Software zur Berechnung von Netzen für die elektrische Energieverteilung in Nieder- und Mittelspannungsbereichen

von Martin Schroll und Benedikt Schmidt

Eidesstattliche Erklärung

Wir erklären an Eides statt, dass wir die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht haben.

Martin Schroll Benedikt Schmidt

Inhaltsverzeichnis

Einleitung

Motivation

Die Anforderungen an die Netze für die Energieverteilungen steigen stetig an. Der Bedarf von den Kunden weist ein ständiges Wachstum auf und zugleich wird von den Netzbetreibern auch größtmögliche Effizienz gefordert. Letzteres ist aber eindeutig auch im Sinne der Betreiber, da geringere Verluste auch eine bessere Konkurrenzfähigkeit auf dem heiß umkämpften Markt mit sich bringen.

Eine exakte Berechnung des Netzes ist für eine fundierte Analyse des Bestandes, und richtiger Dimensionierung von neuen Anlagen, essentiell. Um diese Aufgaben mit akzeptablem Aufwand lösen zu können bietet sich die computergestützte Berechnung an. Produkte für diese Problemstellungen sind bereits auf dem Markt vorhanden, jedoch ist deren Funktionsumfang immens. Dementsprechend sind auch die Kosten für die Anschaffung und die nötige zusätzliche Schulung des Personals erheblich, und versperren kleineren Netzbetreibern damit diese Möglichkeiten.

An diesem Punkt entstand die Idee, dass der große Funktionsumfang häufig gar nicht unbedingt vorhanden sein müsste, um eine simple Analyse der Verluste und die Simulation von einfachen Fehlerfällen durchzuführen. Da es in dieser Spezialisierung kein bereits vorhandenes Produkt gab entschieden wir uns zur Entwicklung eines solchen im Zuge unserer Diplomarbeit.

Aufgabenstellung

Die Entwicklung einer Anwendung, welche Lastflussberechnungen und die Simulation von symmetrischen Kurzschlüssen ermöglicht.

Da die Zielgruppe auf kleinere Netzbetreiber beschränkt ist, kann auch der Spannungsbereich, welcher korrekt simuliert werden soll, auf den Nieder- und Mittelspannungsbereich eingeschränkt werden.

Die Anwendung soll eine intuitive Bedienung durch den Benutzer ermöglichen. Hierzu gehört die Steuerung mithilfe einer Maus, durch welche Komponenten frei platziert und verbunden werden sollen. Außerdem sollen auch gängige Merkmale, wie das Abspeichern eines Plans, Kopieren, Markieren, usw. dem Benutzer verfügbar sein.

Die beiden Hauptvorteile, welche eine computergestützte Berechnung gegenüber der manuellen hat, sollen in keiner Weise beeinträchtigt werden:

1. Keine Näherung in der Berechnung
2. Der schnelle Vergleich unterschiedlicher Varianten, zum Beispiel einer Anlagenerweiterung

Der zweite Punkt bedeutet, dass die Berechnung innerhalb von Sekunden erfolgen muss, da nur dann der Benutzer in wirklich angenehmer Zeit den Einsatz von zum Beispiel unterschiedlichen Kabelquerschnitten vergleichen kann.

Einführung in Netzberechnungen

Die Formeln im Folgenden wurden teilweise aus dem Werk 1 entnommen, jedoch angepasst. Im Falle der manuellen Berechnung wurde das Formelwerk auf die komplexe Rechnung umgesetzt, in der iterativen komplexen Lastflussberechnung fließen Optimierungen, für eine bessere Performance bei der computergestützten Berechnung, ein. Des weiteren sind in der Quelle keine Maßnahmen für eine bessere Konvergenz enthalten, weshalb dieser Punkt ausgearbeitet und erläutert wird.

Händische Berechnung

Die manuelle Berechnung von Netzen ist nur bis zu einem gewissen Grad sinnvoll. Bei größeren Netzen besteht im Endeffekt nur mehr die Möglichkeit das Knotenpunktpotentialverfahren einzusetzen.

Einseitig gespeiste Leitung mit einer Abnahme

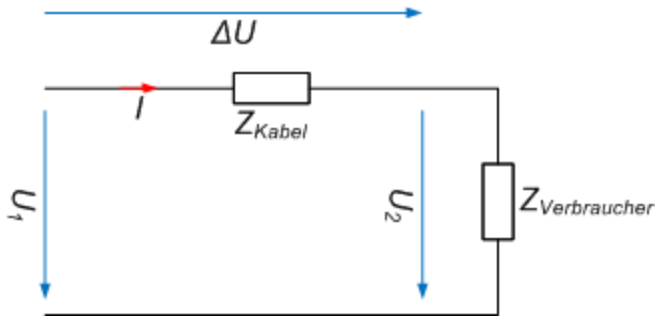


Abbildung 1: einseitig gespeiste Leitung mit einer Abnahme

$\Delta U \ldots Spannungsabfall\, an\, der\, Leitung$

$$\Delta U_{Stern} = I Z_{Kabel} = \frac{P}{\sqrt{3} U_2 \cos\varphi} Z_{Kabel}$$

Da der Unterschied zwischen U_1 und U_2 im Normalfall im Bereich von 5% liegt, kann statt U_2 in der obigen Formel die Nennspannung des Netzes bzw. U_1 eingesetzt werden.

$$\Delta U_{Stern} = \frac{P}{\sqrt{3} U_1 \cos\varphi} Z_{Kabel}$$

$$\Delta U = \sqrt{3}\, \Delta U_{Stern}$$

$$\Delta U = \frac{P}{U_1 \cos\varphi} Z_{Kabel}$$

Abbildung 2: Berechnung einseitig gespeiste Leitung

Einseitige gespeiste Leitung mit verteilten Abnahmen

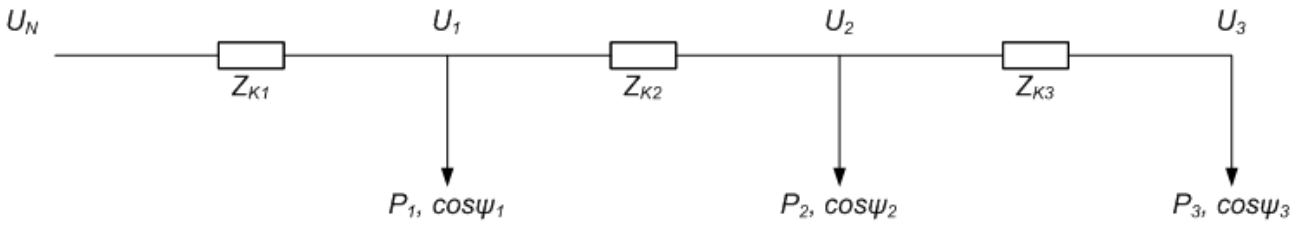


Abbildung 3: einseitig gespeiste Leitung mit verteilten Abnahmen

Die einzelnen Teile dieses Aufbaus können jeweils als einseitige gespeiste Leitung mit einer Abnahme betrachtet werden. So kann man zum Beispiel rechnen, wie wenn am Punkt 1 mit der Spannung U_1 die Leistung P_1, P_2 und P_3 zusammen entnommen werden würden. Die Verluste auf den Leitungen werden hier vernachlässigt, genauso wie auch bei der Berechnung der Spannungsabfälle nicht die vorher bereits abgefallene Spannung abgezogen wird, da sich diese wiederum maximal im Bereich von 5% bewegt.

Die entnommene Leistung am Punkt 1 ist die komplexe Summe der Scheinleistungen der dahinter liegenden Abnahmen.

$$\vec{S} = \frac{P}{\cos\varphi}$$

Der Winkel für S ist der jeweilige Winkel φ der Abnahme.

$$\Delta U_1 = \frac{Z_{K1}}{U_N} \sum_{n=1}^{3} \vec{S}_n$$

Die restlichen Spannungsabfälle berechnen sich analog zur obigen Formel, mit jeweils weniger Abnahmen.

Näherungsweise kann dann die Aussage getroffen werden, dass der gesamte Spannungsabfall über die Leitung die algebraische Summe der einzelnen Spannungsabfälle ist, da diese Spannungen untereinander üblicherweise nur geringfügige Winkel zueinander aufweisen.

$$\Delta U_{ges} = \sum_{n=1}^{3} \Delta U_n = \sum_{n=1}^{3} \frac{Z_{Kn}}{U_N} \sum_{i=n}^{3} \vec{S}_i$$

Abbildung 4: Berechnung einseitig gespeiste Leitung mit verteilten Abnahmen

Einseitige gespeiste, verzweigte Leitung mit verteilten Abnahmen

Dieser Typ eines Netzes hat einen sternförmigen Aufbau und lässt sich ähnlich wie die einseitig gespeiste Leitung mit verteilen Abnahmen berechnen. Um die oben beschriebene Formel anwenden zu können, wird zu Beginn eine Hauptleitung gewählt, an welcher dann die Abzweigungen durch Abnahmen ersetzt werden. Die Leistungen dieser Abnahmen entspricht dann der Summe der Leistungen, welche in dem zu ersetzenden Zweig entnommen werden, da man ja näherungsweise die Leitungsverluste vernachlässigt. Somit kann dann die folgende Formel angewendet werden:

$$\Delta U_{ges} = \sum_{n=1}^{x} \frac{Z_{Kn}}{U_N} \sum_{i=n}^{x} \vec{S}_i$$

$x \ldots Anzahl\, der\ Abnahmen\, an\, der\ Hauptleitung$

Abbildung 5: Einseitig gespeiste, verzweigte Leitung

Nach dem selben Schema können dann auch die Abzweigungen berechnet werden.

Zweiseitig gespeiste Leitung mit verteilten Abnahmen

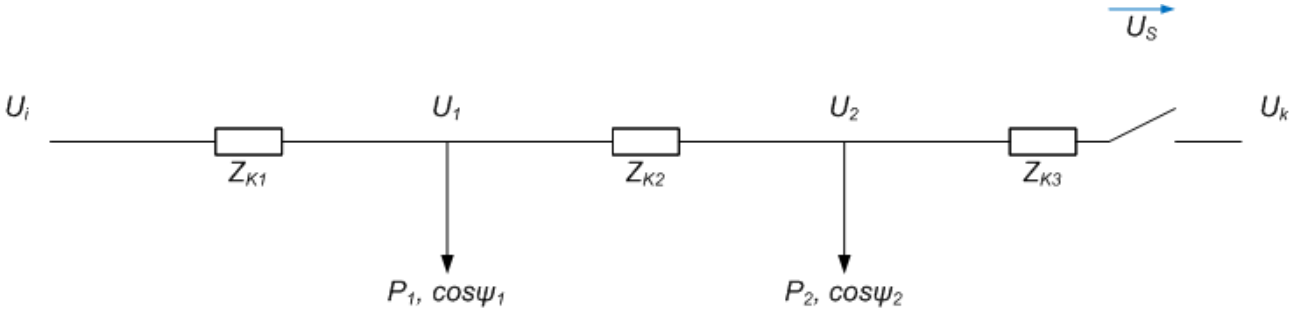


Abbildung 6: zweiseitig gespeiste Leitung

Um ein derartig aufgebautes Netz einfach berechnen zu können müssen mehrere Annahmen zutreffen:

- $U_i \approx U_k \approx U_N$
- $U_i - U_k \neq 0$ muss trotzdem möglich sein

Des weiteren werden wie bereits gehabt die Leitungsverluste vernachlässigt, da sie in der Regel erheblich kleiner als die abgenommenen Leistungen sind.

Im Gegensatz zu den bereits behandelten Aufbauarten ist bei der zweiseitig gespeisten Leitung der Lastfluss nicht bekannt. Deswegen wird der real nicht vorhandene Schalter geöffnet und die Leitung als einseitig gespeist betrachtet.

$$\Delta U_{ges} = \sum_{n=1}^{2} \frac{\vec{Z}_{Kn}}{U_N} \sum_{m=n}^{2} \vec{S}_m$$

Damit ergibt sich dann aber eine Spannungsdifferenz über den Schalter:

$$U_S = (U_i - U_k) + \Delta U_{ges}$$

Um diese Spannungsdifferenz ausgleichen zu können wird, sobald der Schalter geschlossen wird, muss eine Ausgleichsleistung S_k vom Punkt k weg in die Leitung fließen. Da aber bereits alle Abnahmen mit Leistung vom Punkt i versorgt werden kann man sich vorstellen, dass diese Leistung in i abgenommen wird.

$$U_S = \frac{\vec{S}_k}{U_N}(\vec{Z}_{K1} + \vec{Z}_{K2} + \vec{Z}_{K3}) = \frac{\vec{S}_k}{U_N} \vec{Z}_{Kges}$$

Führt man nun diese drei Gleichungen zusammen kann man S_k frei stellen und berechnen.

$$\frac{\vec{S}_K}{U_N}\vec{Z}_{Kges}=(U_i-U_k)+\frac{1}{U_N}\sum_{n=1}^{2}\vec{Z}_{Kn}\sum_{m=n}^{2}\vec{S}_m$$

$$\vec{S}_K=\frac{(U_i-U_k)U_N+\sum_{n=1}^{2}\vec{Z}_{Kn}\sum_{m=n}^{2}\vec{S}_m}{\vec{Z}_{Kges}}$$

Beziehungsweise allgemeiner, mit x Abnahmen:

$$\vec{S}_K=\frac{(U_i-U_k)U_N+\sum_{n=1}^{x}\vec{Z}_{Kn}\sum_{m=n}^{x}\vec{S}_m}{\vec{Z}_{Kges}}$$

Abbildung 7: Lastflussberechnung zweiseitig gespeist

Die Ausgleichsleistung, welche in den Punkt i fließt, muss dementsprechend von der von i zugeführten Leistung abgezogen werden. Damit ist die Leistungsverteilung bekannt und das System kann behandelt werden, wie wenn es zweimal einseitig gespeist werden würde.

$$\vec{S}_i=\sum_{n=1}^{x}\vec{S}_n-\vec{S}_k$$

Die Aufteilung in die zwei getrennten Systeme erfolgt an dem Punkt, welcher von beiden Seiten gespeist wird. Um diesen zu ermitteln, beginnt man auf einer Seite und zieht dann jeweils die abgenommenen Leistungen von der eingespeisten ab. Sobald man im Laufe dieses Vorgangs keine eingespeiste Leistung mehr übrig hat, ist der Punkt erreicht, der von beiden Seiten gespeist wird. Dessen abgenommene Leistung wird dann so aufgeteilt, dass die Summe der Leistungen der beiden einseitig gespeisten Systeme 0 ergibt und dann kann an diesem Punkt die Leitung in Gedanken aufgetrennt werden.

Zweiseitig gespeiste Leitung ohne Abnahmen

Dieser Spezialfall wird bei der Berechnung von vermaschten Netzen häufig benötigt. Er unterscheidet sich zum Regelfall nur dadurch, dass die Summe der abgenommenen Leistungen 0 ist. Dadurch vereinfacht sich die Berechnung der Ausgleichsleistung wie folgt:

$$\vec{S}_K=\frac{(U_i-U_k)U_N}{\vec{Z}_{Kges}}$$

Iterative komplexe Lastflussberechnung

Bei den bisher gezeigten Methoden zur Berechnung ging man immer von einem bestimmten Aufbau des zu berechnenden Netzes aus. Außerdem vernachlässigte man immer die Verlustleistungen über die Leitungen, das heißt, die Summe der zugeführten Leistungen war gleich der abgenommenen. Sollte eine dieser Einschränkungen nicht zulässig sein muss eine exaktere Methode zur Berechnung verwendet werden. Um die Berechnung computergestützt durchzuführen bietet sich ein iteratives Verfahren an.

Hierfür wird das Netz in Netzknoten unterteilt, welche untereinander verbunden sind. Jeder dieser Netzknoten hat die Eigenschaft, dass er eine Scheinleistung bezieht. Um auch Scheinleistung in das Netz liefern zu können wird diese mit einem negativen Vorzeichen behaftet.

