ppe.append_trafo3w(net, **trafo3w_params(dfs, trafoend))
    net.load    = ppe.append_load(net, **load_params(dfs, t1))

    net.bus['island'] = topological_group(without_trafo(net), respect_switches=False)
    net.bus['partition'] = topological_group(net, respect_switches=False)
    net.bus.vn_kv = 1e-3 * bus_voltage_map(net, dfs)
    net.bus.vn_kv = net.bus.vn_kv.fillna(0)
    return net
```

Ideally speaking, there should now only be one partition. We still need to place our slack bus somehow. A problem here is that our CIM data doesn't specify where the grid receives external power from.

7.3.10 Attaching an ext_grid

Most likely, the external grid connection will be at the highest voltage level within a partition. Because of this, placing the ext_grid somewhere at this voltage level should likely be fine.

It might make sense for the client to be able to place the ext_grid manually in the map application. For now, however, we will just place it somewhere and assume that this is "good enough". Since we are not able to verify the accuracy of the simulations through measurements in this project regardless. This can be left to future work and improvements.

In theory you could have one ext_grid for every partition. But we will just attach a single one to the highest voltage level on the largest partition, and discard all other partitions.

7.3.11 Potential causes of divergence

- **Maximum Power Transfer Theorem.**

  How to check: Set loads to 0 and try to run the power flow again

  ```
  net.load.p_mw = 0
  net.load.q_mvar = 0
  ```

- **Numerical instability due to low line impedances**

  Solution: Replace low impedance lines with switches

  ```
  import pandapower_extended as ppe
  net.switch, net.line = ppe.line_to_switch(net, ppe.low_impedance(net.line))
  ```

- **0-values, nan-values or inf-values**

  Solution: Remove or use fallback-values for these elements

  ```
  with pd.option_context('mode.use_inf_as_na', True):

    # Critical! Now, these lines will be turned into switches
    net.line.r_ohm_per_km = net.line.r_ohm_per_km.fillna(0)
    net.line.x_ohm_per_km = net.line.x_ohm_per_km.fillna(0)
    net.line.c_nf_per_km = net.line.c_nf_per_km.fillna(0)
    net.line.length_km = net.line.length_km.fillna(0)
  ```

```
  # Doesn't cause divergence, but does cause invalid loading_percent and some warnings.
  net.line.max_i_ka = net.line.max_i_ka.replace({0: np.nan}).fillna(100)

  # Critical!
  net.trafo.sn_mva = net.trafo.sn_mva.replace({0: np.nan}).fillna(10)
  net.trafo.vk_percent = net.trafo.vk_percent.replace({0: np.nan}).fillna(5)
  net.trafo.vkr_percent = net.trafo.vkr_percent.fillna(net.trafo.vk_percent / 10)

  # Fallback to bus voltage
  net.trafo.vn_hv_kv = net.trafo.vn_hv_kv.fillna(
    net.trafo.merge(net.bus.vn_kv, left_on='hv_bus', right_index=True).vn_kv)
  net.trafo.vn_lv_kv = net.trafo.vn_lv_kv.fillna(
    net.trafo.merge(net.bus.vn_kv, left_on='lv_bus', right_index=True).vn_kv)

  # Basics
  net.trafo.pfe_kw = net.trafo.pfe_kw.fillna(0)
  net.trafo.i0_percent = net.trafo.i0_percent.fillna(0)
```

- **Illegal power transformer values**

  ```
  # net.trafo
  assert (net.trafo.vk_percent != 0).all()
  assert (net.trafo.vk_percent >= net.trafo.vkr_percent).all()

  # net.trafo3w
  assert (net.trafo3w.vk_hv_percent != 0).all()
  assert (net.trafo3w.vk_mv_percent != 0).all()
  assert (net.trafo3w.vk_lv_percent != 0).all()
  assert (net.trafo3w.vk_hv_percent >= net.trafo3w.vkr_hv_percent).all()
  assert (net.trafo3w.vk_mv_percent >= net.trafo3w.vkr_mv_percent).all()
  assert (net.trafo3w.vk_lv_percent >= net.trafo3w.vkr_lv_percent).all()
  ```

- **Large line capacitance**

  A line capacitance allows lines to store charge. This means simply keeping a line at a given voltage starts requiring a lot of both active and reactive power. The active power losses will approach V^2/R, while the reactive power losses will start growing out of control. Voltages will drop towards 0, leading up to an eventual voltage collapse.

  ```
  # This is usually safe
  net.line.c_nf_per_km = 0

  # Alternatively, try making it smaller
  net.line.c_nf_per_km *= 1e-3
  ```

  Power transformer tapping will bring the network closer to its stability margins

7.4 Isolating a subgrid for analysis

It can be valuable to isolate a subgrid, either for the purposes of running quicker power flows, or for visualizing a smaller part of the grid. Pandapower provides a builtin `pp.select_subnet(net, buses)` for isolating a subgrid.

When faced with a large grid, the builtin pandapower function for selecting a subnet given a set of buses were too slow. A pull request was created, that would speed up subnet selection from around 10-20 seconds to around 100ms for a particular grid. (Bergkvist 2020d) How is this kind of speedup even possible?

Loops in Python are notoriously slow, and for that reason it is usually a good idea to avoid them whenever possible. Thanks to NumPy, a vector library for Python written mostly in C, this is possible by “outsourcing” the looping to C code.

In addition to being orders of magnitudes faster when the looping happens in C, it also utilizes the Single Instruction Multiple Data (SIMD) capability of modern CPUs (Sayed Adel 2019).

8 Method: Geospatial data, Simulation Results and Visualization

8.1 Motivation

Visualization of data is not useful to computers. It is however useful to humans that want to understand something. Computers are great at crunching numbers, but they are not yet very good at "big picture"-based behavior - or even image classification compared to humans. This is why Google's captcha uses images to tell whether you are a human or a robot.

Humans and machines make for a very powerful duo, however. But in order for this duo to be efficient - they must communicate efficiently. One of the best ways for machines to communicate intuition-based knowledge/information to a human is through visualizations.

Since electrical grids are generally located somewhere close to the surface of the earth, geospatial visualization can be a good way of creating an intuition for the connectivity and layout of the electrical grid. The grid lines and other components could also be color coded in a continuous or categorical way to provide additional useful information.

Some potentially useful ways of visualizing are:

- Categorical colors for visualizing voltage islands/topological islands
- Threshold-based/discrete colors for visualizing voltage and or capacity problems
- Continuous color scales for visualizing gradual changes in current/voltage

8.2 Preparing geometry for the client

To be able to visualize the results of the powerflow, we are going to need the following two things:

- Geometry of network, such that we are able to draw it on top of a map.
- Results from powerflow analysis

We also need to make sure that our geometry points are provided in the correct projection. Since the projection in the CIM files (EPSG:32632) does not match the one we want to use (EPSG:4326), we will need to reproject the data. For this, we will use the library pyproj (Whitaker 2014), which provides a Python binding to PROJ (Warmerdam 1999). To reproject a lot of points at the same time, we can pass NumPy arrays to the transform function from pyproj.

