FAKULTÄT FÜR !NFORMATIK

Nachrichtenbasierte Simulation von elektrischen Lastflüssen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Informatik

eingereicht von

Stefan Vielguth
Matrikelnummer 9625429

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuer/Betreuerin: O. Univ. Prof. Dipl.-Ing.
Dr.techn. Dietmar Dietrich
Mitwirkung: Univ.Ass. Dipl.-Ing. Dr.techn. Friederich Kupzog

Wien, 02.10.2008 _____ _____
(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Kurzfassung

Elektrische Energienetze bilden das Rückgrat der Versorgung der Menschen mit elektrischer Energie. Im Zuge des zunehmend wachsenden Bedarfs an elektrischer Energie ist ein stetiger Netzausbau für viele Versorgungsregionen unumgänglich. Ursprünglich waren Energienetze als strikte Top-Down-Strukturen angelegt. Das bedeutet, dass die von den Primär-Einspeisungsknoten (Kraftwerken) hin zu den Endverbrauchern benötigte Struktur, einmalig großzügig hinsichtlich der Beanspruchung durch elektrische Randparameter wie Ströme, Spannungen und Leistungen ausgelegt und keine größeren Veränderungen mehr durchgeführt wurden. Im Zuge der Liberalisierung der Energiemärkte und des zunehmenden Energiebedarfs kommt es heutzutage vor allem im Bereich der Mittelspannungs- und Niederspannungsebene zu zusätzlichen Einspeisungen von elektrischer Energie. Beispiele dafür sind Wind- oder Biomassekraftwerke. Dies verändert allerdings die Situation der Energienetze, da neue Einspeisepunkte im Betrieb die elektrischen Parameter im Bereich der betroffenen Spannungsebene verändern. Deshalb ist es notwendig, noch vor der Planung bzw. Inbetriebnahme neuer Kraftwerke, die Auswirkungen auf das betroffene Netz abschätzen zu können. Zu diesem Zweck wurden Verfahren der Lastflussanalyse entwickelt. In dieser Arbeit wird der Fokus auf die Implementierung eines nachrichtenbasierten Lastflussanalyse-Verfahrens, das auf einem Vorwärts-/Rückwärtsschritt-Verfahren basiert, gelegt. Für die Implementierung des Verfahrens wird das Open-Source-Framework OMNeT++ [3] herangezogen, welches es ermöglicht, Graphenmodelle aufzubauen und Nachrichten zwischen den Knoten des Graphen auszutauschen. Weiters wird die Integration des Algorithmus in die Architektur der Simulationsplattform DAVIC beleuchtet. DAVIC ist eine Simulationsplattform des Instituts für Computertechnik, die es ermöglichen soll, Geld-, Kommunikations- und Energieflüsse in Energienetzen zu simulieren. Als Ergebnis wurde eine Lösung implementiert, die es ermöglicht, ein einfaches elektrisches Verteilnetz bestehend aus Leistungs- und Leitungsknoten in Graphenform zu modellieren und für die gegebenen Parameter eine Lastflussanalyse durchzuführen. Weiters wurde in der Lösung ein Ansatz für eine Monte-Carlo-Simulation vorgesehen. Die Berechnungsergebnisse sind mit denen von matrizenbasierten Verfahren bzw. kommerziellen Softwarepaketen vergleichbar.

Abstract

Electrical networks are the backbone for the provision of electrical energy for human beings. Because of the increasing demand of electrical power, a continuous expansion of the underlying electrical networks is needed. Originally these networks were constructed and designed based on an strict top down structure. This means, that the structure which is needed from the primary feeders up to the consumers was designed taking into account the electrical parameters like currents, voltages and power and that there were no crucial modifications needed in the future. Due to the liberalization of the market and the increasing demand of energy, there is an increasing amount of electrical energy injected in the low- and medium voltage grids. Examples are wind mills, biomass power plants or solar cell constructions. The problem arises that these feeds are changing parameters in the network. Therefore it is necessary to analyze the networks in advance to see how the additional feeds are changing the parameters. A lot of algorithms have been developed for this purpose. This work focuses on the message based implementation of a load flow analysis algorithm which is based on a forward and backward sweep algorithm. For the implementation of the algorithm, the Open Source Framework OMNeT++ [3] is used, which makes it possible to create graph models and to exchange messages between the nodes of the graph. In addition the possibility of the integration of the algorithm into the architecture of the simulation platform DAVIC is discussed. DAVIC is a simulation platform of the Institut für Computertechnik, aiming to simulate money, information and energy flows in power networks. As a result a solution was implemented which makes it possible to create a simple electrical distribution network consisting of power and line nodes in a graph type model and to simulate the model with the given parameters. In addition a solution of a Monte Carlo Simulation was developed. The results were comparable to the results of matrice based algorithms or commercial application programs.

Danksagung

Ich möchte mich bei allen bedanken, die mich bei meinem Studium begleitet und oftmals unterstützt haben. Dies gilt vor allem für meine geschätzten Kollegen Enrico und Stephan, die mir während meines berufsbegleitenden Studiums eine große Hilfe waren. Weiters möchte ich mich besonders bei meinen Eltern bedanken, die mich von Beginn an bei diesem Vorhaben unterstützt haben. Leider durften sie die Fertigstellung dieser Arbeit nicht mehr erleben, sie werden im Gedanken trotzdem immer bei mir sein. Weiters möchte ich mich noch bei meinem Betreuer Dipl.-Ing. Dr.techn. Friederich Kupzog für seine äußerst kompetente Betreuung sowohl in fachlicher als auch organisatorischer Hinsicht und bei meiner Freundin Petra für ihre moralische Unterstützung aufs Herzlichste bedanken.

Inhaltsverzeichnis

Abkürzungen

DAVIC	Distributed Automation Via Implicit Control
DCHP	Domestic Combined Heat and Power
DEA	Dezentrale Erzeugungsanlage (Generator, Windkraftwerk, Solaranlage etc.)
FDLF	Fast Decoupled Load Flow
FES	Finite Event Set
GUI	Graphical User Interface
IDL	Interface Definition Language
INI	Initialization
$\mathbf{L_i}$	Leitungsmodell i
$\boldsymbol{Line_i}$	Leitungsknoten i
MSG	Message
$\boldsymbol{Node_i}$	Leistungsknoten i
NED	Network Description
OLTC	On Load Tap Changer (Stufentransformator)
OMG	Object Management Group
RDAP	Radial Distribution Analysis Package
$\mathbf{S_i}$	Leistungsmodell i
UCTE	Union for the Coordination of Transmission of Electricity (UCTE)
$\underline{\mathbf{I}}$	komplexer Strom
$\underline{\mathbf{S}}$	komplexe Leistung
$\underline{\mathbf{U}}$	komplexe Spannung
$\underline{\mathbf{Y}}$	komplexer Leitwert (Admittanz)
$\underline{\mathbf{Z}}$	komplexer Widerstand (Impedanz)

1. Einführung

Elektrische Energienetze stellen die Basis für die Versorgung einer modernen Infrastruktur mit elektrischer Energie dar. Ein elektrisches Netz besteht im Allgemeinen aus Einspeisepunkten und Übertragungsleitungen. An den Einspeisepunkten wird die Energie seitens der Energielieferanten wie z. B. Wasserkraft-, Atom- und Kleinkraftwerken unterschiedlicher Bau- und Machart zur Verfügung gestellt und in das Netz eingespeist. Weiters sind an den Übertragungsleitungen der Netze die Verbraucher angeschlossen, die die eingespeiste Energie aus dem Netz beziehen. Europaweit wird der Ausbau und Betrieb der Energienetze durch die Union for the Coordination of Transmission of Electricity (UCTE, http://www.ucte.org) gewährleistet. Im folgenden Kapitel soll ein Überblick über die Typen und den Aufbau von elektrischen Energienetzen gegeben werden.

1.1 Typen und Aufbau von Netzen

Energienetze können durch die Höhe der anliegenden Spannungen klassifiziert werden. Folgende Spannungsebenen sind im deutschsprachigen Raum Europas üblich:

- Hochspannungsebene ≥ 110 kV
- Mittelspannungsebene ≥ 10 kV
- Niederspannungsebene 230-1000 V

Netze auf Hochspannungsniveau werden primär zur Energieübertragung über weite Strecken verwendet. Der Grund dafür liegt in der Tatsache, dass durch hohe Spannungswerte die Verlustleistung (Wärmeentwicklung) verhindert werden kann. Weiters sind Netze dieser Art aufgrund von Sicherheitsmaßnahmen oftmals vermascht. Das bedeutet, dass im Netz Schleifen vorhanden sind, um im Fall von partiellen Netzausfällen, den Betrieb durch Umschalten auf andere Einspeisepunkte aufrecht erhalten zu können. Der Fall der Mittel- und Niederspannungsebene ist in den elektrischen Verteilnetzen gegeben. Diese dienen primär der Verteilung der elektrischen Energie in der Nähe der Endverbraucher. Weiters sind diese Strukturen meist als radiale Versorgungsstrukturen ausgelegt. Das bedeutet, dass in diesen Netzen keine Schleifen vorhanden sind. Es existieren nur Einfachverbindungen, die die Einspeisepunkte mit den Verbrauchern verbinden. In Abbildung 1 wird die übliche Struktur eines Übertragungsnetzes dargestellt. Aufgrund der einst rein zentral eingespeisten Energie, sind die heutzutage üblich anzutreffenden Netze als hierarchische Strukturen angelegt. Das bedeutet, dass die Auslegung der Belastbarkeit der Übertragungsleitungen im Vorhinein dimensio-

niert wurde bzw. auf die zukünftige Entwicklung hin fix festgelegt wurden. Wichtige Parameter eines Energienetzes lassen sich durch die Leistung $\underline{S}$ und der Spannung $\underline{U}$ der am Energienetz befindlichen Knoten und die Ströme $\underline{I}$ die auf den Leitungen auftreten, charakterisieren. Für die Betriebsführung als auch die Planung und gegebenenfalls notwendige Erweiterungen sind in der Vergangenheit eine Vielzahl von Analyseverfahren entwickelt worden. Vor allem in Hinblick auf Übertragungsnetze seien hier die Verfahren von Gauß-Seidel oder Newton-Raphson erwähnt. In Kapitel 1.3 soll näher auf diese Verfahren eingegangen werden. Als Knoten eines Energienetzes werden die Einspeisungen, Verbraucher und Abzweigungen zu anderen Pfaden und Teilnetzen bezeichnet.

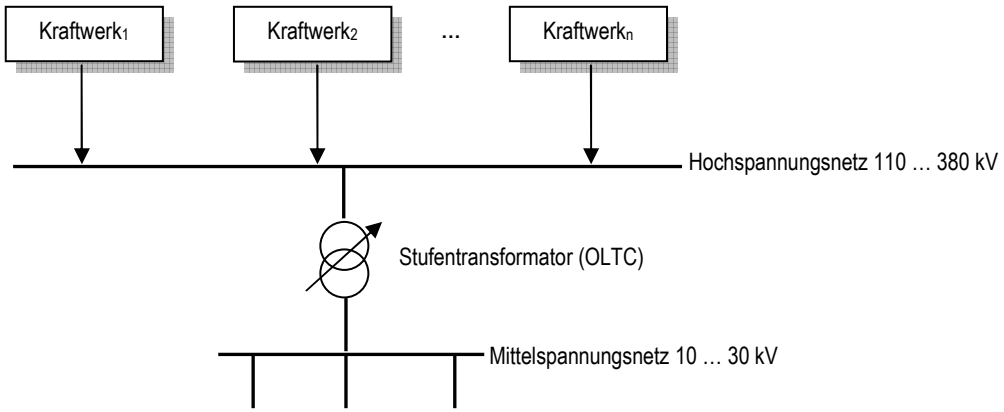


Abbildung 1 – Übertragungsnetz

In Abbildung 2 wird die Struktur eines typischen Verteilnetzes dargestellt. Ausgehend von der Hochspannungsebene werden über das Umspannwerk (On Load Tap Changer OLTC) die sogenannten Sammelschienen mit niedrigerer Spannung versorgt, von denen aus die einzelnen Versorgungslinien mit einer bestimmten Quellenspannung mit Energie gespeist werden. Die schwarzen Punkte stellen die Knoten des Netzes dar. An den Knoten sind z. B. die Verbraucher oder Generatoren oder andere Komponenten angeschlossen. Die Knoten werden durch Übertragungsleitungen miteinander verbunden. Als Komponenten sind u. a. Teilabschnitte, Verbraucher oder Einspeisungen zu betrachten. Als Besonderheit sei hier nochmals die radiale Struktur der Versorgungsäste erwähnt. Das bedeutet, dass in jedem Ast nur ein Pfad von der Versorgung zu den Endknoten bzw. Verbrauchern existiert. Weiters ist zu erwähnen, dass zwischen den Verhältnissen der Wirk- und Blindwiderstände der Leitungen zwischen herkömmlichen Übertragungsnetzen und Verteilnetzen ein Unterschied besteht. In Übertragungsnetzen ist der Wirkanteil im Vergleich zum Blindanteil zumeist geringer, da wie auch schon erwähnt, die Verlustleistung möglichst klein gehalten werden soll. In Verteilnetzen fällt das Verhältnis zwischen dem Wirkanteil R und dem Blindanteil X höher aus [Sto00, Tho03]. In den letzten Jahren kommt es aufgrund des erhöhten Energiebedarfs und der zunehmenden Liberalisierung der Energiemärkte immer öfter auch zu einer Einspeisung von elektrischer Energie in den Unterabschnitten der Verteilnetze. Dies wird auch als dezentrale Stromerzeugung bezeichnet. Dieser Umstand kann aufgrund der eklatanten Ungleichzeitigkeit zwischen Erzeugung und Verbrauch zur Verletzung von Spannungsebenen in den Netzen führen [Lug08].

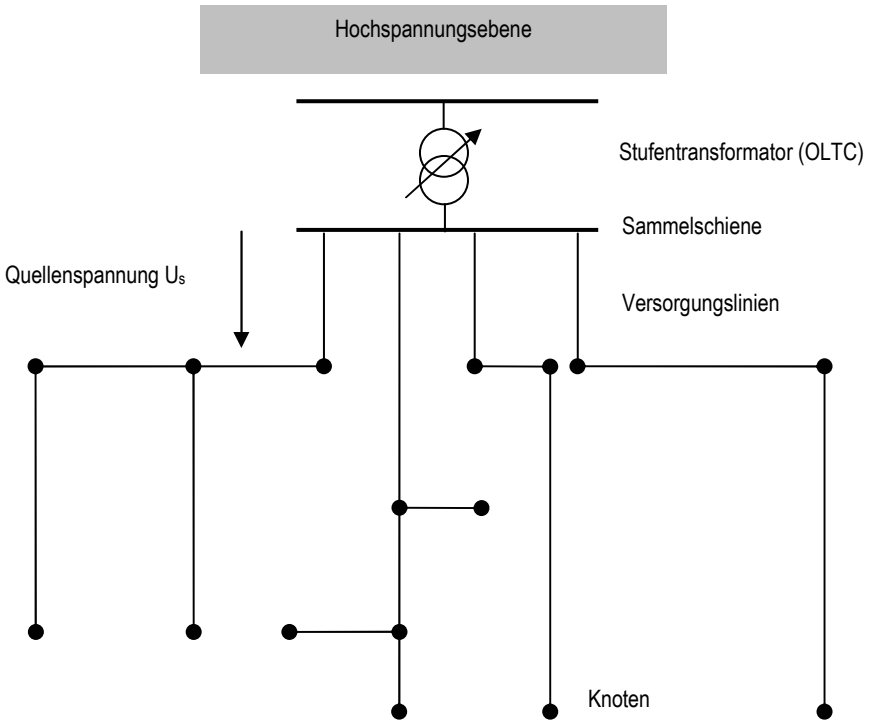


Abbildung 2 - Verteilnetz [Ker07]

1.2 Simulationsplattform DAVIC

Im Folgenden soll auf DAVIC eingegangen werden. DAVIC (Distributed Automation Via Implicit Control) ist eine Entwicklung des Instituts für Computertechnik. Dahinter steckt die Idee eine Simulationsplattform für verteilte Energieoptimierungsalgorithmen zu entwickeln. Im Folgenden seien die Gründe hierfür erwähnt [Pal08]. Zur Zeit existieren am Markt eine Vielzahl von kommerziellen Produkten wie MATLAB [4] oder DIgSILENT Power Factory [7] mit denen Energienetze simuliert werden können. Aufgrund von Forschungszwecken ist es allerdings oftmals notwendig, neue Komponenten die nicht als Standardkomponenten vorhanden sind, manuell zu modellieren. Diese können dann in die Simulation integriert werden. Ein Problem, welches hier auftritt ist, dass kein Wissen über das Standardverhalten bzw. die Implementierung der Standardkomponenten vorliegt, die in den Software-Produkten integriert sind. Es gibt auch Fälle in denen für einen Simulationsfall mehrere Software-Produkte gleichzeitig für unterschiedliche Aufgaben verwendet werden, da jedes Produkt je nach Anwendungsgebiet seine Vor- und Nachteile besitzt. Dies bedingt, dass auch der Datenaustausch zwischen diesen Anwendungen realisiert werden muss. Zusammenfassend lassen sich folgende drei Probleme bei dem genannten Ansatz herausarbeiten [Pal08]:

1. Simulationswerkzeuge unterscheiden sich in deren Spezialisierung und Unterstützung für unterschiedliche Aufgaben
2. Konzept der Objektorientierung ist von Vorteil, wenn Modellhierarchien komplexer werden

3. keines der gegenwärtig verfügbaren Simulationswerkzeuge unterstützt die Objektorientierung in Kombination mit einer genügend hohen Flexibilität um neue und ausgereifte Forschungskonzepte für Netzsimulationen zu implementieren

Die vom Institut neu entwickelte Simulationsplattform soll in Zukunft Last-, Kommunikations- und Geldflüsse simulieren können. In Abbildung 3 wird die konzeptionelle Architektur von DAVIC dargestellt.

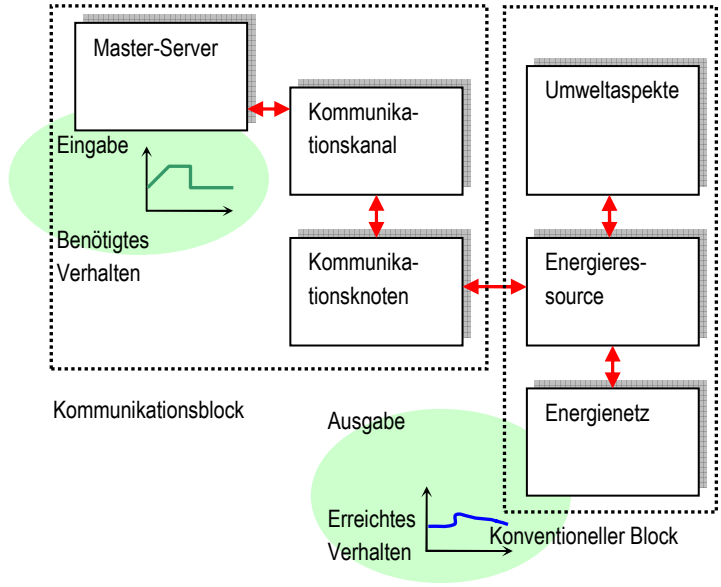


Abbildung 3 – DAVIC Architektur

In dieser Arbeit wird der Schwerpunkt auf den konventionellen Block der Architektur gelegt. Dieser Teil ist für die Modellierung des Verhaltens eines Energienetzes verantwortlich. Der Bereich Umweltaspekte beschreibt dabei den Einfluss von Umweltfaktoren auf das Netz. Darunter sind z. B. Veranstaltungen oder Wetterveränderungen zu verstehen, die eine Änderung im Energiebedarf des Netzes verursachen können. Als Energieressource wird jede Art von Last oder Generator betrachtet. Das Energienetz beschreibt das Verhalten des Netzes, also die Parameter die Energienetze beschreiben, also z. B. Ströme, Spannungen, Lastflüsse etc.. In dieser Arbeit liegt der Schwerpunkt auf der Analyse von Energienetzen. Im Folgenden sollen die gegenwärtigen Analyseverfahren mit Hilfe eines kleinen Beispielnetzes erläutert werden.

1.3 Stand der Technik

Im Bereich der Analyse von elektrischen Energienetzen haben sich in der Vergangenheit hauptsächlich zwei Verfahren etabliert. Namentlich handelt es sich dabei um die Gauß-Seidel-Iteration und das Newton-Raphson-Verfahren. Als Grundlage für die Verfahren werden die Maschen- und Knotengleichungen des zu analysierenden Netzes betrachtet, und in Matrizenform abgebildet. Damit steht ein Gleichungssystem zur Verfügung, das mit Hilfe der erwähnten Verfahren gelöst werden kann. Das in Abbildung 4 dargestellte Netz stellt ein einfaches Verteilnetz dar. Gegeben ist ein elekt-

risches Verteilnetz mit fünf Knoten N_1 bis N_5. Ausgehend von der Hochspannungsebene wird hier über den Stufentransformator eine Sammelschiene (N_1) über einen Stufentransformator mit 30 kV

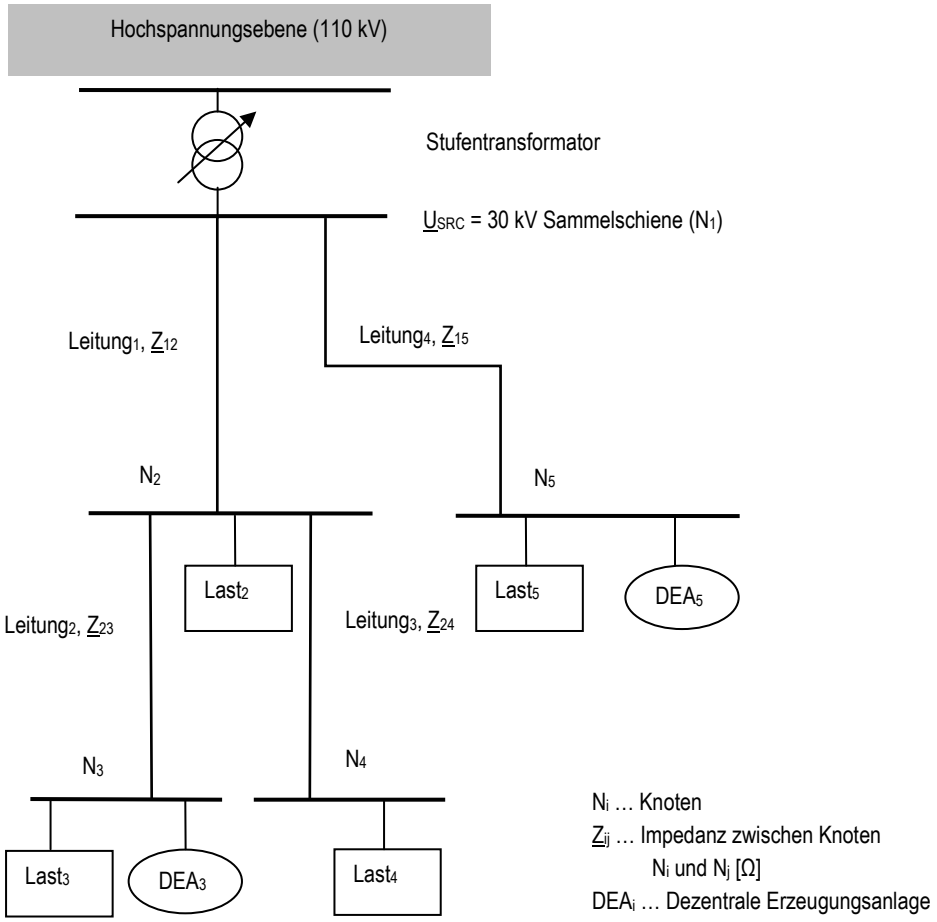


Abbildung 4 – Beispiel-Verteilnetz

versorgt. An dieser Sammelschiene sind zwei Verteiläste angeschlossen, die jeweils die Knoten N_2 bis N_4 und N_5 gesondert mit Energie versorgen. Die Knoten N_i des Netzes stellen Lasten, dezentrale Erzeugungsanlagen (DEA) und Verzweigungspunkte zu anderen Knoten dar. In Tabelle 1 wird eine Übersicht über die Parameter des Netzes gegeben. Dargestellt sind die Impedanzen der Leitungen $\underline{Z}_{ij}$ welche die Knoten N_i und N_j über einen bestimmten komplexen Widerstandswert miteinander verbinden. Weiters sind die Lasten $Last_i$ und die DEAs DEA_i (Generatoren) welche an den Knoten N_i angeschlossen sind, abgebildet. An den Knoten N_3 und N_5 sind Verbraucher und DEAs angeschlossen. Damit ergibt sich im Knoten N_3 eine Nettolast von $\underline{S}_3 = 1 + j1{,}452966$ MVA und im Knoten N_5 eine Nettoeinspeisung von $\underline{S}_5 = -2 + j0{,}4843222$ MVA. Die Verbraucher und Generatoren sind als *PQ*-Knoten modelliert. Das bedeutet, dass für jeden Knoten N_i die komplexe Scheinleistung $\underline{S}_i$ als Summe der Wirkleistung P_i und der Blindleistung Q_i angegeben ist. Der Berechnung wird das Verbraucherpfeilsystem zugrunde gelegt. Damit haben alle Einspeisungen ein negatives Vorzeichen.