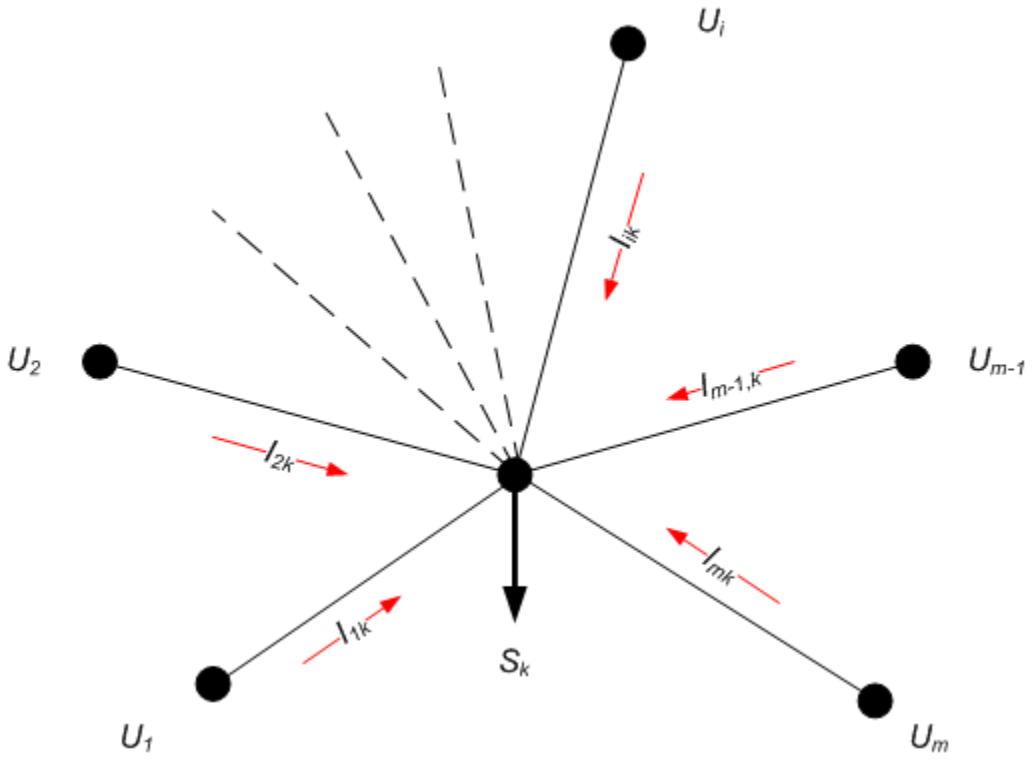


Abbildung 8: Schema eines Knotens

Der Einfachheit halber wird in den folgenden Formeln bei den Strömen und Spannungen immer der Hinweis auf Stern-Größen weg gelassen.

$$\vec{S}_k = 3\,\vec{U}_k \vec{I}_k^{\,\circ}$$

Diese entnommene Leistung wird von den angeschlossenen Knotenpunkten 1 bis m geliefert:

$$\vec{I}_k^{\,\circ} = \vec{I}_{1k}^{\,\circ} + \vec{I}_{2k}^{\,\circ} + \ldots + \vec{I}_{mk}^{\,\circ} = \sum_{i=1}^{m} \vec{I}_{ik}^{\,\circ}$$

$$\vec{S}_k = 3\,\vec{U}_k \sum_{i=1}^{m} \vec{I}_{ik}^{\,\circ}$$

Diese Ströme über die Leitungen verursachen jedoch Spannungsabfälle:

$$\vec{I}_{ik} = \frac{\vec{U}_i - \vec{U}_k}{\vec{Z}_{ik}} = (\vec{U}_i - \vec{U}_k)\,\vec{Y}_{ik}$$

In der Formel wird der konjugiert komplexe Strom benötigt:

$$\vec{I}_{ik}^{\,\circ} = (\vec{U}_i - \vec{U}_k)^{\circ}\,\vec{Y}_{ik}^{\,\circ}$$

Diesen kann man wieder in der im Knotenpunkt k abgenommenen Leistung einsetzen:

$$\vec{S}_k = 3\,\vec{U}_k \sum_{i=1}^{m} (\vec{U}_i - \vec{U}_k)^{\circ}\,\vec{Y}_{ik}^{\circ}$$

Da man das komplex konjugieren aufteilen kann lässt sich der Faktor auf die Summe aufteilen. Des weiteren kann man auch $\vec{U}_k\,\vec{U}_k^{\circ} = U_k^2$ einsetzen.

$$\vec{S}_k = 3\,\vec{U}_k \sum_{i=1}^{m} \vec{U}_i^{\circ}\,\vec{Y}_{ik}^{\circ} - 3\,U_k^2 \sum_{i=1}^{m} \vec{Y}_{ik}^{\circ}$$

Aufgrund dessen, dass in der Formel die gesuchte Größe, U_k, in komplexer und rein reeller Form vorkommt setzt man für die komplexen Größen die Real- und Imaginärteile ein:

$$\vec{Y}_{ik} = Y_{ikRe} + jY_{ikIm}$$
$$\vec{U}_i = U_{iRe} + jU_{iIm}$$

Beziehungsweise man setzt für die konjugiert komplexen Werte ein:

$$\vec{Y}_{ik}^{\circ} = Y_{ikRe} - jY_{ikIm}$$
$$\vec{U}_i^{\circ} = U_{iRe} - jU_{iIm}$$

Der Übersicht halber wird jedoch getrennt eingesetzt. Der erste Summand ergibt dementsprechend folgende Form:

$$3\sum_{i=1}^{m} \vec{U}_i^{\circ}\,\vec{Y}_{ik}^{\circ} = 3\sum_{i=1}^{m} (U_{iRe} - jU_{iIm})(Y_{ikRe} - jY_{ikIm}) = 3\sum_{i=1}^{m} (U_{iRe}\,Y_{ikRe} - U_{iIm}\,Y_{ikIm} - j\,U_{iRe}\,Y_{ikIm} - j\,Y_{ikRe}\,U_{iIm})$$

$$3\sum_{i=1}^{m} \vec{U}_i^{\circ}\,\vec{Y}_{ik}^{\circ} = 3\sum_{i=1}^{m} (U_{iRe}\,Y_{ikRe} - U_{iIm}\,Y_{ikIm}) - j\,3\sum_{i=1}^{m} (U_{iRe}\,Y_{ikIm} + Y_{ikRe}\,U_{iIm}) = A - j\,B$$

Um die Berechnung vereinfachen zu können werden die Werte für Realteil und Imaginärteil einmal vor jedem Durchgang berechnet.

Der zweite Summand besteht nur aus den Werten der angeschlossenen Leitungen und ist damit bei jedem Durchlauf gleich.

$$3\sum_{i=1}^{m} \vec{Y}_{ik}^{\circ} = 3\sum_{i=1}^{m} (Y_{ikRe} - jY_{ikIm}) = 3\sum_{i=1}^{m} Y_{ikRe} - j\,3\sum_{i=1}^{m} Y_{ikIm} = C - j\,D$$

Zusätzlich hierzu muss man auch U_k aufspalten:

$$\vec{U}_k = U_{kRe} + j\,U_{kIm}$$
$$U_k^2 = \left|U_{kRe} + j\,U_{kIm}\right|^2 = \sqrt{U_{kRe}^2 + U_{kIm}^2}^{\,2} = U_{kRe}^2 + U_{kIm}^2$$

Mit diesen Formeln ist es nun möglich die gesamte rechte Seite in reale und imaginäre Anteile zu trennen. Das selbe ist auch auf der linken Seite der Gleichung möglich:

$$\vec{S}_k = P_k + j\,Q_k$$

$$\vec{S}_k = P_k + jQ_k = (U_{kRe} + j\,U_{kIm})(A - jB) - (U_{kRe}^2 + U_{kIm}^2)(C - j\,D)$$

$$P_k + jQ_k = (U_{kRe}\,A + U_{kIm}\,B - U_{kRe}^2\,C - U_{kIm}^2\,C) + j(U_{kIm}\,A - U_{kRe}\,B + U_{kRe}^2\,D + U_{kIm}^2\,D)$$

$$P_k = U_{kRe}\,A + U_{kIm}\,B - U_{kRe}^2\,C - U_{kIm}^2\,C$$

$$Q_k = U_{kIm} A - U_{kRe} B + U_{kRe}^2 D + U_{kIm}^2 D$$

In diesen Gleichungen kommen die gesuchten Größen nur mehr in quadratischer Form vor:

$$0 = -C U_{kRe}^2 + A U_{kRe} + (B U_{kIm} - C U_{kIm}^2 - P_k)$$

$$0 = D U_{kIm}^2 + A U_{kIm} + (D U_{kRe}^2 - B U_{kRe} - Q_k)$$

Damit ergeben sich der reale und imaginäre Teil der gesuchten Knotenpunktspannung folgend:

$$U_{kRe} = \frac{-A \pm \sqrt{A^2 + 4C(B U_{kIm} - C U_{kIm}^2 - P_k)}}{-2C} = \frac{A}{2C} \pm \sqrt{\frac{A^2}{4C^2} + \frac{B U_{kIm} - C U_{kIm}^2 - P_k}{C}}$$

$$U_{kRe} = \frac{A}{2C} \pm \sqrt{\left(\frac{A}{2C}\right)^2 + \frac{B U_{kIm}}{C} - U_{kIm}^2 - \frac{P_k}{C}}$$

$$U_{kIm} = \frac{-A \pm \sqrt{A^2 - 4D(D U_{kRe}^2 - B U_{kRe} - Q_k)}}{2D} = \frac{-A}{2D} \pm \sqrt{\frac{A}{2D}^2 - \frac{D U_{kRe}^2 - B U_{kRe} - Q_k}{D}}$$

$$U_{kIm} = \frac{-A}{2D} \pm \sqrt{\left(\frac{A}{2D}\right)^2 - U_{kRe}^2 + \frac{B U_{kRe}}{D} + \frac{Q_k}{D}}$$

Wie man erkennen kann hängen diese Formeln jedoch noch jeweils vom anderen Teil der Knotenpunktspannung ab, der Realteil vom Imaginärteil und umgekehrt. Hier greift dann der iterative Ansatz dieser Methode an. Die Berechnung wird mit beliebigen Werten gestartet und dann mehrmals durchgeführt, wobei man bei den Abhängigkeiten die alten Werte einsetzt. Dadurch nähert sich das Ergebnis der Berechnung dem tatsächlichen beliebig nahe an. Um möglichst schnell genaue Lösungen zu erhalten bietet es sich an sinnvolle Startwerte zu wählen, zum Beispiel für den Realteil 90% der Nennspannung und für den Imaginärteil 0V.

Es ist aber zu beachten, dass es sich hierbei nur um Sternspannungen handelt!

$$U_{kRe} = \frac{A}{2C} + \sqrt{\left(\frac{A}{2C}\right)^2 + \frac{B U_{kIm}}{C} - U_{kIm}^2 - \frac{P_k}{C}}$$

$$U_{kIm} = -\frac{A}{2D} - \sqrt{\left(\frac{A}{2D}\right)^2 - U_{kRe}^2 + \frac{B U_{kRe}}{D} + \frac{Q_k}{D}}$$

$$A = 3 \sum_{i=1}^{m} (U_{iRe} Y_{ikRe} - U_{iIm} Y_{ikIm}) \qquad B = 3 \sum_{i=1}^{m} (U_{iRe} Y_{ikIm} + Y_{ikRe} U_{iIm})$$

$$C = 3 \sum_{i=1}^{m} Y_{ikRe} \qquad D = 3 \sum_{i=1}^{m} Y_{ikIm}$$

Um die Berechnung computergestützt durchzuführen ist es von Vorteil, wenn man noch ein klein wenig substituiert, wodurch mehrfache Multiplikationen mit 3 vermieden werden können.

$$A = 3K$$
$$B = 3L$$
$$C = 3M$$
$$D = 3N$$

$$U_{kRe} = \frac{3K}{2*3M} + \sqrt{\left(\frac{3K}{2*3M}\right)^2 + \frac{3L U_{kIm}}{3M} - U_{kIm}^2 - \frac{P_k}{3M}} = \frac{K}{2M} + \sqrt{\left(\frac{K}{2M}\right)^2 + \frac{L U_{kIm}}{M} - U_{kIm}^2 - \frac{P_k}{3M}}$$

$$U_{kIm} = -\frac{3K}{2*3N} - \sqrt{\left(\frac{3K}{2*3N}\right)^2 - U_{kRe}^2 + \frac{3LU_{kRe}}{3N} + \frac{Q_k}{3N}} = -\frac{K}{2N} - \sqrt{\left(\frac{K}{2N}\right)^2 - U_{kRe}^2 + \frac{LU_{kRe}}{N} + \frac{Q_k}{3N}}$$

Wenn man nun wieder einsetzt ergeben sich die folgenden Formeln:

$$K = \sum_{i=1}^{m} (U_{iRe} Y_{ikRe} - U_{iIm} Y_{ikIm}) \qquad L = \sum_{i=1}^{m} (U_{iRe} Y_{ikIm} + Y_{ikRe} U_{iIm})$$

$$M = \sum_{i=1}^{m} Y_{ikRe} \qquad N = \sum_{i=1}^{m} Y_{ikIm}$$

$$U_{kRe} = \frac{K}{2M} + \sqrt{\left(\frac{K}{2M}\right)^2 + \frac{LU_{kIm}}{M} - U_{kIm}^2 - \frac{P_k}{3M}}$$

Abbildung 10: iterative Berechnung, Realteil

$$U_{kIm} = -\frac{K}{2N} - \sqrt{\left(\frac{K}{2N}\right)^2 - U_{kRe}^2 + \frac{LU_{kRe}}{N} + \frac{Q_k}{3N}}$$

Abbildung 11: iterative Berechnung, Imaginärteil

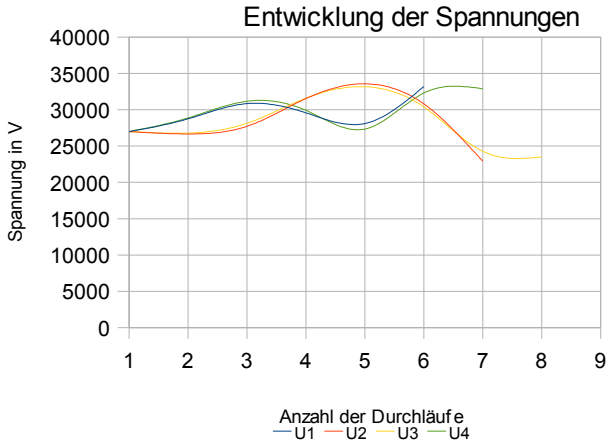


Abbildung 9: iterative Entwicklung der Spannungen, im Fehlerfall

Im Idealfall erhält man dann eine rasche Annäherung an das tatsächliche Ergebnis, und zwar ohne jegliche Näherungen. Die Genauigkeit der Berechnung kann rein theoretisch beliebig genau gewählt werden.

Die hiermit vorgestellte Methode nähert sich, wie bereits beschrieben, dem Ergebnis nur an. In ihrer Funktionsweise ähnelt sie damit dem newtonschen Näherungsverfahren und weist auch eine ähnliche Schwäche auf. Beim newtonschen Näherungsverfahren ist es möglich, wenn die Funktionskurve eine „Delle“ nach unten aufweist, dass das Ergebnis des Verfahrens nicht gegen eine Nullstelle bzw. gegen keinen Wert konvergiert. Ähnliches Verhalten kann auch bei der iterativen Berechnung eines Netzes entstehen. Diese Berechnung kann man sich aber besser als Funktion in Abhängigkeit von mehreren Variablen darstellen, der Einfachheit halber zum Beispiel den Spezialfall, dass ein Knoten nur von einem anderen abhängig ist. Damit ist die Funktion des Knotens nur von 2 Variablen abhängig, sich selber und dem angeschlossenen Knoten. Wenn man von dieser Funktion bei jeder Stelle die Abweichung vom tatsächlichen Wert berechnet (und davon noch den Betrag nimmt) ergibt sich eine Funktionsfläche, welche an einem Punkt (idealerweise) die Ebene berührt, welche die Achsen der abhängigen Variablen aufspannen. Während der iterativen Berechnung wandert nun idealerweise der Funktionswert nach und nach auf diesen Berührungspunkt zu. Es kann jedoch

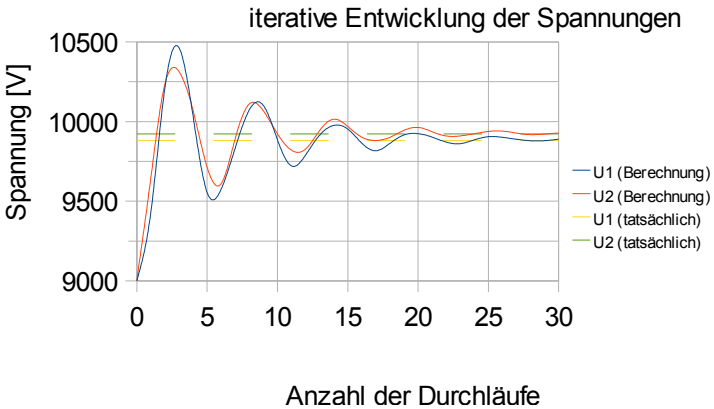


Abbildung 12: iterative Entwicklung der Spannungen, idealerweise

sein, dass diese Ebene eine Delle aufweist, womit das selbe Problem wie beim newtonschen Näherungsverfahren auftritt. Um die Funktion aus dieser Senke wieder herausheben zu können gibt es eine einfache Lösungsvariante. Das Problem wird nämlich dadurch beseitigt, dass nicht der berechnete neue Wert für den nächsten Iterationsschritt verwendet wird, sondern nur ein Prozentsatz der Abweichung (zum Beispiel 50%) vom vorigen hinzu addiert wird. Dies ermöglicht der Berechnung, dass sie selbsttätig aus der Senke wieder herausspringt. Ein Nachteil dabei ist aber, dass das Ergebnis langsamer konvergiert, wodurch mehr Berechnungsschritte nötig sind. Der angesprochene Parameter wird durch den Benutzer vorgegeben, womit dieser, falls die Berechnung in einen Fehler läuft, selbsttätig das Konvergenzverhalten anpassen kann.