```python
from pyproj import transform, Proj
import pandas as pd
import numpy as np

def first_value(rows):
    first_column = list(zip(*rows))[0]
    # Prevent NumPy from broadcasting into multidimensional array
    return np.array([*first_column, None])[:-1]

def location_geometry_map(dfs):
    # By sorting by Location, we ensure that points which belong
    # to the same Location will come right after each other
    sorted_points = dfs.PositionPoint.sort_values(by=['Location', 'sequenceNumber'])
    xPos = sorted_points.xPosition.values
    yPos = sorted_points.yPosition.values
    location = sorted_points.Location.values

    # Reprojecting to latitude and longitude with pyproj
    lat, lon = transform(Proj('epsg:32632'), Proj('epsg:4326'), xPos, yPos)

    unique_location, unique_location_index, coordinate_count = (
        np.unique(location, return_index=True, return_counts=True))

    # Now we can split at the point indices where a new Location appears
    # This way we get a list of points for every unique Location
    coordinates = np.split(np.transpose([lat, lon]), unique_location_index[1:])
    is_point = (coordinate_count == 1)
    return pd.Series(
        [ dict(zip(['type', 'coordinates'], row))
            for row in np.transpose([
                np.where(is_point, 'Point', 'LineString'),
                np.where(is_point, first_value(coordinates), coordinates)
            ])
        ],
        index=unique_location,
    ).rename_axis('Location').rename('geometry')
```

8.2.1 Merging Location data with Features

This is similar in nature to what we did when assigning voltages to pandapower buses earlier.

```
def feature_location_map(dfs):
    return concat.reduce([
        dfs.ACLineSegment.Location.dropna(),
        dfs.Substation.Location.dropna(),
        dfs.Breaker.Location.dropna(),
        dfs.BusbarSection.Location.dropna(),
        dfs.Disconnector.Location.dropna(),
        dfs.EnergyConsumer.Location.dropna(),
        dfs.Fuse.Location.dropna(),
        dfs.GroundDisconnector.Location.dropna(),
        dfs.Jumper.Location.dropna(),
        dfs.LoadBreakSwitch.Location.dropna(),
        dfs.PowerTransformer.Location.dropna(),
        traverse.reduce([
            concat.reduce([
                dfs.ACLineSegment.EquipmentContainer.dropna(),
                dfs.Breaker.EquipmentContainer.dropna(),
                dfs.BusbarSection.EquipmentContainer.dropna(),
                dfs.Disconnector.EquipmentContainer.dropna(),
                dfs.EnergyConsumer.EquipmentContainer.dropna(),
                dfs.Fuse.EquipmentContainer.dropna(),
                dfs.GroundDisconnector.EquipmentContainer.dropna(),
                dfs.Jumper.EquipmentContainer.dropna(),
                dfs.LoadBreakSwitch.EquipmentContainer.dropna(),
                dfs.PowerTransformer.EquipmentContainer.dropna(),
            ]),
            concat.reduce([
                traverse.reduce([
                    dfs.Bay.VoltageLevel.dropna(),
                    dfs.VoltageLevel.Substation.dropna(),
                ]),
                dfs.VoltageLevel.Substation.dropna(),
            ]),
            dfs.Substation.Location.dropna(),
        ])
    ])


# To merge location data with features, we can simply now use:
def feature_geometry_map(dfs):
    return traverse.reduce([
        feature_location_map(dfs),
        location_geometry_map(dfs)
    ])

# This can be saved as a csv-file for later use, or combined into a geojson file
#  - which we could send straight to the client
```

8.3 Simulation results

Based on our use case, we might want to send the geometry straight to the client, with minimal metadata to save valuable bandwidth. The other option is to inject the simulation results into the GeoJSON before sending it. DataFrames in pandas have a useful "to_json"-method for generating JSON data on string format.

```
geojson = df.to_json(orient='records')
```

For our grid overview client, we will be loading a lot of geometry into the client. Having to reload all of this for every simulation would not be ideal. Because of this, we will send geometry and simulation results separately, and merge them in the client.

For our subgrid health client, the geometry will potentially change for every powerflow - since a new subgrid could have been selected. For this reason, the simulation results will be merged into the geojson data on the server side before being sent to the client.

8.4 Communication between client and server

8.4.1 FastAPI and Axios

To facilitate communication between the Python code on the server-side, and the React-code on our client-side, we will be using FastAPI (Ramírez 2018) and Axios (Zabriskie 2014).

FastAPI is a framework that allows for easily creating REST APIs in Python. Not only that, but API documentation is also automatically generated in the process. Creating an endpoint is very easy:

```
# This will create a path-based endpoint where you can write
# http://localhost:5000/sum/5/13
# into your browser, and it will display { "result": 18 } back to you
@app.get('sum/{x}/{y}')
def add(x: float, y: float):
    return { 'result': x + y }
```

Axios is a HTTP request library that we will use on the client-side to send requests to the server, as well as handle responses. Consider the following example

```
import Axios from 'axios'