Tabelle 1 - Parameterübersicht des Beispielnetzes

Parametername	Wert	Einheit
$\underline{Z}_{12}$	4,43 + j1,46	Ω
$\underline{Z}_{15}$	11,075 + j3,65	Ω
$\underline{Z}_{23}$	3,544 + j1,168	Ω
$\underline{Z}_{24}$	4,43 + j1,46	Ω
$Last_2$	5 + j2,421611	MVA
$Last_3$	3 + j1,452966	MVA
$Last_4$	1 + j0,4843222	MVA
$Last_5$	1 + j0,4843222	MVA
DEA_3	-2	MW
DEA_5	-3	MW

1.3.1 Gauß-Seidel-Iteration

Das gegebene Netz aus Abbildung 4 soll nun mit Hilfe der Gauß-Seidel-Iteration [Spr03] analysiert werden. Da Gauß-Seidel-Iteration dient der Lösung von linearen Gleichungssystemen. Gegeben seien die aus Tabelle 1 ersichtlichen Impedanzen Z_{ij} der die Knoten N_i und N_j verbindenden Leitungen, die Quellenspannung U_{SRC} an der Sammelschiene – auch Slack-Knoten genannt – und die Summenleistungen S_i der Knoten N_i. Es sollen die Spannungen an den Knoten N_1 bis N_5 berechnet werden, da hier aufgrund der Einspeisungen Verschiebungen bzw. Anhebungen zu erwarten sind. Im ersten Schritt müssen die Knotengleichungen aufgestellt werden. Diese beschreiben das Gleichgewicht zwischen den abfließenden und zufließenden Strömen bezüglich jedes Knotens N_i. In Anlehnung an das Ohm'sche Gesetz gilt für die komplexe Darstellung $\underline{U} = \underline{Z}\,\underline{I}$ und $\underline{I} = \underline{U}\,\underline{1} / \underline{Z} = \underline{U}\,\underline{Y}$. Daraus kann für das gegebenen Verteilnetz ein Gleichungssystem aufgestellt werden, welches auf der linken Seite des Systems die abfließenden Ströme und auf der rechten Seite die zufließenden Ströme, die aus den gegebenen Leistungsdaten berechnet werden können, darstellt. (1) zeigt das Gleichungssystem für alle Knoten N_1 bis N_5. Auf der linken Seite der Gleichung stehen die Admittanzen $\underline{Y}_{ij}$ die mit der Spannung $\underline{U}_i$ des jeweiligen Knotens N_i multipliziert werden und damit die Ströme berechnen, die vom jeweiligen Knoten in Richtung der Leitung abfließen. Auf der rechten Seite der Gleichung wird der Strom berechnet, der vom jeweiligen Knoten N_i aufgrund der Leistungsabnahme oder Einspeisung $\underline{S}_i$ und der anliegenden Spannung $\underline{U}_i$ abhängig ist, und in die Leitung hineinfließt. S_i bezeichnet die Summe der Einspeisungen abzüglich der Summe der Verbraucherleistungen. Aus diesem Ansatz ergibt sich das folgende Gleichungssystem in (2). Aus diesem System lassen sich nun durch einfache algebraische Umformungen die Unbekannten $\underline{U}_2$ bis $\underline{U}_5$ ausdrücken. $\underline{U}_1$ ist bekannt, diese entspricht der Quellenspannung am Slack-Knoten N_1. (3) stellt die umgeformten Gleichungen dar. Für die nun unbekannten Spannungen $\underline{U}_2$ bis $\underline{U}_5$ werden im Initialschritt Werte angenommen – z. B. $U_i = 1\ (2<=i<=5)$. Die neu berechneten Werte werden dann in die Gleichungen *GL 2'* bis

$$\begin{pmatrix} \underline{Y}_{11} & \underline{Y}_{12} & 0 & 0 & \underline{Y}_{15} \\ \underline{Y}_{21} & \underline{Y}_{22} & \underline{Y}_{23} & \underline{Y}_{24} & 0 \\ 0 & \underline{Y}_{32} & \underline{Y}_{33} & 0 & 0 \\ 0 & \underline{Y}_{42} & 0 & \underline{Y}_{44} & 0 \\ \underline{Y}_{51} & 0 & 0 & 0 & \underline{Y}_{55} \end{pmatrix} \begin{pmatrix} \underline{U}_1 \\ \underline{U}_2 \\ \underline{U}_3 \\ \underline{U}_4 \\ \underline{U}_5 \end{pmatrix} = \begin{pmatrix} (\underline{S}_1 / \underline{U}_1)^* \\ (\underline{S}_2 / \underline{U}_2)^* \\ (\underline{S}_3 / \underline{U}_3)^* \\ (\underline{S}_4 / \underline{U}_4)^* \\ (\underline{S}_5 / \underline{U}_5)^* \end{pmatrix} \quad \textbf{(1)}$$

$\underline{Y}_{ii}$ Summe der Leitwerte aller mit dem Knoten N_i direkt verbundenen Zweige

$\underline{Y}_{ij}$ Leitwert des die Knoten N_i und N_j direkt verbindenden Zweigs mit negativem Vorzeichen

$\underline{U}_i$ Knotenspannung für den Knoten N_i

S_i Summe aus Verbraucher- und DEA-Leistung

$$\begin{aligned}
&GL\ 1 && \underline{Y}_{11}\underline{U}_1 && +\underline{Y}_{12}\underline{U}_2 && +0 && +0 && +\underline{Y}_{15}\underline{U}_5 && = (\underline{S}_1 / \underline{U}_1)^* \\
&GL\ 2 && \underline{Y}_{21}\underline{U}_1 && +\underline{Y}_{22}\underline{U}_2 && +\underline{Y}_{23}\underline{U}_3 && +\underline{Y}_{24}\underline{U}_4 && +0 && = (\underline{S}_2 / \underline{U}_2)^* \\
&GL\ 3 && 0 && +\underline{Y}_{32}\underline{U}_2 && +\underline{Y}_{33}\underline{U}_3 && +0 && +0 && = (\underline{S}_3 / \underline{U}_3)^* \\
&GL\ 4 && 0 && +\underline{Y}_{42}\underline{U}_2 && +0 && +\underline{Y}_{44}\underline{U}_4 && +0 && = (\underline{S}_4 / \underline{U}_4)^* \\
&GL\ 5 && \underline{Y}_{51}\underline{U}_1 && +0 && +0 && +0 && +\underline{Y}_{55}\underline{U}_5 && = (\underline{S}_5 / \underline{U}_5)^*
\end{aligned} \quad \textbf{(2)}$$

GL 5' rückeingesetzt. Wird diese Iteration für eine genügend große Anzahl an Schritten durchgeführt, so erkennt man, dass die Spannungswerte gegen einen bestimmten Wert konvergieren. Ist die Differenz zwischen den alten und den neuen Werten hinreichend gering – z. B. kann hier ein Schwellwert definiert werden – so kann das Verfahren abgebrochen werden. Alle Spannungen im Verteilnetz sind damit berechnet. Die Vorteile des Verfahrens liegen in dessen Einfachheit. Als Nachteil erweist sich die langsame Konvergenz, manchmal auch Divergenz [Spr03]. Für das Beispielnetz aus Abbildung 4 ergeben sich für das Verfahren folgende in Tabelle 2 dargestellte Ergebnisse.

$$\begin{aligned}
&GL\ 1' && \underline{U}_1 = \underline{S}_1^* / \underline{Y}_{11}\ 1 / \underline{U}_1^* - \underline{Y}_{12} / \underline{Y}_{11}\underline{U}_2 - \underline{Y}_{15} / \underline{Y}_{11}\underline{U}_5 \\
&GL\ 2' && \underline{U}_2 = \underline{S}_2^* / \underline{Y}_{22}\ 1 / \underline{U}_2^* - \underline{Y}_{21} / \underline{Y}_{22}\underline{U}_1 - \underline{Y}_{23} / \underline{Y}_{22}\underline{U}_3 - \underline{Y}_{24} / \underline{Y}_{22}\underline{U}_4 \\
&GL\ 3' && \underline{U}_3 = \underline{S}_3^* / \underline{Y}_{33}\ 1 / \underline{U}_3^* - \underline{Y}_{32} / \underline{Y}_{33}\underline{U}_2 \\
&GL\ 4' && \underline{U}_4 = \underline{S}_4^* / \underline{Y}_{44}\ 1 / \underline{U}_4^* - \underline{Y}_{42} / \underline{Y}_{44}\underline{U}_2 \\
&GL\ 5' && \underline{U}_5 = \underline{S}_5^* / \underline{Y}_{55}\ 1 / \underline{U}_5^* - \underline{Y}_{51} / \underline{Y}_{55}\underline{U}_1
\end{aligned} \quad \textbf{(3)}$$

Die Ergebnisse wurden durch die Implementierung des Verfahrens in MATLAB gewonnen. Ab einer Anzahl von 80 Iterationen sind keine nennenswerten Differenzen zwischen den in jedem Iterationsschritt neu berechneten und den alten Werten mehr erkennbar. Man erkennt, dass es im rechten

Tabelle 2 – Ergebnisse Gauß-Seidel-Iteration

Knotenspannung	Wert (Betrag)	Entfernung vom Slack-Knoten
$\underline{U}_1$ (gegeben)	30 kV	0 km
$\underline{U}_2$	28,69 kV	10 km
$\underline{U}_3$	28,46 kV	18 km
$\underline{U}_4$	28,55 kV	20 km
$\underline{U}_5$	30,66 kV	25 km

Ast des Netzes (*Leitung*$_4$) zu einer Spannungsanhebung aufgrund der eingespeisten Leistung durch die Erzeugungsanlage *DEA*$_5$ kommt. Im linken Ast des Netzes kommt es zu einer Spannungsverringerung, da die eingespeiste Leistung durch die Erzeugungsanlage *DEA*$_3$ durch die am Knoten N_3 vorhandene Last kompensiert wird.

1.3.2 Newton-Raphson-Verfahren

Das Newton-Raphson-Verfahren [Kun94] dient der Lösung von nichtlinearen Gleichungssystemen. Für eine Funktion $x = f(x)$ gilt, dass $x_{n+1} = x_n - f(x) / (\partial f(x)/\partial x)$. Es handelt sich dabei um den Spezialfall einer Fixpunktiteration. Das Verfahren kann auf Gleichungssysteme angewendet werden. (4) zeigt ein System von n Gleichungen mit n Unbekannten. Für Matrizen würde die Formel auf $x_{n+1} = x_n - J(x)^{-1} f(x)$ lauten. $J(x)$ bezeichnet die Jacobi-Matrix, also die partiellen Ableitungen der Funktion

$$\begin{aligned} &f_1(x_1, x_2, \dots, x_n) = b_1 \\ &f_2(x_1, x_2, \dots, x_n) = b_2 \\ &\dots \quad \dots \quad \dots \quad \dots \quad \dots \\ &f_n(x_1, x_2, \dots, x_n) = b_n \end{aligned} \qquad \textbf{(4)}$$

nach allen Variablen. Die Invertierung von $J(x)$ ist allerdings numerisch ungünstig, deshalb wird die Gleichung auf die Form $\Delta x_n \, J(x) = f(x)$ umgeschrieben, und nach Lösung des Systems $x_{n+1} = \Delta x_n + x_n$ berechnet. Da dieses Verfahren ebenso ein Iterationsverfahren darstellt, wird auch hier mit Schätzwerten gestartet, und mittels Neuberechnung durch Rückeinsetzung auf die Konvergenz des Verfahrens gegen einen bestimmten Wert für die gegebenen Parameter, gesetzt. Die Initialwerte seien mit den Parametern $x_1^0, x_2^0, \dots, x_n^0$ gegeben. Dadurch ergeben sich die Korrekturwerte die notwendig sind um das Gleichungssystem zu erfüllen als $\Delta x_1^0, \Delta x_2^0, \dots, \Delta x_n^0$. (5) zeigt diesen Sachverhalt. Nach Anwendung des Taylorschen Lehrsatzes [Bar95] kann jede beliebig oft differenzierbare Funktion in eine Reihe entwickelt werden. Durch die untenstehende Gleichung in (5) mit dem Korrekturfaktor Δ und des Prinzips des Rückeinsetzens der neu berechneten Werte, kann das System in (6) umgewandelt werden. Die Matrix J wird als Jacobi-Matrix bezeichnet. Δf bezeichnet die Fehler die in jedem

$$
\begin{aligned}
&f_1(x_1^0 + \Delta x_1, x_2^0 + \Delta x_2, \ldots, x_n^0 + \Delta x_n) = b_1 \\
&f_2(x_1^0 + \Delta x_1, x_2^0 + \Delta x_2, \ldots, x_n^0 + \Delta x_n) = b_2 \\
&\ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \ldots \\
&f_n(x_1^0 + \Delta x_1, x_2^0 + \Delta x_2, \ldots, x_n^0 + \Delta x_n) = b_n
\end{aligned}
\qquad \Delta_i \ldots \text{Korrekturwert zu Startwert bzw. Letztwert um Gleichungssystem zu erfüllen} \qquad \textbf{(5)}
$$

$$
\underbrace{\begin{pmatrix} b_1 - f_1(x_1^0, x_2^0, \ldots, x_n^0) \\ b_2 - f_2(x_1^0, x_2^0, \ldots, x_n^0) \\ \ldots \\ b_n - f_n(x_1^0, x_2^0, \ldots, x_n^0) \end{pmatrix}}_{\Delta \boldsymbol{f}} = \underbrace{\begin{pmatrix} (\partial f_1/\partial x_1)^0 & (\partial f_1/\partial x_2)^0 & \ldots & (\partial f_1/\partial x_n)^0 \\ (\partial f_2/\partial x_1)^0 & (\partial f_2/\partial x_2)^0 & \ldots & (\partial f_2/\partial x_n)^0 \\ \ldots & \ldots & \ldots & \ldots \\ (\partial f_n/\partial x_1)^0 & (\partial f_n/\partial x_2)^0 & \ldots & (\partial f_n/\partial x_n)^0 \end{pmatrix}}_{\boldsymbol{J}} \underbrace{\begin{pmatrix} \Delta x_1 \\ \Delta x_2 \\ \ldots \\ \Delta x_n \end{pmatrix}}_{\Delta \boldsymbol{x}} \qquad \textbf{(6)}
$$

Schritt gemacht werden. Δx stellt die Korrekturwerte dar, die notwendig sind um die Gleichung zu erfüllen. Weiters werden die Produkte der Spannungen in Eulerscher Darstellung ausgedrückt. Als Grundlage für das System gelten die Zusammenhänge $\underline{S}_k = P_k + jQ_k = \underline{U}_k \underline{I}_k^*$ für den Knoten N_k. Für den Strom $\underline{I}_k$ *gilt* $\underline{I}_k = \sum \underline{Y}_{km} \underline{U}_m$, also die Summe aus allen zu- und abfließenden Strömen (siehe auch (1)) zwischen den Knoten N_k und N_m. Durch Einsetzen von $\underline{I}_k$ in $\underline{S}_k$ kann die Gleichung umgeschrieben werden auf $\underline{S}_k = P_k + jQ_k = \underline{U}_k \sum (G_{km} - jB_{km}) \underline{U}_m^*$ wobei gilt, dass $\underline{Y} = G + jB$ und daher $\underline{Y}^* = G - jB$ (siehe (7)).

$$
\begin{aligned}
\underline{S}_1 &= P_1 + jQ_1 = \underline{U}_1 ((G_{11} - jB_{11}) \underline{U}_1^* + (G_{12} - jB_{12}) \underline{U}_2^* + (G_{13} - jB_{13}) \underline{U}_3^* + (G_{14} - jB_{14}) \underline{U}_4^* + (G_{15} - jB_{15}) \underline{U}_5^*)) \\
\underline{S}_2 &= P_2 + jQ_2 = \underline{U}_2 ((G_{21} - jB_{21}) \underline{U}_1^* + (G_{22} - jB_{22}) \underline{U}_2^* + (G_{23} - jB_{23}) \underline{U}_3^* + (G_{24} - jB_{24}) \underline{U}_4^* + (G_{25} - jB_{25}) \underline{U}_5^*)) \\
\underline{S}_3 &= P_3 + jQ_3 = \underline{U}_3 ((G_{32} - jB_{32}) \underline{U}_2^* + (G_{32} - jB_{32}) \underline{U}_2^* + (G_{33} - jB_{33}) \underline{U}_3^* + (G_{34} - jB_{34}) \underline{U}_4^* + (G_{35} - jB_{35}) \underline{U}_5^*)) \\
\underline{S}_4 &= P_4 + jQ_4 = \underline{U}_4 ((G_{42} - jB_{42}) \underline{U}_2^* + (G_{42} - jB_{42}) \underline{U}_2^* + (G_{43} - jB_{43}) \underline{U}_3^* + (G_{44} - jB_{44}) \underline{U}_4^* + (G_{45} - jB_{45}) \underline{U}_5^*)) \\
\underline{S}_5 &= P_5 + jQ_5 = \underline{U}_5 ((G_{51} - jB_{51}) \underline{U}_1^* + (G_{52} - jB_{52}) \underline{U}_2^* + (G_{53} - jB_{53}) \underline{U}_3^* + (G_{54} - jB_{54}) \underline{U}_4^* + (G_{55} - jB_{55}) \underline{U}_5^*))
\end{aligned}
\qquad \textbf{(7)}
$$

Für die Eulersche Darstellung des Produkts aus $\underline{U}_k \underline{U}_m^*$ für jeden Knoten N_k folgt, dass

$$(U_k e^{j\theta k}) (U_m e^{-j\theta m}) = U_k U_m e^{j(\theta k - \theta m)} \qquad \textbf{(8)}$$

wobei $\theta_{km} = (\theta_k - \theta_m)$ gilt und somit

$$\underline{U}_k \underline{U}_m^* = U_k U_m (cos(\theta_{km} + jsin(\theta_{km}) \qquad \textbf{(9)}$$

gilt. Aus dem in Gleichung (7) dargestellten Gleichungssystem und der Eulerschen Darstellung aus Gleichung (8) und (9) lassen sich nun die Wirk- und Blindleistung als gesonderte Gleichungen wie in Gleichung (10) und Gleichung (11) gezeigt, in Realform darstellen. Das heißt, dass

$$P_k = U_k \sum (G_{km} U_m \, cos\theta_{km} + B_{km} U_m \, sin\theta_{km}) \qquad \textbf{(10)}$$

und

$$Q_k = U_k \sum (G_{km} U_m \sin\theta_{km} - B_{km} U_m \cos\theta_{km}) \quad \textbf{(11)}$$

gilt. Wirk- und Blindleistungen stellen damit Funktionen vom Betrag U und Phasenverschiebung θ dar. In (12) wird dieser Zusammenhang dargestellt. Ausgehend von diesem Gleichungssystem und dem System aus (6), lässt sich nun aus den gegebenen Gleichungen das System mit der Jacobi-Matrix J aufstellen. Die Jacobi-Matrix beschreibt dabei das Differential der Wirk- und Blindleistung jedes Knotens N_k nach den Spannungen U_m und Phasen θ_m ($k \neq m$), also jenen Spannungen und Phasen die auf den Bussen, die mit dem Knoten N_k in Verbindung stehen, auftreten.

$$\begin{aligned}
&P_1 = U_1 ((G_{11} U_1 \cos\theta_{11} + B_{11} U_1 \sin\theta_{11}) + (G_{12} U_2 \cos\theta_{12} + B_{12} U_2 \sin\theta_{12}) + 0 + 0 + (G_{15} U_5 \cos\theta_{15} + B_{15} U_5 \sin\theta_{15}))\\
&P_2 = U_2 ((G_{21} U_1 \cos\theta_{21} + B_{21} U_1 \sin\theta_{21}) + (G_{22} U_2 \cos\theta_{22} + B_{22} U_2 \sin\theta_{22}) + (G_{23} U_3 \cos\theta_{23} + B_{23} U_3 \sin\theta_{23}) + (G_{24} U_4 \cos\theta_{24} + B_{24} U_4 \sin\theta_{24}) + 0)\\
&P_3 = U_3 (0 + (G_{32} U_2 \cos\theta_{32} + B_{32} U_2 \sin\theta_{32}) + (G_{33} U_3 \cos\theta_{33} + B_{33} U_3 \sin\theta_{33}) + 0 + 0)\\
&P_4 = U_4 (0 + (G_{42} U_2 \cos\theta_{42} + B_{42} U_2 \sin\theta_{42}) + 0 + (G_{44} U_4 \cos\theta_4 + B_{44} U_4 \sin\theta_{44}) + 0)\\
&P_5 = U_5 ((G_{51} U_1 \cos\theta_{51} + B_{51} U_1 \sin\theta_{51}) + 0 + 0 + 0 + (G_{55} U_1 \cos\theta_{55} + B_{55} U_1 \sin\theta_{55}))\\
&Q_1 = U_1 (0 + (G_{12} U_2 \sin\theta_{12} - B_{12} U_2 \cos\theta_{12}) + 0 + 0 + (G_{15} U_5 \sin\theta_{15} - B_{15} U_5 \cos\theta_{15}))\\
&Q_2 = U_2 ((G_{21} U_1 \sin\theta_{21} - B_{21} U_1 \cos\theta_{21}) + (G_{22} U_2 \sin\theta_{22} - B_{22} U_2 \cos\theta_{22}) + (G_{23} U_3 \sin\theta_{23} - B_{23} U_3 \cos\theta_{23}) + (G_{24} U_4 \sin\theta_{24} - B_{24} U_4 \cos\theta_{24}) + 0)\\
&Q_3 = U_3 (0 + (G_{32} U_2 \sin\theta_{32} - B_{32} U_2 \cos\theta_{32}) + (G_{33} U_3 \sin\theta_{33} - B_{33} U_3 \cos\theta_{33}) + 0 + 0)\\
&Q_4 = U_4 (0 + (G_{42} U_2 \sin\theta_{42} - B_{42} U_2 \cos\theta_{42}) + 0 + (G_{44} U_4 \sin\theta_4 - B_{44} U_4 \cos\theta_{44}) + 0)
\end{aligned} \quad \textbf{(12)}$$

$$\begin{pmatrix}
P_1^{def} - P_1(\theta_{11}^0, \theta_{12}^0, 0, 0, \theta_{15}^0, U_1^0, U_2^0, 0, 0, U_5^0)\\
P_2^{def} - P_1(\theta_{21}^0, \theta_{22}^0, \theta_{23}^0, \theta_{24}^0, 0, U_1^0, U_2^0, U_3^0, U_4^0, 0)\\
P_3^{def} - P_1(0, \theta_{32}^0, \theta_{33}^0, 0, 0, 0, U_2^0, U_3^0, 0, 0)\\
P_4^{def} - P_1(0, \theta_{42}^0, 0, \theta_{44}^0, 0, 0, U_2^0, 0, U_4^0, 0)\\
P_5^{def} - P_1(\theta_{51}^0, 0, 0, 0, \theta_{55}^0, U_1^0, 0, 0, 0, U_5^0)\\
Q_1^{def} - Q_1(\theta_{11}^0, \theta_{12}^0, 0, 0, \theta_{15}^0, U_1^0, U_2^0, 0, 0, U_5^0)\\
Q_2^{def} - Q_1(\theta_{21}^0, \theta_{22}^0, \theta_{23}^0, \theta_{24}^0, 0, U_1^0, U_2^0, U_3^0, U_4^0, 0)\\
Q_3^{def} - Q_1(0, \theta_{32}^0, \theta_{33}^0, 0, 0, 0, U_2^0, U_3^0, 0, 0)\\
Q_4^{def} - Q_1(0, \theta_{42}^0, 0, \theta_{44}^0, 0, 0, U_2^0, 0, U_4^0, 0)\\
Q_5^{def} - Q_1(\theta_{51}^0, 0, 0, 0, \theta_{55}^0, U_1^0, 0, 0, 0, U_5^0)
\end{pmatrix} = \begin{pmatrix}
\partial P_1/\partial\theta_1 \dots \partial P_1/\partial\theta_5 \; \partial P_1/\partial U_1 \dots \partial P_1/\partial U_5\\
\partial P_2/\partial\theta_1 \dots \partial P_2/\partial\theta_5 \; \partial P_2/\partial U_1 \dots \partial P_2/\partial U_5\\
\partial P_3/\partial\theta_1 \dots \partial P_3/\partial\theta_5 \; \partial P_3/\partial U_1 \dots \partial P_3/\partial U_5\\
\partial P_4/\partial\theta_1 \dots \partial P_4/\partial\theta_5 \; \partial P_4/\partial U_1 \dots \partial P_4/\partial U_5\\
\partial P_5/\partial\theta_1 \dots \partial P_5/\partial\theta_5 \; \partial P_5/\partial U_1 \dots \partial P_5/\partial U_5\\
\partial Q_1/\partial\theta_1 \dots \partial Q_1/\partial\theta_5 \; \partial Q_1/\partial U_1 \dots \partial Q_1/\partial U_5\\
\partial Q_2/\partial\theta_1 \dots \partial Q_2/\partial\theta_5 \; \partial Q_2/\partial U_1 \dots \partial Q_2/\partial U_5\\
\partial Q_3/\partial\theta_1 \dots \partial Q_3/\partial\theta_5 \; \partial Q_3/\partial U_1 \dots \partial Q_3/\partial U_5\\
\partial Q_4/\partial\theta_1 \dots \partial Q_4/\partial\theta_5 \; \partial Q_4/\partial U_1 \dots \partial Q_4/\partial U_5\\
\partial Q_5/\partial\theta_1 \dots \partial Q_5/\partial\theta_5 \; \partial Q_5/\partial U_1 \dots \partial Q_5/\partial U_5
\end{pmatrix} \begin{pmatrix}
\Delta\theta_1\\ \Delta\theta_2\\ \Delta\theta_3\\ \Delta\theta_4\\ \Delta\theta_5\\ \Delta U_1\\ \Delta U_2\\ \Delta U_3\\ \Delta U_4\\ \Delta U_5
\end{pmatrix} \quad \textbf{(13)}$$

Δf $\qquad$ J $\qquad$ Δx

(13) zeigt nun das Ausgangsgleichungssystem mit Jacobi-Matrix J. Δf stellt die Differenzen zwischen dem jeweils letzten Leistungswert (P_i^{def} und Q_i^{def}) und dem aktuell berechneten Wert aufgrund

der Schätzwerte für θ_i und U_i dar. Die Jacobi-Matrix beschreibt die partiellen Ableitungen der Leistungen P_i nach θ_m und U_m und Δx die notwendigen Korrekturwerte um das System zu erfüllen. Als Grundlage für das Verfahren wird angenommen, dass alle Knoten als *PQ*-Knoten modelliert sind. Das bedeutet, dass die konsumierte bzw. eingespeiste Leistung in der komplexen Form $\underline{S} = P + jQ$ gegeben ist. Die Besiedelung der Jacobi-Matrix *J* entspricht der der Knoten-Admittanz-Matrix in Gleichung (1) die schon im Gauß-Seidel-Verfahren als Grundlage herangezogen wurde. Das Gleichungssystem kann mittels Verfahren, die auf die Besiedelungsdichte der Matrizen in Betracht ziehen, gelöst werden [Kun94]. Auf die Durchrechnung wird hier aufgrund des Umfanges verzichtet, es sollte gezeigt werden, wie die gegebenen Netzparameter zum Aufbau des Gleichungssystems verwendet werden können. Das Newton-Raphson-Verfahren ist heute eines der meist gebräuchlichsten Verfahren in der Lastfluss-Analyse und wird von vielen kommerziellen Software-Produkten am Markt implementiert. Das Verfahren besitzt eine sehr gute Konvergenzrate unter der Voraussetzung, dass die Startwerte nahe am tatsächlichen Wert liegen [Kun94].