Häufige Probleme in der Berechnung entstehen auch durch den Bezug von großen Leistungen durch einen Verbraucher. Dieser produziert, rechnerisch, damit einen sehr großen Strom, und liefert während einer Iteration einen Fehler. Um dies zu umgehen, wird die Leistung schrittweise angehoben. Im Zusammenspiel mit dem Parameter für den Übernahmefaktor, welcher zuvor erläutert wurde, ist es somit möglich, dass zu Beginn der Berechnung die oszillatorischen Ausgleichsvorgänge ablaufen, und dann allmählich die Spannungen der Knoten durch die steigenden Leistungen nach unten gezogen werden.

Knotenpunktpotentialverfahren

Die Leistung eines Knoten, die durch ihn entnommen wird, muss gleich der Leistungen sein, die zu dem Knoten zu fließen.

$$\vec{S}_{Entnahme} = \vec{S}_{1zu} + \vec{S}_{2zu} + \dots$$

Jede einzelne Leistung lässt sich, durch die näherungsweise Vernachlässigung des Spannungsabfalls, wie bereits bei der einseitig gespeisten Leitung, durch die Spannungsabfälle und Leitwerte der Verbindungen darstellen:

$$\vec{S}_{izu} = (\vec{U}_i - \vec{U}_j)\,\vec{U}_N\,\vec{\lambda}_{ij}$$

Stellt man nun diese Gleichungen für alle Knoten auf und formt sie um, erhält man ein lineares Gleichungssystem, welches in folgende Matrizengleichung umgewandelt werden kann:

$$Knotenpunktleitwerte * Knoten = Speisepunktleitwerte * Einspeisungen - 1/U_N * Leistungsmatrix$$

Wobei das Schema für das Befüllen der Matrizen folgend lautet:

In die Knotenpunktleitwertmatrix wird in die Hauptdiagonale jeweils die Summe aller Verbinungsleitwerte des jeweiligen Knotens gesetzt, wobei diese Matrix bei n (Entnahme-)Knoten die Dimensionen n x n besitzt, und jede Spalte wie Zeile für einen Knoten steht. An den restlichen Positionen steht der Verbindungsleitwert der Knoten, negativ, und sollte keine Verbindung bestehen 0. Dadurch wird in realistischen Netzen diese Matrix nur spärlich besetzt, was einen erheblichen Vorteil für die spätere Lösung der Gleichung bedeutet.

Die Knotenpunktleitwertmatrix besitzt so viele Zeilen wie Entnahmeknoten und Spalten wie Speiseknoten. In ihr werden wiederum die Leitwerte der Verbindung eingesetzt, jedoch positiv. Der damit zu multiplizierende Vektor der Speisespannungen enthält die jeweiligen Speisespannungen, wobei diese in einem Kurzschlussfall auch 0 sein können. Der Vektor der Leistungen wird, wie es der Name bereits sagt, mit den entnommenen Leistungen befüllt.

Für nähere Informationen bezüglich der Herleitung dieses Schemas sei auf die Quelle 1 verwiesen.

Zur Lösung der Gleichung kann das Gauß-Jordan-Verfahren angewandt werden, welches im Endeffekt darauf abzielt, durch Elimination die Knotenpunktleitwertmatrix zu einer Einheitsmatrix umzuformen. Damit entfällt die nicht besonders performante Berechnung der inversen Matrix.

Berechnung über mehrere Netzebenen

Für die Berechnung über mehrere Netzebenen wäre es möglich, alle Impedanzen auf eine bestimmte Spannungsebene bezogen zu berechnen. Da Transformatoren häufig aber nicht die Nennübersetzung einsetzen, sondern mit einer etwas höheren Spannung die Unterspannungsseite speisen, kann diese Methode nicht angewandt werden. Es besteht unseres Wissens nach keine Möglichkeit, eine abweichende Übersetzung in die Impedanz des Trafos einzubeziehen.

Um dieses Problem zu umgehen, wird jede Ebene auf ihrer Nennspannung berechnet. Die einzige Verbindung zwischen den Ebenen, die Transformatoren, besitzen in diesem Fall dann zwei Knoten, einen auf jeder Seite. Die beiden Knoten wissen, mit welcher Übersetzung sie zu dem jeweils anderen verbunden sind, und beziehen diese ein, wenn sie die Spannung des jeweils anderen ermitteln.

Symmetrische Kurzschlüsse

Symmetrische Kurzschlüsse (bzw. Erdkurzschlüsse) werden in der Berechnung durch Knoten mit einer festen Spannung von 0V repräsentiert. Die Platzierung von Kurzschlüssen kann in Kabeln erfolgen, da die Kurzschlussposition bei Bedarf auch auf den Anfang oder das Ende gesetzt werden kann.

Da ein Knoten mit 0V einen sehr hohen Unterschied und eine große Belastung des Netzes darstellt, ist es sehr wahrscheinlich, dass dessen Umgebung bereits in dem ersten Iterationsschritt einen Fehler produziert. Um dies zu vermeiden wird die Kurzschlussspannung langsam auf 0V abgesenkt.

Berechnung der Verluste

Verluste in einem Netz entstehen bei der Berechnung nur durch die Impedanzen zwischen den Knoten. Diese setzen sich aus den Netzimpedanzen der Einspeisungen, den Leitungen und den Transformatoren zusammen. Da die Knoten nach der Berechnung keine Informationen mehr darüber haben, wie sie entstanden sind, wird die Auswertung der Ergebnisse von den Komponenten übernommen. Jene sind nämlich in der Lage zu rekonstruieren, von welchen Knoten bzw. Impedanzen sie im berechneten Netz dargestellt werden.

Einspeisung und Kabel

Die Verluste einer Einspeisung, sowie auch die eines Kabels, beziehen sich auf deren Impedanz und den Spannungen der Knoten, zwischen denen sie liegen. Da es häufig so ist, dass beide

Impedanzen zusammen zwischen einem Knoten vorkommen, wird zur Berechnung der Verluste der Strom ermittelt, welcher zwischen den Knoten fließt:

$$\vec{I}_{12} = \frac{\vec{U}_1 - \vec{U}_2}{\vec{Z}_{12}}$$

Wobei sich die gesamte Impedanz folgend zusammensetzt:

$$\vec{Z}_{12} = \vec{Z}_{Kabel} + \vec{Z}_{Netz}$$

$$Z_{Kabel} = l(R' + jX')$$

$$Z_{NetzIm} = \frac{\frac{U_N^2}{S}}{\sqrt{1 + \frac{R^2}{X}}}$$

$U_N \ldots Nennspannung\ , bei\ der\ eingespeist\ wird$

$S \ldots Kurzschlussleistung\ des\ vorgelagerten\ Netzes$

$\frac{R}{X} \ldots Verhältnis\ der\ Leitungsbeläge\ des\ vorgelagerten\ Netzes$

$$Z_{NetzReal} = Z_{NetzIm}\frac{R}{X}$$

Die gesamte Verlustleistung setzt sich dann aus denen der drei Phasen zusammen. Im symmetrischen Fall kann die Berechnung vereinfacht werden:

$$\vec{S}_{Verlust} = \sqrt{3}(\vec{U}_1 - \vec{U}_2)\vec{I}_{12}$$

Abbildung 13:
Berechnung der Verluste zwischen zwei Knoten

Transformator

Die Verlustleistung des Transformators ergibt sich aus den Knotenspannungen der US- und OS-Seite. Im Grunde verhält sich der Transformator wie ein Kabel, es müssen nur die Spannungen auf die selbe Ebene gebracht werden, mithilfe der Übersetzung ü.

Softwareentwurf

Benutzeroberfläche

Die Benutzeroberfläche besteht im Grunde genommen nur aus einem Bild, welches mithilfe der Maus manipuliert werden kann. Durch Bewegungen der Maus, beziehungsweise Klicks, wird der Plan manipuliert.

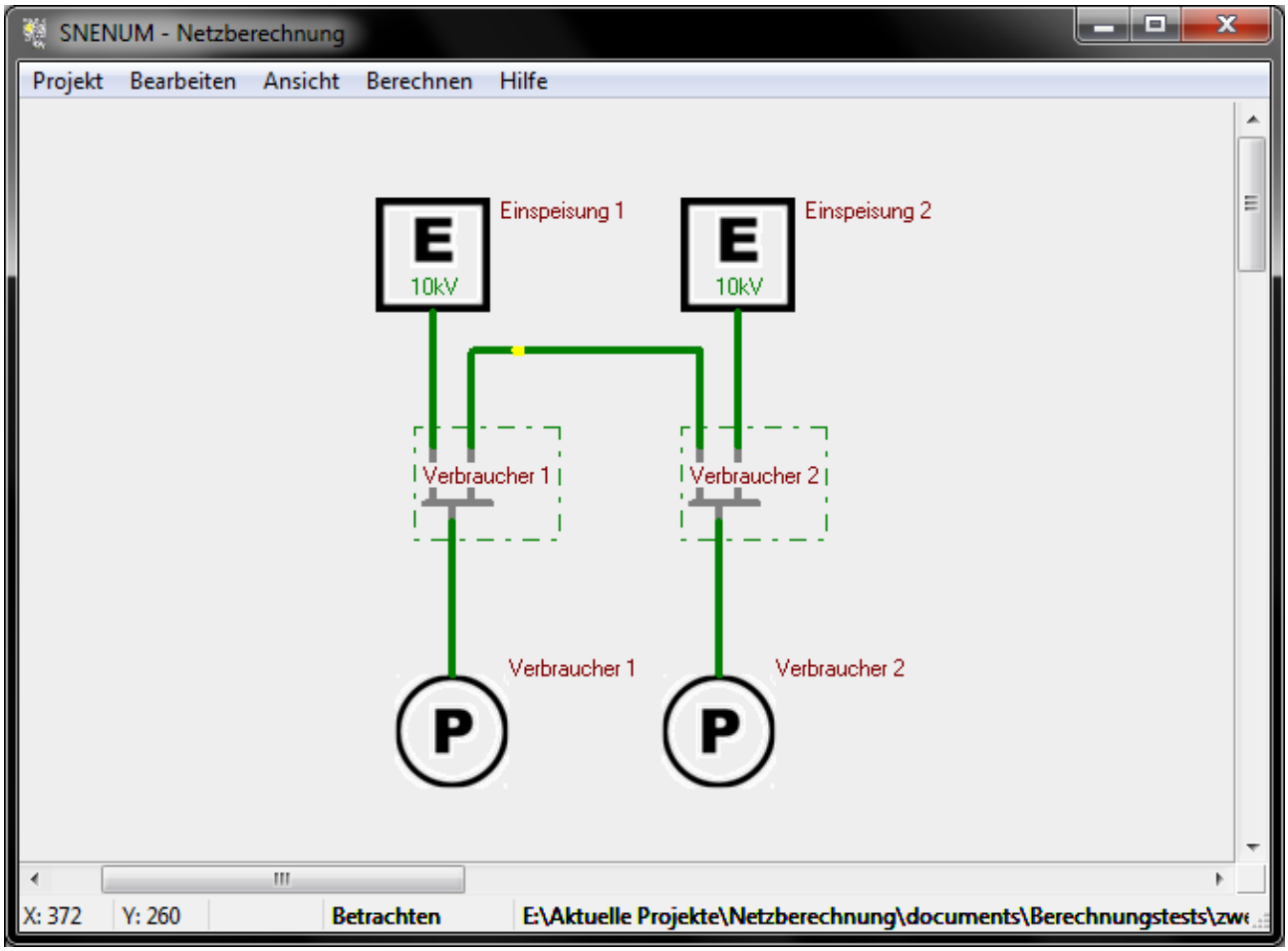


Abbildung 14: Benutzeroberfläche

Alle Objekte, die der Benutzer somit erstellt und manipuliert, werden in einem Array zusammengefasst. Da sich die Objekte teilweise sehr stark unterscheiden, und auch in sehr unterschiedliche Bereiche unterteilt werden können, werden sie alle von abstrakten Basen abgeleitet. Als Beispiel im Folgenden sehen sie die Vererbungshierarchie eines Verbrauchers. Wie man erkennen kann wird Mehrfachvererbung eingesetzt. Da es zum Diamondproblem kommt muss virtuelle Mehrfachvererbung[1] angewandt werden.

1 Eine Erklärung findet sich in Kapitel 4 der Quelle 2

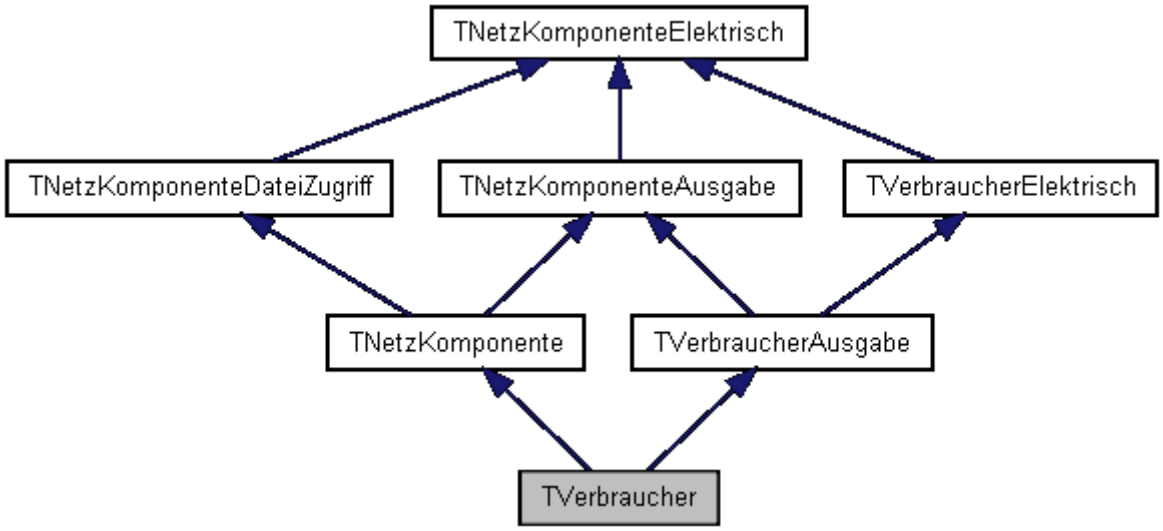


Abbildung 15: Vererbungshierarchie TVerbraucher

Mithilfe dieser Hierarchie werden die wesentlichen Aspekte einer Komponente jeweils in einen eigenen Zweig ausgelagert und die spezifischen Implementierungen bleiben übersichtlich.

Bedienung

Wie schon angesprochen erfolgt die Bedienung des Planes groß teils mit der Maus. Um dies übersichtlicher zu programmieren und für Erweiterungen jeglicher Art vorbereitet zu sein, haben wir uns für das Zustands[2] Entwurfsmuster entschieden. Im folgenden sind nun alle wichtigen Zustände aufgelistet.

Betrachten

Dies ist der standardmäßig aktive Zustand. Man könnte ihn auch als Ruhezustand bezeichnen. Im Konstruktor dieses Zustandes wird *NeuZeichnenGesamt* aufgerufen.

Platzieren

Wenn in diesen Zustand gewechselt wird, wird im Konstruktor der Speicher für die zu platzierenden Komponenten reserviert und nachher werden sie in das Array aller am Plan vorhandenen Komponenten mit eingefügt. Bei vorzeitiger Beendigung des Vorganges werden diese Komponenten wieder aus der Liste entfernt.