async function requestSum(x, y) {
  const { data } = await Axios.get(`http://localhost:5000/${x}/${y}`)
  return data.result
}
```

8.4.2 Cross-Origin Resource Sharing (CORS)

When sending requests across multiple domains, you will have to take the browser CORS policies into account. You might not be allowed to read the response from the server in the client. There are a few ways of working around this.

1. Use a reverse proxy, like Nginx (Sysoev 2002) to make sure that both the server and the client is served on the same host name/domain. You can use rules based on the url path to decide where to route the requests.
2. Add the HTTP Headers `Access-Control-Allow-Origin: *` or `Access-Control-Allow-Origin: <specific origin>` to the server responses. This will tell your browser that your client-application is allowed to read the response. This can be done quite easily in FastAPI by adding a builtin middleware.

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # Alternatively, a list of allowed origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

8.5 Component layout with React-Mosaic

Now, it is time to consider some design aspects of the client. As mentioned in the end of the background section on “The modern web and programming paradigms”, React-Mosaic is great in reducing mental effort because it allows us to not have to think about layout of component groups on the screen.

As a reminder, React-Mosaic is a tiling window manager for the browser, written in React - and with a BlueprintJS theme. React allows us to separate state from view logic, which is a principle React-Mosaic follows. This means that other components and actions are able to potentially control over the window layout.

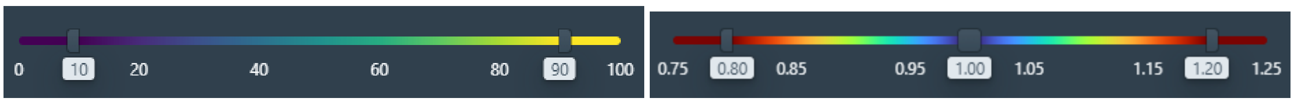
The React-Mosaic state is a tree structure, based around horizontal and vertical splits - as well as split percentages. To find a nice initial layout, we will place the windows/panels in initially random positions - and drag them around until we are happy. Then we will copy this tree structure, and paste it into our code as an initial value.

8.6 Color coding of simulation results in the client

One of the most important things when using continuous color scales for visualization - is to have some kind of reference/color bar to that tells us which values the colors represent. Imagine if you could use this color bar as a controller for changing the color-scale in real-time as well.

8.6.1 Color scale slider

BlueprintJS already has something known as a “MultiSlider” component, which is a slider with multiple draggable handles. By default, it used to be relatively limited in how the tracks between the handles could be styled. As of such, a contribution to the BlueprintJS library was made as part of this project, enabling the tracks to visualize color scales (Bergkvist 2020a), as shown below:



The slider serves 2 purposes:

- As a colorbar, telling the user what numbers the colors represent.
- As an intuitive controller for changing the thresholds of the color scale.

8.7 Geospatial visualization

For geospatial visualization on maps in the web browser, there are a several popular open source libraries. Some of these are:

- **OpenLayers**
  “OpenLayers is a high-performance, feature-packed library for creating interactive maps on the web. It can display map tiles, vector data and markers loaded from any source on any web page.” (Christopher Schmidt 2006)
- **Leaflet**
  “Leaflet is the leading open-source JavaScript library for mobile-friendly interactive maps. Weighing just about 37 KB of gzipped JS code, it has all the mapping features most developers ever need.” (Agafonkin 2010)
- **Mapbox GL JS**
  “Mapbox GL JS is a JavaScript library for interactive, customizable vector maps on the web. It takes map styles that conform to the Mapbox Style Specification, applies them to vector tiles that conform to the Mapbox Vector Tile Specification, and renders them using WebGL.” (Mapbox 2013)

Out of these, the library that provides the best performance for visualizing data on top of a map is Mapbox GL JS. It also allows the map to be tilted for a 3D effect - along with displaying 3D buildings. Much of the API is imperative, however. Is there a way we could make this fit into our React-way of thinking?

It turns out Uber will save us a lot of work, since they have created a React binding for Mapbox GL JS called react-map-gl.

Not only that, but Uber has also created the data visualization library deck.gl, which integrates seamlessly with react-map-gl.

8.7.1 react-map-gl

react-map-gl is a "React friendly API wrapper around MapboxGL JS" created by Uber, and donated to the Urban Computing Foundation that ports Mapbox GL JS to React. (Uber 2015) react-map-gl is essentially a wrapper that allows for controlling/keeping track of the map state externally.

Perhaps the biggest selling point of react-map-gl, however - is that it integrates with the data visualization library deck.gl, also created by Uber.

8.8 deck.gl

deck.gl is a React and WebGL2-framework that integrates with React-Map-GL to provide large scale performant data visualizations in the browser (Uber 2016). This is made possible by minimizing the communication between the CPU and the GPU. Deck.GL maintains a separate render loop from React, which is necessary, since React was not built to manipulate HTML5 canvases through WebGL. The render loops being seperate can cause synchronization challenges.

One of the most impressive things about deck.gl is its performance. Nothing else that has been tested has even come remotely close to the performance that deck.gl is able to provide for large amounts of data. With deck.gl it is possible to visualize hundreds of thousands or even millions of assets at the same time. deck.gl raises the bar for what is possible in terms of large-scale data visualization in the browser.

8.8.1 react-context-toolkit

Due to a problem with deck.gl and react-redux (Bergkvist 2020b), these libraries currently cannot be used together. It turns out that React Context does not have the same problem. As of such, a toolkit library that makes React Context feel more like react-redux and redux-toolkit was created as part of this project. This library is called `react-context-toolkit`, and is available on GitHub (Bergkvist 2020c).

This library is made possible thanks to `use-context-selector`, a userland implementation of useContextSelector (Kato 2019), which is not yet implemented in the official React library, but is currently an active proposal on GitHub (Story 2019).

Context selectors allows for selecting a slice of the application state, and only rerendering the components that uses the parts of the application state that has changed. Without context selectors, any change to the state will rerender all components that depend on any part of it.

8.9 Visualizing tabular data

Something else that is needed, is the ability to display interactive tabular data. BlueprintJS has a table-module which could be used, although provides a more spreadsheet-like feeling. Instead we will be using a different library.

8.9.1 material-table

material-table is a "Datatable for React based on material-ui's table with additional features" (Baran 2018). It provides useful features such as search, action buttons, and sorting. There are a couple of challenges, however.

The default styling of material-table doesn't fit the color profile of the BlueprintJS-theme, so we will need to manually change the styling to reuse colors from BlueprintJS.

material-table mutates its input data, which is an ugly antipattern (Royer 2019). The result of this is that material-table doesn't work together with Immer by default, since Immer freezes the state it produces to enforce immutability.