1.3.3 Fast Decoupled Load Flow (FDLF)

Erwähnt werden soll auch noch die Methode des Fast Decoupled Load Flow (FDLF) [Kun94, Sto74, Ame89], welche eine Erweiterung bzw. Näherung des Newton-Raphson-Verfahrens dargestellt. Bei dieser Methode werden einige Vereinfachungen bezüglich der Jacobi-Matrix *J* durchgeführt und damit der Berechnungsaufwand reduziert. Dabei werden die schwache physikalische Kopplung zwischen *P* und *U* und zwischen *Q* und θ in Betracht gezogen. Die allgemeine Form des Newton-Raphson-Verfahrens, wie in (14) gezeigt, kann dann wie in (15) dargestellt vereinfacht werden. Diese Form wird nach weiteren Vereinfachungen, wie das Weglassen von Elementen die hauptsächlich den reaktiven Lastfluss beeinflussen, erreicht [Kun94].

$$\begin{pmatrix} \Delta P \\ \Delta Q \end{pmatrix} = \begin{pmatrix} \partial P/\partial\theta & \partial P/\partial U \\ \partial Q/\partial\theta & \partial Q/\partial U \end{pmatrix} \begin{pmatrix} \Delta\Theta \\ \Delta U \end{pmatrix} \qquad \textbf{(14)}$$

(15) zeigt weiters wie aus dieser Form durch Iteration der jeweils aktuelle ΔP- und ΔQ-Wert (*i-ter* Iterationsschritt) berechnet werden kann. Die Berechnung wird abgebrochen wenn, $\Delta P/U$ und $\Delta Q/U$ kleiner einem bestimmten Schwellwert sind.

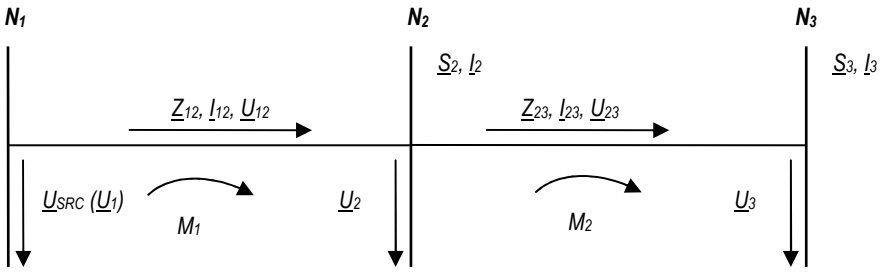
$$\begin{aligned} &\Delta P/U = B'\Delta\theta && B'.\ B'' \ldots \textit{Blindleitwerte des Netzwerkes} \\ &\Delta Q/U = B''\Delta U && \\ &\Delta P = P^{def} - P(\theta^{i-1}, U^{i-1}) && i \ldots \textit{i-ter Iterationsschritt} \\ &\Delta Q = Q^{def} - Q(\theta^{i-1}, U^{i-1}) && \end{aligned} \qquad \textbf{(15)}$$

Abschließend sei erwähnt, dass von diesem Verfahren zwei Typen, namentlich das XB- und BX-Schema existieren [Ame89]. Dabei wird auf die Konvergenzeigenschaften des Verfahrens in System mit geringen (XB) bzw. hohen (BX) *R/X*-Verhältnissen der Leitungen eingegangen, da sich gezeigt

hat, dass das ursprüngliche Verfahren (XB) eher schlecht bei Netzwerken mit hohen *R/X*-Verhältnissen konvergiert [Kun94][Ame89].

1.3.4 Ladder Iterative Technique

Die bisher vorgestellten matrizenbasierten Verfahren sind die gebräuchlichsten und werden vor allem zur Analyse von Übertragungsnetzen verwendet. Verteilnetze weisen radiale Topologien auf, also nur Einfachverbindungen von den Einspeisepunkten zu den Endverbrauchern. Es finden sich zumeist keine Schleifen in derartigen Systemen. Iterative matrizenbasierte Verfahren werden in diesen Netzen eher wenig verwendet, da diese Verfahren auf radiale Strukturen angewendet, eine eher schlechte Konvergenz aufweisen [Ker07, Tre70]. Gegeben sei ein einfaches Verteilnetz in Abbildung 5 mit drei Knoten N_i ($1 <= i <= 3$) und den dazugehörigen komplexen Leistungen $\underline{S}_i$ und



Z_{ij} … komplexe Impedanz zwischen Knoten N_i und N_j (i ≠ j), $Z_{ij} = R + jX\ [\Omega]$

S_i … komplexe Leistung am Knoten N_i, $S_i = P + jQ\ [VA]$

I_i, I_{ij} … komplexer Strom am Knoten N_i bzw. zwischen Knoten N_i und N_j (i ≠ j)

U_{SRC} … Quellenspannung am Slack-Knoten *[V]*

U_i … Spannung am Knoten N_i *[V]*

M_i … i-te Masche, es wird das Verbraucherpfeilsystem angenommen

Abbildung 5 – Variablendefinition in einem einfachen Verteilnetz

den komplexen Impedanzen $\underline{Z}_{ij}$ der Leitungen zwischen den Knoten N_i und N_j (i ≠ j). Die verstärkten senkrechten Linien in der Abbildung bezeichnen die Busse bzw. Knoten des Netzes. Die waagrechten Linien stellen die Verbindungsleitungen zwischen den Bussen dar. Zuerst wird am Endknoten N_3 die zugehörige Spannung $\underline{U}_3$ berechnet. Daher wird angenommen, dass an den Knoten keine Lasten vorhanden sind. Damit ergibt sich die Spannung $\underline{U}_3 = \underline{U}_{SRC}$, und am Knoten N_3 kann der zugehörige Strom aus $\underline{I}_3 = (\underline{S}_3/\underline{U}_3)^*$ berechnet werden. Dieser Schritt wird auch als Initialschritt bezeichnet, d. h. alle Endknoten der am Einspeisepunkt angeschlossenen Teiläste des Verteilnetzes werden mit der am Einspeisepunkt vorhandenen Quellenspannung $\underline{U}_{SRC}$ initialisiert. Sind alle Endknoten initialisiert, kann mit dem Rückwärtsschritt des Verfahrens begonnen werden. Dazu werden beginnend bei den an den Endknoten angeschlossenen Verbindungen bzw. Leitungen die Spannungsabfälle die auf den Leitungen auftreten, berechnet. Aufgrund der gegebenen Impedanz $\underline{Z}_{23}$ kann nun der Spannungsabfall $\underline{U}_{23} = \underline{Z}_{23}\,\underline{I}_{23}$ und $\underline{I}_3 = \underline{I}_{23}$ berechnet werden. Die Maschengleichung M_2 ergibt sich zu $-\underline{U}_2 +$

$\underline{U}_{23} + \underline{U}_3 = 0$. Daraus folgt $\underline{U}_2 = \underline{U}_3 + \underline{U}_{23}$. Am Knoten N_2 kann jetzt der Strom aus $\underline{S}_2$ und $\underline{U}_2$ mittels $\underline{I}_2 = (\underline{S}_2/\underline{U}_2)^*$ berechnet werden. Der Spannungsabfall $\underline{U}_{12}$ ergibt sich zu $\underline{U}_{12} = \underline{Z}_{12}\,\underline{I}_{12}$. $\underline{I}_{12}$ ergibt sich aus der Kirchhoffschen Knotenregel zu $\underline{I}_{12} = \underline{I}_2 + \underline{I}_{23}$. Aus der Maschengleichung M_1 kann nun $\underline{U}_1 = \underline{U}_2 + \underline{U}_{12}$ berechnet werden. Aus dem neu berechneten Wert $\underline{U}_1$ und dem gegebenen Wert $\underline{U}_{SRC}$ kann eine Differenz berechnet werden. Es sei $\Delta = ||\underline{U}_{SRC}| - |\underline{U}_1||$ und $t = 0{,}001\ \underline{U}_{SRC}$ ein gegebener Schwellwert. Die *0,1%* der Quellenspannung $\underline{U}_{SRC}$ werden für gewöhnlich als akzeptable Schwankungsbreite um die Quellenspannung $\underline{U}_{SRC}$ herum akzeptiert. Daher kann nun $\Delta <= t$ verglichen werden und bei Erfüllung dieser Bedingung das Verfahren abgebrochen werden. Als Resultat stehen alle Knotenspannungen und Ströme fest. Wird die Bedingung nicht erfüllt, beginnt die Berechnung von vorne. Im Unterschied zum Initialschritt stehen nun alle Ströme auf den Leitungen bzw. Knoten fest. Im nun folgenden Vorwärtsschritt werden die neuen Spannungen berechnet. Die Spannung $\underline{U}_2$ kann erneut mittels Maschengleichung aus $M_1 = -\underline{U}_1 + \underline{U}_{12} + \underline{U}_2 = 0$ zu $\underline{U}_2 = \underline{U}_1 - \underline{U}_{12}$ berechnet werden. Die Spannung am Endknoten N_3, $\underline{U}_3$ kann analog dazu aus $\underline{U}_3 = \underline{U}_2 - \underline{U}_{23}$ berechnet werden. Abbildung 6 zeigt nochmals den Ablauf des Verfahrens.

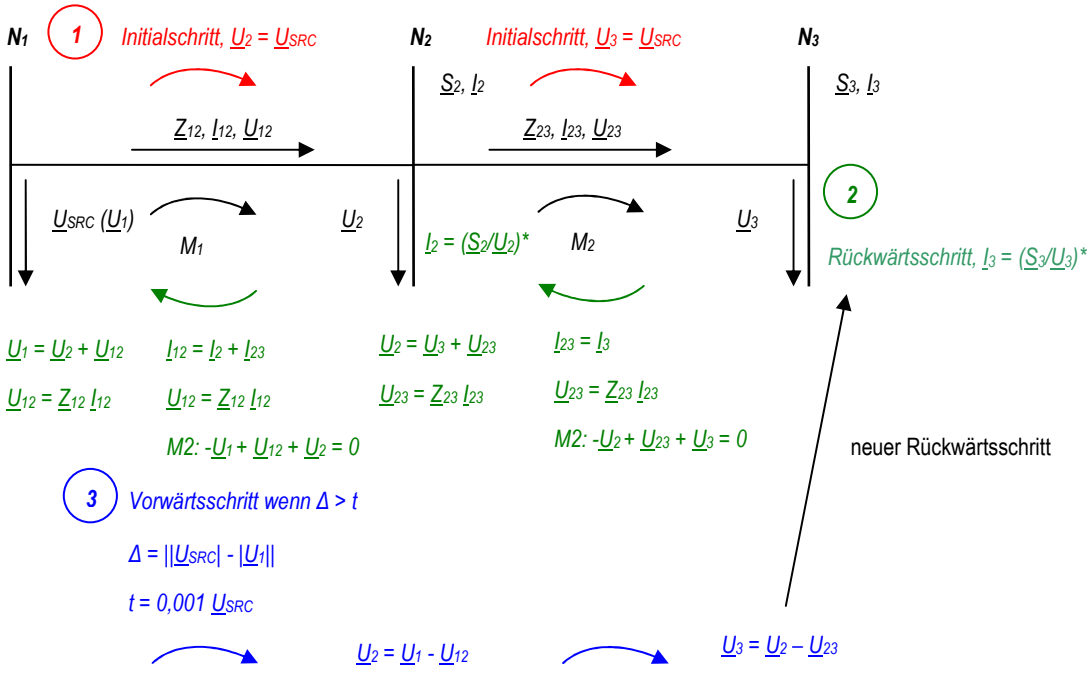


Abbildung 6 – Ladder Iterative Technique

Aus Abbildung 6 lassen sich folgende allgemeine Zusammenhänge für die Berechnung der Knotenströme und -spannungen sowie Spannungsabfälle auf den Leitungen ableiten:

Im Initialschritt (1) werden alle Spannungen von allen Knoten N_i des Verteilnetzes ausgehend vom Slack-Knoten N_1 mit der Quellenspannung initialisiert, d. h. $\underline{U}_i = \underline{U}_{SRC}$ $(2 <= i <= n)$, n … Anzahl Knoten. Im darauf folgenden Rückwärtsschritt (2) werden die Ströme in den Endknoten N_{Ti} mittels $\underline{I}_{Ti} = (\underline{S}_{Ti}/\underline{U}_{Ti})^*$ berechnet. In den restlichen Knoten N_i ergeben sich die Ströme aufgrund der Kirchhoffschen Knotenregel zu $\underline{I}_i = (\underline{S}_i/\underline{U}_i)^* + \sum \underline{I}_j$ $(i+1 <= j <= n)$, n … Anzahl Nachfolgeknoten, also

aus der Summe des Stromes der sich am Knoten durch die Leistung und die gegebene Spannung berechnet und der Summe der Ströme aus allen Nachfolgerknoten. Für die Spannungsabfälle auf den Leitungen, die die Knoten N_i und N_j verbinden ergibt sich $\underline{U}_{ij} = \underline{Z}_{ij}\ \underline{I}_{ij}$, wobei $\underline{I}_{ij} = \underline{I}_j$ gilt. Nach Abarbeitung des Rückwärtsschrittes wird die Abweichung Δ von der Quellenspannung berechnet und mit dem anfänglich festgelegten Schwellwert t verglichen. Sofern $\Delta > t$ wird mit dem Vorwärtsschritt (3) fortgesetzt. Die neuen Spannungen $\underline{U}_i$ für die Knoten N_i ergeben sich dann aufgrund der Kirchhoffschen Maschenregel zu $\underline{U}_i = \underline{U}_{i-1} - \underline{U}_{ij}$ $(1 < i <= n)$, n … Anzahl Knoten, wobei $\underline{U}_{ij}$ den Spannungsabfall auf der Leitung die die Knoten N_i und N_j verbindet, beschreibt. Wird ein Endknoten N_{Ti} erreicht, so wird erneut mit dem Rückwärtsschritt (2) fortgesetzt. Wird nun am Slack-Knoten die Bedingung $\Delta <= t$ erfüllt, so ist die Berechnung abgeschlossen. Sodann stehen alle Spannungen und Ströme im Netz fest. Bislang wurde das zu betrachtende Netz dahingehend einfach gehalten, dass keine Verteilknoten N_{Vi} in dem Netz enthalten waren. Als Verteilknoten N_{Vi} sei ein Knoten mit mehreren angeschlossenen Verbindungslinien bezüglich Nachfolgeknoten gegeben. Abbildung 7 zeigt das Beispielnetz aus Abbildung 4 mit Kennzeichnung der Knoten anhand der soeben durchgeführten Klassifikation.

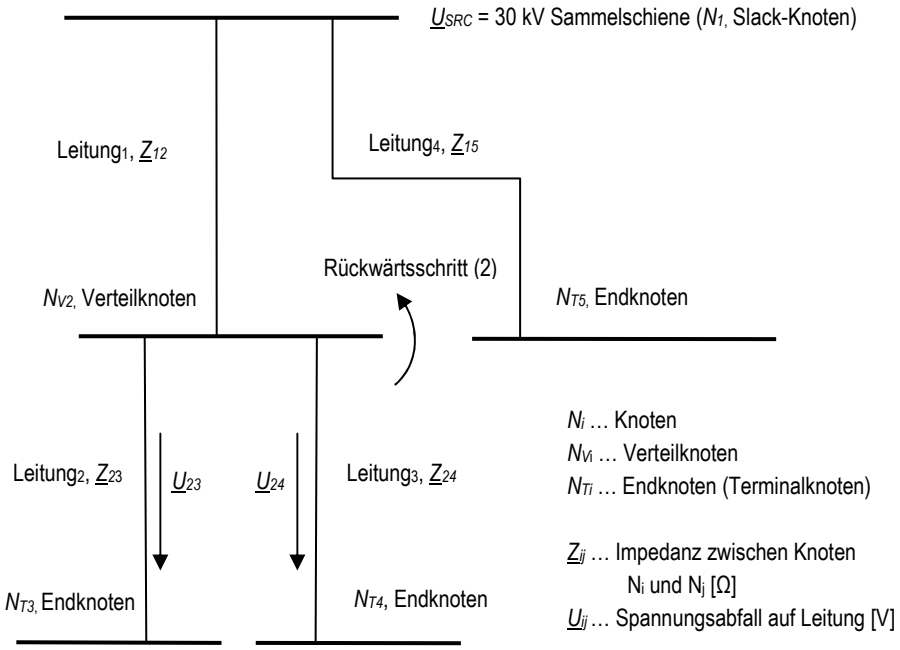


Abbildung 7 - Knotenklassifikation Beispielnetz

Für die Knoten N_1, N_{T3}, N_{T4} und N_{T5} ist die Berechnung aufgrund der Beschreibung im letzten Absatz klar ersichtlich. Ein Problem ergibt sich für den Verteilknoten N_{V2} im Fall des Rückwärtsschrittes (2). Aufgrund der an N_{V2} angeschlossenen Leitung$_2$ und Leitung$_3$ ergeben sich unter der Annahme unterschiedlicher Impedanzen $\underline{Z}_{23}$ und $\underline{Z}_{24}$ unterschiedliche Spannungsabfälle $\underline{U}_{23}$ und $\underline{U}_{24}$. Einer dieser Spannungsabfälle wird für N_{V2} als anliegenden Spannungswert $\underline{U}_2$ herangezogen um den Strom $\underline{I}_2 = (\underline{S}_2/\underline{U}_2)^* + \underline{I}_3 + \underline{I}_4$ zu berechnen. $\underline{U}_2$ wird in diesem Fall mit $\underline{U}_{24}$ angenommen. Als Entscheidungsgrundlage soll im Weiteren angenommen werden, dass im Fall eines Verteilknotens N_{Vi} im Fall des Rückwärtsschrittes bei der Berechnung des Knotenstromes aufgrund der gegebenen Knotenleistung $\underline{S}_i$ die Knotenspannung $\underline{U}_i$ gleich des Spannungsabfalls auf der letzten zu berechnen-

den Leitung sein soll [Ker07]. Abschließend sei erwähnt, dass die Laufzeit des Verfahrens mit den matrizenbasierten Verfahren vergleichbar ist, ja diese sogar unterbietet [Tho03].

1.4 Aufgabenstellung

In dieser Arbeit soll der konventionelle Block von DAVIC durch die Implementierung eines Lastflussanalyse-Verfahrens erweitert werden. Da es in jüngster Zeit aufgrund der Liberalisierung des Energiemarktes und des zunehmenden Bedarfes an elektrischer Energie vermehrt zur Einspeisung von Energie auf der Mittel- und Niederspannungsebenen kommt, soll dabei ein besonderes Augenmerk auf die elektrischen Verteilnetze in diesen Spannungsebenen gelegt werden. Dabei sind vor allem die Integrierbarkeit und Erweiterbarkeit der Lösung in Betracht zu ziehen, als auch die bisher üblichen Verfahren auf deren Umsetzbarkeit in dem nachrichtenbasierten Framework OMNeT++ [3] näher zu betrachten. Um dies zu bewerkstelligen ist eine solide Literaturrecherche bezüglich üblicher Lastflussanalyse-Verfahren und dementsprechender energietechnischer Grundlagen durchzuführen sowie die Konzepte des Software Engineerings auf die Problemstellung anzuwenden.

Der Umfang der Arbeit lässt sich also wie folgt zusammenfassen:

- Einarbeitung Grundlagen (Energietechnik, Lastflussanalyse, Verteilnetze)
- Einarbeitung DAVIC bzw. OMNeT++
- Auswahl eines geeigneten Verfahrens zur Berechnung der Lastflüsse in einem elektrischen Verteilnetz
- Implementierung des Verfahrens und Vergleich der Ergebnisse mit Referenzlösungen
- bestmögliche Integration des Verfahrens in die Architektur von DAVIC
- Modularität bzw. Erweiterbarkeit der Lösung
- Probleme der Lösung bzw. Ausblick und Verbesserungsvorschläge

Die Einarbeitungsphase umfasst dabei im Besonderen die Einarbeitung in die notwendigen elektrotechnischen Grundlagen, im Speziellen die Einarbeitung in die Theorie der Energieübertragung betreffend elektrischer Verteilnetze. Weiters sind die üblichen Lastflussanalyse-Verfahren auf deren Anwendbarkeit und Performance hinsichtlich der Implementierung in OMNeT++ [3] hin zu untersuchen. Als Voraussetzung ist dabei die Kenntnis des Frameworks und dessen bisherige Verwendung für das Projekt DAVIC zu betrachten. Danach soll ein geeignetes Verfahren ausgewählt und mittels OMNeT++ [3] umgesetzt werden. Die Ergebnisse der Implementierung sind mit Testnetzen zu verifizieren bzw. mit Ergebnissen aus kommerziellen Anwendungspaketen wie DIgSILENT [7] zu vergleichen. Im Anschluss daran sollen die Probleme bzw. Grenzen der Lösung aufgezeigt werden, und ein Ausblick auf Verbesserungsvorschläge gegeben werden.

Der Schwerpunkt der Arbeit liegt somit auf der Auswahl eines geeigneten Verfahrens hinsichtlich Laufzeit und Speicherauslastung als auch der Möglichkeit das Verfahren in die Gesamtarchitektur von DAVIC integrieren zu können.