Verschieben

Alle angewählten Komponenten werden in einen temporären Vektor von Zeigern gespeichert. Wenn die Maustaste bewegt wird werden die Koordinaten denen der Maus angepasst. Bei Abbruch wird der letzte Befehl rückgängig gemacht und anschließend aus der History gelöscht.

2 Eine Erklärung zu diesem Entwurfsmuster gibt es aus der 3. Quelle

Verbinden

In diesen Zustand wird nach Anwahl eines noch nicht besetzten Anschlusses gesprungen. Im Konstruktor wird Speicher für ein neues Kabel reserviert und anschließend dem Vektor von den Zeigern der Komponenten hinzugefügt. Der Verbindungsvorgang kann durch zwei Arten beendet werden, einerseits durch Abbruch, oder durch erfolgreiches Beenden. Beim Beenden werden die elektrischen Verbindungen zwischen den jeweiligen Komponenten erstellt. Dies bedeutet: Verbindung zwischen der *Komponente A* und dem *Kabel* und die Verbindung zwischen der *Komponente B* und dem *Kabel* werden erstellt.

Messen

In diesem Zustand werden lediglich die Ergebnisse der jeweiligen Komponente angezeigt. Hierzu haben wir eine eigene Komponente erstellt und in diese geben wir die Ergebnisse aus, welche jede Komponente für sich abgespeichert hat. Da es nur eine temporäre Komponente ist wird diese nicht dem großen Vektor der Komponenten hinzugefügt.

Markieren

Hierbei wird nach jeder Maus-Bewegung abgefragt, welche Komponenten sich in dem gezeichneten Rechteck befinden und diese anschließend als markiert auf dem Plan dargestellt. Ebenso fällt die Entscheidung, ob diese gelöscht, verschoben oder kopiert werden, noch in diesem Zustand.

Berechnen

Um die Veränderung des Planes während der Berechnung zu verhindern haben wir diesen Zustand eingefügt. Er ist lediglich während der Berechnung aktiv und unterbindet jegliche Interaktion des Benutzers mit dem Plan.

Löschen

Wenn nach dem Markieren in diesen Zustand gewechselt wird, werden die zuvor markierten Komponenten in dem Konstruktor gelöscht. Nach erfolgreichem Abschluss dieses Löschvorganges wird wieder in den Ausgangszustand zurückgesprungen.

Bearbeiten

Wenn die elektrischen Daten einer Komponente verändert werden, ist dieser Zustand aktiv. Somit ist es wiederum möglich, wie schon bei der Berechnung, die Änderung des Planes zu unterbinden und somit Fehlern, welche dadurch entstehen könnten, zuvorzukommen. Wir verwenden ihn allerdings auch für andere Formulare.

Kabel verschieben

Wie der Name schon verrät, werden in diesem Zustand nur Kabel verschoben. Wenn keine Eckpunkte im Kabel vorhanden sind, wird dieser Zustand zugleich wieder verlassen, andernfalls wird eine Seite des Kabels bei Bewegung der Maus verschoben.

Zustandsdiagramm

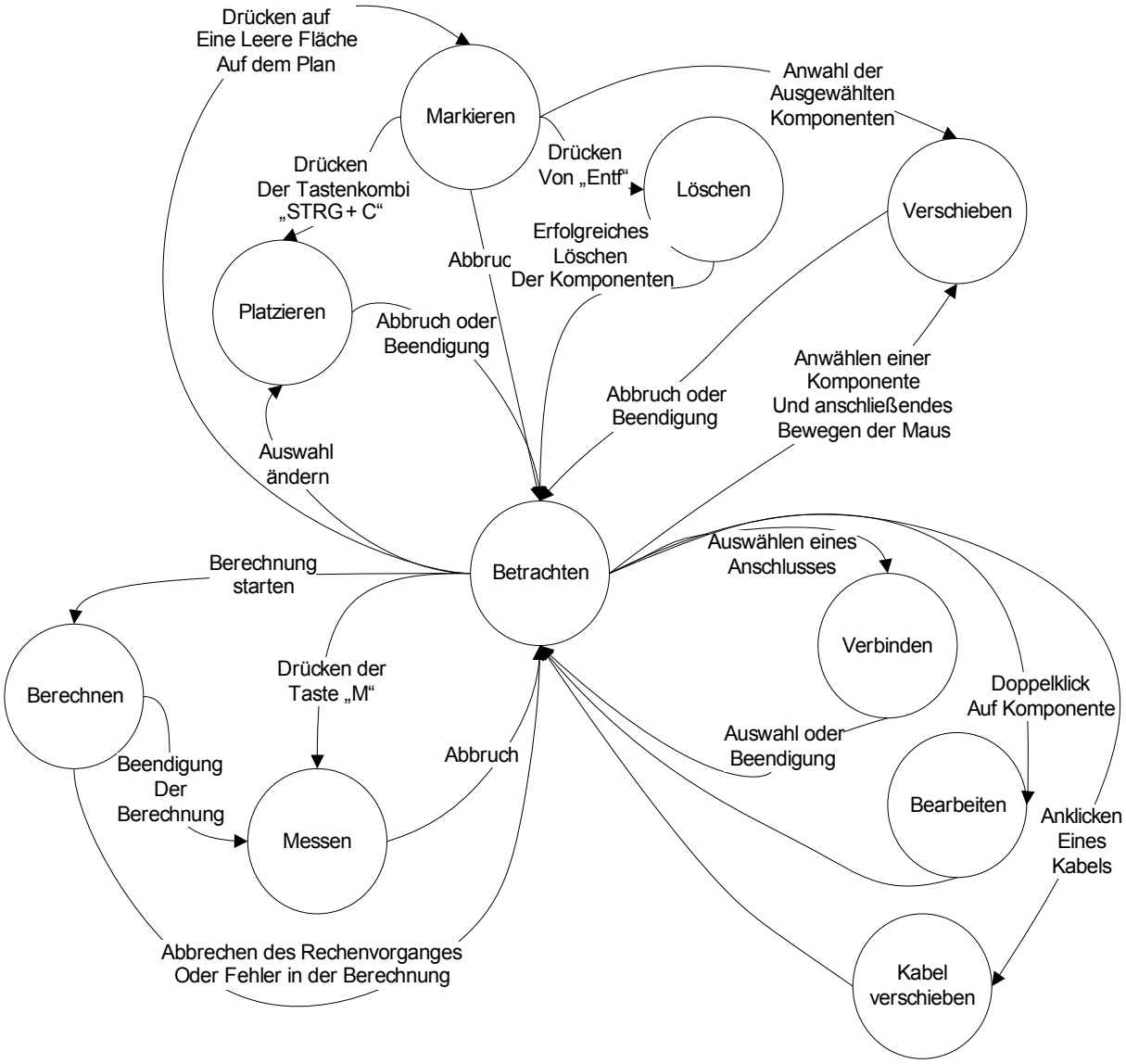


Abbildung 16: Diagramm, zur Veranschaulichung der Zustände

Oben abgebildet ist eine grobe Übersicht der Zustände und deren Abhängigkeiten untereinander. Es soll nur veranschaulichen wieviele Zustände wir haben und wie schwierig und unübersichtlich die Umsetzung dieser ganzen Fälle wäre, wenn man diese in einer ewig langen Reihe an IF-Abfragen realisieren würde. Einem jeden dürfte ganz schnell klar werden, dass es nicht sinnvoll umsetzbar sein dürfte.

Komponenten

Die unterschiedlichen Typen von Komponenten sind alle von abstrakten Basen abgeleitet, wodurch sie gleich behandelt werden können. Zudem wurde auch eine Unterteilung in den Part der Ausgabe, der elektrischen Daten und in den des Dateizugriffs gewählt.

Im Anschluss folgt hier nun ein Diagramm, auf welchem die Klassenhierarchie von *TNetzKomponente* zu sehen ist.

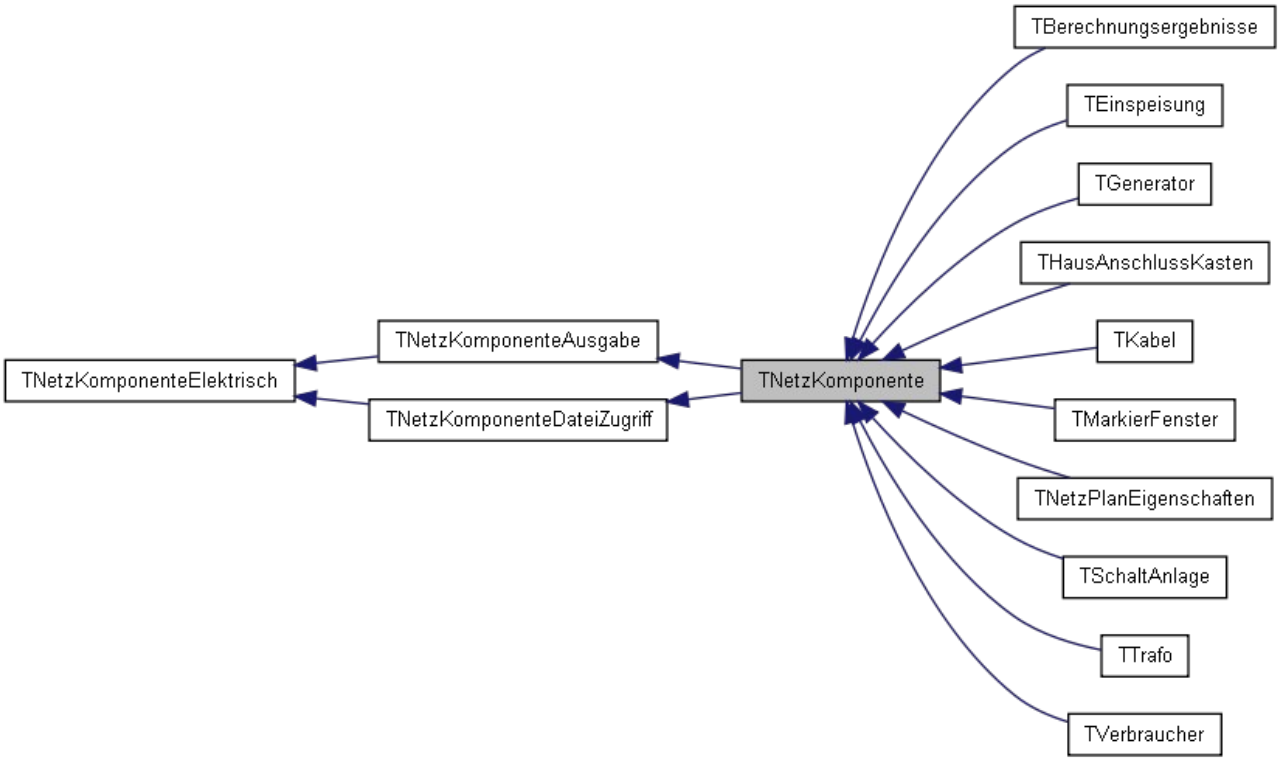


Illustration 17: Klassenhierarchie von TNetzKomponente

Wie hier in dieser Abbildung unschwer zu erkennen ist, erben alle Komponenten von *TNetzKomponente.* So haben wir die Möglichkeit, trotz unterschiedlicher Komponenten alle gleich zu behandeln. Somit müssen beispielsweise allgemeine elektrische Daten wie die Nennspannung nicht für jede Komponente für sich definiert werden, sondern nur einmal.

Verbraucher und Einspeisung

Der Verbraucher und die Einspeisung stellen die primitivsten Komponenten dar, sowohl elektrisch, als auch von der Ausgabe her. Die Anzahl der Ein- und Ausgänge ist beschränkt auf einen, bzw. gar keine. Der Verbraucher übergibt seinem Knoten zudem noch seine bezogene Leistung, und die Einspeisung setzt eine feste Spannung.

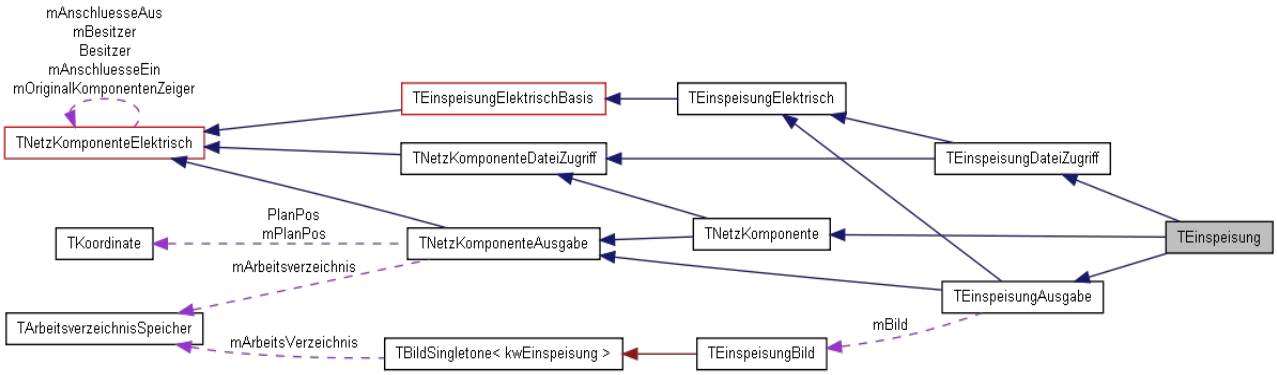
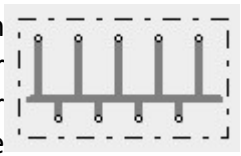


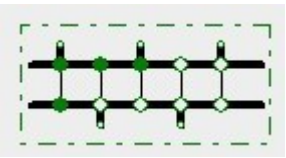
Illustration 18: Abhängigkeiten der Einspeisung

Hausanschlusskasten



Hausanschlusskästen sind die primitivsten Verteiler, welche im Plan verwendet werden können. Sie stellen ebenfalls einen Knoten in der Berechnung dar, welcher keine besonderen Eigenschaften aufweist. Der repräsentative Knoten besitzt jedoch als angeschlossene Knoten all jene Komponenten, welche auch der Benutzer verbunden hat.

Schaltanlage



Die Schaltanlage besteht im Hintergrund aus mehreren Hausanschlusskästen und fängt alle Anschlussversuche ab. Aufgrund der Information, welche sie vom Benutzer bekommt, welche Sammelschienen mit welchen Ein- bzw. Ausgängen verbunden sind, werden dann die Sammelschienen mit den extern angeschlossenen Komponenten verbunden.

Transformator

Transformatoren sind die einzigen Komponenten welche in der Lage sind unterschiedliche Netzebenen zu verbinden. Bei allen anderen Versuchen Netzebenen zu verbinden, welche ungleich sind, wird der Benutzer von der Anwendung eingeschränkt. Auch beim Transformator ist es nicht möglich beliebige Ebenen anzuschließen, sondern nur die beiden angegebenen an den jeweiligen Anschlüssen.

Berechnung der Knoten

Die Berechnung der Knoten erfolgt komplett getrennt von der Objekthierarchie, welche der Benutzer mit dem Plan bearbeitet. Die dementsprechend nötige Umwandlung in Objekte vom Typ *TNetzKnoten* wird von einer abstrakten Fabrik[3] durchgeführt:

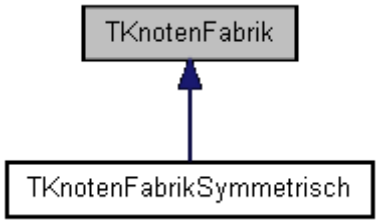


Abbildung 19: Vererbungshierarchie TKnotenFabrik

Diese Klasse wird aber auch nicht direkt vom TNetzPlan verwendet, sondern dieser erhält nur den Algorithmus zur Berechnung von dem Benutzer-Objekt übergeben. Damit ist die Berechnung rein prinzipiell nicht abhängig von der Art der Berechnung, wodurch eine spätere Erweiterung für zum Beispiel unsymmetrische Berechnungen leicht implementiert werden kann.

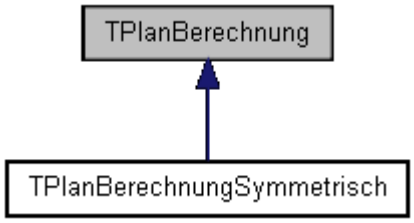


Abbildung 20: Vererbungshierarchie TPlanBerechnung

Die Berechnung an sich wurde zwar rein prinzipiell so ausgeführt, dass sie überladen werden kann, aber eigentlich sollten hier keine Änderungen mehr nötig sein. Durch die modulare Konvertierung dürften alle möglichen, stationären, Berechnungsarten abgedeckt sein. Getrennt von der Berechnung erfolgt jedoch die Auswertung der Ergebnisse, da hierzu wieder spezifisch auf die Art der Berechnung eingegangen werden muss. Diese Funktion muss von allen Erben von *TPlanBerechnung* überladen werden.