```
Object.freeze(object)
// Trying to modify the properties of object will now throw an error
```

To work around this, we will have to disable freezing in our Immer instance, or overwrite the Object.freeze-function globally to do nothing (which is far from an elegant solution). Another alternative is to deep clone the table state before passing it to the table, which could have performance implications.

9 Results

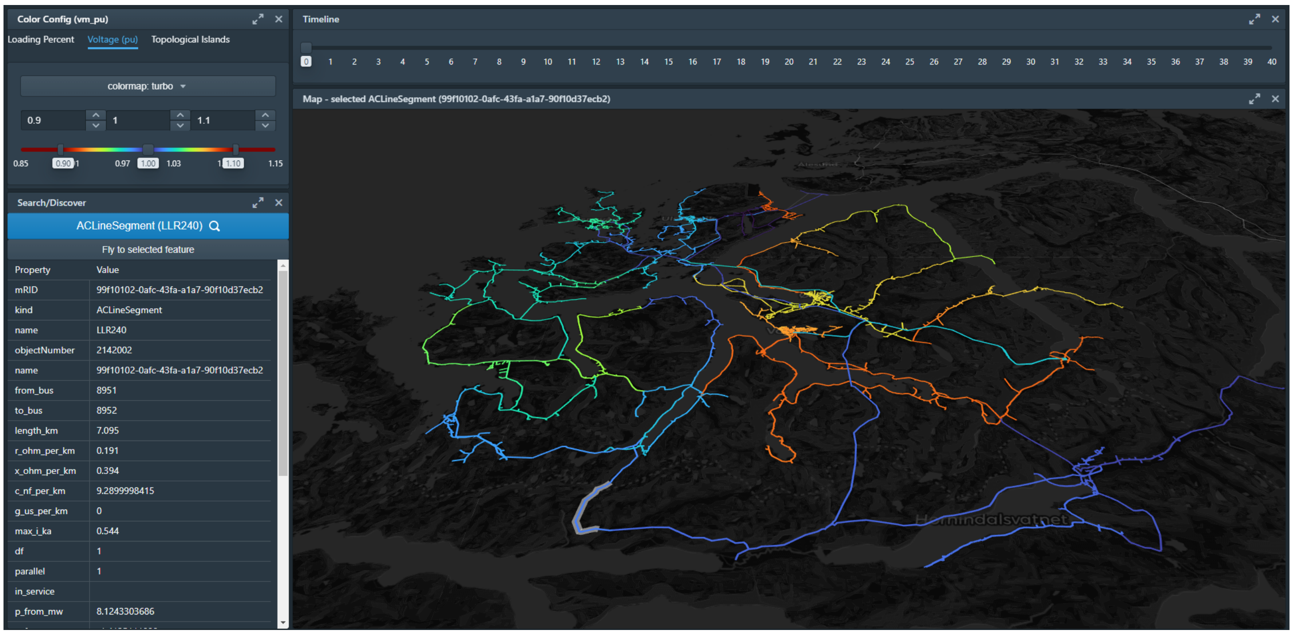
9.1 Grid Overview - Visualizing an entire power grid in the browser

We will now look at our first result: A user-facing client that can visualize and run powerflows on a distribution grid with tens of thousands of customers, and display everything as a web application in the browser.

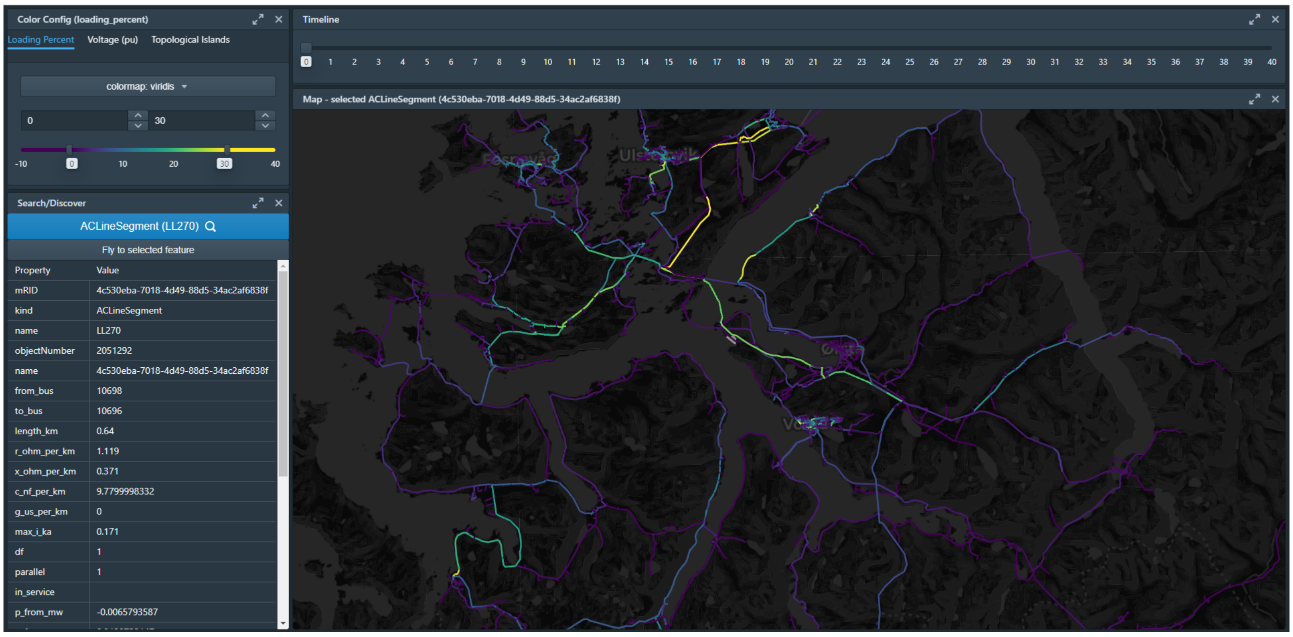
9.1.1 Feature overview

- Switch between viewing capacity, current, and topological islands.
- Switch between different color schemes based on which view mode you have selected.
- Drag a timeline to trigger different simulations over the course of 40 hours with scaled consumptions.
- Geometry and simulation results are merged in the client. This means the geometry doesn't need to be reloaded in the client when a new simulation is run.
- Click assets in the map or search for them to select them.
- Trigger a fly-to animation in the map to find the selected asset.
- See input parameters and simulation results for selected elements

9.1.2 Screenshots: voltage deviations and line loading



Property	Value
mRID	99f10102-0afc-43fa-a1a7-90f10d37ecb2
kind	ACLineSegment
name	LLR240
objectNumber	2142002
name	99f10102-0afc-43fa-a1a7-90f10d37ecb2
from_bus	8951
to_bus	8952
length_km	7.095
r_ohm_per_km	0.191
x_ohm_per_km	0.394
c_nf_per_km	9.2899998415
g_us_per_km	0
max_i_ka	0.544
df	1
parallel	1
in_service	
p_from_mw	8.1243303686



Property	Value
mRID	4c530eba-7018-4d49-88d5-34ac2af6838f
kind	ACLineSegment
name	LL270
objectNumber	2051292
name	4c530eba-7018-4d49-88d5-34ac2af6838f
from_bus	10698
to_bus	10696
length_km	0.64
r_ohm_per_km	1.119
x_ohm_per_km	0.371
c_nf_per_km	9.7799998332
g_us_per_km	0
max_i_ka	0.171
df	1
parallel	1
in_service	
p_from_mw	-0.0065793587

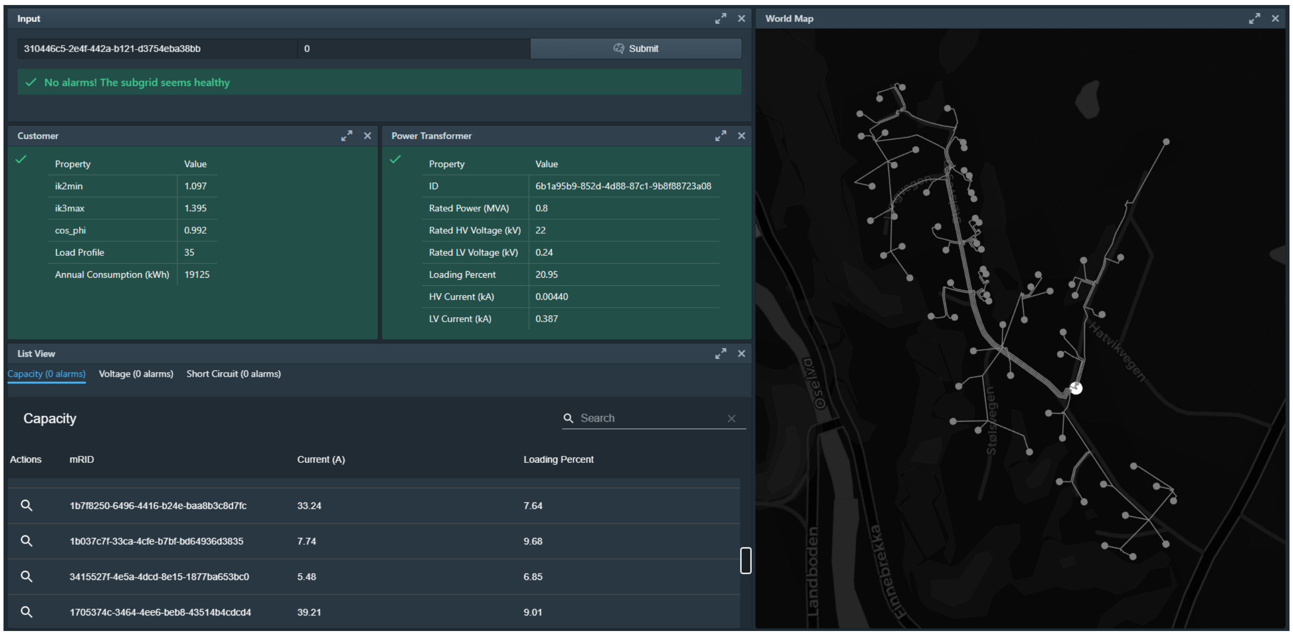
9.2 Subgrid Health - Measuring the health of subgrids

Now, for our second result: a prototype/client that can potentially be used by a caseworker to tell you if you application for an electric car charger is approved, without the need for a specialist to calculate anything by hand.

9.2.1 Feature overview

- Enter a customer ID, and a requested additional consumption/load.
- The topological island that you customer belongs to (below a power transformer) will be isolated, and a power flow calculation will be performed. Geometry and simulation results will be merged on the server before being sent to the client.
- A clear overview over the number of problems, different types of problems, and their location.
- If no subgrid alarms are triggered, the application for an electric car charger can be immediately accepted.

9.2.2 Screenshots: An added load of 100kW causes capacity and voltage problems



310446c5-2e4f-442a-b121-d3754eba38bb | 0

✓ No alarms! The subgrid seems healthy

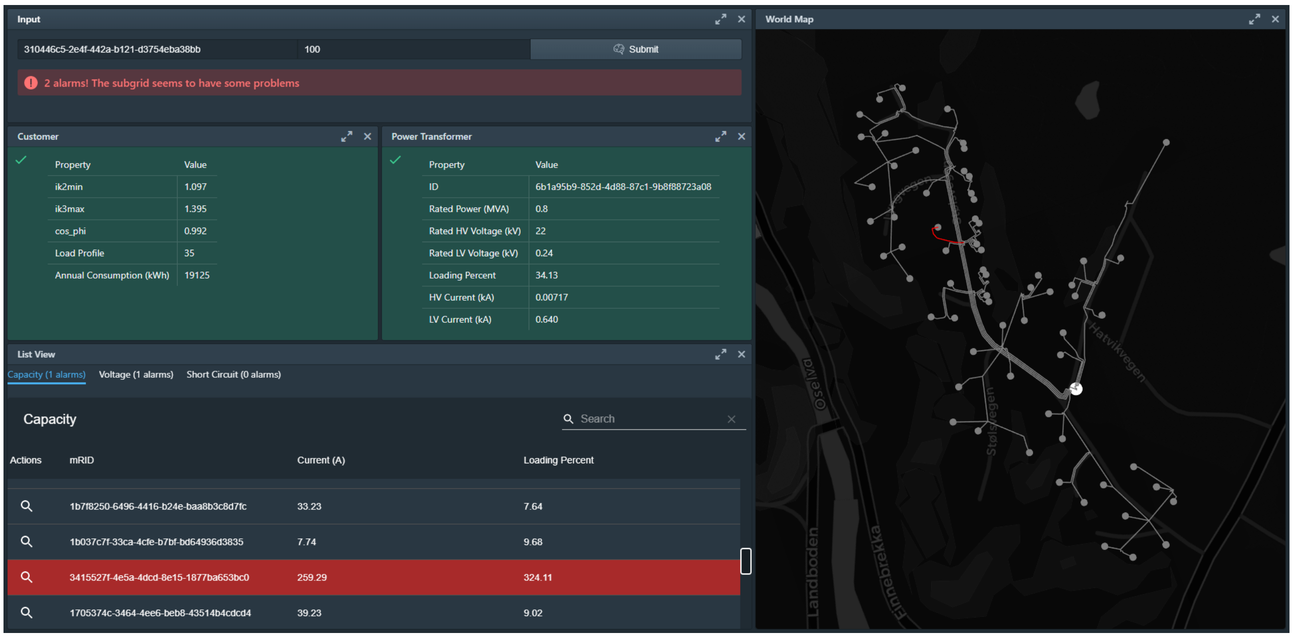
Property	Value
ik2min	1.097
ik3max	1.395
cos_phi	0.992
Load Profile	35
Annual Consumption (kWh)	19125

Property	Value
ID	6b1a95b9-852d-4d88-87c1-9b8f88723a08
Rated Power (MVA)	0.8
Rated HV Voltage (kV)	22
Rated LV Voltage (kV)	0.24
Loading Percent	20.95
HV Current (kA)	0.00440
LV Current (kA)	0.387

Capacity (0 alarms) Voltage (0 alarms) Short Circuit (0 alarms)

Capacity

Actions	mRID	Current (A)	Loading Percent
🔍	1b7f8250-6496-4416-b24e-baa8b3c8d7fc	33.24	7.64
🔍	1b037c7f-33ca-4cfe-b7bf-bd64936d3835	7.74	9.68
🔍	3415527f-4e5a-4dcd-8e15-1877ba653bc0	5.48	6.85
🔍	1705374c-3464-4ee6-beb8-43514b4cdcd4	39.21	9.01



310446c5-2e4f-442a-b121-d3754eba38bb | 100

! 2 alarms! The subgrid seems to have some problems

Property	Value
ik2min	1.097
ik3max	1.395
cos_phi	0.992
Load Profile	35
Annual Consumption (kWh)	19125

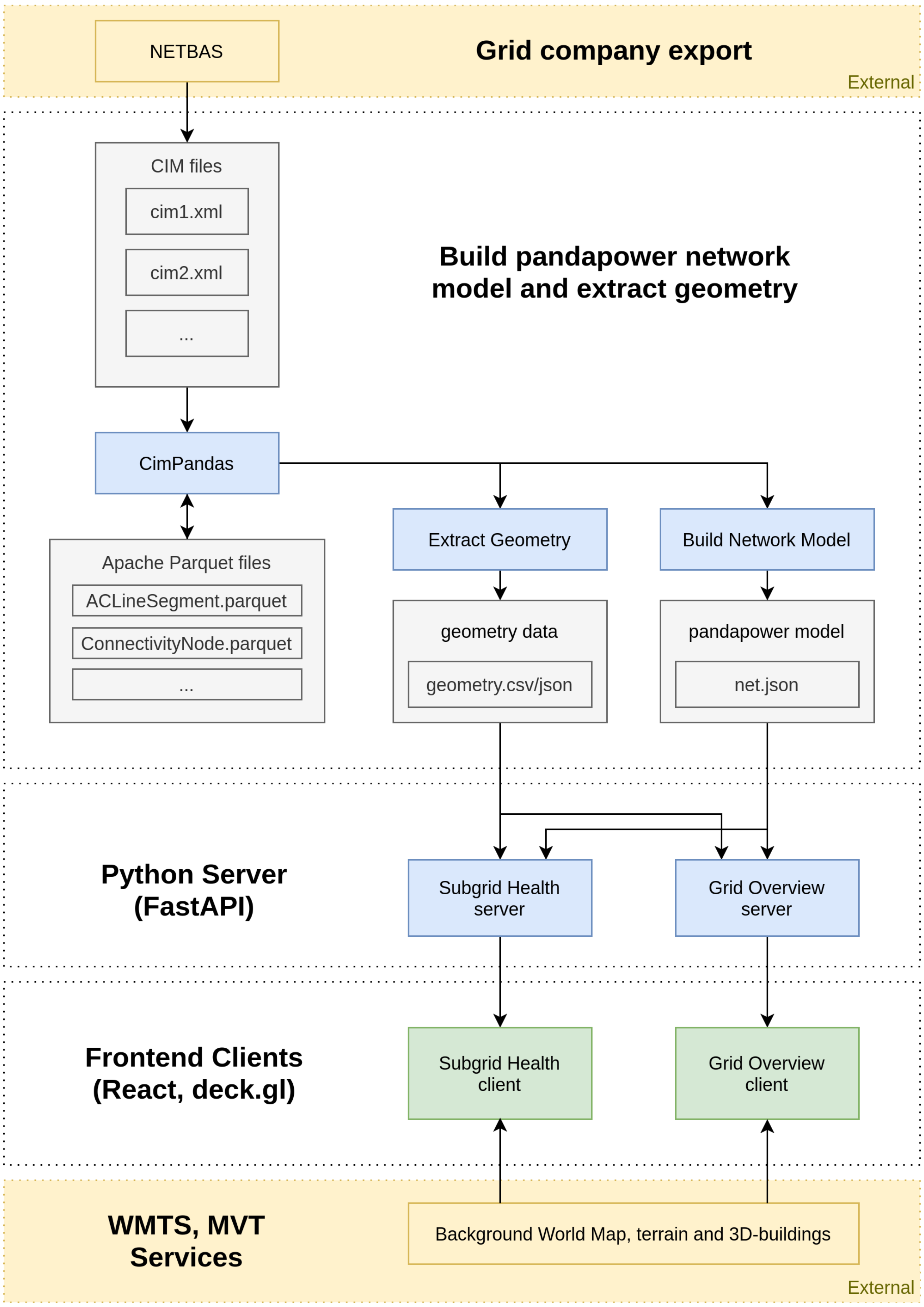
Property	Value
ID	6b1a95b9-852d-4d88-87c1-9b8f88723a08
Rated Power (MVA)	0.8
Rated HV Voltage (kV)	22
Rated LV Voltage (kV)	0.24
Loading Percent	34.13
HV Current (kA)	0.00717
LV Current (kA)	0.640

Capacity (1 alarms) Voltage (1 alarms) Short Circuit (0 alarms)

Capacity

Actions	mRID	Current (A)	Loading Percent
🔍	1b7f8250-6496-4416-b24e-baa8b3c8d7fc	33.23	7.64
🔍	1b037c7f-33ca-4cfe-b7bf-bd64936d3835	7.74	9.68
🔍	3415527f-4e5a-4dcd-8e15-1877ba653bc0	259.29	324.11
🔍	1705374c-3464-4ee6-beb8-43514b4cdcd4	39.23	9.02