1.5 Verwandte Arbeiten

In den folgenden Abschnitten soll auf einige Arbeiten eingegangen werden, die sich in jüngerer Vergangenheit mit der Implementierung von Algorithmen zur Lastflussanalyse auseinandergesetzt haben. Dabei sind einerseits Arbeiten, die die elektrotechnischen Grundlagen der Verfahren bzw. deren Anwendbarkeit zur Analyse von elektrischen Verteilnetzen beleuchten und andererseits jene Arbeiten die Abbildung von elektrischen Netzen und deren technische Zusammenhänge in eine softwaremäßig saubere Implementierung ermöglichen, zu betrachten. Wie schon in Kapitel 1.3 beschrieben, existieren zur Analyse von Energienetzen einerseits die matrizenbasierten Verfahren, die vor allem zur Analyse von Übertragungsnetzen verwendet werden, da sie dort aufgrund der Parameter der Netze sehr gute Konvergenzeigenschaften besitzen und sich auch in der praktischen Anwendbarkeit bewährt haben. Andererseits haben sich vor allem im Bereich der elektrischen Verteilnetze Varianten der Ladder Iterative Technique [Ker07] durchgesetzt [Tho03, Shi88, Khu06].

1.5.1 Analyseverfahren

In Kersting [Ker07] wird die Modellierung und Analyse von elektrischen Verteilnetzen behandelt. Das Buch behandelt im Wesentlichen die mathematische Modellierung von allen Komponenten eines Verteilnetzes. Dabei wird z. B. auf die Art von Lasten, auf die Impedanzen und Admittanzen von Freileitungen und verlegten Leitungen, auf Leitungsmodelle, Spannungsregulierung und Lastmodelle näher eingegangen. Das für diese Arbeit wesentliche Kapitel behandelt die Lastflussanalyse von Verteilnetzen, wobei das vom Autor entwickelte Verfahren der Ladder Iterative Technique für den balancierten und unbalancierten Lastfall zum Einsatz kommt. Die grundsätzliche Vorgangsweise des Verfahrens wurde schon in Kapitel 1.3.4 beschrieben.

In der Arbeit von Thomson et al. [Tho03] wird die Analyse von elektrischen Verteilnetzen im Niederspannungsbereich (unter 500 V, auch als Secondary Distribution Network bezeichnet) behandelt. Weiters wird auch auf den Fall der verteilten Generierung von Energie (Distributed Generation, DG) eingegangen. Im Folgenden sollen die wesentlichen Aussagen der Arbeit erläutert werden. Die verteilte Einspeisung von Energie kann zu lokalen Überspannungen bzw. thermischen Überlastungen der Leitungen führen. Daher stellt das Risiko einer Überspannung im Netz oftmals ein Hindernis für den Anschluss von verteilten Energieeinspeisungen wie z. B. Windkraftwerken o. ä. dar. Deshalb wurde von den Autoren eine Simulationssoftware entwickelt, um die Effekte von Solarenergie-Technologien und häuslicher Energieerzeugung (Domestic Combined Heat and Power, DCHP) abschätzen zu können. Das Newton-Raphson-Verfahren versagt manchmal bezüglich der Konvergenz im Bereich der elektrischen Verteilnetze, aufgrund der hohen *R/X*-Verhältnisse der Verbindungsleitungen. Deshalb wird auch hier für die Analyse die Ladder Iterative Technique [Ker07] herangezogen. Als Vorteile werden hier nicht nur die besseren Konvergenzeigenschaften, sondern auch der verminderte Aufwand bei der Berechnung erwähnt. Die Manipulation von großen Admittanz-Matrizen entfällt hier. Ein Problem der Anwendung des Verfahrens in Niederspannungsnetzen, stellt die hohe stochastische Natur der individuellen Lasten dar. Eine Steady-State-Betrachtung (Momentaufnahme) bzw. Auswertung und Analyse der Daten, ist daher nur für einen sehr kurzen Zeitraum gültig bzw. hat nur einen dementsprechend geringen Wert. Als eine Möglichkeit sei hier die Anwen-

dung von Diversitäts-Faktoren genannt, die aufgrund einer eingehenden Auswertung von lokalen Lasten bestimmt werden könnten. Für gewöhnlich verlässt man sich eher auf den historischen Verlauf, und versucht daraus typische Lastprofile die den Erwartungswert für die Zukunft am besten beschreiben, zu entwickeln. Im Fall der zunehmend verteilten Generierung von Energie erweist sich diese Vorgangsweise als schwierig bzw. undurchführbar. In der Arbeit wird weiters auf die Realisierung einer Monte-Carlo-Variante des Algorithmus eingegangen. Darunter ist eine wiederholte Simulation unter Berücksichtigung von sich verändernden Parametern zu verstehen, d. h. es wird eine Steady-State-Analyse für jeden Zeitschritt, in dem sich Parameter geändert haben, durchgeführt. Das Verfahren wurde sowohl für den balancierten als auch unbalancierten Fall in MATLAB [4] implementiert und die Ergebnisse mit kommerziellen Produkten verglichen. Der Unterschied zwischen einem balancierten bzw. unbalancierten Netz besteht darin, dass in einem Dreiphasennetz auch einphasige Verbraucher existieren können. Das bedeutet weiters, dass die Energieabnahme in den drei Phasen unterschiedlich sein kann, es damit zu einem Ungleichgewicht zwischen Energiezufuhr und Abnahme in den einzelnen Phasen kommen kann (siehe auch Abschnitt 2.1). Als Testsystem wurde ein Netzwerk mit 40 Knoten die 21 Häuser als Verbraucher versorgen, betrachtet. Als Lastmodell wurden gegebene konstante Lasten herangezogen. Es wurde ein voller Tag mit 1-Minuten-Intervallen simuliert. In diesen 1440 Schritten wurden alle komplexen dreiphasigen Spannungen und Ströme aller Knoten berechnet. Die Analyse des gesamten Netzes dauerte 100 s. Es sind noch weitere Erweiterung der Analyse notwendig um verteilte Einspeisungen behandeln zu können. Die Arbeit betrachtet also hauptsächlich die Ladder Iterative Technique im unbalancierten Lastfall und erweitert diese um das Monte-Carlo-Verfahren um auch Parameteränderungen berücksichtigen zu können [Tho03].

Das von Mok et al. [Mok99] verfasste Papier beschreibt eine Methode zur Analyse von Verteilnetzen für den balancierten und unbalancierten Lastfall. Die Methode wurde in C++ implementiert. Aufgrund der bereits in den anderen Abschnitten erwähnten Nachteilen der matrizenbasierten Systemen, wird auch hier das Netzwerk als System bestehend aus Bussen die durch Verbindungsleitungen oder Schalter die an einen Quellenbus angeschlossen sind, modelliert. Jeder Bus kann eine bestimmte Last, Kompensationslast (Kapazität oder Induktanz) oder Generator beinhalten. Die Leitungen werden als Serie von Impedanzen $\underline{Z} = R + jX$ aufgefasst. Die Lasten an einem Bus werden über ein allgemeines Modell implementiert, welches es erlaubt jede Last als konstante Leistung, konstanten Strom, konstante Impedanz oder ein exponentielles Modell zu implementieren [Mok99]. Das eigentliche Verfahren entspricht der Ladder Iterative Technique, beruht also auf einem Vorwärts- und Rückwärtsschrittverfahren in dem abwechselnd die Ströme und Spannungen der Busse berechnet werden bis die Bedingung am Quellenbus erfüllt ist. Ein Unterschied besteht in der Berechnung von Teilabzweigungen (Kreuzungspunkten) im Netz. Dies ist notwendig da ein Netz mehrere Ebenen von Verzweigungen haben kann, also Abzweige von Abzweigen existieren können. Zuerst werden alle Busse der Hauptversorgung mit der Spannung des Slack-Knotens initialisiert, wie dies aus der Ladder Iterative Technique bekannt ist. Der Rückwärtsschritt wird vom höchsten Level von Teilabzweigen aus in Richtung der Hauptversorgung durchgeführt. Damit wird die gesamte Last für jeden Abzweig berechnet. Danach werden die Ströme an jedem Knoten der Hauptversorgungslinie an dem auch die Teilabschnitte angeschlossen sind, berechnet. Danach werden die neuen Spannungen für jeden Knoten berechnet und im Fall der Hauptversorgungslinie mit der Spannung des Slack-

Knotens verglichen und bei Bedarf eine neue Iteration durchgeführt. Für das nächst höhere Level werden nun ebenso die Knotenspannungen mit der Spannung am jeweiligen Quellen-Knoten des Teilabzweigs initialisiert und die Ströme für diese Knoten des Teilabzweigs berechnet. Danach werden die Spannungen der Knoten berechnet und ein Vergleich am jeweiligen Startknoten des Netzes durchgeführt und bei Bedarf eine neue Iteration gestartet. Dies wird solange durchgeführt bis alle Levels von Teilabzweigen abgearbeitet sind. Jeder Teilabzweig wird somit gesondert berechnet. Deshalb müssen die jeweiligen neu berechneten Spannungswerte in einem Array dessen Länge der Anzahl der Teilabschnitte entspricht, gespeichert werden. Die Ergebnisse wurden mit den Ergebnissen der existierenden Software ERACS [8] verglichen und stimmten überein. Die Performance ist im Vergleich zu matrizenbasierten Verfahren verbessert. Die Dauer der Konvergenz wird durch die Höhe der Verzweigungsgrade bestimmt. Dabei stellte sich heraus, dass ab einem Grad >= 5 bzw. bei sehr stark belasteten Systemen die Konvergenzdauer zunimmt. Als verwendete Lastmodelle wurden statische Modelle herangezogen. Dies wird als noch verbesserungswürdig von den Autoren erwähnt [Mok99].

Auch Khushalani et al. [Khu06] behandelt im Wesentlichen die Ladder Iterative Technique zur Behandlung von elektrischen Verteilnetzen. Als Komponenten des Netzes wird zwischen Komponenten die in den Ästen (Verbindungslinien) vorhanden sind und den Knotenkomponenten des Netzes unterschieden. Als erstere werden Modelle für Leitungen, Transformatoren und Umschalter bezeichnet. Als Modelle für Knoten werden Kapazitäten, Lasten und verteilte Generatoren angegeben. Im Besonderen wird die Modellierung von Generatoren im 3-Phasen-Fall behandelt, wobei hier auf die Unterschiede zwischen der Modellierung als *PQ*-Knoten und *PV*-Knoten eingegangen wird. Im letzterem Fall ist der Generator als spannungsabhängige Stromquelle zu betrachten, welche vom Algorithmus nicht direkt zu berechnen ist. Bei *PV*-Knoten ist nur die Wirkleistung *P* und Spannung *U* gegeben. Das Problem liegt dabei darin, dass nach einem durchgerechneten Lastfluss für jede Phase des *PV*-Knotens ein bestimmter neu berechneter Spannungswert anliegt. Sofern diese drei Spannungen nicht balanciert sind, so resultieren daraus natürlich auch drei unbalancierte Ströme die vom Generator eingespeist werden. Sofern der Unterschied zwischen den Strömen zu groß ist, käme es zu einer Abschaltung des Generators [Khu06]. Die Generatorspannung kann typischerweise durch die Angabe eines Mitsystems (Positive Sequence Component) [Spr03] kontrolliert werden. Daraus folgt, dass auch die Ströme in einem unbalancierten Fall berechnet werden können. Zusammenfassend lässt sich erwähnen, dass sich nach einem Vergleich der neu berechneten Spannung mit den gegebenen Klemmenspannungen des Generators eine Differenz berechnen lässt und diese mit einem Schwellwert verglichen wird. Bei Verletzung der Bedingung wird reaktive Leistung *Q* berechnet, um die Spannungskompensation am Knoten durchzuführen und weiters die Ströme zu berichtigen. Das Programm wurde von den Autoren in MATLAB [4] implementiert. Simulationen wurden für den unbalancierten 3-Phasen-Fall für *PQ*- und *PV*-Knoten als Generatoren durchgeführt. Die Ergebnisse wurden mit RDAP (Radial Distribution Analysis Package) verglichen. Die Ergebnisse waren für Systeme ohne Generatoren und mit Generatoren modelliert als *PQ*-Knoten vergleichbar. Für *PV*-Knoten war kein direkter Vergleich möglich, da RDAP keine Modellierung von *PV*-Knoten unterstützt [Khu06].

Shirmohammadi et al. [Shi88] beschreibt ebenso ein Vorwärts-/Rückwärtsschritt-Verfahren analog der Ladder Iterative Technique von [Ker07]. In den Rückwärtsschritten werden die Ströme der Lei-

tungen in den Vorwärtsschritten die Spannungen der Knoten berechnet. Das Verfahren lässt sich auf radiale Strukturen als auch auf schwach vermaschte Netze anwenden. Die Lösung liegt darin, dass bestimmte Knoten ausgewählt werden und damit die Maschen im Netz aufgelöst werden um im Endeffekt eine radiale Struktur zu erzeugen. An beiden entstehenden Endknoten einer aufgelösten Masche müssen demnach entsprechende Ströme eingespeist werden um die Betriebsbedingungen des Netzes nicht zu verändern [Shi88]. Die Autoren führen an, dass sich die Methode als einfach und berechnungstechnisch effizient und numerisch robust herausgestellt hat. Nach eingehenden Tests wurde weiters festgestellt, dass das Verfahren im Vergleich zum Newton-Raphson-Verfahren signifikant besser hinsichtlich der Performance abschneidet, wenn es zur Lösung von radialen Strukturen bzw. schwach vermaschten Netzen herangezogen wird [Shi88].

Thomson et al. [Tho07] beschreibt auch die Anwendung der Ladder Iterative Technique anhand einem realen Netz in UK. Betrachtet wird ein Niederspannungsnetz bei dem in den vernetzten Haushalten eine große Anzahl von Kleinsterzeugern eingespeist durch Solarzellen betrachtet wird. Für die Simulation wurde ein Testnetz mit 1262 Haushalten bzw. Kunden herangezogen. Für knapp die Hälfte nämlich 629 wurde die Annahme einer Mikrogenerierung von Energie in Form von Solarzellen angenommen. Das Netz wurde mit 1-Minuten-Zeitschritten simuliert. Die simulierten Werte wurden mit aktuellen Messwerten von drei ausgewählten Testhäusern verglichen. Die Abweichungen zwischen den errechneten und den Messwerten waren akzeptabel.

1.5.2 Software-Modellierung der Algorithmen

Die folgenden drei Arbeiten beleuchten die softwaremäßige Modellierung von Lastflussanalyse-Verfahren für elektrische Verteilnetze. Dabei wird vor allem auf die Modularisierung bzw. Entwurfsmuster-basierte Realisierung der Methoden Wert gelegt.

In der Arbeit von Bouchard et al. [Bou98] werden die Systemkomponenten eines Energieverteilnetzes und deren Abbildung in SW-Objekten behandelt. Energieverteilsysteme können als Graphen aufgefasst werden. Die Komponenten eines Systems umfassen Leitungen, Kabel, Transformer, …, usw. Die Arbeit lässt sich wie folgt zusammenfassen. Zunächst wird ein SW-Entwurfsmuster definiert. *„A pattern can be thought of as an abstraction with a name that provides a solution to a recurring problem within a certain context, and in the presence of numerous competing concerns“* [Bou98, S1]. Im Weiteren wird in dieser Arbeit vor allem die Sinnhaftigkeit der Verwendung des Abstract-Factory- [Fre04], Composite- [Fre04] und Iterator-Entwurfsmusters [Fre04] im Allgemeinen beschrieben, ohne jedoch konkret auf die Anwendung im Fall eines Verteilnetzes einzugehen. Auf die konkrete Anwendbarkeit bzw. Sinnhaftigkeit dieser Entwurfsmuster wird im Abschnitt 4.3 dieser Arbeit noch eingegangen.

Das von Selvan et al. [Sel04] verfasste Papier beschreibt die Abbildung eines Energieverteilnetzes in ein objektorientiertes SW-Design. Abbildung 8 zeigt das Klassendiagramm das die Autoren vorschlagen. Anhand dieser Abbildung lässt sich ein Verteilnetz wie folgt aufbauen und kategorisieren. Ein Verteilsystem (Distribution System) besteht aus einem Bus (Bus), Versorger bzw. Sammelschiene (Feeder), Last (Load) und Nebenwiderstand (Shunt). Als Typen von Bussen existieren die Spezialvarianten Hauptknoten (Root Node), Verzweigknoten (Fork Node) und Endknoten (Terminal

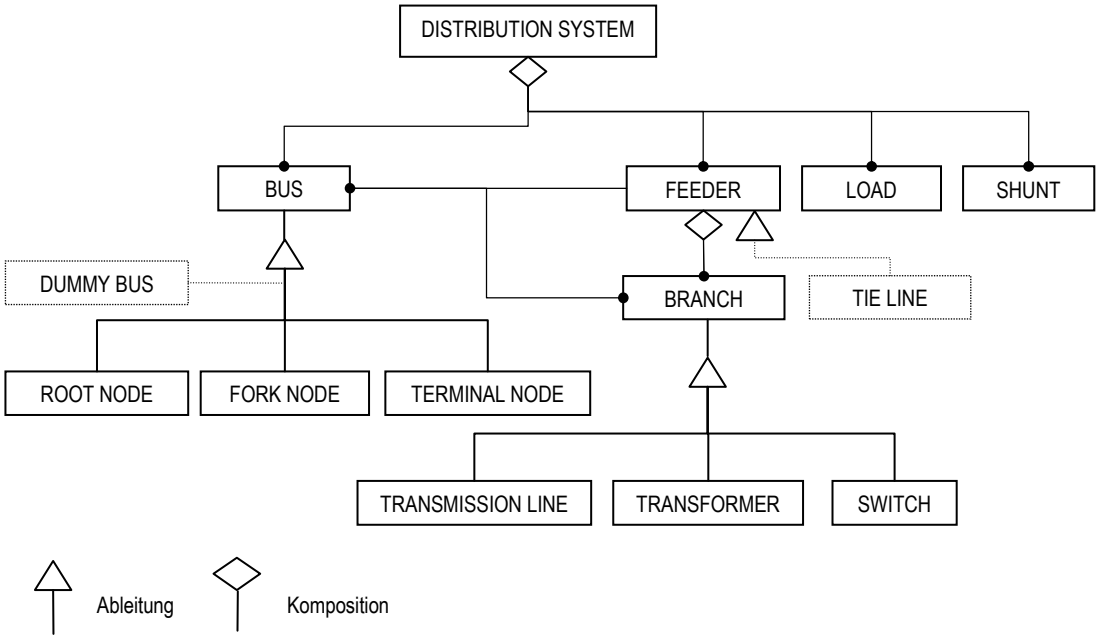


Abbildung 8 – Verteilsystem Klassenentwurf [Sel04]

Node). Ein Feeder besteht aus Zweigen (Branch). Als Typen von Zweigen existieren Übertragungsleitungen (Transmission Lines), Transformatoren (Transformer) und Schalter (Switches). Dieses Konzept ist für rein radiale Systeme gut geeignet. Im Fall eines schwach vermaschten Systems, kann für jede Masche ein sogenannter Loop Break Point eingeführt werden. Aus diesem ergeben sich dann für die Modellierung zwei zusätzliche Komponenten Dummy Bus und Anknüpfung (Tie Line). Auch diese Elemente werden in Abbildung 8 als strichlierte Elemente dargestellt. Im Folgenden werden die Definition der Elemente des Klassenentwurfs wie in der Arbeit von Selvan et al. [Sel04] beschrieben zusammengefasst. Als Last (Load) wird ein Basiselement im elektrischen Lastfluss bezeichnet, welches Leistung aufnimmt oder erzeugt. Das entsprechende Objekt beinhaltet die Leistung und Spannung als primäre Attribute. Als Nebenwiderstand (Shunt) werden Elemente bezeichnet die reaktive Leistung für das Energienetz zur Verfügung stellen können. Das primäre Attribut des Objekts Nebenwiderstand ist durch die konstante Admittanz (Leitwert) gegeben, die aus der reaktiven Leistung und der Spannung berechnet werden kann. Als Bus wird ein Verknüpfungspunkt (Knoten) bezeichnet an dem verschiedene Elemente wie Leitungen, Transformatoren, Lasten und Nebenwiderstände angeschlossen sind. Das entsprechende Objekt enthält die Attribute Busnummer (Bus Number), Spannungsbetrag (Voltage Magnitude), Winkel (Angle) und die Referenzen der angeschlossenen Elemente (Objekte). Als Zweig (Branch) wird ein Objekt bezeichnet, welches zwischen zwei Bussen platziert ist, als konkrete Instanzen kommen eine Übertragungsverbindung (Transmission Line), ein Transformator (Transformer) oder ein Schalter (Switch) in Frage. Als Versorger bzw. Sammelschiene (Feeder) wird eine Instanz bezeichnet die aus Zweigen (Branches) besteht, und die Energie vom Umspannwerk in die verschiedenen Abschnitte eines Verteilnetzes einbringt. Das Verteilnetz selbst, kann als Komposition der soeben beschriebenen Objekte aufgefasst werden. Wie schon erwähnt, werden für Maschen zwei zusätzliche Komponenten eingeführt um eine

rein radiale Struktur zu erzeugen. Ein Dummy Bus leitet sich vom allgemeinen Bus-Objekt ab. Erzeugt wird ein Dummy Bus durch ein Parent Bus Objekt am Loop Break Point. Das Objekt Tie Line leitet sich vom Objekt Feeder ab, und kann als Feeder aufgefasst werden, welcher an einem Knoten startet und an einem Dummy Bus endet [Sel04]. Als Algorithmus für die Berechnung eines Netzes wird ein Vorwärts- bzw. Rückwärtsschritt-Verfahren verwendet. Aufgrund der Vorteile des objektorientierten Designs, können die Berechnung in jedem Schritt lokal innerhalb des Feeder-Objektes durchgeführt werden. Das Ergebnis der Arbeit lässt sich wie folgt zusammenfassen. Das Design wurde in MS Visual C++ 6.0 implementiert auf einer Pentium III Maschine ausgeführt. Jede Klasse (Objekt) besitzt Methoden um Referenzen auf andere Klassen (Objekte) abzubilden. Die Methode der Klasse Bus *injectcurrent()* bewerkstelligt die Stromeinspeisung am Loop Break Point, welcher den Stromfluss durch eine Schleife simuliert. Die Methoden *doforwardcalculations()* und *dobackwardcalculations()* des Objekt Feeder, berechnen die Spannungen bzw. Ströme in den Vorwärts- bzw. Rückwärtsschritten des Algorithmus. Die Methode *updatevoltage()* aktualisiert die Spannung des empfangenden Endbusses während des Vorwärtsschrittes [Sel04]. Durch die objektorientierte Modellierung kann die Struktur des Netzes abgebildet werden. Der Ansatz führt zu erhöhter Flexibilität und Erweiterbarkeit bezüglich Entwicklung von elektrischen Energieverteilsystemen. Das Design wurde verwendet um eine Lastflussanalyse für Verteilnetze zu entwickeln. Die Ergebnisse des Programms stimmten exakt mit Ergebnissen aus der Literatur überein [Sel04].

In dieser Arbeit [Li04] wird ein Framework für die Entwicklung von Energieverteilsystemen vorgeschlagen. Als Framework wird hier ein Ansatz bezeichnet, um SW-Wiederverwendung für ein bestimmte Problemdomäne zu erreichen. In der Arbeit wird eine Schichtenarchitektur mit strikter Top-Down-Abhängigkeit vorgeschlagen, um Abhängigkeiten zu verringern. Ein Framework kann auch als halbfertiggestellte Anwendung interpretiert werden [Li04]. Die Arbeit präsentiert ein High-Level-Design eines SW-Frameworks, welches die gemeinsamen Attribute und Verhalten in Energieverteilsystem-Analysealgorithmen abstrahiert [Li04]. Das Ergebnis der Arbeit lässt sich wie folgt zusammenfassen. Das Framework wurde entwickelt um eine möglichst hohe Wiederverwendung bzw. Erweiterbarkeit zu gewährleisten. Die Anwendungsentwickler müssen ihre Applikation nicht von Neuem entwickeln. Anstatt dessen, können sie das Framework wiederverwenden und nach ihren Bedürfnissen abändern bzw. erweitern. Das Framework unterstützt vor allem die Entwicklung von typischen Verteilsystemalgorithmen, wie Lastflussanalyse, Kurzschlussberechnungen und Verlässlichkeitsabschätzungen (Reliability Assessment). Zu diesem Zweck müssen die Topologie der Komponenten und des Systems modelliert werden. Um verteilte Berechnungen zu ermöglichen, stellt das Framework auch eine Schnittstelle für Distributed Computing mit dem Concurrent-Server-Client-Mode zur Verfügung. Das Ziel des Frameworks ist wie folgt zusammenzufassen:

- Multiphasenmodell bereitstellen
- Modellierung der Topologie und Traversierungsmethoden
- Modellierung von fundamentalen Algorithmen
- Modellierung von Distributed Computing

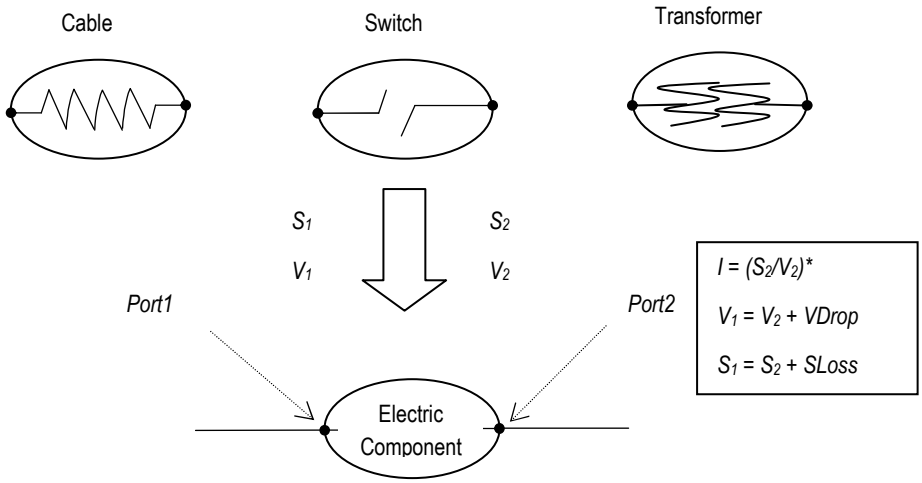


Abbildung 9 – Port-Modell Komponenten [Li04]

Als Traversierungsmethoden werden Algorithmen zum Durchlaufen der Topologie bezeichnet. Um die Komponenten zu modellieren wurde ein sogenanntes Zweiknoten-Modell (Two Node Model) wie in Abbildung 9 gezeigt eingeführt. Jede Komponente beinhaltet einen Upstream Port oder Port1, und einen Downstream Port oder Port2. Der Upstream Port liegt näher bei der Spannungsquelle, währenddessen der Downstream Port näher an den Verbrauchern liegt. Spannung, Strom und Lastfluss an jedem Port werden als individuelle interne Attribute jeder Komponente betrachtet. Deshalb ist bei diesem Ansatz keine Notwendigkeit gegeben, eine knoten- bzw. kantenbasierte Darstellung der Topologie oder Matrizenabbildung des Netzes durchzuführen. Durch die Ports jeder Komponente können dieser miteinander verbunden werden.