Die Vererbungshierarchie von *TPlanBerechnung* wurde nach dem Entwurfsmuster der Strategie[4] entwickelt, und dementsprechend wird nur ein Objekt, dessen Basis *TPlanBerechnung* ist, dem *TNetzPlan* übergeben. Dieser führt dann die Berechnung durch, abhängig davon welche Strategie gewählt wurde.

3 Dieses Entwurfsmuster wird im Werk 3 näher erläutert
4 Eine exakte Definition dieses Musters findet sich in der Quelle 3

Die eigentliche Berechnung ist dann eine iterative Annäherung an das Ergebnis, welche solange durchgeführt wird, bis sich die Ergebnisse nicht mehr so stark, vom Benutzer einstellbar, von ihrem Mittelwert abweichen.

```
mAnzahlIterationen =    0;
do
{
      KnotenBerechnungBeendet =     0;
      mAnzahlIterationen++;

       for (unsigned  int i =  0; i < KnotenZuBerechnenAnzahl; i++)
            FehlerInBerechnung = FehlerInBerechnung ||
                                 !KnotenZuBerechnen[i]->BerechnungsSchritt();

       for (unsigned  int i =  0; i < KnotenZuBerechnenAnzahl; i++)
             if (KnotenZuBerechnen[i]->BerechnungsSchrittBeenden())
                  KnotenBerechnungBeendet++;

       for (unsigned  int i =  0; i < KnotenNichtBerechnenAnzahl; i++)
             if (KnotenNichtBerechnen[i]->BerechnungsSchrittBeenden())
                  KnotenBerechnungBeendet++;

       if (mZielFenster != NULL)
            PostMessage(mZielFenster,CM_BERECHNUNGSINFO,mAnzahlIterationen,          0);

} while
 (KnotenBerechnungBeendet < KnotenZuBerechnenAnzahl + KnotenNichtBerechnenAnzahl
  && !FehlerInBerechnung);
```

Abbildung 21: Implementierung der Berechnung

Jeder Knoten berechnet sich selber, abhängig von den Spannungen der angeschlossenen Knoten und der eigenen. Da es in der Berechnung zu Fehlern kommen kann, welche sich durch eine negative Wurzel zeigen, müssen die Diskriminanten zuvor überprüft werden.

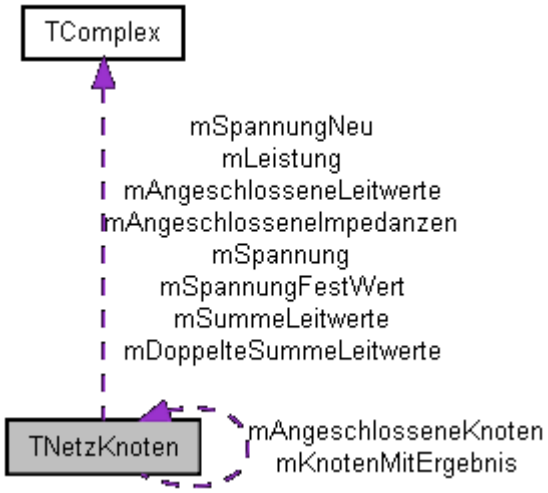


Abbildung 22: Zugehörigkeiten TNetzKnoten

Wie bereits in der Abbildung *Implementierung der Berechnung* zu sehen ist, wird nach jedem Berechnungsschritt eines Knotens auch jeweils eine Funktion namens *BerechnungsSchrittBeenden* aufgerufen. Diese sorgt unter anderem dafür, dass das neue Ergebnis für die Spannung übernommen wird. Es ist nicht möglich, dieses sofort im eigentlichen Berechnungsschritt zu

übernehmen, da ansonsten angeschlossene Knoten eventuell schon auf das neue Ergebnis zugreifen, und damit die Berechnung verfälschen. Außerdem wird nicht exakt das neue Ergebnis übernommen, sondern nur ein gewisser Prozentsatz der Differenz zum neuen. Damit erreicht man eine bessere Konvergenz der Knotenspannungen.

Des weiteren wird in *BerechnungsSchrittBeenden* auch ermittelt, ob die Abweichungen zu dem Mittelwert sich in einem gewissen Rahmen bewegen. Ist dieser Punkt erreicht, wird dem Aufrufer mitgeteilt, dass dieser Knoten fertig berechnet wäre.

Um eine bessere Performance in dieser Überprüfung zu erreichen, wird nicht tatsächlich der Mittelwert jedes Mal berechnet, und mit jedem Ergebnis, der zum Beispiel letzten 20 Iterationen, verglichen. Hierbei kommt die gleitende Mittelwert-Berechnung zum Einsatz, sowohl für das arithmetische Mittel, als auch für die Differenzen zu eben diesem.

```
//! Berechnung der Hilfsgrößen
for (unsigned  int i =  0; i < mAnzahlAngeschlossene; i++)
{
     TempSpannung = mAngeschlosseneKnoten[i]->mSpannung;
     TempLeitwert = mAngeschlosseneLeitwerte[i];

     TempSpannung *= mUebersetzungZuKnoten[i];

     HilfsGroesseK +=        TempSpannung.Real * TempLeitwert.Real -
                             TempSpannung.Im * TempLeitwert.Im;
     HilfsGroesseL +=        TempSpannung.Real * TempLeitwert.Im +
                             TempLeitwert.Real * TempSpannung.Im;
}

//! Realteil berechnen
Diskriminante = (HilfsGroesseK / mDoppelteSummeLeitwerte.Real) *
     (HilfsGroesseK / mDoppelteSummeLeitwerte.Real) +
     HilfsGroesseL * mSpannung.Im / mSummeLeitwerte.Real -
     mSpannung.Im * mSpannung.Im - mLeistung.Real / (       3 * mSummeLeitwerte.Real);

if (Diskriminante <    0)
      return  false;

mSpannungNeu.Real = HilfsGroesseK / mDoppelteSummeLeitwerte.Real + sqrt(Diskriminante);

//! Imaginärteil berechnen
Diskriminante = (HilfsGroesseK / mDoppelteSummeLeitwerte.Im) *
     (HilfsGroesseK / mDoppelteSummeLeitwerte.Im) -
     mSpannung.Real * mSpannung.Real +
     HilfsGroesseL * mSpannung.Real / mSummeLeitwerte.Im +
     mLeistung.Im / (   3 * mSummeLeitwerte.Im);

if (Diskriminante <    0)
      return  false;

mSpannungNeu.Im = - HilfsGroesseK / mDoppelteSummeLeitwerte.Im - sqrt(Diskriminante);

return  true;
```

Abbildung 23: Implementierung der Berechnung eines Knotens

$$Mittelwert_{i+1} = Mittelwert_i(1 - Gewichtung) + NeuerWert * Gewichtung$$

Abbildung 24: Berechnung des gleitenden Mittelwerts

$$Abstand_{i+1} = Abstand_i(1 - Gewichtung) + NeuerAbstand * Gewichtung$$

Abbildung 25: Berechnung der Differenz zum Mittelwert

Ausgabe der Ergebnisse

Die Ergebnis Ausgabe muss hier noch unterteilt werden, da wir uns dazu entschlossen, einerseits die Ergebnisse direkt auf dem Plan anzeigen zu lassen, und andererseits dem Benutzer zusätzlich auch eine Möglichkeit der Ergebnisausgabe in tabellarischer Form zu bieten.

Eine weitere Differenzierung muss getroffen werden, da bei einer anderen Berechnungsmethode als der symmetrischen die Ausgabe ebenfalls angepasst werden muss. Hierzu haben wir eben eine Basisklasse *TplanBerechnung,* von welcher *TPlanBerechnungSymmetrisch* erbt, wie aus der Abbildung *Vererbungshierarchie TPlanBerechnung* hervorgeht. In *TPlanBerechnungSymmetrisch* wird die Funktion Auswertung aus *TPlanBerechnung* überladen. Dadurch wäre die Ergebnis-Auswertung anderer Berechnungsarten sehr leicht zu implementieren und steht somit für allerlei neues offen und lädt schier dazu ein, die Implementierung anderer Berechnungsarten ebenfalls umzusetzen und in die Applikation mit einzubinden.

Plan

Die Ergebnisausgabe direkt auf dem Plan wurde mittels eines eigenen Zustandes realisiert. In diesen sogenannten „Messen“-Zustand, wird automatisch nach erfolgreicher Beendigung des Rechenvorganges gewechselt. Wenn der Benutzer nun mit der Maus über die Komponenten fährt werden die Berechnungsergebnisse für die jeweilige Komponente angezeigt.

Mithilfe eines zusätzlichen Auswertungsformulars, in welchem alle Ergebnisse aufgelistet sind, ist es möglich einzelne Komponenten auf dem Plan zu suchen, denn wenn eine beliebige Komponente ausgewählt wird, wird diese mithilfe eines linearen Helligkeitsfilter etwas heller auf dem Plan dargestellt.

Tabellen

Um die Berechnungsergebnisse etwas kompakter und übersichtlicher auszugeben, kann man 3 verschiedene Tabellen ausgeben lassen. In der ersten sind alle sogenannten Knotenpunkte aufgelistet. Diese umfassen alle Einspeisungen und Verbraucher des Netzplanes. In der zweiten Tabelle sind die Transformatoren untergebracht und in der dritten dieser Tabellen sind alle Hausanschlusskästen und Schaltanlagen des jeweiligen Planes aufgelistet.

Diese Tabellen erstellen wir ganz einfach, indem wir für jede dieser drei Tabellen einen *stringstream* in der jeweiligen Berechnungsstrategie implementiert haben. Nach Beendigung der Berechnung, erfolgt automatisch die Auswertung der Berechnungsergebnisse, parallel dazu schreibt sich jede Komponente selbst in die jeweilige Tabelle. Diese Ausgabe in die Tabellen erfolgt im **.csv–Format*. Ursprünglich wollten wir diese Ausgabe im Standardmäßigem **.xml–Format* von Microsoft realisieren, doch unzureichende Möglichkeiten im Borland Builder 6 trieben uns dazu, dies noch einmal zu überdenken, da es ansonsten zu viel Zeit in Anspruch genommen hätte.

Die Auswertung erfolgt hier bei den verschiedenen Berechnungsarten nicht anders, denn hier

werden einfach die Daten, welche bei den jeweiligen Knoten von Interesse sind beim Aufruf von *TabellenAuswertung* in der Parameterliste mitgegeben und müssen somit nicht von neuem ermittelt werden.

Export von Plänen

Wenn ein Netzplan mühevollst mit einem Programm nachgebildet wurde, ist es meistens schade, wenn man diesen nur sehr umständlich auf ein Blatt Papier bringen kann, also wenn hierzu keine Funktionen vorhanden sind. Deswegen haben wir uns gedacht, eine Möglichkeit zum Exportieren der Netzpläne zu implementieren.

Bild

Da wir anfangs ein wenig Respekt vor der Ausdruck-Funktion hatten, überlegten wir uns, wie wir den Netzplan anders ausdrucken könnten. Somit kamen wir auf den schlichten Einfall, den Plan auf ein *TImage* zu zeichnen. Dieses Bild könnte sich der Benutzer anschließend selbst unterteilen und ausdrucken.

In *TNetzPlan* haben wir hierzu eine Funktion implementiert, welche einen vorgegebenen Bereich des gesamten Planes absucht, und alle dort liegenden Komponenten einfach auf ein neues *TImage* zeichnet und dieses anschließend als Rückgabewert liefert.

Es kann entweder der gesamte Netzplan als Bild exportiert werden, oder eben nur der Bereich, welcher aktuell angezeigt wird.

Ausdrucken

Bei jeder Software ist standardmäßig eine Druckfunktion mit eingebaut. Da wir unsere Software so komfortabel wie möglich gestalten wollten, entschlossen wir uns nach Absprache mit unserem Betreuungslehrer, der sich sehr für eine solche Funktion aussprach, dazu, dies zu verwirklichen.

Beim Ausdrucken wird zuerst die Anzahl an erforderlichen Blättern ermittelt. Hierzu wird in Zeilen und Spalten unterschieden. Wenn wir diese Zahlen haben, wird noch die Abweichung ermittelt, welche zum Zentrieren des Planes über die Blätter wichtig ist. Nach diesen Vorbereitungen werden alle Spalten und Zeilen durch iteriert. Jede Seite wird hierbei mithilfe der *Printer()*-Funktion dem Drucker übermittelt. Damit die Blätter zugeordnet werden können, wird eine Fußzeile ausgeben, welche die Zeile, Spalte, den Dateinamen und einen Zeitstempel enthält.

Damit eine Anpassung des Ausdruckes ermöglicht werden kann, bauten wir noch zusätzlich hierfür ein Vorschauformular ein. In diesem kann ein konstanter Wert verändert werden, welcher die Größe des Planes auf dem Blatt reguliert.

Dateiformate

Um die Netzpläne und Standardtypen auch abspeichern zu können, sind Dateiformate von Nöten. Die Standardtypen werden als binäre Dateien abgespeichert und können somit nur mit der dazu geschriebenen Klasse *TStandardLesenSchreiben* sinnvoll ausgelesen und abgespeichert werden.

Die Netzpläne werden ebenfalls binär in die Datei geschrieben, jedoch haben diese nicht **.bin* als Dateiendung, sondern die neu definierte **.ntz.*

Da alle Komponenten von *TNetzKomponenteDateiZugriff* abgeleitet sind, können alle Komponenten ganz einfach und praktikabel in die Datei hinterlegt und auch wieder ausgelesen werden.

Der Header einer Komponente sieht folgendermaßen aus:

Versionsnummer	Die ID der Komp.	Name der Komp.	Nennspannung	Planposition	Elektrischen Daten der Komp.

Abbildung 26: Dateiheader jeder einzelnen Komponente

Standardtypen

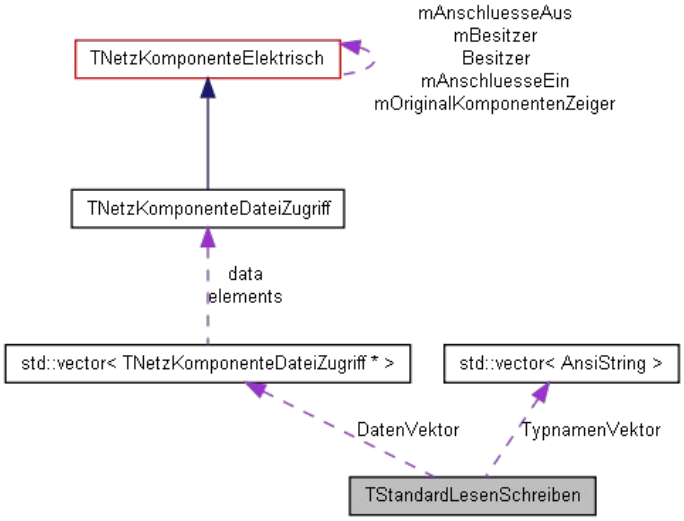


Illustration 27: Includierungshierarchie von TStandardLesenSchreiben

Wie aus der Grafik oben hervorgeht, beinhaltet die Klasse *TStandardLesenSchreiben* einen Vektor an *TNetzKomponenteElektrisch*-Objekten und einem weiteren Vektor in welchem alleinig die Namen der jeweiligen Typen hinterlegt sind. Diese Klasse ermöglicht nun ein bequemes Schreiben, bzw. Lesen der Standardtypen in und aus der Datei.

Bevor die elektrischen Daten abgespeichert werden, ist in jeder Datei ein kleiner Dateiheader, welcher wie folgt aussieht:

Anzahl Typen	Position bzw. Name des Typs	Die Komponentendaten an sich, je Typ

Abbildung 28: Dateiheader der Standardtypen

Hier sind nun kurz alle wichtigen elektrischen Daten der jeweiligen Komponenten aufgelistet, welche in den Standardtypen gespeichert werden:

Kabel

Vom Kabel werden der induktive Belag, der ohmsche Belag und die Betriebskapazität in die Datenbank aufgenommen.

Trafo

Beim Transformator werden in der Datei die Bemessungskurzschlussspannung, die Nenneisenverluste, der relative Spannungsabfall, die Nennleistung, die Übersetzung des Trafos und die Nennspannung, Ober- sowie Unterseite, gespeichert.