9.3 Architecture



10 Conclusion

Given a CIM export from NETBAS from a grid company, we are now able to build a model for running powerflow simulations (even in the face of imperfect data quality). We are also able to extract geometry as geojson - and combine this with simulation results for large-scale data visualization in a web browser.

We have created a simple prototype that allows for assessing the health of a subgrid, with some additional requested capacity to see how the grid will handle it. If this is based on realistic load profiles, tuned to better match reality, and properly adapted to case worker needs, it could save a lot of money and time - both for the case workers, and for the people who want to install electric car chargers in their garages.

In conclusion - we don't have a production-ready digital twin by any means, but this wasn't the goal either. This is more than anything a foundation, that demonstrates what is possible. There will likely be some significant advancements in the energy sector in the years to come - in terms of digital systems.

10.1 Future work

There are a lot of improvements to consider and interesting directions to move from here. Some things to consider are:

- If these prototypes are to be made into actual products, this would require authentication systems - support for multitenancy, and automation in the onboarding process.
- Right now, one of the biggest bottlenecks is parsing and reading XML data into Python. This can be improved by writing this part of the solution in a lower-level language than Python - like C++ or Rust. An alternative might also be to use something like Cython or Numba - in order to stay within the Python ecosystem.
- Once data sets become too large to fit into the memory of the computer, alternatives to pandas like Dask and Apache Spark starts to look interesting. Perhaps pandapower could be extended to run powerflows on distributed data in Apache Spark. In this case, it might also no longer be feasible to send all the geometry to the client at the same time. Using something like PostGIS for querying tiles as MVT might be an interesting solution.
- Using real-time sensor data to run real-time simulations could allow for quickly detecting outages and problems in the grid in real-time.
- Using realistic load profiles are important for getting realistic results. These could be based on weather forecasts and historical data to predict future problems before they happen.

11 Development Environment

11.1 JupyterLab

JupyterLab gives you a so called "notebook" interface, which is perfect for experimenting with and prototyping out code, as well as running one-time scripts to create plots or similar. Its biggest environment is likely around Python, but it supports a ton of different languages.

11.2 TypeScript and Visual Studio Code

TypeScript is a superset of JavaScript, which adds types to the language (and compiles to regular JavaScript). This allows for static checking of your code, reducing the number and types of tests needed for your code.

Visual Studio Code is a free to use editor created by Microsoft. Since both TypeScript and VSCode was created by Microsoft, you can imagine they work quite well together, providing very good autocomplete suggestions (Intellisense) while writing code. This might as big a motivation for using TypeScript as the static checking it provides.

11.3 Yarn and React Fast Refresh

Yarn was created as an alternative to the node package manager (npm) for node.js (server-side JavaScript). Its main advantage is that it is significantly faster than npm when it comes to installing dependencies.

React Fast Refresh is quite new, and as of such is not yet officially part of create-react-app (Abramov 2016). It can easily be added by customizing some files, however. It allows you to edit a file, and have the website refresh with the update without losing any of its state!

11.4 Docker and Docker-Compose

Docker is a tool for working with, creating and running containerized applications. Containers fill the same market share as virtual machines for development in many cases, but are more lightweight and efficient than virtual machines, since they reuse anything they have in common with each other or the host operating system.

You can embed your application into what is known as a docker image. This can then be pushed to a docker registry, such as Docker Hub. This allows someone else to download and run your image. Since the images run in isolated environments, they will "just work" essentially everywhere.