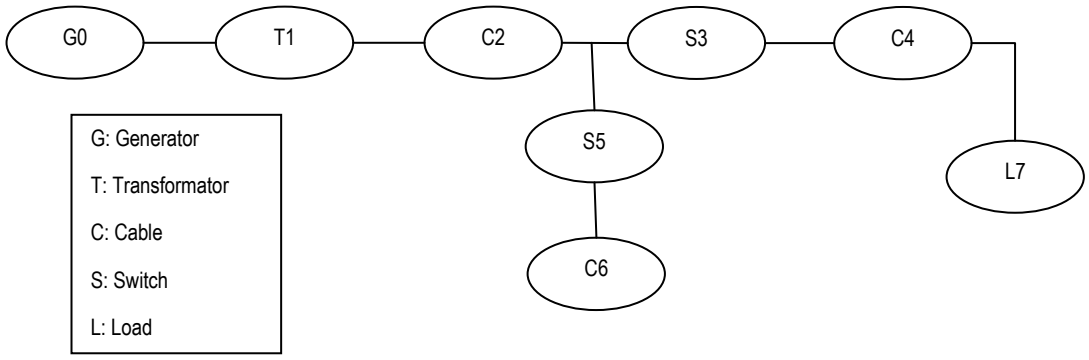


Abbildung 10 – Frameworkmodellierung (einfaches Netz) [Li04]

Abbildung 10 zeigt die daraus resultierende komponentenbasierte Darstellung eines einfachen Netzes. Das Framework wurde mittels der von der Object Management Group (OMG) definierten Inteface Definition Language (IDL) offen gegenüber heterogenen Umgebungen entwickelt. Mittels IDL, kann das Framework auf jede gängige objektoriente Sprache wie z. B. C++ umgesetzt werden. Das Framework selbst ist in C++ implementiert und bietet seine Schnittstellen im IDL-Stil gegenüber den zu entwickelnden Applikationen an (siehe Abbildung 11). Das Framework bietet als Schnittstellen in IDL-Implementierung an, die von den Applikationen aus aufgerufen werden können. Das

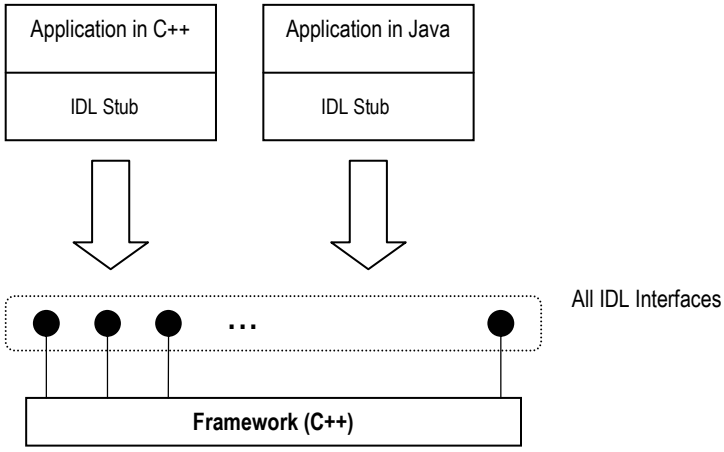


Abbildung 11 – Framework (IDL) [Li04]

Framework selbst besteht aus vier verschiedenen Schichten (Layers). Dieses Schichtenmodell besitzt eine strikte Top-Down-Abhängigkeit. Jede Schicht hängt immer nur von der unmittelbar nachfolgenden Schicht ab. Damit sind niedrigeren Schichten einfacher wiederverwendbar als die weiter oben liegenden. Der Aufbau des Schichtenmodells wird kurz in Abbildung 12 wiedergegeben.

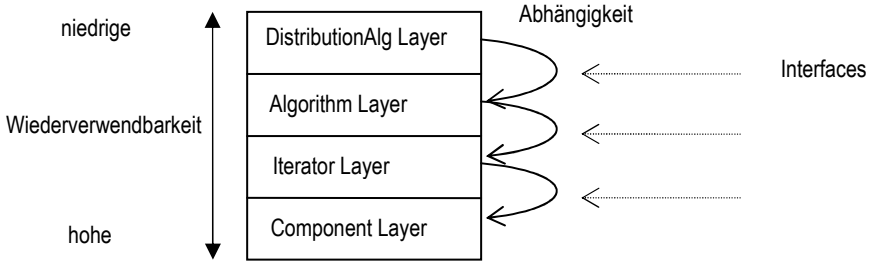


Abbildung 12 – Schichtenmodell (Abhängigkeit)

Alle gemeinsamen Verantwortlichkeiten bzw. Verhalten von allen Objekten einer Schicht (Layer) werden zusammengefasst, und ergeben damit ein Interface für diese Schicht. Die Klassen in allen Schichten werden identifiziert bzw. kategorisiert nach der Vererbungsarchitektur, die sich basierend auf den physikalischen Modellen der realen Komponenten ergibt [Li04]. Im Folgenden soll ein Auszug aus der Arbeit bezüglich des Component Layer und des Algorithm Layer erläutert werden, da der Aufbau und die Funktion dieser Layer für diese Diplomarbeit von Nutzen sein könnten. Alle Funktionen der in Abbildung 12 beschriebenen Layer können nur über die öffentlichen Schnittstellen der betreffenden Layer erreicht werden. Die Schnittstellen umfassen Funktionalitäten betreffend Verbindungen, Berechnung von Lastflüssen und der Simulation bzw. Berechnung von Verlässlichkeiten betreffend der verwendeten Komponenten. Das Component Layer umfasst die folgenden Funktionen, die über die folgenden öffentlichen Schnittstelle ausgeführt werden können:

- IConnectivity (setzt die Konnektivität zu anderen Komponenten)
- ILoadFlow (setzt Parameter die für die Berechnung eines Lastfluss notwendig sind, z. B. Spannung, Strom, Leistung etc.)

- IReliability (setzt Parameter für die Verlässlichkeit einer Komponente, z. B. Ausfallsrate, Reparaturzeit etc.)

Basierend auf dem IConnectivity-Interface weiß somit jede Komponente über seine Mutter- und Kindkomponenten Bescheid. Damit kann eine Topologie, die einem Netz entspricht, aufgebaut werden. Die Komponentenhierarchie kann damit wie in Abbildung 13 dargestellt aufgebaut werden:

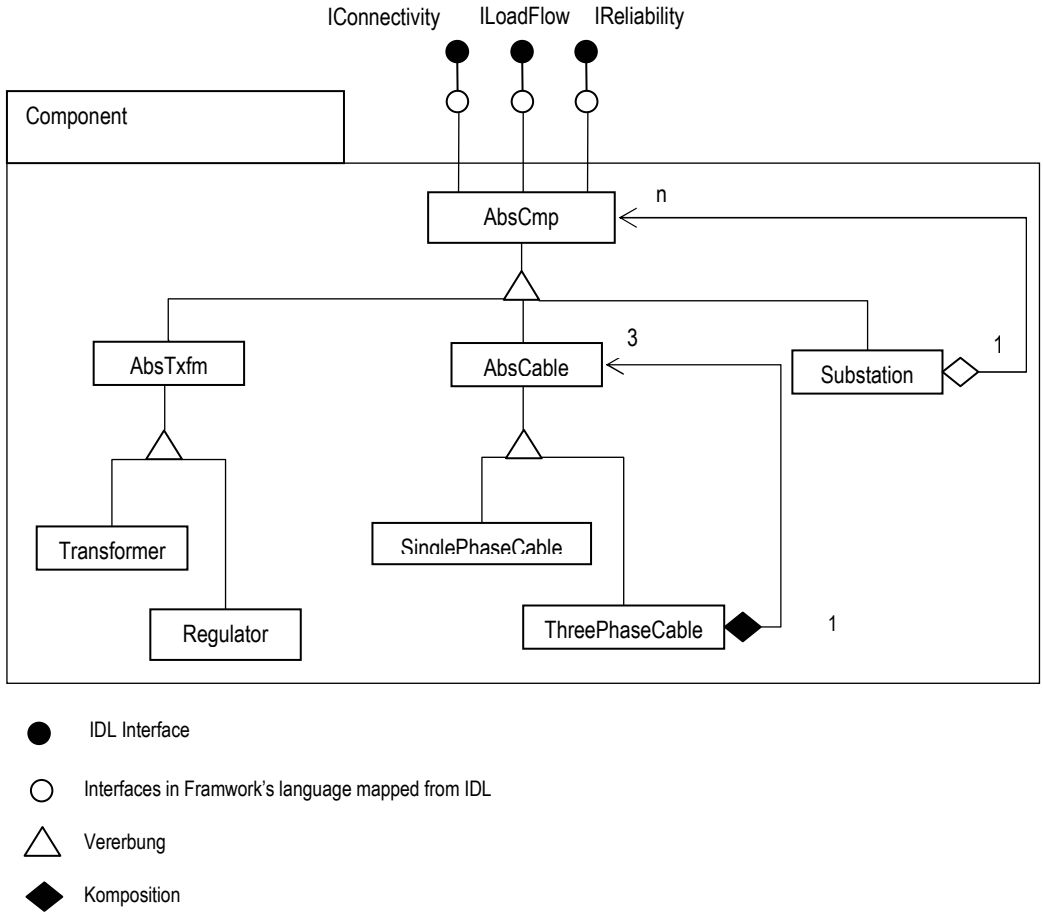


Abbildung 13 – Component (Framework) [Li04]

Wie in Abbildung 13 ersichtlich, sind hier die abstrakten Komponenten AbsCmp, AbsTxfm und AbsCable definiert. Von diesen können dann die konkret zu implementierenden Komponenten wie z. B. Transformator (Transformer), Einphasen- bzw. Dreiphasen-Kabel (SinglePhaseCable, ThreePhaseCable) abgeleitet werden. Eine Substation kann aus vielen Komponenten bestehen bzw. besitzt Referenzen auf viele Komponenten, da von dieser ja alle Teile versorgt werden. Auf die Komponenten wird das Composite-Entwurfsmuster [Fre04] bzw. Iterator-Entwurfsmuster [Fre04], zum einfachen Durchlaufen bzw. zur Implementierung von Teil-Ganzes-Beziehungen angewandt [Li04].

Das Algorithm Layer des Frameworks ist konzeptionell gleich aufgebaut wie der Component Layer. Das Interface trägt hier den Namen IExecAlg. Das Interface unterstützt im Wesentlichen die Aus-

führung von Algorithmen. Als abstrakte Klasse wird hier die Klasse AbsAlgorithm implementiert, von der aus die abstrakten Klassen für AbsLoadFlow (Lastfluss-Algorithmus), AbsShortCircuit (Kurzschlussanalyse) und AbsRelAssess (Verlässlichkeitsabschätzung) ableiten. Die jeweiligen Bereiche sind in die Packages LoadFlow, ShortCkt und RelAssess gegliedert. Von diesen drei abstrakten Klassen können dann die konkret zu implementierenden Algorithmen abgeleitet werden [Li04]. Abbildung 14 zeigt den Aufbau des Algorithm Layers.

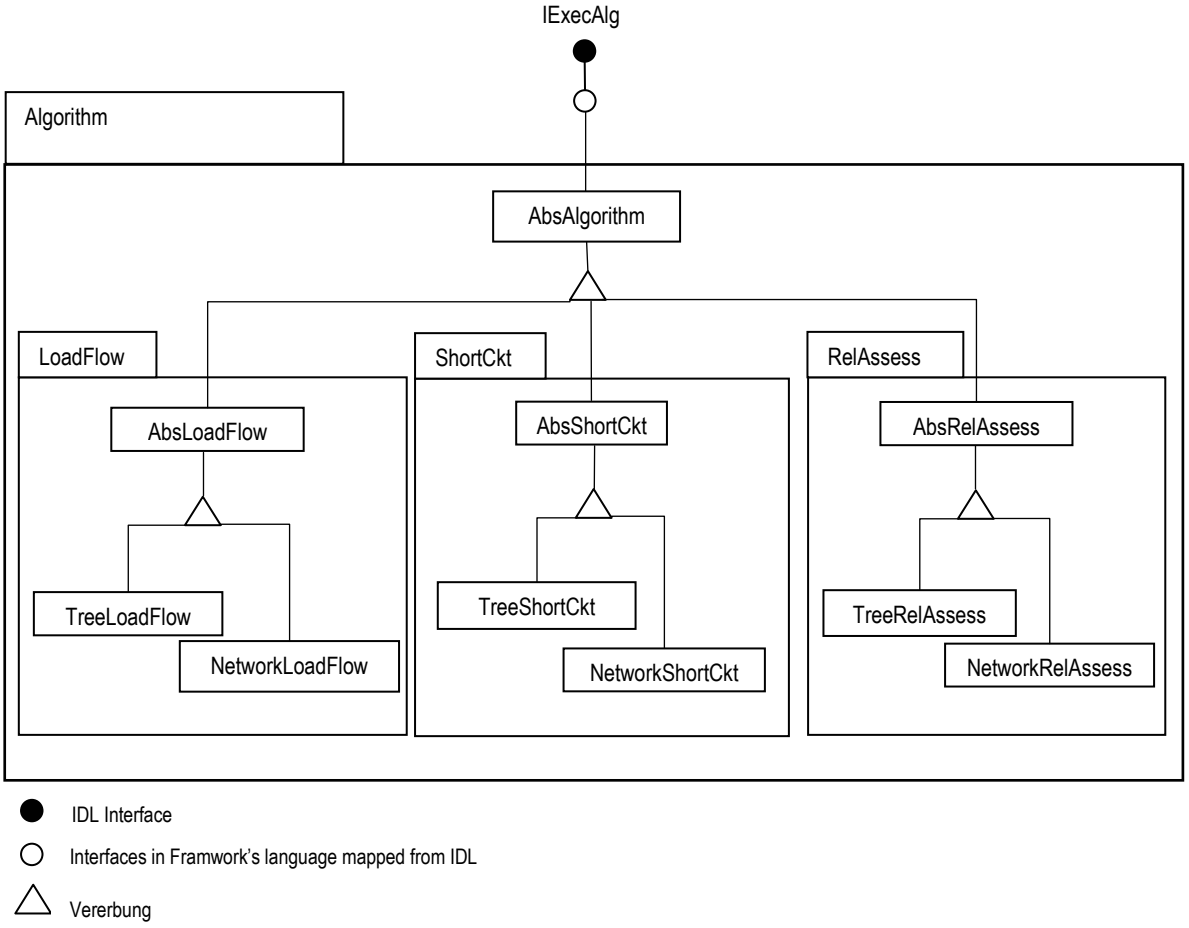


Abbildung 14 – Algorithm (Framework) [Li04]

Aufgrund der Top-Down-Abhängigkeiten der vier Layers kommt es zu reduzierten Kosten und einer erweiterten Wiederverwendbarkeit bezüglich der Komponenten. Die interne Klassenhierarchie jeder Schicht folgt einer strikten hierarchischen Struktur, die einfach zu erweitern ist. Standard-Entwurfsmuster werden verwendet um das Framework zu implementieren. Sofern das Design des Frameworks verstanden wurde, ist es für die Anwender einfach jegliche Anwendung die auf diesem Framework basiert zu verstehen. Die Abhängigkeiten der Schichten basieren auf der Implementierung von wohldefinierten Interfaces. Die interne Struktur der Schichten wird damit versteckt. Die Interfaces werden in der sprachenunabhängigen Beschreibungssprache IDL verfasst, und werden durch viele gängige Sprachen unterstützt [Li04].

Die Abschnitte 1.5.1 und 1.5.2 zeigen einerseits die Möglichkeiten zur Analyse von elektrischen Verteilnetzen und andererseits die Herangehensweisen bezüglich der Abbildung in Software. Es zeigt sich, dass es bereits einige Ansätze zur Analyse von elektrischen Verteilnetzen entwickelt wurden. Im Besonderen sind hier die Vorwärts-/Rückwärtsschritt-Verfahren zu erwähnen, von denen es wie in den letzten Abschnitten gezeigt, verschiedene Varianten der Implementierung gibt aber im Kern das gleiche Prinzip verfolgen. Eines dieser Verfahren ist die Ladder Iterative Technique von Kersting [Ker07]. Bezüglich der Implementierung der Verfahren stellt sich eindeutig eine Orientierung an objektorientierten Konzepten heraus. Diese Herangehensweise erlaubt es auch im Bereich von elektrischen Verteilnetzen, die wesentlichen Komponenten möglichst real nachzubilden und dabei die Modularität bzw. Erweiterbarkeit der Lösung im Auge zu behalten. In Abschnitt 4.3 dieser Arbeit wird auf diese Techniken verwiesen bzw. deren Anwendbarkeit in dieser Arbeit näher untersucht.

In den nun folgenden Kapiteln wird auf den Hauptteil der Arbeit eingegangen, wobei hier zunächst die technischen Grundlagen die für diese Arbeit notwendig sind erläutert werden. Nachfolgend wird die Implementierung im Open-Source-Framework OMNeT++ [3] beleuchtet um im Anschluss daran die Ergebnisse zu diskutieren. Der darauf folgende Schlussteil rundet die Arbeit ab.

2. Technische Grundlagen

In diesem Teil der Arbeit werden die Technischen Grundlagen der für die Arbeit notwendigen Werkzeuge sowie elektrotechnischen Grundlagen erörtert. Danach werden die verschiedene Lösungsansätze miteinander verglichen und deren Probleme erörtert. Folgend wird die Implementierung und die Ergebnisse der Arbeit präsentiert.

Wie schon in der Einführung erörtert, dient ein elektrisches Netz der Übertragung und Verteilung von elektrischer Energie. Grundsätzlich kann in der Elektrizitätslehre zwischen Gleich- und Wechselstrom unterschieden werden. Der Beweggrund der Verwendung von Wechselstrom in den heutzutage üblichen Energienetzen liegt in der höheren Übertragungsreichweite und in der Fähigkeit von Wechselstromgeneratoren, höhere Spannungen zu generieren. Um die Stromwärmeverlustleistung in den Energieübertragungsleitungen gering zu halten, muss mit kleinen Leitungsströmen bei hohen Übertragungsspannungen gearbeitet werden. Als Daumenregel gilt dabei 1 kV/1 km. Für Wechselstrom gibt es bis heute keine mit vernünftigen Wirkungsgrad arbeitende Motoren. Deshalb werden heute aufgrund des besseren Wirkungsgrades für Motoren Dreiphasenwechselstrom- bzw. Drehstrom-Asynchronmotoren eingesetzt. Grundsätzlich kann der Asynchronmotor auch als Generator zur Erzeugung von Strom eingesetzt werden. Als Nachteil ist hier anzumerken, dass die Maschine allerdings Blindleistung benötigt. Der Asynchronmotor ist allerdings nicht in der Lage, Blindleistung für alle Komponenten im Netz, die Blindleistungen benötigen, zu liefern. Deshalb werden heutzutage zur Generierung von Strom in den Kraftwerken Synchronmaschinen eingesetzt, die eine Leistung von bis zu 1500 MVA [Anm. in Deutschland] und für Spannungen bis zu 30 kV gebaut werden können. Ein problemlose Übertragung der Energie ist mit dieser Technik bis etwa 1000 km Leitungslänge möglich [Spr03].

2.1 Elektrische Parameter der elektrischen Energieübertragung

Für die Energieübertragung spielen die Verhältnisse von Strom, Spannung und Leistung eine entscheidende Rolle. Grundsätzlich ist man daran interessiert, eine möglichst große Leistung bei geringen Strömen zu übertragen. Es gilt, dass die Übertragungskapazität einer Leitung mit dem Quadrat der Übertragungsspannung wächst und dem induktiven Widerstand (Reaktanz) der Leitung umgekehrt proportional ist [Spr03]. Das bedeutet, dass sich die maximal übertragbare Leistung zu $P_{max} = U^2 / X$ ergibt [Spr03]. In Abbildung 15 sei der Zusammenhang von der Spannung U, dem Strom I und der Leistung P für die Zeit t im einphasigen Fall für einen rein ohmschen Widerstands R darge-

stellt. Der Mittelwert für die Leistung *P* ergibt sich hierbei aus dem Produkt der Effektivwerte der Spannung *U* und dem Strom *I* zu $P = U\,I$, in zeitabhängiger Form zu $p(t) = u(t)\,i(t)$, welcher dann zwischen 0 und dem Maximalwert schwankt . Wie in Abbildung 15 ersichtlich, ist hier der Strom *I* und die Spannung *U* in Phase. In der Graphik wird das Verbraucherpfeilsystem angenommen.

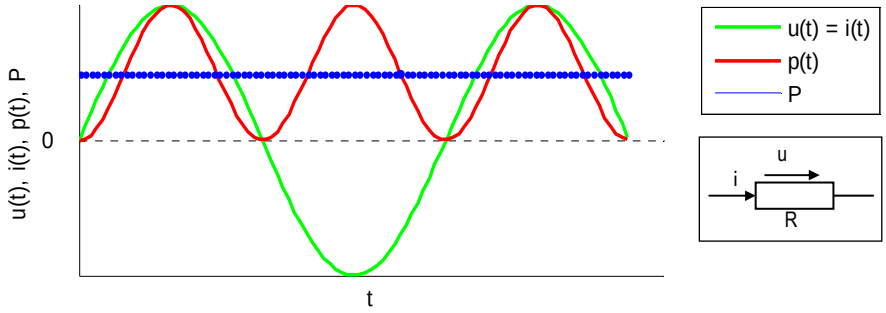


Abbildung 15 – Leistung (ohmsch)

Für den Fall einer induktiven Last *L* wie sie z. B. ein Motor darstellt, kommt es zu einem Hin- und Herpendeln der Leistung. Im Fall einer rein induktiven Last eilt die Spannung *U* dem Strom *I* um 90° voraus. Abbildung 16 zeigt diesen Sachverhalt, es wird das Verbraucherpfeilsystem angenommen. Weiters ist hier ersichtlich, dass die mittlere Leistung $P = 0$ ergibt, die Induktivität also im Zeitmittel keine Leistung aufnimmt bzw. abgibt.