Einspeisung

Bei der Einspeisung werden das Netzverhältnis, die Nennspannung und die Kurzschlussleistung verzeichnet. Wir zogen ebenfalls in Betracht, für die Generatoren Standardtypen zu hinterlegen, da sich jedoch kaum Generatoren ähneln, hätte dies nur sehr wenig Sinn gehabt.

Verbraucher

Hierbei werden lediglich die Nennspannung, die bezogene Leistung (komplex) und das dem Vorgelagerten-Netz zugeschriebene Netz-Verhältnis vermerkt.

Plan

Um den Plan und dessen benutzerdefinierten Optionen sinnvoll speichern zu können, gibt es hierzu für jede Komponente eine eigene Klasse, welche für nichts anderes zuständig ist als für den Datei-Zugriff. Das heißt wenn von der jeweiligen Komponente die Funktionen *Read* oder *Write* aufgerufen wird, werden die für diese Komponente wichtigsten elektrischen Daten gespeichert, oder wieder aus der Datei ausgelesen. Für die Optionen haben wir einfach eine Dummy-Komponente *TNetzPlanEigenschaften* implementiert, somit konnten wir diese leicht in die bereits bestehende Hierarchie einbauen. Der Header dieser Plan-Datei, sieht dann beispielsweise wie folgt aus:

Abbildung 29: Dateiheader des Planes

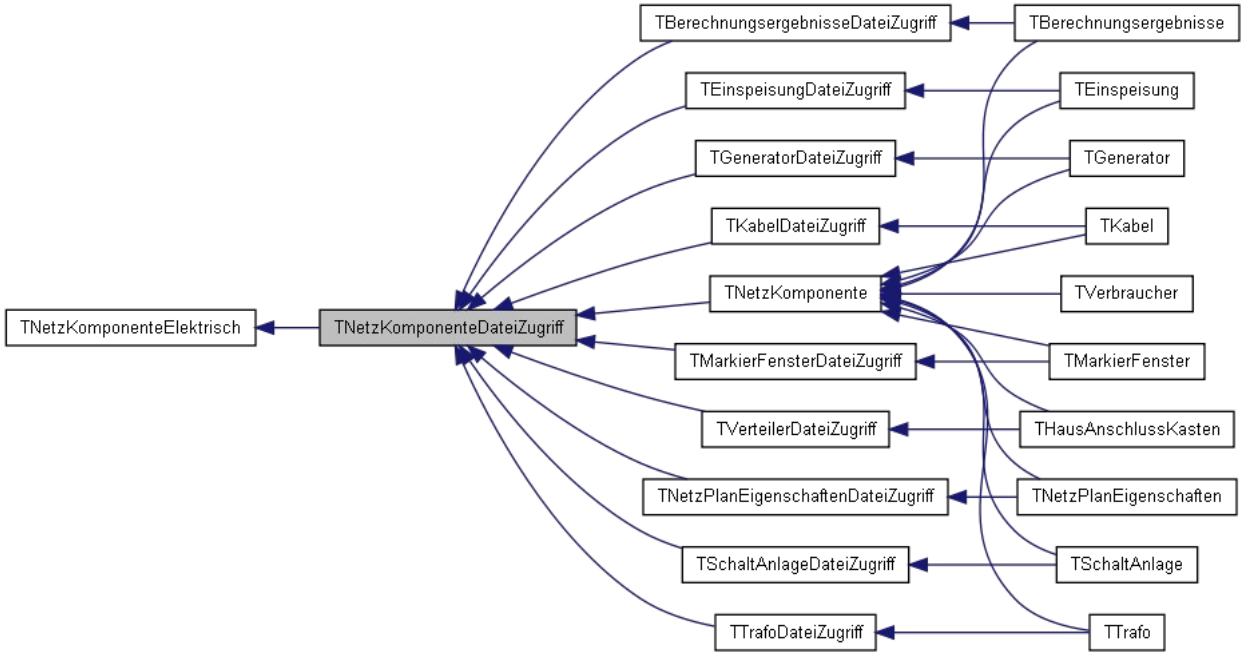


Illustration 30: Vererbungshierarchie von TNetzKomponenteDateiZugriff

Übersichtsmäßig werden hier die Zugriffsklassen der einzelnen Komponenten und letztere dargestellt. Wichtig hierbei, alle erben von TNetzKomponenteDateiZugriff, deswegen können alle zusammen in eine Datei geschrieben werden.

Performance

Die Performance der Anwendung hatte einen hohen Stellenwert in der Entwicklung inne, jedoch nur in zwei Bereichen. Auf der einen Seite muss nämlich die Benutzeroberfläche flüssig reagieren, und auf der anderen soll die Berechnung möglichst schnell durchgeführt werden. Glücklicherweise beeinflussen sich diese Bereiche nicht gegenseitig.

Im Zuge der ersten Tests fiel auf, dass beim Bedienen vor allem die Geschwindigkeit, in der der Plan neu gezeichnet wird, im Vordergrund steht. Einfache Interaktionen, wie die Veränderung von Werten, werden, ohne besondere Maßnahmen, innerhalb eines Zeitraumes durchgeführt, den der Benutzer nicht bemerkt. Werden hingegen Komponenten auf dem Plan verschoben, ist eine hohe Frequenz beim Aktualisieren der Oberfläche von Nöten. Deswegen lag auf diesem Bereich während der Entwicklung ein spezielles Augenmerk und es wurde hierfür sehr viel Zeit investiert.

Neuzeichnen des Planes

Anfangs hatten wir sehr große Probleme mit dem Neuzeichnen des Planes, da wir in regelmäßigen Abständen (alle 20ms) den gesamten Plan neu zeichnen ließen. Bei kleineren Netzen machte dies kaum etwas aus, doch bei größeren wurde das Verändern des Planes immer schwieriger, da es den Prozessor sehr stark aus lastete und somit alles sehr träge schien.

Dies veranlasste uns schlussendlich dazu, das Zeichnen des Planes völlig zu überdenken. Wir haben die Performance-Optimierung hierbei wiederum in zwei unterschiedliche Teile, welche uns sofort ins Auge stachen, unterteilt und haben diese sogleich auch verändert und optimiert.

Bereiche

Als Erstes entschieden wir uns einmal dafür, nicht immer den gesamten Plan neu zeichnen zu lassen, sondern immer nur einen gewissen Bereich, eine ROI (=region of interest) neu zeichnen zu lassen. Hierzu wird der Bereich zuerst geleert anschließend mithilfe von *SucheKomponenten* die in diesem Bereich liegenden Komponenten gesucht und neu auf den Plan gezeichnet.

Bilder in Arbeitsspeicher laden

Da wir einige Komponenten nicht mithilfe der uns zur Verfügung stehenden Funktionen von Canvas zeichneten, sondern einfach ein Bild hierfür laden, war dies noch ein sehr zeitaufwändiger Code, da anfangs die Bilder immer wieder von der Festplatte geladen wurden, was es eben sehr langsam war. Um dies einzudämmen, verwenden wir nun das Entwurfsmuster Singletone[5], mit welchem wir die Bilder zu Beginn einmal in den Arbeitsspeicher laden und später dieses für die Komponenten immer wieder verwenden.

Weiters konnte die Ausführung bestimmter Codeteile von dem Zeitpunkt, an dem der Benutzer die Komponenten verschiebt, weg verlagert werden. Die einzelnen Bilder der Komponenten, ausgenommen das des Kabels, verändern sich nämlich durch diese Manipulation nicht. Eine Änderung der Darstellung ist nur bei der Änderung von den elektrischen Daten nötig.

In der Vererbungshierarchie ist dementsprechend eine Übertragung von Information von oben nach unten notwendig. Dazu enthalten die elektrischen Komponenten eine virtuelle Funktion, mit leere Implementierung, *DatenGeandert*. Diese wird aufgerufen, sobald die elektrischen Daten eine Änderung der Daten verzeichnen. Weiter unten in der Hierarchie, bei der Ausgabe, wird die Funktion überladen und löst eine Aktualisierung der Ausgabe aus. Damit wird, während die elektrischen Daten einer Komponente manipuliert werden, unter Umständen das Bild für die Darstellung auf dem Plan mehrfach gezeichnet. Zu diesem Zeitpunkt ist die Performance jedoch komplett irrelevant, da 100ms mehr oder weniger dem Benutzer nicht auffallen. Während dann aber der Plan bearbeitet wird, müssen die Bilder nicht mehr neu erstellt werden, sondern werden schlicht an der passenden Stelle im Plan kopiert.

Berechnung

Die Performance während der Berechnung stellte nie ein besonders großes Problem dar, da bereits von Beginn an darauf geachtet wurde, dass mit den Ressourcen sparsam umgegangen wurde. Die hierzu eingesetzten Maßnahmen sind vielfältig. Die Formel für einen Iterationsschritt wurde zum Beispiel dahin gehend modifiziert, als das Divisionen durch 3 vermieden werden können. Des weiteren wurde im Bereich der Berechnung auch ganz gezielt auf den Einsatz von STL-Komponenten verzichtet, im speziellen auf den Einsatz von STL-Vektoren. Deren Implementierung ist nämlich abhängig vom Compiler, und deswegen kann unter Umständen zu großen Problemen führen. Ein paar Tests ergaben zwar, dass der Iterator eines Vektors beim Borland C++ Builder 6 im Prinzip nur eine Typdefinition auf einen Zeiger des jeweiligen Datentyps ist, jedoch kann dies bei einem Wechsel des Compiler wiederum anders gehandhabt werden. Es kam auch ganz bewusst nicht der Datentyp *complex* zum Einsatz. Im Gegenzug wurde die Klasse TComplex entwickelt, welche speziell an die Anforderungen angepasst ist. Hierzu gehört unter anderem auch, dass die Informationen in Koordinatenform abgespeichert werden, und Multiplikation, genauso wie die Division, in dieser Darstellungsart durchgeführt werden. Die Einbußen, welche man bei der

5 Dieses Entwurfsmuster ist in der Quelle 3 hinterlegt.

Berechnung von Logarithmen oder Potenzen durch die kartesische Form hat, sind irrelevant, da sie nicht benötigt werden. Zudem wurde damit auch erreicht, dass man wiederum nicht von der Implementierung des jeweiligen Compiler-Herstellers abhängig ist.

Die Beschränkung beim Datentyp des Real- und Imaginärteils auf doppelte Genauigkeit der Gleitkommazahlen ist eine wirkliche Einschränkung, da in der Realität bereits Abweichungen von einigen hundert Millivolt ausreichend genau sind.

Die Implementierung zahlreicher überladener Operatoren verhindert häufige implizite Konstruktoraufrufe. Außerdem ist es so möglich, Objekte vom Typ *TComplex* wie gewöhnliche Zahlen im Code zu verwenden. Wie in der Abbildung *Header der Klasse TComplex* auf der nächsten Seite ersichtlich ist.

```
class TComplex
{
        private:
                //! Karthesische Koordinaten
                double mReal, mIm;

        private:
                void SetReal(double Real);
                void SetIm(double Im);
                double GetBetrag() const;
                void SetBetrag(double Radius);
                double GetWinkel() const;
                void SetWinkel(double Phi);
                void KarthToPol(double &Betrag, double &Winkel);
                void PolToKarth(double Betrag, double Winkel);

        public:
                //! Stellt mReal dar
                __property double Real={read=mReal,write=SetReal};
                //! Stellt mIm dar
                __property double Im={read=mIm,write=SetIm};
                //! Stellt den Betrag dar
                __property double Betrag={read=GetBetrag,write=SetBetrag};
                //! Stellt den Winkel dar (in rad)
                __property double Winkel={read=GetWinkel,write=SetWinkel};

        public:
                TComplex();
                TComplex(double Real, double Im);
                TComplex(double Real);

        public:
                TComplex operator+(const double& rhs);
                TComplex operator-(const double& rhs);
                TComplex operator*(const double& rhs);
                TComplex operator/(const double& rhs);
                void operator=(const double& rhs);
                void operator+=(const double& rhs);
                void operator-=(const double& rhs);
                void operator*=(const double& rhs);
                void operator/=(const double& rhs);

                TComplex operator+(const double& rhs) const;
                TComplex operator-(const double& rhs) const;
                TComplex operator*(const double& rhs) const;
                TComplex operator/(const double& rhs) const;

                TComplex operator+(const TComplex& rhs);
                TComplex operator-(const TComplex& rhs);
                TComplex operator*(const TComplex& rhs);
                TComplex operator/(const TComplex& rhs);
                void operator=(const TComplex& rhs);
                void operator+=(const TComplex& rhs);
                void operator-=(const TComplex& rhs);
                void operator*=(const TComplex& rhs);
                void operator/=(const TComplex& rhs);
                bool operator==(const TComplex &rhs);
                bool operator!=(const TComplex &rhs);

                TComplex operator+(const TComplex& rhs) const;
                TComplex operator-(const TComplex& rhs) const;
                TComplex operator*(const TComplex& rhs) const;
                TComplex operator/(const TComplex& rhs) const;
                bool operator==(const TComplex &rhs) const;
                bool operator!=(const TComplex &rhs) const;

        public:
                TComplex AbsolutBetrag();
                TComplex KonjugiertKomplex() const;
                AnsiString ToString() const;
};
```

Illustration 31: Header der Klasse TComplex

Die Dauer der Berechnung kann zudem noch von dem Benutzer über die *Simulationsparameter* beeinflusst werden. Bei diesen ist es nicht wirklich möglich eine einheitliche Wahl zu treffen, weshalb der Anwender selber die Parameter justieren muss. Maßgeblich für die Geschwindigkeit sind der Übernahmefaktor, die Genauigkeit, die Geschwindigkeit, mit der Kurzschlussspannungen abgesenkt werden, die Geschwindigkeit, mit der die Leistungen erhöht werden, die Anzahl, wie oft das Ergebnis, welches im gewünschten Genauigkeitsbereich liegt berechnet werden soll, bevor die Berechnung abgebrochen wird, und der Startwert kann entweder berechnet werden oder in Prozent der Nennspannung angegeben werden.

Ob die getroffene Wahl wirklich auch ausreichend genau war, kann der Benutzer anhand der Summe der Leistungen entscheiden. Diese wird während der Auswertung ermittelt, und ist die Summe aller Verlustleistungen und eingespeisten Leistungen. Sollte diese ungleich 0, bzw. sehr stark von 0 abweichen, sind die Parameter ungünstig gewählt. Im Besonderen die Genauigkeit hat einen sehr starken Einfluss auf die Qualität der Ergebnisse und die Berechnungsdauer.

Weitere Merkmale

Um eine benutzerfreundliche Handhabung der Anwendung bieten zu können, ist die Implementierung von weiteren Features nötig. Die Eingabe eines Netz-Planes mit größeren Dimensionen brachte uns zum nachdenken. Wenn wir uns also andere Programme ansahen, welche Komfortabel zu bedienen sind, erkannten wir viele Ähnliche Funktionen, welche für ein solches Programm einfach unumgänglich sind und somit haben wir uns dazu entschlossen, die Wichtigsten dieser Funktionen in unsere Applikation zu integrieren.

Standardtypen

Die Idee der Standardtypen war, dass somit dem Benutzer Standardkomponenten, wie Kabel oder Transformatoren und deren elektrischen Daten, zur Verfügung stehen und nicht ständig händisch neu eingegeben werden müssen, sondern lediglich ausgewählt. Um diese Datei zu erstellen, in welcher die Standardtypen verzeichnet sind, haben wir eine zusätzliche Applikation geschrieben, um eine komfortable Eingabe der Standardtypen zu ermöglichen. Standardtypen sind für das Kabel, die Einspeisung, den Transformator und für den Verbraucher implementiert.

Rückgängig und Wiederherstellen

Ein sehr wichtiger Punkt war es, dass wir eine Rückgängig/Wiederherstellen-Funktion in den Plan integrierten. Hierzu verwendeten wir das Command[6] Entwurfsmuster. Hierbei handelt es sich um eine Art Container, in welchem alle Befehle „hineingeworfen“ werden. Diese im LIFO-Format (=last in first out) angesammelten Daten, können nun auch wieder abgerufen werden. Für jeden einzelnen Befehl ist es also notwendig ein eigenes Objekt zu schreiben, in welchem die Daten für die Ausführung und für Reversion eines Befehls gespeichert sind.

Diese Befehle erben alle von *TBefehl* und können somit ähnlich wie die Komponenten alle gleich behandelt werden und somit auch in einen Vektor hinterlegt werden. Somit kann die Ausführung der Befehle nach vorne, bzw. nach hinten einfach realisiert werden. Im folgenden ist die

6 Die genauerer Beschreibung ist aus der Quelle 3 zu entnehmen.

Vererbungshierarchie von *TBefehls* zu erkennen und alle untergeordneten Befehle, welche für alle Befehle, welche vom Plan hervorgerufen werden können, implementiert werden mussten. Den vorhin genannten Container bildet *TBefehlsListe*. In diesem befindet sich ein *deque* an *Tbefehl*-Objekten, wie in der folgenden Abbildung (links) zu sehen ist.