Docker-Compose allows for starting and stopping multiple containers that can talk to each other and use shared folders. All of this is also possible using just docker, but docker-compose streamlines the process. Its main purpose is for development, and should not be used in production environments. For this, Kubernetes is more suitable.

12 Kubernetes

Kubernetes is an open source distributed operating system created by Google used for production environments (Google 2014). Its main purposes are to provide high availability, horizontal scalability and to allow you to define infrastructure as code for your applications.

Kubernetes allows for describing the desired state of the system declaratively - and the Kubernetes engine will make a "best effort" to realize this state, as well as maintain it, given the available resources. A bit like the cruise control on your car will try to keep your car moving at a certain speed.

To do this, it defines abstractions for processes, communication and data/persistence. Under the hood it uses prebuilt container images to run every "process" in an isolated and predictable environment.

13 Abbreviations and Terms

- TSO: Transmission System Operator
- DSO: Distribution System Operator
- CIM: Common Information Model (Electricity)
- XML: Extensible Markup Language
- HTML: HyperText Markup Language
- CSS: Cascading Style Sheets
- API: Application Programming Interface
- CSV: Comma Separated Values
- trafo: Power Transformer
- trafo3w: 3-Winding Power Transformer
- ext_grid: External Grid/Slack bus/Signal generator
- NaN: Not a number (represents a missing or invalid value)
- df: Used as shorthand variable name for a DataFrame instance in the context of the pandas Python library
- pp: Shorthand variable name used when importing the pandapower library
- pd: Shorthand variable name used when importing the pandas library
- np: Shorthand variable name used when importing the NumPy library
- WMS: Web Map Service
- WMTS: Web Map Tile Service
- WFS: Web Feature Service
- MVT: Mapbox Vector Tile
- REST: REpresentational State Transfer
- CORS: Cross-Origin Resource Sharing

14 Appendix

14.1 CIM classes and their properties

Notice that the parenthesis after classes show which classes they inherit from. These are sometimes neccesary to understand what references are pointing at.

14.1.1 Substation (EquipmentContainer)

- Location (reference)
- PSRType (reference)
- name
- alias
- objectNumber
- descripion

14.1.2 Location

- CoordinateSystem (reference)
- mainAddress

14.1.3 PositionPoint

- Location (reference)
- xPosition
- yPosition
- sequenceNumber

14.1.4 PSRType

- name

14.1.5 VoltageLevel (EquipmentContainer)

- Substation (reference)
- BaseVoltage (reference)
- name

14.1.6 Bay (EquipmentContainer)

- VoltageLevel (reference)
- name

14.1.7 BaseVoltage

- nominalVoltage

14.1.8 CoordinateSystem

- crsUrn

14.1.9 ACLineSegment (ConductingEquipment)

- Location (reference)
- PSRType (reference)
- BaseVoltage (reference)
- name
- length
- b0ch
- bch
- r
- r0
- x
- x0
- value (current limit)
- description
- objectNumber
- EquipmentContainer (reference)

14.1.10 Terminal

- ConductingEquipment (reference)
- ConnectivityNode (reference)
- sequenceNumber

14.1.11 ConnectivityNode

No properties

14.1.12 BusbarSection (ConductingEquipment)

- EquipmentContainer (reference)
- BaseVoltage (reference)
- Location (reference)
- name

14.1.13 GroundDisconnector (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- name
- normalOpen
- scadaident
- objectNumber
- BaseVoltage (reference)

14.1.14 Fuse (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- name
- normalOpen
- ratedCurrent
- scadaident
- objectNumber
- BaseVoltage (reference)

14.1.15 Disconnector (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- name
- normalOpen
- ratedCurrent
- scadaident
- objectNumber BaseVoltage (reference)

14.1.16 LoadBreakSwitch (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- name
- normalOpen
- ratedCurrent
- scadaident
- objectNumber
- BaseVoltage (reference)

14.1.17 Breaker (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)

- normalOpen
- ratedCurrent
- breakingCapacity
- scadaident
- objectNumber
- BaseVoltage (reference)

14.1.18 Jumper (Switch, ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- normalOpen
- objectNumber
- BaseVoltage (reference)

14.1.19 PowerTransformer (ConductingEquipment)

- Location (reference)
- EquipmentContainer (reference)
- name
- objectNumber

14.1.20 PowerTransformerEnd

- BaseVoltage (reference)
- PowerTransformer (reference)
- Terminal (reference)
- endNumber
- ratedU
- ratedS