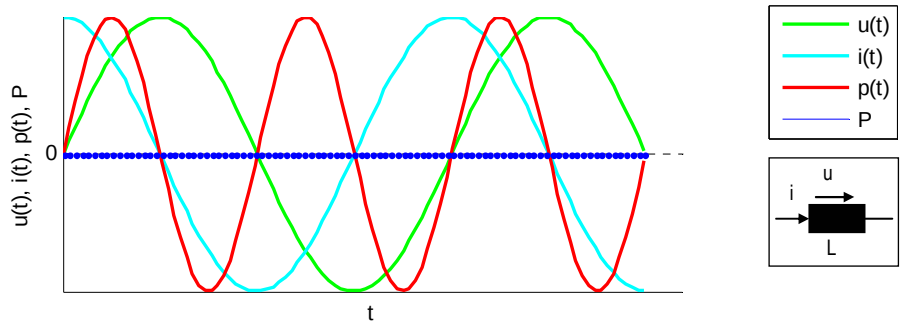


Abbildung 16 – Leistung (induktiv)

Für kapazitive Lasten gilt ebenfalls eine Phasenverschiebung zwischen der Spannung *U* und dem Strom *I* um 90°. Allerdings eilt hier der Strom *I* der Spannung *U* voraus. In unseren (Anm. Deutschland und Österreich) Netzen dominieren vor allem Lasten ohmschen-induktiven Charakters, bei denen der Strom der Spannung um einen bestimmten Wert nacheilt [Spr03]. Danach können für die Momentanwerte von Spannung, Strom und Leistung folgende Gleichungen wie in (16) bis (25) dargestellt, aufgestellt werden. Die Gleichungen (16) und (17) beschreiben dabei den Momentanwert von Strom und Spannung, die Phasenverschiebung des Stroms um -90° wird durch φ beschrieben. Zur Berechnung des Momentanwertes *p* der Leistung mittels Gleichung (19) wird Glei-

chung (16) - Gleichung (18) herangezogen, wobei durch Substitution von Gleichung (20) und Gleichung (21) in Gleichung (19) die Gleichung (22) zur Berechnung des Momentanwertes p der Leistung herangezogen werden kann. Diese Gleichung kann auch anders angeschrieben werden und man gelangt damit zur Darstellung der Momentanleistung *p* aus zwei zeitabhängigen Anteilen, nämlich der Wirkleistung *P* und der Blindleistung *Q* wie in Gleichung (25) dargestellt.

$$u = U_{max} \sin \acute{\omega} t \quad \textbf{(16)}$$

$$i = I_{max} \sin (\acute{\omega} t - \varphi) \quad \textbf{(17)}$$

$$p = u\, i = U_{max}\, I_{max} \sin \acute{\omega} t \sin (\acute{\omega} t - \varphi) \quad \textbf{(18)}$$

$$p = U_{eff}\, I_{eff} \cos \varphi - U_{eff}\, I_{eff} \cos (2\acute{\omega} t - \varphi) \quad \textbf{(19)}$$

$$P = U_{eff}\, I_{eff} \cos \varphi \quad \textbf{(20)}$$

$$S = U_{eff}\, I_{eff} \quad \textbf{(21)}$$

$$p = P - S \cos (2\acute{\omega} t - \varphi) \quad \textbf{(22)}$$

$$p = Ueff\, Ieff \cos \varphi\, (1 - \cos 2\acute{\omega} t) - U\, I \sin \varphi \sin 2\acute{\omega} t) \quad \textbf{(23)}$$

$$Q = U_{eff}\, I_{eff} \sin \varphi \quad \textbf{(24)}$$

$$p = P(1 - \cos 2\acute{\omega} t) - Q \sin 2\, 2\acute{\omega} t \quad \textbf{(25)}$$

Damit ergibt sich für den allgemeinen Lastfall eine Darstellung wie in Abbildung 17 gezeigt. Die Wirkleistung *P* ist konstant und zeitunabhängig. Die Blindleistung *Q* pendelt zwischen seinem Minimal- und Maximalwert hin und her und kann nicht genutzt werden. Bei Leistungsangaben sind die Arten der Leistung an der Einheit zu erkennen. Die Einheit der Wirkleistung *P* ist Watt W, die Einheit der Blindleistung *Q* Volt-Ampere-reaktiv Var, die Einheit der Scheinleistung $S = U\, I$ ist Voltampere VA.

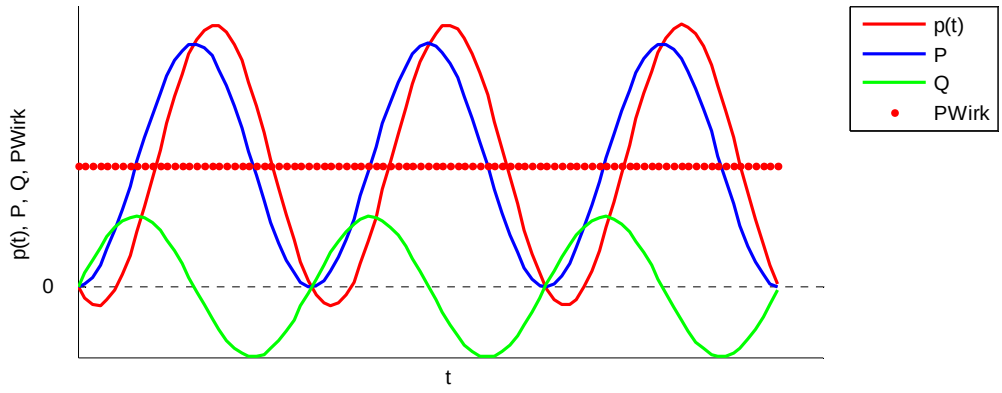


Abbildung 17 – Darstellung des zeitlichen Verlaufs der Leistungsgrößen im allg. Lastfall

Bisher wurde nur auf den einphasigen Fall der elektrischen Parameter eingegangen. Wie schon erwähnt, existieren keine effizient arbeitenden Motoren für Einphasenwechselstrom. Deshalb ist es üblich, elektrische Energie in Form von Dreiphasenwechselstrom oder Drehstrom zu übertragen. In diesem Fall werden die Generatoren und die Verbraucher bzw. zu speisenden Netze im Allgemeinen

durch drei Leiter verbunden. Die Leiter seien mit L_1, L_2 und L_3 gekennzeichnet und bezeichnen die sogenannten Hinleiter der drei Stromkreise bzw. Phasen, deren Spannungen U_1, U_2 und U_3 und Ströme I_1, I_2 und I_3 betragsmäßig gleich groß, aber jeweils um 120° gegeneinander in der Phase verschoben sind. Der Rückleiter ist hinfällig, da die Summe der Leiterströme null ergibt (dies kann im Zeigerdiagramm leicht nachvollzogen werden) [Spr03]. Die folgende Abbildung 18 zeigt die drei Phasen zwischen dem Generator und einer Last. Weiters ist in der Abbildung der Neutralleiter *N* eingezeichnet, der als gemeinsamer Rückleiter der drei Phasen fungiert.

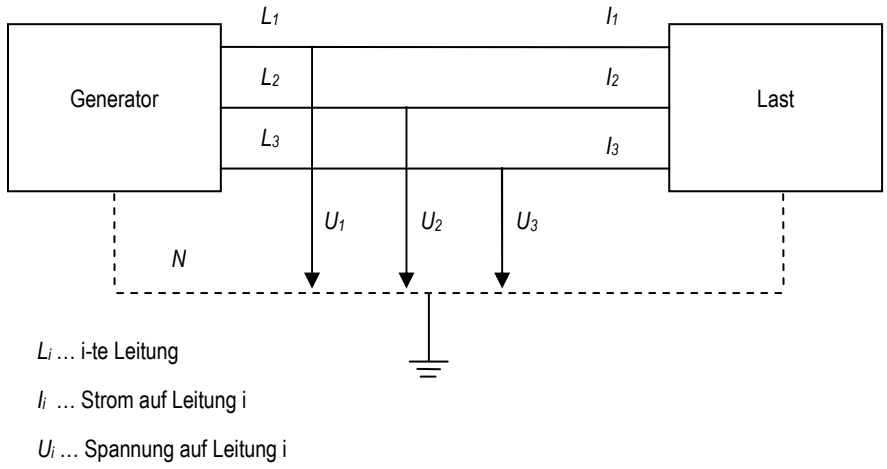


Abbildung 18 – Leistungsübertragung mit Drehstrom [Spr03]

In einem Dreiphasen-Netz gibt es natürlich auch einphasige Verbraucher. Bei gleichmäßiger Verteilung der einphasigen Lasten bzw. Verbraucher auf die drei Phasen eines Netzes spricht man von einem symmetrischen Lastfall, d. h. auch in diesem Fall ist die Summe der Ströme null. Im Fall von Hochspannungs- und Mittelspannungsleitungen ist dies in der Regel der Fall. Im Fall von Verteilnetzen im Niedrigspannungsbereich (z. B. 400 V), kann es bei einer kleinen Anzahl von Verbrauchern zu einem unsymmetrischen Lastfall kommen. Das bedeutet, dass die Summe der Ströme nicht mehr null ergibt. Deshalb ist dort auch ein gemeinsamer Rückleiter für die drei Phasen vorgesehen, der mit der Erde verbunden ist, das Potential null besitzt, die Bezeichnung *N* trägt und nur bei im unsymmetrischen Lastfall unter Strom steht. Bei Anschluss von Einphasenwechselstromverbrauchern können diese entweder zwischen einem der so genannten Außenleiter L_1, L_2 oder L_3 und dem Neutralleiter *N* oder zwischen je zwei Außenleiter angeschlossen werden. Die Spannung zwischen einem Außenleiter und dem Neutralleiter wird als Leiter-Erd-Spannung bezeichnet, die Spannung zwischen zwei Außenleitern als Leiter-Leiter-Spannung. In Niederspannungs-Verteilnetzen ist die übliche Leiter-Erdspannung durch $U = 400\ V / \sqrt{3}$ (230 V) gegeben, wobei $\sqrt{3}$ der die Verknüpfung zwischen Leiter-Erd- und Leiter-Leiter-Spannung beschreibende Verkettungsfaktor bei Dreiphasenwechsel- oder Drehstrom ist. Für den Augenblickswert der Leistung ergibt sich der Ausdruck $p(t) = 3\ U\ I \cos \varphi$ [Spr03], damit ist der Augenblickswert im Dreiphasenbetrieb zeitunabhängig. Die Leistung entspricht damit $P = 3\ P_{Wirk}$ (Gleichung (20)) also dem Dreifachen der in einem Stromkreis bzw. Leiter übertragenen Wirkleistung. Damit vereinfachen sich die Berechnungen, da man sich unter der Voraussetzung eines symmetrischen Lastfalls, auf die Berechnung einer Phase beschrän-

ken kann. Für die Blindleistung $Q = 3\ Q_{Blind}$ (Gleichung (24)) und die Scheinleistung $S = 3\ U\ I$ gilt derselbe Zusammenhang [Spr03].

2.2 Komplexe Darstellung der elektrischen Leistung

In der Elektrotechnik ist es üblich, technische Zusammenhänge des Wechselstroms in komplexer Darstellung anzugeben. Ein komplexe Zahl $z = x + jy$ besteht demnach aus einem Realteil x und dem Imaginärteil y, welche auf der x- und y-Achse des kartesischen Koordinatensystems aufgetragen werden. Komplexe Zahlen können in kartesischer, exponentieller Form bzw. Polarform dargestellt werden.

Auch für Leistungsparameter kann die in der Elektrotechnik übliche komplexe Darstellung angewandt werden. Gegeben sei ein komplexer Widerstand $\underline{Z}$, auch als Impedanz bezeichnet, der an einer Wechselspannung mit dem Effektivwert U_{eff} angeschlossen ist. Diese Spannung kann in Zeigerform als

$\underline{U} = U_{eff} e^{j\varphi_u}$ [Spr03]

und der Strom als

$\underline{I} = I_{eff} e^{j\varphi_i}$ [Spr03]

dargestellt werden. Als Grundlage wird hier die Eulersche Identität herangezogen die besagt, dass

$e^{j\varphi} = \cos(\varphi) + j\sin(\varphi) = 1\angle\varphi$ [Ham97]

und

$e^{-j\varphi} = \cos(\varphi) - j\sin(\varphi) = 1\angle -\varphi$ [Ham97]

gilt. Weiters kann daraus für eine komplexe Zahl der Form

$A\angle\varphi = Ae^{j\varphi}$ [Ham97]

geschrieben werden. Dies wird auch als exponentielle Darstellung einer komplexen Zahl bezeichnet. Für die beiden gegebenen Werte $\underline{U}$ und $\underline{I}$ kann nun wie für gegebene Gleichströme bekannt, die Leistung $P = U\ I$ als $\underline{S} = \underline{U}\ \underline{I}$ ausgedrückt werden. Für die Leistung S ergibt sich demnach

$\underline{S} = \underline{U}\ \underline{I} = U_{eff} e^{j\varphi_u}\ I_{eff} e^{j\varphi_i} = U_{eff} I_{eff} e^{j(\varphi_u + \varphi_i)}$.

Das heißt, die Leistung würde sich in diesem Fall aus dem Produkt der Effektivwerte und der Summe der Phasenwinkel ergeben, was physikalisch nicht sinnvoll zu deuten ist, da für die Leistungsberechnung die Differenz der Phasenwinkel heranzuziehen ist (siehe 2.1). Deshalb wird für die Leistungsberechnung die konjungiert komplexe Darstellung des Stromes

$\underline{I}^* = I_{eff} e^{-j\varphi_i}$ [Spr03]

herangezogen. Damit ergibt sich für die Scheinleistung

$\underline{S} = U_{eff} I_{eff} e^{j(\varphi_u - \varphi_i)}$ [Spr03]

oder auch

$\underline{S} = U_{eff} I_{eff} e^{j\varphi}$ für $\varphi = \varphi_u - \varphi_i$ [Spr03].

Diese Darstellung kann aufgrund der Eulerschen Identität auf die Form

$\underline{S} = U_{eff} I_{eff} (\cos \varphi + j \sin \varphi) = U_{eff} I_{eff} \cos \varphi + j\, U_{eff} I_{eff} \sin \varphi$ [Spr03]

umgeschrieben werden. Aus Abschnitt 2.1 ist bereits bekannt, dass

$P = U_{eff} I_{eff} \cos \varphi$ und $Q = U_{eff} I_{eff} \sin \varphi$

gilt. Daraus folgt, dass $\underline{S} = P + jQ$ [Spr03] ist.

2.3 Aufbau von elektrischen Netzen

Ein elektrisches Netz dient der Energieübertragung und Verteilung von elektrischer Energie. Netze können durch die Höhe der anliegenden Spannungsniveaus unterschieden werden. Abbildung 19 zeigt den konzeptionellen Aufbau eines Energienetzes. Hochspannungsnetze dienen primär der Übertragung von Energie über weite Distanzen, wobei Verteilnetze eher der Versorgung der Endverbraucher dienen und die dort benötigten niedrigeren Spannungsebenen zur Verfügung stellen. Die

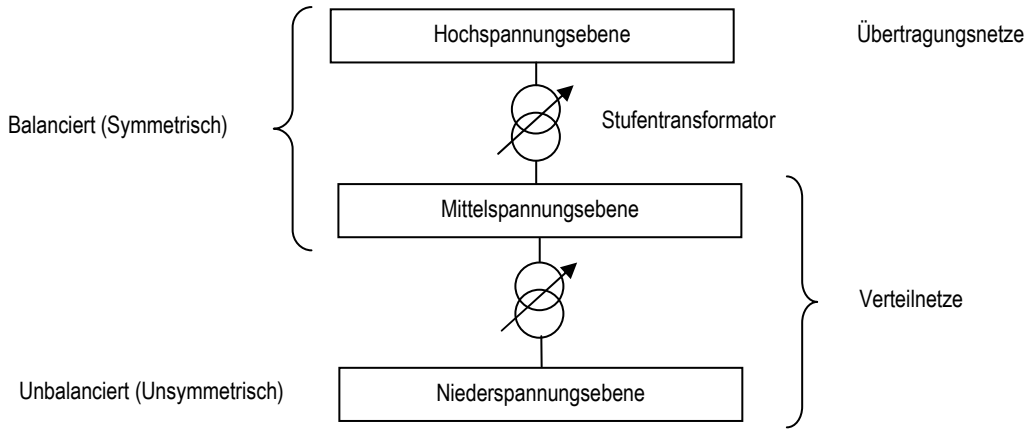


Abbildung 19 – Elektrisches Netz (konzeptionell)

Netze sind durch sogenannte Stufentransformatoren getrennt, die je nach Belastung der miteinander verbundenen Netze die Spannungsstufen regeln können und dadurch Spannungsbandverletzungen vermeiden können. Als Spannungsband wird jenes Intervall bezeichnet, in dem sich die Spannungswerte des Netzes bewegen dürfen. Es liegt also eine bestimmte Quellen- bzw. Referenzspannung an, und aufgrund von Lasten bzw. Verbrauchern und Einspeisungen auf den Leitungen, kommt es zu lokalen Spannungsabfällen bzw. Spannungsanhebungen. Das Spannungsband beschreibt die zulässigen Unter- bzw. Überschreitungen bezogen auf die Referenz. Wie schon in Abschnitt 2.1 erwähnt,

wird die elektrische Energie mittels Dreiphasenwechselstrom übertragen. Die eingespeiste und abgenommene Energie hält sich im Normalbetrieb zu jedem Zeitpunkt die Waage. Nachdem aber vor allem im Niederspannungsbereich viele einphasige Verbraucher angeschlossen sind und diese einen sehr variablen Lastgang aufweisen, kommt es sehr häufig zu unterschiedlichen Auslastung der drei Phasen und damit zu einer Verletzung der Balance bzw. Unsymmetrie.

Die Energie wird über Übertragungsleitungen die die Komponenten des Netzes miteinander verbinden übertragen. Eine Leitung kann wie in Abbildung 20 dargestellt, modelliert bzw. aufgefasst werden. Die Leitung wird dabei als Widerstand bzw. komplexe Impedanz $\underline{Z}$ mit Wirkanteil R und Blindanteil (Reaktanz) X modelliert, welche die Knoten N_i und N_j miteinander verbindet. Der Blind-

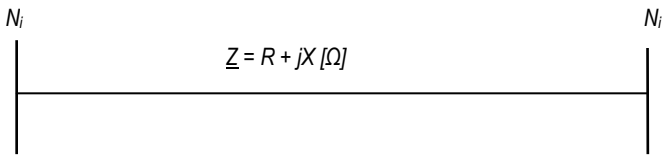


Abbildung 20 – Leitungsmodell

anteil X kommt durch den Abstand der Leiter in der Übertragungsleitung zustande, siehe auch [Spr03]. In dieser Arbeit liegt der Schwerpunkt auf der Analyse von Verteilnetzen, bei denen der Wirkwiderstand R im Gegensatz zu den Hochspannungsnetzen respektive Übertragungsnetzen nicht verschwindend klein ist, sondern eine wichtige Rolle spielt. Die folgenden Eigenschaften sind typisch für Verteilnetze [Sto00]:

- radiale Struktur (nur Einfachverbindungen, keine Maschen)
- große Anzahl an Serienwiderständen R und Reaktanzen X
- hohes R/X-Verhältnis

2.4 Spannungsregelung in Verteilnetzen

Aufgrund des zunehmenden Energiebedarfs und der fortschreitenden Liberalisierung der Energiemärkte kommt es vermehrt zu Einspeisungen von elektrischer Energie nicht nur in den Übertragungsnetzen, sondern auch zur Einbringung von Energie in die Verteilnetze der Mittel- und Niederspannungsebene. Dies bedeutet für die Elektrizitätsnetzbetreiber eine zunehmende Herausforderung hinsichtlich des Managements und der Planung von neuen alternativen Einspeisepunkten [Lug08].

Im Zusammenhang mit dieser dezentralen Stromerzeugung entstehen die Probleme vor allem in den untersten Spannungsebenen, die eine eklatante Ungleichzeitigkeit zwischen Erzeugung und Verbrauch von elektrischer Energie aufweisen. Das Kernproblem besteht in der Spannungshaltung im gesamten Netz bei sich umkehrenden Lastflüssen – also von den Enden des Netzes hin zu den ursprünglich zentralen Einspeisepunkten [Lug08].

Vor allem in ländlichen Gebieten besteht das Problem der Spannungshaltung wenn z. B. Kleinkraftwerke (Wasser-, Solartechnik) Energie in das Netz einspeisen. Die bisher verbreitete Strategie ist die Leitungsverstärkung, um Spannungsbandverletzungen (siehe Abschnitt 2.3) bzw. Überlastungen der Leitungen zu vermeiden. In Zukunft ist es notwendig, mittels intelligenten Steuerungs- und Regelungsmechanismen erweiterte Reserven der Leitungen zu nutzen. Diese Vorgehensweise wird im Allgemeinen als „aktiver Verteilernetzbetrieb“ bezeichnet [Lug08].

Die Entwicklung in Europa betreffend der Energieversorgung verläuft im Moment verstärkt in Richtung Förderung von alternativen erneuerbaren Energieträgern. Weiters fordern die Verbraucher eine hohe Flexibilität und Qualität der Versorgung bei steigendem Leistungsbedarf. Weiters steigt der Kostendruck auf die Verteilnetzbetreiber. Daraus folgt, dass die Entwicklung nur möglich ist, wenn die dezentrale Leistungseinspeisung auch durch die Netzbetreiber ermöglicht wird. Aufgrund der zunehmenden Streichung der anerkannten Netzkosten bei den Netzbetreibern und der zunehmenden dezentralen Leistungseinspeisung, ist die Leitungsverstärkung die ungünstigste Lösung. Deshalb sind aktive Verteilernetze zu bevorzugen [Lug08].

Die Stromerzeugung aus regenerativen Energieträgern erfolgt meist direkt dort, wo das Angebot gegeben ist (z. B. Kleinwasserkraft). Die Einspeisepunkte sind über das gesamte Verteilnetz hinweg verteilt und liegen oftmals auch an den abgelegenen Gebieten an den Netzrändern, wo die Netze weniger leistungsstark sind. Aufgrund der meist mittleren Anlagengrößen speisen die Anbieter im Bereich der Mittel- und Niederspannungsnetze ihre Energie ein, wobei sich das Problem ergibt, dass die Verteilnetze für einen Zuwachs an dezentraler Einspeiseleistung nicht konzipiert und gebaut sind. Deshalb kann es bei unkoordinierter bzw. nicht geplanter Einspeisung – unter der Voraussetzung, dass die Leitungen nicht verstärkt werden – zu einer Anhebung der Spannungswerte und damit zu einer Verletzung des Spannungsbandes kommen. Die Verstärkung der betroffenen Netzabschnitte und die damit verbundenen Kosten führen oftmals dazu, dass potentielle dezentrale Einspeisemöglichkeiten wirtschaftlich uninteressant und damit nicht durchgeführt werden. Sofern keine Gleichzeitigkeit zwischen Verbrauch und Erzeugung besteht, verbrauchen zusätzliche Einspeiser einen Teil des Spannungsbandes, die eigentlich zur Versorgung weiterer Verbraucher zur Verfügung stünden [Lug08].

Die Übertragungskapazität einer Leitung ist durch den Bemessungsstrom begrenzt. In Verteilnetzen stellen die Spannungsbandgrenzen eine zusätzliche Einschränkung dar, so dass die Leitungen nur mit einem Teil der Volllast betrieben werden können. Mittels Steuerungs- und Regelungstechnik, die während des Betriebs des Netzes die Spannungen aktiv beeinflussen, ist es möglich zusätzliche Reserven der Leitungen zu nutzen [Lug08].