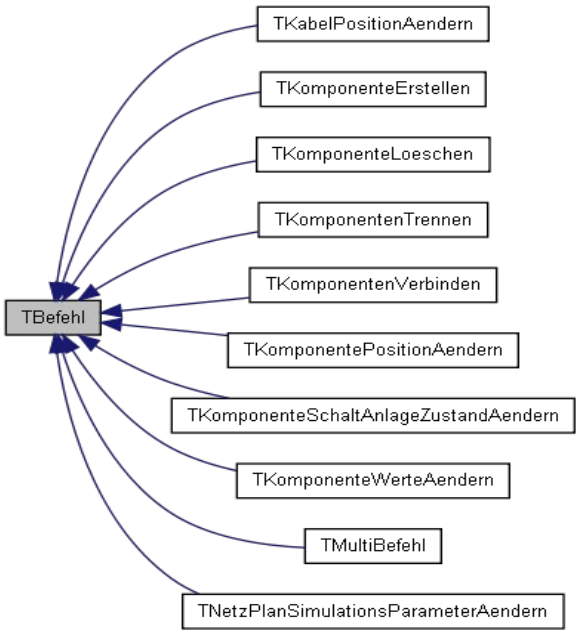


Illustration 32: Vererbungshierarchie von TBefehl

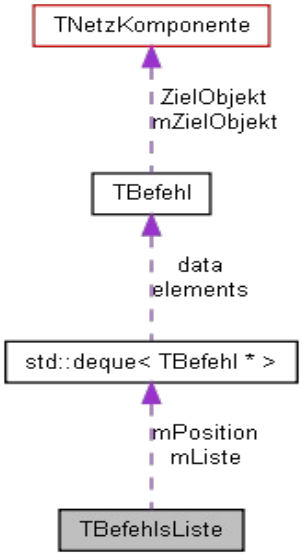


Illustration 33: Abhängigkeiten von TBefehlsListe

Befehle, welche für den Benutzer der Logik nach als ein einziger interpretiert werden, werden in einem einzigen *MultiBefehl* zusammengefasst. Dieser wird anschließend wieder in den Container hinzugefügt und kann genau gleich wie die anderen Befehle behandelt werden.

Kopieren und Verschieben von Objekten

Um Komponenten zu verschieben, oder zu kopieren, muss zuerst entschieden werden, welche Komponenten zu kopieren sind. Da wir ja das Entwurfsmuster der Zustände[7] verwenden, wird eben in diesem entschieden, welche Komponenten nun markiert (und somit ausgewählt) wurden. Je nach dem, wie sich der Benutzer dann entscheidet, werden diese kopiert, verschoben oder gelöscht.

Bei ersterem wird in den *„Platzieren“-Zustand* gewechselt. Im *Konstruktor* werden dort Kopien der markierten Komponenten erstellt. Differenziert muss man hierbei das Kabel betrachten, denn dieses wird nur dann kopiert, wenn die Komponenten, welche dieses verbindet ebenfalls markiert wurden. Da wir uns anschließend noch immer im *„Platzieren“-Zustand* befinden, werden bei Bewegen der Maus die Koordinaten der Komponenten an die der Maus angepasst und können somit auf dem Plan an die gewünschte Stelle platziert werden.

Wenn der Benutzer allerdings diese Komponenten verschieben möchte, dann wird in den *„Verschieben“-Zustand* gewechselt. Hierbei wird einfach ident vorgegangen, wie beim Kopieren, nur das die Komponenten, welche bewegt werden nicht vorher kopiert werden, sondern die originalen Komponenten verwendet werden.

Beim Wechsel in den *„Löschen“-Zustand*, werden im *Konstruktor* schon die ganzen Komponenten gelöscht und anschließend wird wieder in den *„Betrachten“-Zustand* gewechselt.

Rasterung des Plans

Die Eingabe für den Benutzer wird vereinfacht, wenn der Plan gerastert ist. Vor allem die Verlegung von Kabeln wird dadurch erheblich komfortabler. Die Einschränkung, dass nur bestimmte Punkte beim Platzieren oder Verlegen angewählt werden können, wurde erst nachträglich eingeführt. Dementsprechend wäre es, technisch gesehen, möglich, eine beliebige Rasterung einzusetzen. Da die Bilder der Komponenten aber an dem jetzt gewählten Raster orientiert aufgebaut sind, wurde diese Einstellung dem Benutzer vorenthalten.

Hilfe

Damit bei auftauchenden Problemen ein Nachschlagewerk vorhanden ist, haben wir eine kleine Hilfe eingebaut, in welcher die Wichtigsten grundlegenden Funktionen und Funktionsweisen des Programms kurz beschrieben werden.

Um hier nicht irgendeine Exotische Hilfe zu erstellen, haben wir die Standard-Hilfe von Microsoft verwendet. Hierzu müssen lediglich für jedes Thema, welches beschrieben werden sollte, eine HTML-Seite erstellt. Diese mussten anschließend noch verlinkt werden.

7 Eine genauere Erklärung dieses Entwurfsmusters, können sie aus der Quelle 3 entnehmen.

Projektaufteilung

Die Projektaufteilung erfolgte ganz grob gesagt in die Bereiche der Oberflächengestaltung und der Berechnung des Netzes. Für die Oberfläche der Applikation war hauptsächlich Martin Schroll zuständig und für die Berechnung Benedikt Schmidt.

Somit war es uns möglich am Programm zu arbeiten, ohne den Bereich des anderen zu berühren. Dies war jedoch nur anfangs möglich, im späteren Verlauf des Projektes überlagerten sich unsere Arbeitsbereiche, deswegen entschieden wir uns dazu, das Zusammenführen der Dateien mithilfe einer Versionsverwaltungssoftware durchzuführen. In unserem Fall entschieden wir uns für *RapidSVN*, und zum Vergleichen zweier Textdateien verwendeten wir *Kdiff 3.* Mithilfe dieser Applikationen konnten wir unsere Fortschritte am Projekt sehr bequem dem *Repository* hinzufügen, also dem Server, auf welchem immer die aktuellste Version gespeichert wurde.

Projektentwicklung

Passend zur Weihnachtspräsentation, funktionierte bereits die symmetrische Berechnung. Jedoch waren die Ausgabe der Ergebnisse und die Handhabung der Applikation alles andere als angenehm. Die Berechnung funktionierte allerdings auch nur für sehr einfache und kleine Netze.

Im späteren Verlauf, konnten wir beide alles neu gestalten. Einerseits stellten wir die Handhabung mit dem Plan komplett auf den Kopf und andererseits dasselbe mit der Berechnung.

Bei der Handhabung war es notwendig, da wir ansonsten keinen Überblick mehr über die ganzen Zustände gehabt hätten. Anschließend wurden nun wichtige grundsätzliche Funktionen, wie beispielsweise das Verschieben von mehreren Komponenten oder das Kopieren eingebunden. Ebenso wurde die Ausgabe der Ergebnisse um einiges angenehmer gestaltet.

Bei der Berechnung war die Umgestaltung notwendig, um das Einbinden der Berechnung von unsymmetrischen Fehlern eleganter zu ermöglichen. Hierzu wurde die gesamte Berechnung kompakter gestaltet und eine *Abstrakte Fabrik*[8] implementiert. In dieser Fabrik werden alle Knoten und deren Zusammenhänge für die Berechnung soweit vorbereitet, dass die eigentliche Berechnung nicht mehr so umfangreich sein muss und auch nicht mehr von der Berechnungsart abhängt. Weitere Probleme, welche nach der Weihnachtspräsentation gelöst wurden, waren die Implementierung einer Schaltanlage, sowie die Einbeziehung von Übersetzungen. Zuvor wurde die gesamte Berechnung auf einer fiktiven Spannungsebene durchgeführt. Damit konnten aber Transformatoren nicht Übersetzungen, welche nicht der Nennübersetzung entsprachen, in die Berechnung einfließen. Erst nach dieser Umstrukturierung war es möglich, dass jeder Knoten auf der Spannungsebene berechnet wird, welche er auch tatsächlich inne hat.

8 Zur näheren Beschreibung dieses Entwurfsmusters, sei auf Quelle 3 verwiesen.

Projektfortschritte

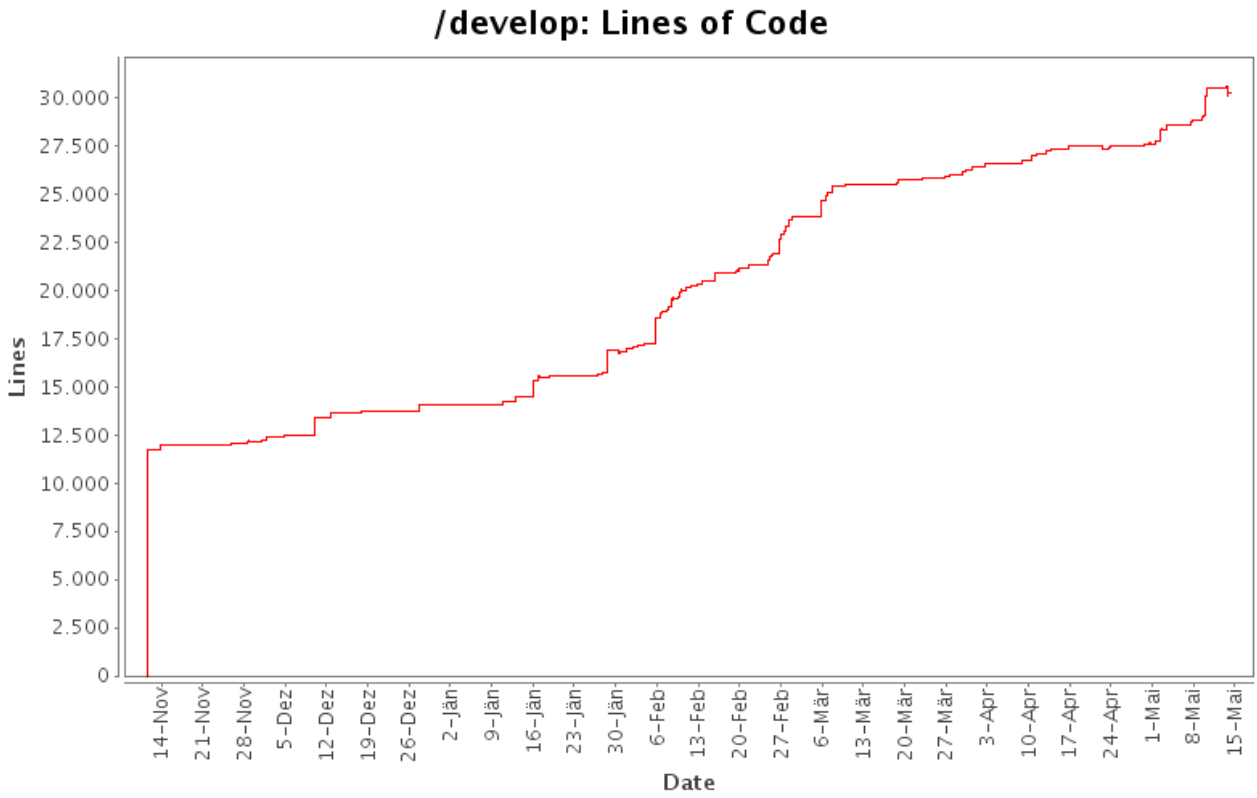


Abbildung 34: Zeilen Code, Verlauf über 5 Monate hinweg

Diese Grafik zeigt, zu welcher Zeit wie viel am Projekt effektiv gearbeitet wurde. Der große Sprung am Anfang repräsentiert die Arbeit, welche vor der Verwendung der Objektverwaltungssoftware geleistet wurde. Die Statistik zeigt einfach, wie viele Codezeilen dem Projekt hinzugefügt wurden.

Bevorzugte Arbeitszeiten

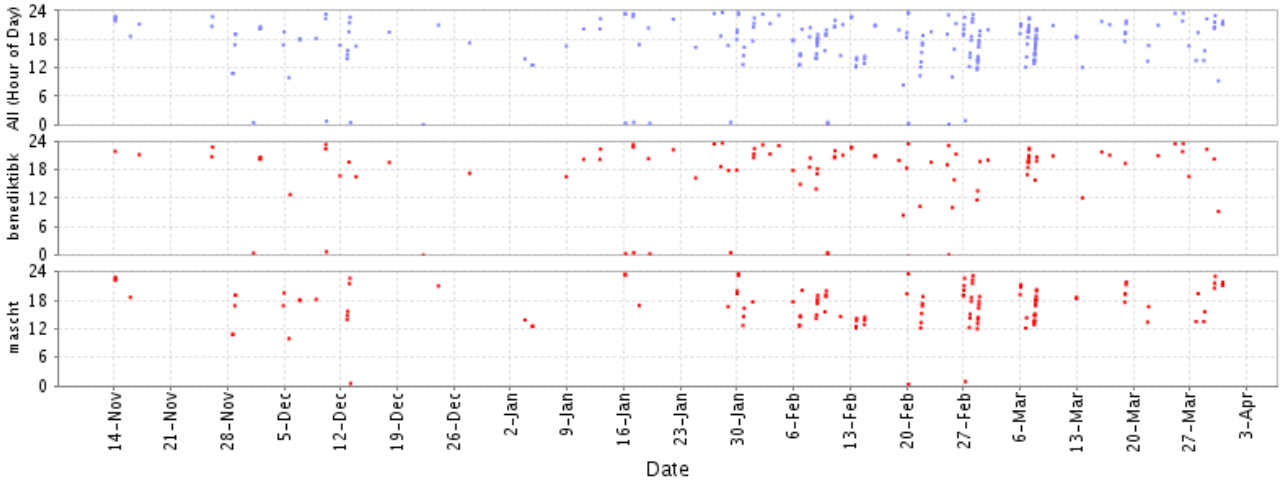


Abbildung 35: Bevorzugten Arbeitszeiten

Diese Grafik, ebenfalls automatisch erstellt von der Versionsverwaltung, stellt sehr schön die bevorzugten Arbeitszeiten dar. Wie man unschwer erkennen kann, begannen wir meistens erst nach dem Mittagessen mit der Arbeit, unter anderem bedingt durch den Besuch der letzten Schulstufe.

Gesamter Aufwand

Die gesamte Arbeitszeit, welche in das Projekt investiert wurde, kann leider nur im Nachhinein geschätzt werden. Die jeweiligen Zeiten, welche pro Tag in das Projekt flossen, wurden ab September von Martin Schroll dokumentiert. Basierend auf diesen Aufzeichnungen und auf den Statistiken, welche man der Versionsverwaltung entnehmen kann, lässt sich die restliche Zeit näherungsweise extrapolieren.

Für diese Berechnung wird das Verhältnis von Code, welcher erstellt und modifziert wurde, zu der Anzahl an Code, welcher dann tatsächlich im Zuge der Arbeiten entstand, benötigt. Diese Kennziffer kann aus der Versionsverwaltung ermittelt werden, im Zeitraum von Anfang Dezember 2009 bis Ende März:

$$\frac{24113\,LOC}{13970\,LOC} = 1{,}7261$$

Aus den Aufzeichnungen von Schroll ist ersichtlich, dass er im selben Zeitraum wie oben 220,25 h an Arbeit erbrachte. Währenddessen modifizierte bzw. erstellte er 11132 Zeilen Code.

$$\frac{220{,}25\,h}{11132\,LOC} = 1{,}1871\,min\ pro\ modifizierter\ Codezeile$$

Aus diesen beiden Kennziffern und der gesamten Anzahl an Zeilen Code ergibt sich ungefähr der totale Arbeitsaufwand:

$$\frac{29312 * 1{,}7251 * 1{,}1871}{60} = 1000{,}45\,h \approx 1000\,h$$

Insgesamt überstieg die investierte Arbeitszeit dementsprechend bei weitem den Umfang von 250 h pro Diplomand, welcher maximal für eine Diplomarbeit vorgesehen ist.