- connectionKind
- phaseAngleClock
- loss
- r
- endBaseVoltage
- lossZero
- uk

14.1.21 RatioTapChanger

- TransformerEnd (reference to PowerTransformerEnd)
- stepVoltageIncrement
- highStep
- lowStep
- neutralStep
- neutralU
- normalStep

14.1.22 EnergyConsumer (ConductingEquipment)

- Location (reference)
- BaseVoltage (reference)
- name
- objectNumber
- EquipmentContainer (reference)

14.1.23 UsagePoint

- Equipments (reference to EnergyConsumer)
- annualconsumption
- objectNumber
- LoadGroup
- EquipmentContainer (reference)

References

Abramov, Dan. 2015. "React-Redux - Official React Bindings for Redux." 2015. https://react-redux.js.org/.

———. 2016. "Create React App: Set up a Modern Web App by Running One Command." 2016. https://create-react-app.dev/.

———. 2020. "Redux: Implementing Undo History." 2020. https://redux.js.org/recipes/implementing-undo-history.

afc163, Wei Zhu, Benjy Cui. 2015. "Ant Design: A Ui Design Language and React Ui Library." 2015. https://ant.design/.

Agafonkin, Vladimir. 2010. "Leaflet - a Javascript Library for Interactive Maps." 2010. https://leafletjs.com/.

Alexandrov, Oleg. 2007. "Illustration of Complex Number in 2D Plane." 2007. https://commons.wikimedia.org/wiki/File:Complex_conjugate_picture.svg.

Apache. 2013. "Apache Parquet: Columnar Storage Format." 2013. https://parquet.apache.org/.

———. 2016. "Apache Arrow: A Cross-Language Development Platform for in-Memory Data." 2016. https://arrow.apache.org/.

Baran, Mehmet. 2018. "Material-Table: Datatable for React Based on Material-Ui's Table with Additional Features." 2018. https://material-table.com.

Bergkvist, Tobias. 2020a. "BlueprintJS Pull Request: Add trackStyleBefore and trackStyleAfter to Multi-slider.handle." 2020. https://github.com/palantir/blueprint/pull/4130.

———. 2020b. "Deck.gl Issue: Map Flickers on Transition When viewState Is Stored in Redux." 2020. https://github.com/visgl/deck.gl/issues/4550.

———. 2020c. "React-Context-Toolkit: Toolkit for React Context Api Heavily Inspired by Reduxjs/Toolkit and React-Redux." 2020. https://github.com/bergkvist/react-context-toolkit.

———. 2020d. 2020. https://github.com/e2nIEE/pandapower/pull/765.

Christopher Schmidt, Schuyler Erle. 2006. "OpenLayers: A High-Performance, Feature-Packed Library for All Your Mapping Needs." 2006. https://openlayers.org/.

Collins Dictionary. n.d. "Definition of Power Grid." Accessed June 6, 2020. https://www.collinsdictionary.com/dictionary/english/power-grid.

"Common Information Model." n.d. Accessed June 6, 2020. https://www.entsoe.eu/digital/common-information-model/.

Dodds, Kent C. 2018. "React Prop Drilling." 2018. https://kentcdodds.com/blog/prop-drilling.

Edison Tech Center. n.d. "The History of Electrification." Accessed June 6, 2020. https://edisontechcenter.org/HistElectPowTrans.html.

Erikson, Mark. 2018. "Redux-Toolkit: The Official, Opinionated, Batteries-Included Toolset for Efficient Redux Development." 2018. https://github.com/reduxjs/redux-toolkit.

Faassen, Martijn. 2005. "Lxml - Xml and Html with Python." 2005. https://lxml.de/.

Google. 2009. "Protocol Buffers - Google's Data Interchange Format." 2009. https://developers.google.com/protocol-buffers/.

———. 2014. "Kubernetes: Production-Grade Container Orchestration." 2014. https://kubernetes.io/.

Grøn, Øyvind. 2019. "SNL: Vekselstrøm." 2019. https://snl.no/vekselstr%C3%B8m.

Hai Nguyen, Olivier Tassinari. 2014. "Material Ui: React Components for Faster and Easier Web Development." 2014. https://material-ui.com/.

Hawkins, Nehemiah. 1917a. "Hawkins Electrical Guide, Volume 4." In, Page 1026. Copyright 1917 by Theo. Audel & Co.

———. 1917b. "Hawkins Electrical Guide, Volume 7." In, Page 1979. Copyright 1917 by Theo. Audel & Co.

Howard Butler, Allan Doyle, Martin Daly. 2016. "The GeoJSON Format." RFC 7946. IETF Tools; Internet Requests for Comments; IETF Tools. https://tools.ietf.org/html/rfc7946.

Jack Lukic, Levi Thomason. 2013. "Semantic Ui React: User Interface Is the Language of the Web." 2013. https://react.semantic-ui.com/.

John W. Simpson-Porco, Francesco Bullo, Florian Dörfler. 2016. "Voltage Collapse in Complex Power Grids." *Nature Communications* 7 (10790). https://doi.org/10.1038/ncomms10790.

Jonnes, Jill. 2004. *Empires of Light: Edison, Tesla, Westinghouse, and the Race to Electrify the World.* Random House.

Kato, Daishi. 2019. "React useContextSeelctor Hook in Userland." 2019. https://github.com/dai-shi/use-context-selector.

Kennelly, A. E. 1893. "Impedance." *Transactions of the American Institute of Electrical Engineers* X: 172–232. https://doi.org/10.1109/T-AIEE.1893.4768008.