In der folgenden Abbildung 21 wird der Zusammenhang dargestellt. In der Grafik (A) ist die Maximal- und die Minimalspannung im Netz abgebildet und deren Verlauf über einen bestimmten Zeitraum. Es ist zu erkennen, dass der Verlauf der Spannung innerhalb des Spannungsbandes, welches durch die unteren und oberen Spannungsgrenzen U_{min} und U_{max} begrenzt wird, liegt. In Grafik (B) ist das gleiche Netz nochmals dargestellt, allerdings unter der Bedingung, dass sowohl die Lasten als auch die dezentralen Einspeisungen zugenommen haben. Weiters ist zu erkennen, dass in diesem Fall das Spannungsband sowohl im Fall des minimalen als auch des maximalen Spannungsverlauf unter- bzw. überschritten wird. Die daraus folgenden Lösungsmöglichkeiten bestehen einerseits in

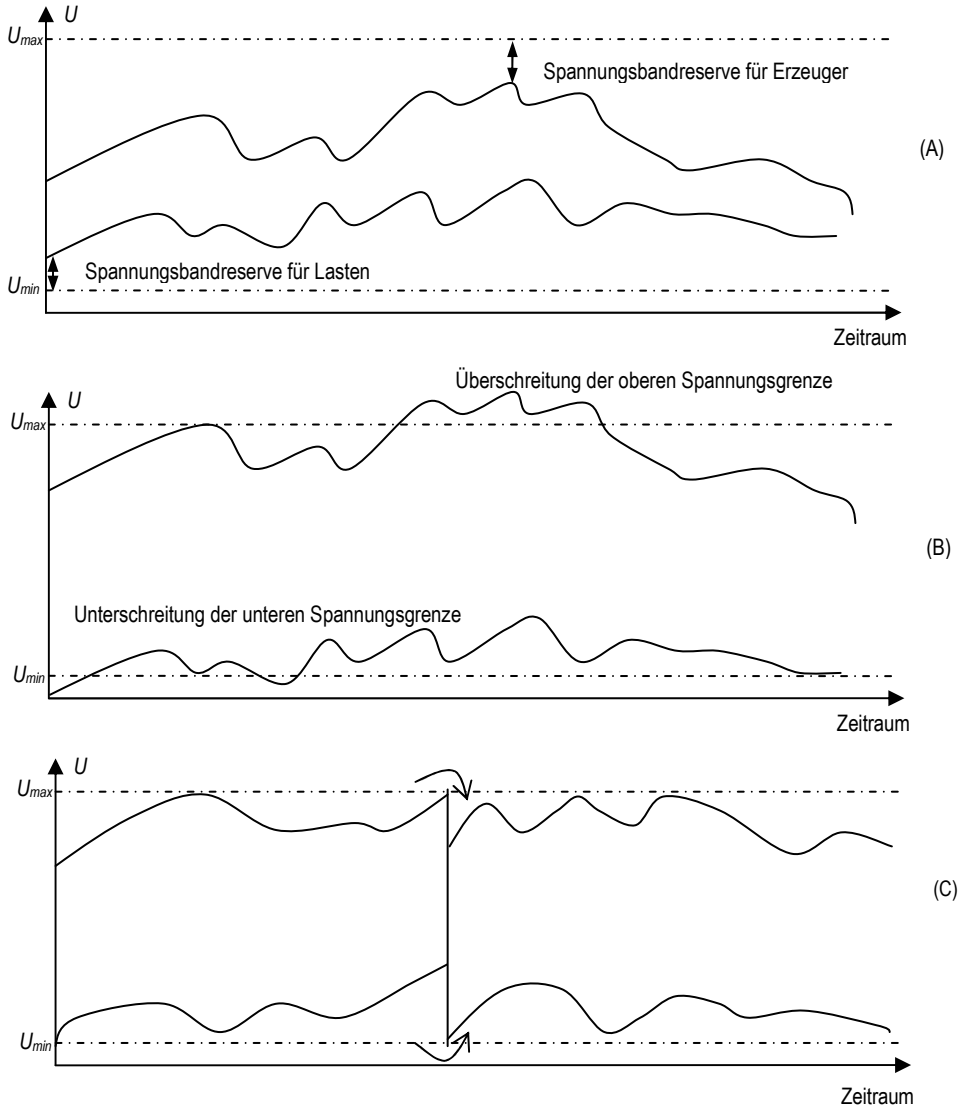


Abbildung 21 – Spannungsverlauf im aktiven Verteilernetz [Lug08]

der schon erwähnten Leitungsverstärkung oder im Einsatz eines aktiven Verteilernetzes. In der Grafik (C) wird der Spannungsverlauf unter Berücksichtigung eines aktiven Verteilsystems dargestellt. Man erkennt die Verschiebung des Bandbedarfs, wenn in die Spannungsverläufe eingegriffen wird [Lug08].

Der Nutzen von aktiven Verteilernetzen lässt sich damit durch die kostengünstige Möglichkeit der verstärkten dezentralen Einspeisung und einer Steigerung der Entnahmeleistung für die Verbraucher umschreiben. Dabei soll die Leitungsverstärkung die jeweils letzte Möglichkeit darstellen [Lug08].

Bei der Beurteilung von neuen potentiellen Einspeisungen ermittelt die Netzplanung durch Simulation des Netzes, ob Verletzungen des Spannungsbandes zu erwarten sind. Trotz des hohen Aufwands in die Netzmodelle weisen diese Berechnungen allerdings immer noch Unschärfen auf. Reale Messreihen an ausgewählten Messpunkten des Netzes stellen gegenüber den Berechungsergebnissen der

Simulation reale Fakten dar. Die Notwendigkeit des Netzausbaus bzw. der Leitungsverstärkung können dadurch noch besser abgeschätzt werden. Bei jedem Netzausbau sollten Überkapazitäten eingeplant werden, um zukünftige Netzbenutzer einfacher einschließen zu können. Im Fall von aktiven Verteilernetzen können diese Ausbauten in kleineren Schritten erfolgen. Als aktives Verteilnetz wird ein Netz bezeichnet, welches es ermöglicht die Spannung U an den Transformatoren (OLTC) bzw. die Wirkleistung *P* und Blindleistung *Q* an den Einspeisepunkten in Echtzeit zu regeln [Lug08].

2.5 Lastflussberechnung im elektrischen Netz

Zur Berechnung der kritischen Größen U und I im Netz werden verschiedene Methoden der Lastflussberechnung eingesetzt. Die Berechnungsmethoden für Lastflüsse können in drei Gruppen eingeteilt werden [Sto00]:

1. Leistungssummationsmethode
2. Stromsummationsmethode
3. Impedanzsummationsmethode

Jede der Methoden hat ihre Vor- und Nachteile. Die Anwendbarkeit hängt von den Bedingungen des Netzes und des Typs der Lastparameter also der Lasttypen ab. Die Lasttypen eines Netzes können als

- konstante Leistung
- konstanter Strom
- konstante Impedanz

modelliert werden [Sto00]. Das Problem ist, dass kein reales Netz nur aus einem dieser Modelltypen besteht. In einem Netzwerk, ja sogar Knoten, sind immer mehrere verschiedene Typen möglich. Weiters sind die Lasten zeitlich veränderbar und damit in einem Großteil der Fälle unvorhersehbar. Damit ist auch eine exakte Netzwerkberechnung praktisch unmöglich. Weiters kommt noch die Variabilität der Spannung ausgehend von den Stufentransformatoren in den Schaltwerken hinzu. In der Berechnung von Verteilnetzen der Mittel- und Niederspannungsebene werden die Lasten zumeist als konstanten Leistungen modelliert [Sto00]. Knoten, die eine bestimmte Leistung generieren, also in das Netz einspeisen, werden als negative Leistungen angegeben [Ker06]. Im Folgenden sollen die technischen Grundlagen der bereits erwähnten Ladder Iterative Technique von [Ker07] erörtert werden. Diese ist mit der Methode von [Shi88] vergleichbar und zählt damit zu den Stromsummationsmethoden [Sto00].

Die Abänderung der Ladder Network Theory für lineare Systeme führt zur erwähnten Ladder Iterative Technique für Lastflussanalyse [Ker76]. Ein elektrisches Verteilnetz ist nichtlinear, da die meisten Lasten als konstante Leistungen in Watt W und Voltampere reaktiv Var angegeben werden. Der Ansatz für lineare Systeme kann dennoch so verändert werden, dass die nichtlinearen Eigenschaften

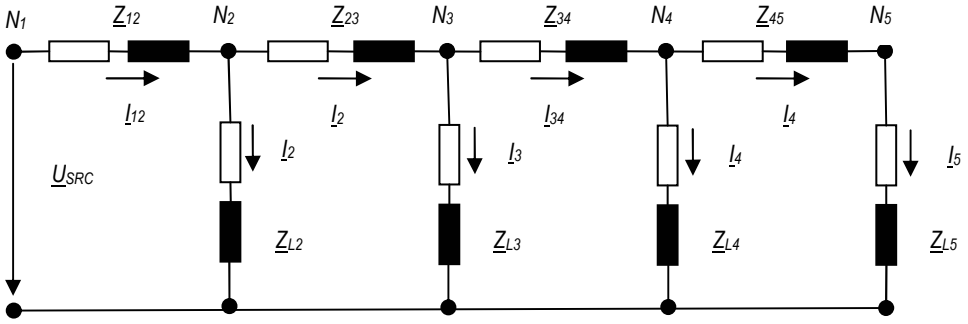


Abbildung 22 – Linear Ladder Network [Ker07]

von Verteilnetzen berücksichtigt werden können [Ker07]. Abbildung 22 zeigt ein einfaches lineares Netzwerk in Form einer Kettenschaltung (Ladder Network). In diesem Netzwerk wird angenommen, dass alle Leitungs- und Lastimpedanzen sowie die Spannung an der Quelle $\underline{U}_{SRC}$ bekannt sind. Um eine Lösung für das Netz zu erhalten, wird mit einem Vorwärtsschritt begonnen bei dem die Spannung am Endknoten N_5, bei dem angenommen wird, dass im Netzwerk keine Lasten vorhanden sind, berechnet werden soll. Nachdem keine Lasten vorhanden sind, ergeben sich damit auch keine Lastströme und damit gleicht die Spannung am Endknoten $\underline{U}_5 = \underline{U}_{SRC}$. Mithilfe des Ohmschen Gesetzes kann daraus $\underline{U} = \underline{Z}\,\underline{I}$ und $\underline{I}_5 = \underline{U}_5 / \underline{Z}_{L5}$ berechnet werden. Für den Endknoten N_5 entspricht der Strom $\underline{I}_{45}$ dem neu berechneten Laststrom $\underline{I}_5$. Für den nun folgenden Rückwärtsschritt kann durch Anwendung des Kirchhoffschen Gesetzes die Spannung an Knoten N_4 durch $\underline{U}_4 = \underline{U}_5 + \underline{Z}_{45}\,\underline{I}_{45}$ berechnet werden. Daraus folgt, dass der Strom $\underline{I}_4$ berechnet werden kann und mit Hilfe des Kirchhoffschen Knotenstromgesetzes der Strom $\underline{I}_{34} = \underline{I}_{45} + \underline{I}_4$ berechnet werden kann. Nun kann bis zum Knoten N_1 in abwechselnder Reihenfolge mit Hilfe der Kirchhoff-Regeln die Spannung $\underline{U}_i$ bzw. der Strom $\underline{I}_i$ berechnet werden, bis die neu berechnete Spannung $\underline{U}_1 = \underline{U}_{SRC}$ berechnet werden kann. Das Verhältnis zwischen Quellenspannung $\underline{U}_{SRC}$ und neu berechneter Spannung $\underline{U}_1$ kann als *Ratio* = $\underline{U}_{SRC} / \underline{U}_1$ angegeben werden. Nachdem es sich um ein lineares Netzwerk handelt, können nun alle Leitungs- und Lastströme sowie Knotenspannung im Netzwerk mit diesem Faktor multipliziert werden, um das Endresultat für das Netzwerk zu erhalten [Ker07]. Soll nun dieses Prinzip auf nichtlineare Systeme wie ein elektrisches Verteilnetz angewandt werden, so ist bei der Berechnung des Stromes $\underline{I}_i$ in jedem Knoten N_i zu berücksichtigen. Der Strom kann, basierend auf den Grundlagen wie schon in Abschnitt 2.2 erwähnt, zu $\underline{I}_i = (\underline{S}_i / \underline{U}_i)^*$, also aufgrund der Leistung $\underline{S}_i$, die in jedem Knoten N_i als gegeben betrachtet wird, berechnet werden. Als wesentliche Modifikation der Ladder Network Theory ist die Einführung eines Vorwärtsschrittes anzuführen [Ker07]. Dieser Vorwärtsschritt wird durchgeführt, wenn die in jedem Gesamtiterationsschritt neu berechnete Spannung $\underline{U}_1^i$ (i-ter Schritt) von der ursprünglich gegebenen Quellenspannung $\underline{U}_{SRC}$ um einen Wert abweicht, der größer einem gegebenen Schwellwert ist. Das Wesentliche daran ist, dass die im Rückwärtsschritt aufgrund der gegebenen Leistungen und Spannungen die Ströme, und im jeweils darauf folgenden Vorwärtsschritt die Spannungen, aufgrund der berechneten Ströme berechnet werden, wobei als Startwert für die

Spannung am Startknoten N_1 wieder die ursprünglich gegebene Quellenspannung $\underline{U}_{SRC}$ herangezogen wird. Jeder der Schritte verwendet die Kirchhoffsche Maschen- bzw. Knotenstromregel. Damit kommt es zu einer sukzessiven Annäherung an einen bestimmten Spannungs- bzw. Stromwert für jeden Knoten und jede Leitung. Als Beispiel sei auf den Abschnitt 1.3.4 verwiesen, in dem die Ladder Iterative Technique anhand eines Beispielnetzes und die konkreten Berechnungsschritte für eine konstantes Leistungs-Lastmodell erläutert werden.

In dieser Arbeit wird untersucht, wie ein Lastflussanalyse-Verfahren entwickelt werden kann, das möglichst einfach in DAVIC integriert werden kann. DAVIC (siehe Abschnitt 1.2) ist wie schon erwähnt eine Simulationsplattform für Simulation von Geld-, Kommunikations- und Energieflüssen. Die bisherig fertiggestellten Teile der Plattform wurden in OMNeT++ [3] entwickelt auf das im nachfolgenden Abschnitt näher eingegangen wird. OMNeT++ [3] soll im Gegensatz zu dem am Markt erhältlichen kommerziellen Produkten die Möglichkeit bieten, mittels objektorientierten und offenen Softwarekonzepten eine Plattform zu erstellen, die der Anforderung für eine ausreichend hohe Flexibilität bezüglich neuer Forschungskonzepte genügt. Im folgenden Abschnitt wird auf das Framework OMNeT++ [3] eingegangen und dessen technische Eigenschaften näher erläutert.

2.6 Framework OMNeT++

Das Framework OMNeT++ [3] ist ein auf der Sprache C++ basierendes objektorientiertes Framework. Es dient zu Modellierung und Simulation von Kommunikationsnetzen auf Basis von diskreten Ereignissen (*discrete events*). OMNeT++ [3] besteht aus hierarchisch verschachtelten Modulen. Die Verschachtelungstiefe der Module ist dabei nicht begrenzt. Die logische Struktur der zu modellierenden bzw. simulierenden Anwendung kann in dieser Hierarchie abgebildet werden. Die Kommunikation zwischen den Modulen läuft über einen Nachrichtenaustausch ab. Die Nachrichten selbst können komplexe Datenstrukturen beinhalten und damit Daten zwischen den Modulen übermitteln. Die Nachrichten werden entweder direkt zwischen den Modulen oder über Verbindungslinien über g*ates* bzw. c*onnections* versendet. Die Module können Parameter besitzen. Die Parameter der Module dienen der Verhaltenssteuerung der Module bzw. der Topologieparametrisierung des zu modellierenden Systems. Innerhalb einer Modulhierarchie besitzen nur die untersten Module – auf unterster Ebene – ein Verhalten, das vom Benutzer des Frameworks implementiert werden muss. Diese Module werden als Simple-Module bezeichnet und in C++ auf Basis der vom Framework zur Verfügung gestellten Library implementiert. Die Simulation selbst kann in einem GUI oder im Batch-

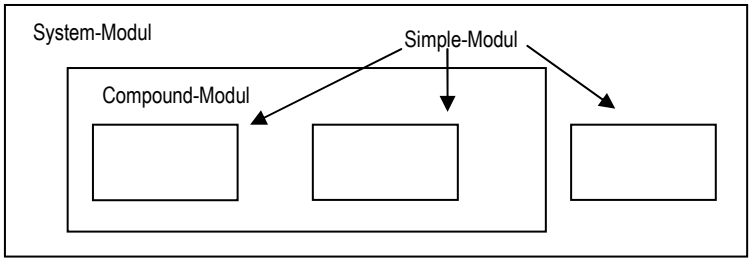


Abbildung 23 – Modulhierarchie OMNeT++ [1]

Modus ausgeführt werden [1]. Das Konzept der grundsätzlichen Modulhierarchie ist in Abbildung 23 dargestellt.

OMNeT++-Modelle werden auch als Netzwerke bezeichnet. Das Top-Level-Modul ist das System-Modul. Darin eingebettet sind die Strukturen die vom Benutzer in Form von Compound-Modulen und Simple-Modulen abzubilden sind. Sowohl die Compond-Module als auch die Simple-Module sind Instanzen von Modultypen. Bei der Beschreibung eines Simulationsmodelles werden die Modultypen spezifiziert. Instanzen dieser Modultypen stellen die Komponenten für komplexere Modultypen dar. Letztendlich erstellt der Benutzer ein System-Modul, als Instanz aller im Vorfeld definierter Modultypen. Alle Module des Netzwerkes werden als Sub-Modul und Sub-Sub-Module des Sys-

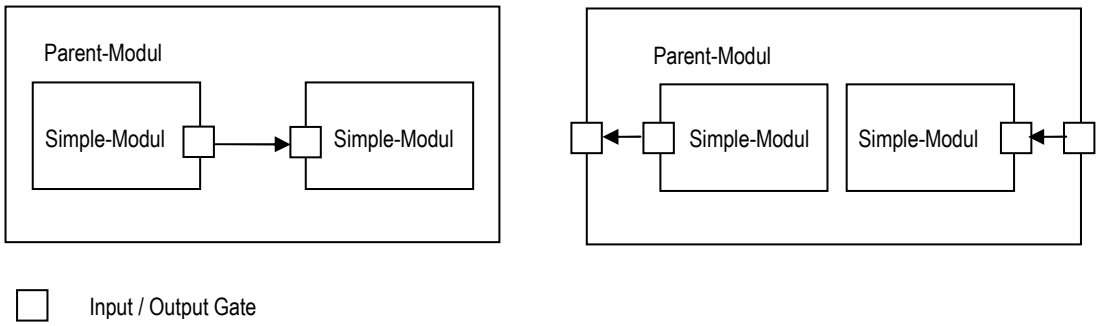


Abbildung 24 – Interfaces, OMNeT++-Module [1]

tem-Moduls instantiiert [1]. Damit ist es möglich Komponenten rekursiv aus Komponenten des Typs Compound-Modul bzw. Simple-Modul aufzubauen.

Module kommunizieren über einen Nachrichtenaustausch. Nachrichten können z. B. Frames oder Pakete in einem Computer-Netzwerk darstellen. Die Nachrichten können direkt an ein Modul oder über *connections,* also vordefinierte Verbindungspfade, verschickt werden. Die lokale Simulationszeit eines Moduls schreitet voran, wenn eine Nachricht empfangen wird. Nachrichten werden von anderen Modulen, oder vom empfangenden Modul selbst, verschickt. *Gates* stellen die Interfaces der Module zur Außenwelt dar. Es existieren *input gates* zum Nachrichtenempfang und *output gates* zum Nachrichtenversand an andere Module oder sich selbst [1]. Abbildung 24 zeigt die Möglichkeiten der Verbindung von Modulen.

Verbindungen sind also zwischen Module derselben Hierarchieebenen, aber auch zu umgebenden Compound-Modulen möglich. Durch die gegebenen Interfaces und Verbindungen ist es möglich, Nachrichten über das gesamte Netzwerk hinweg in beliebige Richtung zu verschicken.

Die durch *connections* modellierten Standardverbindungen zwischen Modulen können über drei Parameter parametrisiert werden [1]:

1. *propagation delay* (Verzögerungszeit)
2. *bit error rate* (BER) (Bitfehler-Rate, Wahrscheinlichkeitswert)
3. *data rate* (Durchsatz in Bit/s)

Das heißt es kann die Verzögerungszeit, die Fehlerrate bei der Übertragung von Nachrichten und die Datendurchsatzrate in Bit/s für eine Standardverbindung spezifiziert werden. Diese Einstellungen können im Abschnitt *channel* der Datei vorgenommen werden [1]. Wie schon erwähnt, können für Module auch Parameter definiert werden. Diese können in einer Datei namens omnepp.ini (Anm. im Standardfall) oder in einer zu jeder Anwendung gehörigen Topologiedatei angegeben werden. Parameter können die Typen String, Numeric, Boolean und XML-Strukturen umfassen, siehe auch [1].

Zusammenfassend lässt sich der Aufbau einer Applikation wie folgt darstellen:

1. Erstellung Topologie mit Modultypen
2. Implementierung des Verhaltens der in der Topologie enthaltenen Simple-Module

2.6.1 Netzwerktopologie in OMNeT++

Im folgenden Abschnitt wird auf die Modellierung einer Netzwerktopologie eingegangen. Die Struktur einer auf OMNeT++ [3] basierenden Anwendung, also die Komponenten eines Netzes und deren Verbindungen (*connections*) werden in einer sogenannten NED-Datei hinterlegt. Diese Datei besitzt folgenden Aufbau, wie in Abbildung 25 gezeigt.

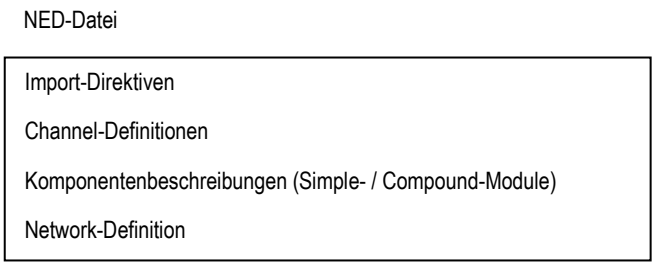


Abbildung 25 – Aufbau NED-Datei

Die Import-Direktiven dienen dazu, andere NED-Dateien und damit Netzwerk-Topologie-Beschreibungen einzubinden. In den Channel-Definitionen können die schon erwähnten Parameter für Fehlerrate, Datendurchsatz und Verzögerungszeit angegeben werden. In den Komponenten-

```
simple name
parameters:
        paramname1: type1,
        paramname2: type2,
        paramname3: type3,
        …
        paramnamen: typen;
gates:
        in: name,
        out[]: name;
endsimple
```

Listing 1 – Definition Simple-Modul

beschreibungen wird der strukturelle Aufbau der im Netzwerk vorhanden Modultypen und deren Parameter spezifiziert. Die Definition der Simple-Modultypen umfasst die Angabe des Namens, die Parameter und die *gates*, also die Spezifikation der Interfaces zur Außenwelt. Der Aufbau ist in Listing 1 zu sehen.

Eine Definition für ein Simple-Modul wird also durch die Schlüsselwörter *simple* und *endsimple* begrenzt und durch die Schlüsselwörter *parameters* und *gates* erweitert. Es können für ein Simple-Modul *n* verschiedene Parameter unterschiedlichen Typs, siehe [1], angegeben werden. Als Interfaces zur Außenwelt, also Anknüpfungspunkte, werden die Schlüsselwörter *in* und *out* verwendet. Diese dienen zur Definition der *input* und *output gates*. Wie auch in Listing 1 ersichtlich, ist es dabei möglich einzelne *gates* auch als Arrays von *gates* zu definieren. Die einzelnen Anschlüsse des *gate arrays* können dann über einen Index angesprochen werden. Die Definition eines *gate arrays* hat über die Angabe von eckigen Klammern [] zu erfolgen.

```
module name
parameters:
        paramname1: type1,
        paramname2: type2,
        paramname3: type3,
        …
        paramnamen: typen;
gates:
        in: name,
        out[]: name;
submodules:
        submodulename1: moduletype1,
        parameters:
         …
        gatesizes:
        …
        submodulename2: moduletype2,
        parameters:
        …
        gatesizes:
        …
        submodulename3: moduletype3,
        parameters:
        …
        gatesizes:
        ….
        …
        submodulenamen: moduletypen;
        parameters:
        …
        gatesizes:
        ….
connections:
…
endmodule
```

Listing 2 – Definition Compound-Modul

Zusammengesetzte oder auch als Compound-Modul bezeichnete Module bestehen aus 1 bis *n* Simple-Modulen. Compound-Module besitzen ebenso Parameter und *gates*. Der Unterschied besteht hier

allerdings in der Verwendbarkeit, da Compound-Module kein eigenes Verhalten implementieren, sondern nur als Struktureinheiten fungieren. Es können damit die Simple-Module gruppiert oder logisch zusammengefasst und die Anwendung damit besser strukturiert werden. Die gegebenen Parameter und *gates* können dazu verwendet werden, Parameter an die Submodule zu übergeben bzw. Verbindungen mit diesen herzustellen. Listing 2 zeigt diesen Zusammenhang.

Die Definition eines Compound-Moduls wird durch die Schlüsselwörter *module* und *endmodule* begrenzt, und durch die Schlüsselwörter *submodules* und *connections* ergänzt. Im Bereich *submodules* können nun die verschiedenen Submodule mit Typ Simple-Modul oder Compound-Modul eingefügt werden, und damit komplexe Strukturen erzeugt werden. Im Bereich *parameters* können die jeweiligen Parameter für das Compound- als auch die Submodule angegeben werden. Im Abschnitt *gatesizes* ist es möglich die Größe für *gate arrays* zu definieren und damit die Anzahl der Anschlüsse für jedes *input* und *output gate* anzugeben. Der Bereich *connections* dient der Spezifikation der Verbindungen des Compound-Moduls mit den zugeordneten Submodulen.