Sprache und Entwicklungsumgebung

Die Wahl der Sprache fiel schlussendlich sehr schnell. Im Endeffekt bestanden nur die beiden Optionen C++ und C#. Letztere Sprache schied aber aus zweierlei Gründen aus:

1. In C# ist keine Mehrfachvererbung möglich. Es können zwar mehrere Interfaces als Basen verwendet werden, aber diese sind nicht in der Lage, Daten zu enthalten. Wahrscheinlich wäre es möglich gewesen, den ursprünglichen Ansatz zu manipulieren, so dass C# wieder ein realistischer Kandidat geworden wäre, aber der intuitiv gewählte schien recht elegant zu sein.
2. Die beiden Entwickler hatten bisher mehr Erfahrungen mit C++ gesammelt. Das mag zwar im ersten Moment als nicht besonders schwerwiegendes Argument erscheinen, aber um ein so umfangreiches Projekt zu realisieren ist einiges an Vorwissen nötig. Zudem war die Wahl des Themas auch riskant, und zusätzliche Risiken sollten deswegen nicht aufgenommen werden.

Im Zuge der Wahl der Programmiersprache legte man sich auch relativ bald auf die Entwicklungsumgebung Borland C++ Builder 6 fest, da dieser bereits häufig zum Einsatz kam und seine Tücken größtenteils bereits bekannt waren. Da dieses Produkt aber nicht mehr vom Hersteller mit Updates versorgt wird, mussten einige Workarounds angewandt werden.

Analyse des Netzes der Haller Stadtwerke

Als praxisnahen Test der Anwendung wurde eine grobe Analyse des Netzes der Haller Stadtwerke durchgeführt. Genauer gesagt, es wurde ermittelt wie sich die Spannungsabfälle im Mittelspannungsnetz verhalten werden bei der bereits seit langem geplanten Umstellung von 25kV auf 30kV. Hierzu wird das Netz bereits seit Jahren umgebaut, beziehungsweise neue Anlagenteile für die Umstellung ausgelegt. Durch die höhere Übertragungsspannung werden sich die Verluste im Netz verringern, was einen effizienteren Betrieb des Netzes für die Haller Stadtwerke bedeutet.

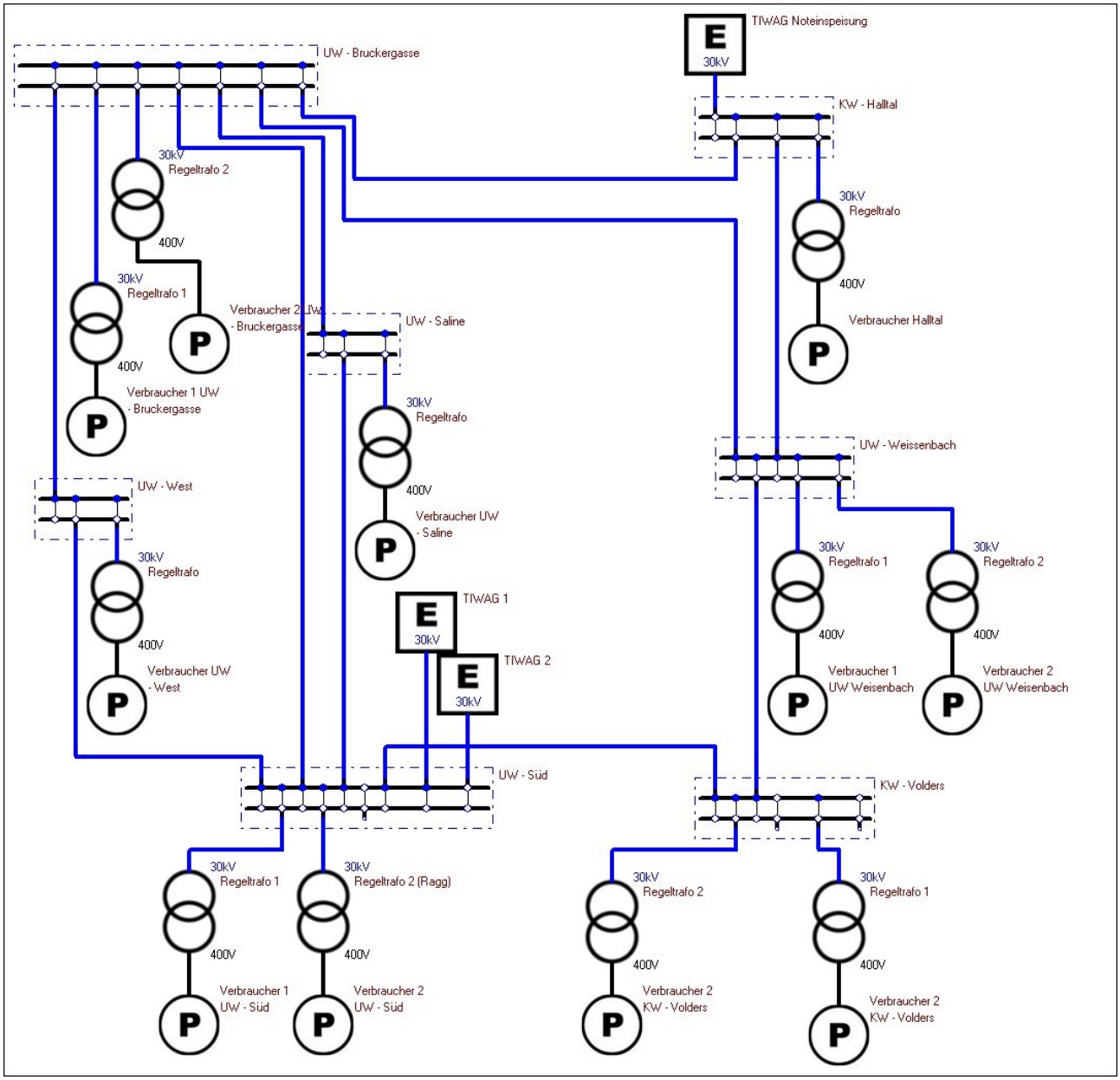


Abbildung 36: 30kV-Netz der Haller Stadtwerke

Wie man sehen kann, wurden für die Berechnung die Generatoren der Stadtwerke nicht miteinbezogen, sondern ging man von einem Betrieb alleine durch die Versorgung der TIWAG aus, exklusive der Noteinspeisung im Halltal, aufgrund des leichteren Aufbaus.

Aufgrund der Lastflussverteilung ergaben sich folgende Speisespannungen für das 10kV-Netz:

Name	Typ	Spannung [kV]	Wirkleistung [kW]	cosφ
TIWAG Noteinspeisung	Einspeisung	30,000	0	1,000
Verbraucher 1 UW Weisenbach	Verbraucher	0,393	2165	0,900
Verbraucher 2 UW Weisenbach	Verbraucher	0,393	2165	0,900
Verbraucher Halltal	Verbraucher	0,398	170	0,900
Verbraucher 2 KW - Volders	Verbraucher	0,389	3464	0,900
Verbraucher 2 KW - Volders	Verbraucher	0,385	3464	0,900
Verbraucher 1 UW - Süd	Verbraucher	0,390	3464	0,900
Verbraucher 2 UW - Süd	Verbraucher	0,392	2078	0,900
Verbraucher UW - Saline	Verbraucher	0,389	3897	0,900
Verbraucher 2 UW - Bruckergasse	Verbraucher	0,388	3897	0,900
Verbraucher 1 UW - Bruckergasse	Verbraucher	0,389	3464	0,900
Verbraucher UW - West	Verbraucher	0,390	3464	0,900
TIWAG 1	Einspeisung	30,000	32012	0,885
TIWAG 2	Einspeisung	30,000	0	1,000

Illustration 37: Ergebnistabelle der Knoten für das 30kV-Netz der Haller Stadtwerke (ohne TIWAG-Noteinspeisung im Halltal)

Die maximale Spannungsabsenkung ergibt sich damit in Volders, mit 3,817%. Dies fällt jedoch in den Bereich des erlaubten, und mit einer etwas niedrigeren Regelstufe des Trafos kann diese Absenkung ausgeglichen werden.

Zudem zeigt sich, dass der eingespeiste Leistungsfaktor der TIWAG unter 0,9 liegt, verursacht vor allem durch die niedrig gewählten Leistungsfaktoren der Verbraucher. Hier kommen außerdem noch die Phasenverdrehungen der hauptsächlich induktiven Leitungen hinzu.

Durch eine zusätzliche Einspeisung im Halltal durch die TIWAG, bei einer dortigen Kurzschlussleistung des vorgelagerten Netzes von 10GVA, könnten die Speisespannungen verbessert werden:

Name	Typ	Spannung [kV]	Wirkleistung [kW]	cosφ
TIWAG Noteinspeisung	Einspeisung	30,000	5112	0,869
Verbraucher 1 UW Weisenbach	Verbraucher	0,393	2165	0,900
Verbraucher 2 UW Weisenbach	Verbraucher	0,393	2165	0,900
Verbraucher Halltal	Verbraucher	0,399	170	0,900
Verbraucher 2 KW - Volders	Verbraucher	0,390	3464	0,900
Verbraucher 2 KW - Volders	Verbraucher	0,385	3464	0,900
Verbraucher 1 UW - Süd	Verbraucher	0,390	3464	0,900
Verbraucher 2 UW - Süd	Verbraucher	0,392	2078	0,900
Verbraucher UW - Saline	Verbraucher	0,389	3897	0,900
Verbraucher 2 UW - Bruckergasse	Verbraucher	0,389	3897	0,900
Verbraucher 1 UW - Bruckergasse	Verbraucher	0,390	3464	0,900
Verbraucher UW - West	Verbraucher	0,391	3464	0,900
TIWAG 1	Einspeisung	30,000	13441	0,888
TIWAG 2	Einspeisung	30,000	13441	0,888

Illustration 38: Ergebnistabelle der Knoten für das 30kV-Netz der Haller Stadtwerke (mit TIWAG-Noteinspeisung im Halltal)

Im folgenden noch die Lastflüsse bei nur einer Einspeisung der TIWAG im UW Süd:

UW - Bruckergasse

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9161kV*
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9161kV*
UW - West	0,073	11315	0,883		
Regeltrafo 1	0,067	10465	0,885		
Regeltrafo 2	0,076	11784	0,883		
UW - Süd	-0,102	-15915	-0,884		
UW - Saline	-0,065	-10087	-0,884		
UW - Weissenbach	0,078	12092	0,879		
KW - Halltal	0,019	2977	0,902		

UW - West

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9516kV*
UW - Bruckergasse	0,082	12832	0,883		
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9516kV*
UW - Süd	-0,140	-21797	-0,885		
Regeltrafo	0,067	10467	0,886		

UW - Süd

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.999kV*
UW - West	0,158	24679	-0,885		
UW - Bruckergasse	0,116	18058	-0,884		
UW - Saline	0,159	24804	-0,884		
KW - Volders	0,141	22034	0,888		
TIWAG 1	0,696	108471	0,885		
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.999kV*
Regeltrafo 1	0,067	10470	0,885		
Regeltrafo 2 (Ragg)	0,040	6277	0,888		

UW - Saline

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9524kV*
UW - Bruckergasse	0,073	11421	-0,884		
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.9524kV*
UW - Süd	-0,141	-21885	-0,884		
Regeltrafo	0,076	11785	0,883		

KW - Halltal

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.8961kV*
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.8961kV*
UW - Bruckergasse	0,019	2974	0,902		
UW - Weissenbach	-0,016	-2465	-0,903		
Regeltrafo	0,003	509218	0,898		

UW - Weissenbach

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.879kV*
UW - Bruckergasse	0,089	13744	0,879		
KW - Halltal	0,018	2729	-0,903		
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.879kV*
KW - Volders	-0,010	-1500	-0,813		
Regeltrafo 1	0,042	6519	0,890		
Regeltrafo 2	0,042	6519	0,890		

KW - Volders

1. Sammelschiene - Eingänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.8623kV*
UW - Süd	0,141	21934	0,887		
UW - Weissenbach	0,012	1843	-0,813		
1. Sammelschiene - Ausgänge	*Strom [kA]*	*Wirkleistung [kW]*	*cosφ*	*-> Spannung:*	*29.8623kV*
Regeltrafo 2	0,068	10502	0,879		
Regeltrafo 1	0,067	10461	0,886		

Illustration 39: Berechnungsergebnistabelle, wenn nur eine Einspeisung im UW Süd

Resümee und Ausblick

Insgesamt muss gesagt werden, dass sehr viel Arbeit investiert werden musste. Es entstanden immer wieder neue Probleme, bedingt durch zu wenig Erfahrung, sowohl im Bereich der Softwareentwicklung, als auch von der elektrotechnischen Seite. Besonders bei letzterem konnte uns aber unser Betreuter, Prof. Mag. Werner Sejkora, immer wieder unterstützen. Trotzdem tauchten aber ständig neue, nicht erwartete Probleme auf, weshalb die geplanten Zeiten nicht eingehalten werden konnten. Aufgrund des doch sehr großen Puffers, und auch, weil bereits im April 2009 mit den ersten Vorarbeiten begonnen wurde, war eine rechtzeitige Fertigstellung möglich.

Die Arbeit an dieser Diplomarbeit bedeutet sehr viel Zeitaufwand, brachte aber auch viele neue Erfahrungen mit sich. Außerdem war, da man sich sehr intensiv mit dem Thema der Softwareentwicklung beschäftigte, ein bedeutender Schritt in der Planung der weiteren Ausbildung beziehungsweise bei der Wahl der gewünschten späteren Anstellung dadurch möglich.

Erweiterungen

Die gesamte Anwendung wurde ständig mit dem Augenmerk dahingehend entwickelt, dass die Implementierung von anderen Berechnungsarten, wie zum Beispiel unsymmetrischen Fehlern, leicht möglich sein sollte. Die Berechnung an sich sollte in diesem Fall wenige bis keine Manipulation erfordern, es ist nur von Nöten, dass das repräsentative Netz von Knoten zu Beginn anders erstellt, und im Anschluss passend ausgewertet wird.

Die wahrscheinlich erforderlichen Modifikationen beziehungsweise Neuentwicklungen sind folgende:

- Angepasste abstrakte Fabrik, zur Erstellung der Knoten
- Einführung einer neuen, reinen virtuellen Funktion in *TNetzKomponenteElektrisch*, und deren Überladung, in welcher jede Komponente für sich die Knoten erstellt.
- Einführung einer neuen, reinen virtuellen Funktion in *TNetzKomponenteElektrisch*, und deren Überladung, in welcher jede Komponente für sich die Verbindungsdaten der Knoten ermittelt.
- Einführung einer neuen, reinen virtuellen Funktion in *TNetzKomponenteElektrisch*, und deren Überladung, in welcher jede Komponente für sich die passenden Verbindungsdaten an den Aufrufer übergibt.
- Einführung einer neuen, reinen virtuellen Funktion in *TNetzKomponenteElektrisch*, und deren Überladung, in welcher jede Komponente für sich die Auswertung der Ergebnisse der Berechnung durchführt.
- Überladung von *TPlanBerechnung*, welche die passende Auswertung für jede Komponente aufruft.

Es würde uns freuen, wenn jemand sich mit der Weiterentwicklung beschäftigen würde, und stehen diesem auch gerne hilfreich zur Seite.

Verzeichnis der Abbildungen

Literaturverzeichnis

1: René Flosdorff, Günther Hilgarth, Elektrische Energieverteilung, 2005, ISBN 3-519-36424-7

2: Jürgen Wolf, C++ von A bis Z - Das umfassende Handbuch, 2009, ISBN 9783836214292

3: Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software, 2004, ISBN 9783827321992

Danksagungen

In erster Linie möchten wir uns natürlich bei unserem Betreuer, Prof. Mag. Werner Sejkora, bedanken, welcher uns bezüglich elektrotechnischen Fragen eine große Hilfe war. Hierbei fehlte uns schlicht und ergreifend Wissen, welches wir im Laufe der Ausbildung an der HTL nicht vermittelt bekommen konnten, da es den zeitlichen Rahmen gesprengt hätte.

Weiterer Dank gilt auch an die Haller Stadtwerke, welche uns die Daten ihres gesamten Netzes zukommen ließen, und damit einen praxisnahen Test der Anwendung ermöglichten.

Natürlich wollen wir uns auch noch bei Stefan Schroll, Johannes Welebil, Markus Rendl und Fabian Marth bedanken, welche uns als Beta-Tester tatkräftig unterstützten, denn durch sie wurden so einige Bugs an der Software entdeckt und konnten anschließend gleich behoben werden. Sie trugen auf diesem Wege ihren Teil zur erfolgreichen Fertigstellung der Applikation bei, und dafür gibt es von unserer Seite ein großes Dankeschön.

Außerdem gilt auch ein großes Dankeschön an unsere Familien, welche uns diese hervorragende Ausbildung finanzierten und viel Geduld während der 5 Jahre, speziell während den Arbeiten an der Diplomarbeit, aufbrachten.