Kirchhoff, Gustav. 1845. "Ueber Der Durchgang Eines Elektrischen Stromes Durch Ebene, Insbesibdere Durch Eine Kreisformige." *Annalen Der Physik Und Chemie* LXIV (4).

"KogniTwinGrid." 2019. 2019. https://www.kongsberg.com/no/digital/solutions/kognitwingrid/.

"Kongsberg Digital." 2015. 2015. https://www.kongsberg.com/no/digital/.

Mapbox. 2013. "Mapbox Gl Js - a Javascript Library for Interactive, Customizable Vector Maps on the Web." 2013. https://github.com/mapbox/mapbox-gl-js.

———. 2014. "Mapbox Vector Tile Specification." 2014. https://github.com/mapbox/vector-tile-spec.

McKinney, Wes. 2010. "Data Structures for Statistical Computing in Python." In *Proceedings of the 9th Python in Science Conference*, 445:51–56. Austin, TX.

———. 2011. "Pandas: A Foundational Python Library for Data Analysis and Statistics." *Python for High Performance and Scientific Computing* 14.

Mozilla. 2020. "MDN Web Docs: Window.localStorage." 2020. https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage.

Nedkov, Simeon. 2015. "WMTS: Tile Pyramid Image." 2015. https://github.com/Geonovum/PDOK-NGR-documentatie/blob/master/docs/images/tile-pyramid.png.

"Nordic System Operation Agreement." 2019. 2019. https://www.statnett.no/contentassets/2270ef6fc48e42ae84 53833bb86df746/nordic-soa-2019-annex-lfcr.pdf.

Norwegian oil and energy department. 2004. "Regulation on Supply Quality in the Power System." https://lovdata.no/dokument/SF/forskrift/2004-11-30-1557.

NVE. 2018. "Current Reports: How Will a Comprehensive Electrification of the Transport Sector Affect the Power System?" 2018. https://www.nve.no/energy-consumption-and-efficiency/energy-consumption-in-norway/current-reports/.

Omegatron. 2006. "Illustration of Kirchhoffs Current Law." 2006. https://commons.wikimedia.org/wiki/File:KCL.png.

OS Awards JavaScript. 2019. "The Most Impactful Contribution to the Community (Immer)." 2019. https://osawards.com/javascript/2019.

OS Awards React. 2019. "Breakthrough of the Year (Immer)." 2019. https://osawards.com/react/2019.

Palantir. 2016. "Blueprint: A React-Based Ui Toolkit for the Web." 2016. https://blueprintjs.com/.

Powel. n.d. "NETBAS: Grid Management Software." Accessed June 6, 2020. https://www.powel.no/smarte-nettselskap/nettinformasjonssystem/Netbas-copy.

Python. 2005. "Xml.etree.ElementTree — the Elementtree Xml Api." 2005. https://docs.python.org/3/library/xml.etree.elementtree.html.

Ramírez, Sebastián. 2018. "FastAPI: Web Framework for Building Apis with Python 3.6+." 2018. https://github.com/tiangolo/fastapi.

Rhodes, Brandon. 2014. "The Clean Architecture in Python." https://pyvideo.org/pyohio-2014/the-clean-architecture-in-python.html.

Royer, Stéphane. 2019. "Material-Table Issue: Rows Are Modified with a New Property tableData." 2019. https://github.com/mbrn/material-table/issues/666.

Santilli, Sandro. 2005. "ZM Values and Srid for Simple Features." PostGIS. https://github.com/postgis/postgis/blob/2.1.0/doc/ZMSgeoms.txt.

Sayed Adel, Ralf Gommers, Matti Picus. 2019. "NumPy Nep 38 - Using Simd Optimization Instructions for Performance." 2019. https://numpy.org/neps/nep-0038-SIMD-optimizations.html.

Story, Josh. 2019. "React Rfc: Context Selectors Proposal." 2019. https://github.com/reactjs/rfcs/pull/119.

Sysoev, Igor. 2002. "Nginx Http and Reverse Proxy Server." 2002. http://nginx.org/.

Thurner, L., A. Scheidler, F. Schäfer, J. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun. 2018. "Pandapower — an Open-Source Python Tool for Convenient Modeling, Analysis, and Optimization of Electric Power Systems." *IEEE Transactions on Power Systems* 33 (6): 6510–21. https://doi.org/10.1109/TPWRS.2018.2829021.

Uber. 2015. "React-Map-Gl - a Suite of React Components for Mapbox Gl Js." 2015. http://visgl.github.io/react-map-gl.

———. 2016. "Decl.gl: WebGL2 Powered Geospatial Visualization Layers." 2016. https://deck.gl.

Verdieck, Kevin. 2017. "React Mosaic: A React Tiling Window Manager." 2017. https://github.com/nomcopter/react-mosaic.

Warmerdam, Frank. 1999. "PROJ - Cartographic Projections and Coordinate Transformations Library." 1999. https://github.com/OSGeo/PROJ.

Weststrate, Michel. 2017. "Immer: Create the Next Immutable State by Mutating the Current One." 2017. https://github.com/immerjs/immer.

Whitaker, Jeffrey. 2014. "Pyproj: Python Interface to Proj (Cartographic Projections and Coordinate Transformations Library)." 2014. https://github.com/pyproj4/pyproj.

Zabriskie, Matt. 2014. "Axios: Promise Based Http Client for the Browser and Node.js." 2014. https://github.com/axios/axios.