Aus dem bisher Beschriebenen ergibt sich, dass für eine Anwendung die Topologie bzw. deren Module anhand der NED-Datei fest vorgeschrieben wird. Es gibt allerdings die Möglichkeit, Vorlagen für Submodul-Typen zu definieren und den eigentlichen Modultyp als Parameter zu definieren. Dies kann sinnvoll sein, wenn die einzubindenden Module die gleichen Anschlüsse (*gates*) bzw. Parameter (*parameters*) besitzen, die Art der Implementierung jedoch unterschiedlich ist. Dies kann z. B. für die Implementierung von verschiedenen Algorithmen, die dasselbe Problem lösen, sinnvoll sein. Listing 3 zeigt, wie dies bewerkstelligt werden kann.

```
module name
parameters:
        moduletype1: string,
        moduletype2: string,
        moduletype3: string,
        …
        moduletypen: string;
gates:
        …
submodules:
        module1: moduletype1 like templatemodule1;
        module2: moduletype2 like templatemodule2;
        module3: moduletype3 like templatemodule3;
        …
        modulen: moduletypen like templatemodulen;
connections:
…
endmodule
```

Listing 3 – Submodul-Typ als Parameter

Die dynamisch einzubindenden Module werden als Zeichenkette im Abschnitt *parameters* definiert. Dort wird der Name des einzubindenden Moduls angegeben. Die einzubindenden Module werden dann im *submodules* Abschnitt angegeben, wobei als Vorlage jene Typen angegeben werden, die in der Abbildung als $templatemethod_1$ bis $tempatemethod_n$ bezeichnet werden. Diese Vorlagen müssen in der NED-Datei nur definiert werden, um die Vorgaben hinsichtlich zu erfüllender Parameter und

gates zu spezifizieren. Ein Implementierung der Vorlagen ist nicht erforderlich. Damit können z. B. verschiedene Module in C++ umgesetzt werden und in der NED-Datei der Name der implementierten Module angegeben werden. Diese werden dann in der in der Datei definierten Struktur dynamisch eingebunden. Für Parameter kann schon in der NED-Datei eine Wertzuweisung durchgeführt werden. Dies kann entweder direkt in der NED-Datei oder über die zu jeder Anwendung gehörenden Datei omnetpp.ini passieren. Auch eine interaktive Eingabe beim Start der Anwendung ist möglich, siehe auch [1].

Verbindungen werden in der Datei im Bereich *connections* definiert. Dies gilt für alle Compound-Module die aus 1 bis *n* Submodulen bestehen. Nachdem jedes Submodul auch *gates* bzw. *gate arrays* – also *input* und *output gates* mit *n* Anschlüssen – besitzt, können dieses mit den anderen Modulen verbunden werden. In Listing 4 wird dies dargestellt.

```
module name
parameters:
        ...
gates:
        ...
submodules:
        ...
connections:
        submodule1.out → submodule2.in; // Fall1
        submodule1.in ← submodule2.out;
        …
        submodule3.out++ → submodule4.in++; // Fall2
        submodule3.in++ ← submodule4.out++;
        …
        submodule5.out → channelname → submodule6.in; // Fall3
        submodule5.in ← channelname ← submodule6.out;
endmodule
```

Listing 4 – OMNeT++-Verbindungen (*connections*)

Wie aus der Abbildung ersichtlich werden die Submodule durch die definierten *gates in* und *out* miteinander verbunden. Die *gates* werden durch den Namen des Submoduls und dem Namen des *gates* getrennt durch Punkt, angesprochen. Im ersten Fall wird eine Verbindung in beide Richtungen zwischen den Submodulen $submodule_1$ und $submodule_2$ realisiert. In diesem Fall ist bei jedem Modul ein einfaches *gate in* und *out* mit jeweils einem Anschluss vorhanden. An diesem Beispiel zeigt sich, dass ein *gate* nur mit einem anderen *gate* verbunden werden kann. Deshalb ist für eine beidseitige Verbindung an jedem Modul jeweils ein *input* und *output gate* notwendig, die in jeweils umgekehrter Richtung miteinander verbunden werden müssen. Im zweiten Fall wird auf ein *gate array* zugegriffen. Die Operatoren ++ bedeuten, dass der jeweils nächste freie Anschluss des *gate arrays* als Verbindungspunkt herangezogen werden soll. Die dritte Möglichkeit besteht darin, einen Kanal (*channel*) mit den bereits in Abschnitt 2.6 besprochenen Eigenschaften dazwischen zu platzieren. Damit besitzt die Verbindung dann die Eigenschaften des Übertragungskanals.

Die erwähnten Modultypen Compound-Modul und Simple-Module können in der Anwendung noch nicht simuliert werden. Dazu ist eine Netzwerkdefinition notwendig. Eine Netzwerkdefinition refe-

renziert ein Compound-Modul, welches aus mehreren Simple-Modulen besteht und die Verbindungen zwischen diesen definiert. Prinzipiell ist es auch möglich ein Simple-Modul in eine Netzwerkdefinition einzubinden, damit ist es allerdings nicht möglich, einen Kommunikationsgraphen aufzubauen, da Simple-Module keine Verbindungen (*connections*) definieren. Listing 5 zeigt den konzeptionellen Aufbau einer simulationsfähigen Anwendung unter OMNeT++ [3].

```
simple simplemodulename1
   parameters:
         ...
   gates:
         ...
endsimple
...
simple simplemodulenamen
   ...
endsimple

module compoundmodulename
   parameters:
         ...
   gates:
         ...
submodules:
         submodulename1: simplemodulename1
         ...
         submodulenamen: simplemodulenamen
connections:
         ...
endmodule

network networkname: compoundmodulename
   parameters:
         ...
endnetwork
```

Listing 5 – Netzwerkdefinition OMNeT++

Wie in Listing 5 gezeigt, werden *n* Simple-Module mit den modultypischen Anschlüssen (*gates*) und Parametern (*parameters*) definiert. Danach wird ein Compound-Modul erstellt und die definierten Simple-Module eingebunden und die Verbindungen (*connections*) zwischen diesen Modulen definiert. Basierend auf der Compound-Modul-Definition wird ein Netzwerk von diesem Typ instantiiert. Simulationsmodelle in OMNeT++ [3] stellen Instanzen eines Modultyps dar [1].

2.6.2 Basismodule und Nachrichten in OMNeT++

Wie im Abschnitt 2.6.1 gezeigt, dient die zu jedem Simulationsmodell gehörende NED-Datei zur Spezifikation der Topologie des zu simulierenden Netzes. Das tatsächliche Verhalten der Kommunikationsknoten, wird nur in den Simple-Modulen der Anwendung definiert, da die restlichen Elemente nur der zusätzlichen Strukturierung der Netzwerkdefinition dienen. Eine OMNeT++-

Anwendung [3] basiert auf der Simulation diskreter Ereignisse. Zustandsänderungen werden nur zu diskreten Zeitpunkten registriert, d. h. es können keine Ereignisse zwischen zwei aufeinanderfolgenden Zeitpunkten stattfinden [1]. Die Ereignisse in einer Anwendung werden im Finite Event Set (FES), also einer Menge von endlichen Ereignissen verwaltet. Eine Simulation läuft solange bis das FES leer ist, oder aber ein vom Benutzer festgelegtes Zeitlimit abgelaufen ist. In einer Simulation wird zwischen der Modellzeit – Zeit innerhalb des Modells – und realer Zeit – CPU-Zeit – unterschieden [1].

Das Verhalten der Module – Knoten im Kommunikationsgraphen – wird in C++ implementiert. Das Framework stellt dazu eine Klasse *cSimpleModule* zur Verfügung. Jedes Modul muss von dieser Klasse abgeleitet und die Funktionalität implementiert werden. Das Verhalten jeder implementierten Klasse wird über die Reaktion auf eingehende Nachrichten und gegebenenfalls einer Zustandsänderung – Veränderung der Membervariablen – des Knotens realisiert [1]. Die Klasse stellt neben Konstruktur und Destruktor und anderen Methoden die sich aus der im Framework abgebildeten Vererbungshierarchie [2] ergeben, vier weitere Methoden zur Verfügung die vom Anwender des Frameworks implementiert bzw. überschrieben werden müssen, um eine Funktionalität zu erzeugen. Abbildung 26 zeigt einen Ausschnitt der Vererbungshierarchie und die Methoden, die von den

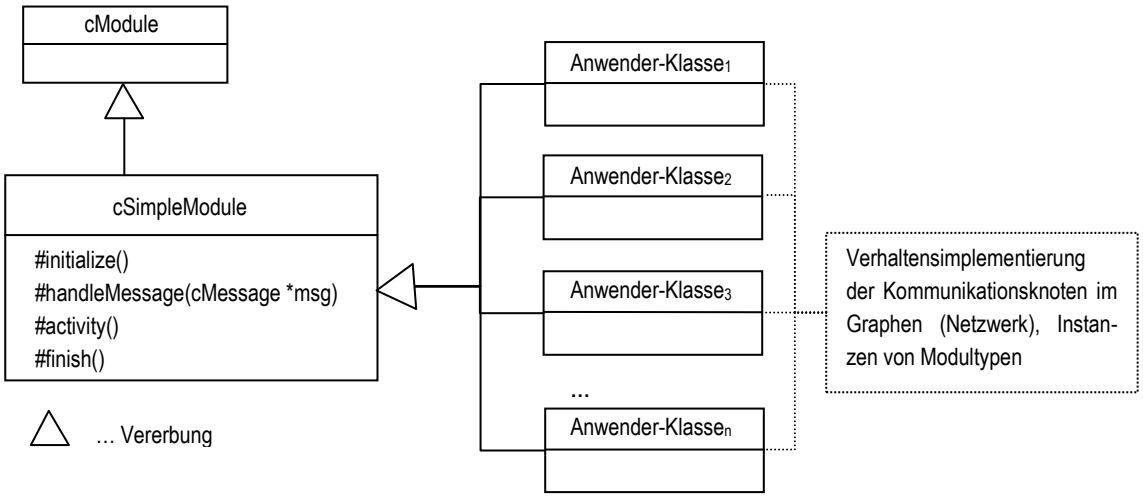


Abbildung 26 – cSimpleModule und Implementierung Klassen

Anwender-Klassen implementiert werden müssen. Die Methode *initialize()* dient der Initialisierung der betreffenden Knoten und wird in der Initialisierungsphase von OMNeT++ [3] aufgerufen. In dieser Methode ist eine erste *message* zu generieren, um den Kommunikationsablauf der Anwendung zu starten. Sie dient also neben dem Lesen von Parametern und der Initialisierung von Klassenvariablen als Bootstrap-Mechanismus. Die Methode *finish()* wird am Ende der Simulation aufgerufen, zu diesem Zeitpunkt können z. B. Protokolle bzw. Statistiken erstellt werden. In der Methode *handleMessage(cMessage *msg)* wird auf eingehende Nachrichten reagiert. D. h. es können neue Nachrichten generiert und verschickt werden. Damit hängt der Simulationsablauf im Wesentlichen von dieser Methode ab. Die Methode *activity()* stellt eine prozessorientierte bzw. threadorientierte Möglichkeit der der Nachrichtenabarbeitung dar. Die in dieser Arbeit umgesetzte Implementierung verwendet allerdings nur die Methode *handleMessage(sMessage *msg)*, deshalb soll hier im Weite-

ren nicht näher darauf eingegangen werden. Sofern benötigt, können auch für die Anwender-Klassen Vererbungshierarchien aufgebaut werden. Für die detailliertere Beschreibung aller Methoden bzw. Hinweise bzgl. der Verwendung sei auf [1] verwiesen.

Der Kommunikationsablauf in einem Netzwerk wird also über einen Nachrichtenaustausch, der über die *handleMessage(cMessage *msg)*-Methode jedes Knotens abgewickelt wird, vollzogen. Simple-Module können Nachrichten erzeugen, senden, empfangen, speichern und löschen. Weiters ist es für Simple-Module möglich *self messages* zu erzeugen. Das bedeutet, dass eine Nachricht zu einem bestimmten Zeitpunkt in der Zukunft an das zu dem Modul gehörende *input gate* verschickt wird. Damit ist es für Simple-Module möglich, einen Timer zu implementieren, in dem z. B. zyklisch s*elf messages* generiert werden. Die Nachrichten selbst, die einer OMNeT++-Anwendung [3] verschickt werden können, werden durch die Klasse *cMessage* implementiert. Eine Nachricht besitzt neben vielen anderen Attributen – siehe auch [1] – in jedem Fall einen Namen und Felder verschiedenen Typs, in denen die Inhalte gespeichert werden können. Um eine *message* zu erzeugen, kann entweder die *cMessage*-Klasse mittels Vererbung erweitert werden oder aber die Nachrichteninhalte bzw. die verschiedenen Nachrichtentypen in einer MSG-Datei definiert werden. Letzteres ist die einfachere und für diese Arbeit gewählte Methode. Innerhalb der MSG-Datei werden die verschiedenen Namen der gewünschten Nachrichtentypen definiert und die jeweiligen Felder mit den Datentypen definiert. In Listing 6 wird eine exemplarische MSG-Datei gezeigt.

```
message MyMessage1
{
   fields:
        type  name1;
        type name2;
        type name3;
        …
        type namen;
}
…
message MyMessagen
{
   fields:
        type  name1;
        type name2;
        type name3;
        …
        type namen;
}
```

Listing 6 – Definition *message*

Hier werden *n messages* definiert. Die Datentypen umfassen die üblichen primitiven Typen, es können aber auch komplexere Strukturen eingebunden werden [1]. Das Framework kann aus der MSG-Datei die notwendigen C++-Dateien und Header erzeugen, welche dann in die Anwendung inkludiert werden. Damit haben die Simple-Module Zugriff auf die *messages* und deren Inhalte.

Die notwendigen Schritte, um eine lauffähige OMNeT++-Simulation zu erzeugen, lassen sich damit wie folgt zusammenfassen:

- Erstellung Topologiedatei (NED-Datei)
- Erstellung Nachrichtendatei (MSG-Datei)
- Implementierung der Simple-Modul-Typen (Verhalten, Reaktion auf eingehende Nachrichten)
- Kompilieren und Linken aller Dateien [1]
- Verarbeitung der INI-Datei (omnetpp.ini) bei Programmstart

Die folgende Abbildung 27 zeigt den Zusammenhang aller Schritte sowie aller notwendigen Dateien. Zu erkennen sind die NED- und die MSG-Dateien. Die NED-Datei definiert die Netzstruktur, die von der Laufzeitumgebung des Frameworks bei Simulationsstart aufgebaut wird. Die MSG-Datei definiert die verschiedenen Nachrichtentypen und deren Inhalte, die zwischen den Instanzen der Modultypen zur Laufzeit der Anwendung ausgetauscht werden können. Die C++-Dateien implementieren das Verhalten der Modultypen (Knotentypen) des Graphenmodells. Die beim Start der Anwendung eingelesene INI-Datei liefert eventuelle Startwerte bzw. Initialisierungswerte für die Parameter der Knoten deren Definition in der NED-Datei vorgenommen wird.

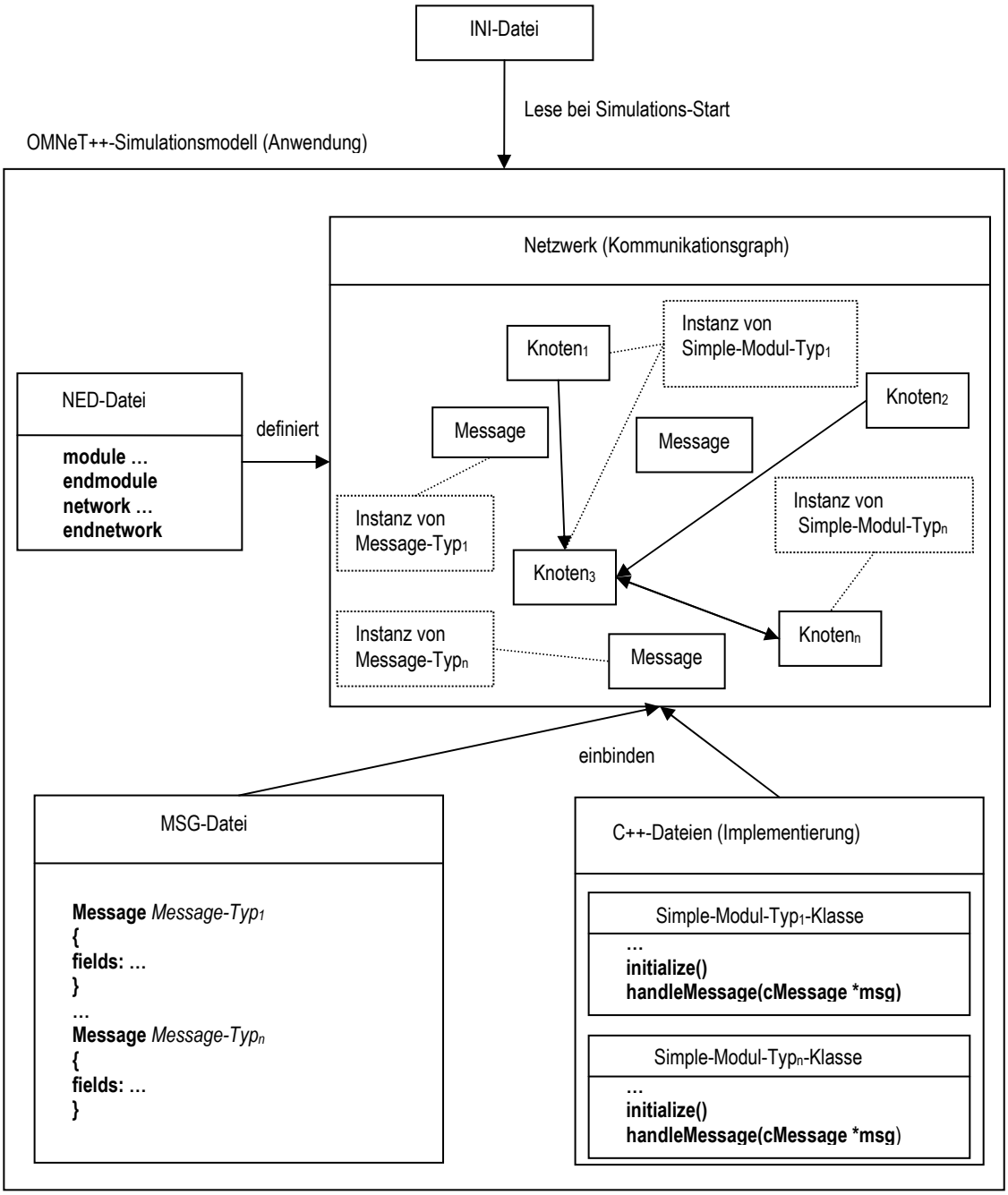
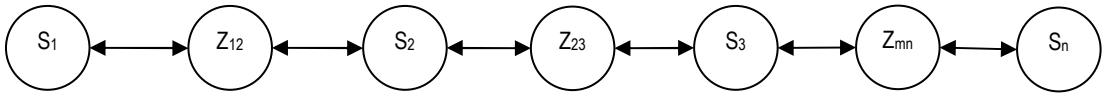


Abbildung 27 – OMNeT++ Simulationsmodell

3. Gewählter Lösungsansatz und Implementierung

In den Abschnitten 2.3 und 2.6 wurden die elektrotechnischen Grundlagen und Eigenschaften des Frameworks OMNeT++ [3], welches zur Implementierung einer Lastflussanalyse-Verfahrens für Verteilnetze herangezogen werden soll, erläutert. In diesem Abschnitt wird eine Übersicht über die möglichen Implementierungsansätze sowie deren Probleme, Nutzen und letztendlich die Gründe für die in dieser Arbeit gewählte Methode erläutert. Wie bereits in Abschnitt 1.3 und 1.5.1 erwähnt, gibt es zwei Hauptgruppen von Analyseverfahren für elektrische Netze. Dies sind matrizenbasierte Verfahren, bei denen die Topologie der Netze in Form von Knotengleichungen abgebildet wird, und das iterative Stromsummationsverfahren bzw. Abwandlungen der Ladder Iterative Technique. In dieser Arbeit soll bei der Abbildung der Struktur des Netzes aufgrund des sonst großen Umfangs, der Schwerpunkt vor allem auf die Leistungsknoten – Leistungsabnahme und -einspeisung – und auf die Modellierung der Leitungen gelegt werden. Jedes elektrische Netz kann im Prinzip als Graph, also als Menge von Knoten und Kanten, aufgefasst werden. Als Lastmodell soll für Leistungsknoten ein konstantes Leistungsmodell in der Form $\underline{S} = P + jQ$ (*PQ*-Knoten) und für Leitungen die Impedanz $\underline{Z} = R + jX$ herangezogen werden.

Die gewählte Lösung soll die Möglichkeit bieten, ein gegebenes Netz in DAVIC zu berechnen. Ausgegangen wird von einem elektrischen Netz, das im konventionellen Block von DAVIC als Kommunikationsgraph im konzeptionellen Bereich Elektrisches Netz abgebildet ist. Nachdem wie schon in Abschnitt 2.6 und in [1] erwähnt, die Standardverbindungen von Modulen, respektive Knoten in OMNeT++ [3], keine Parametrisierung außer den erwähnten zulassen, muss für die Leitungen eines Energienetzes ein eigener Modultyp mit der Eigenschaft Impedanz $\underline{Z}$ eingeführt werden. Die Leistungsknoten besitzen die Eigenschaft der Leistung $\underline{S}$. Zwischen den Leistungs- und Leitungsknoten seien die Standardverbindungen von OMNeT++ [3] vorhanden. Abbildung 28 zeigt das Modell eines einfachen linearen elektrischen Netzes ohne Verteilknoten.



S_i … Leistungsknoten ($\underline{S} = P + jQ$) , Instanz vom Modultyp Leistungsknoten

Z_{ij} … Leitungsknoten ($\underline{Z} = R + jX$) zwischen Knoten N_i und N_j, Instanz vom Modultyp Leitungsknoten

↔ … Standardverbindung in OMNeT++

Abbildung 28 – Lineares Netz (OMNeT++)

3.1 Matrizenbasierte Umsetzung in OMNeT++

Für die matrizenbasierte Umsetzung der Analyse könnte folgender Ansatz in OMNeT++ [3] umgesetzt werden:

- Wissen über Struktur des Netzes ermitteln (Topologie)
- Parameter (Leistungen und Impedanzen) der Knoten beziehen
- Berechnung an einem besonders markierten Knoten (Kalkulationsknoten)
- Werte an alle Knoten propagieren

Das bedeutet, dass in das Netz z. B. ein neuer Knoten eingebracht werden kann, der die Berechnung des Verfahren aus Abschnitt 1.3 durchführt. Um die Knotengleichungen aufzustellen, muss dieser Knoten allerdings Wissen über die Struktur besitzen. Die Strukturinformation kann über die Klasse *cTopology* der Framework-API zur Verfügung gestellt werden. Basierend auf der Strukturinformation können die Knotengleichungen (Anm. z. B. Gauß-Seidel-Iteration, Newton-Rapshon-Verfahren) aufgestellt werden. Die Werte der Knoten können entweder zu Beginn der Simulation aus der NED-Datei, oder über Nachrichten von den Knoten zur Verfügung gestellt werden. Nachdem auch die Möglichkeit einer Parameteränderung des Knotens in Betracht gezogen werden soll, und diese Änderungen ohnehin an den Kalkulationsknoten kommuniziert werden müssen, um einen neuen Simulationslauf zu starten, ist der nachrichtenbasierte Ansatz besser geeignet. Es ergeben sich damit zwei Möglichkeiten:

1. jeder Knoten verschickt Nachricht mit Wert
2. eine Nachricht zum Sammeln aller Werte

Im ersten Fall verschickt also jeder Knoten eine Nachricht mit den Parametern an den Kalkulationsknoten. Das bedeutet, dass zu Beginn der Simulation und nach Bestimmung der Strukturen erhält der Kalkulationsknoten im Fall eines linearen Netzes wie in Abbildung 29 gezeigt, n Nachrichten von den Leistungsknoten und n-1 Nachrichten von den Leitungsknoten, in Summe also $2n$-1 Nachrichten. Die Nachrichtengröße ist dabei durch die Speicherung eines Wertes pro Nachricht beschränkt. Nach Berechnung des Netzes müssen die jeweiligen Knoten über ihren aktuellen Wert – Spannungs- bzw. Stromwert – in Kenntnis gesetzt werden. Dazu ist bei diesem Ansatz wieder eine Nachricht fällig, d. h. die Gesamtanzahl der Nachrichten verdoppelt sich daher auf $2(2n\text{-}1) = 4n\text{-}2$ Nachrichten.

Im zweiten Fall wird ausgehend vom Kalkulationsknoten eine Nachricht in das Netz hineingeschickt. Beim Passieren jedes Knotens – Leitungs- und Leistungsknoten – trägt jeder Knoten seinen Wert in eine in der Nachricht vorgesehene Datenstruktur (Liste bzw. dynamisches Array) ein. Beim Erreichen des Endknotens wird die Nachricht an den Kalkulationsknoten zurückgeschickt. Daraus folgt, dass bei diesem Ansatz eine Nachricht mit der Anzahl von $2n$-1 Parametern (n Leistungsknoten und n-1 Leitungsknoten) notwendig ist. Nach der Berechnung des Netzes ist eine Nachricht mit den neu berechneten Spannungs- und Stromwerten zu erstellen und in das Netz hineinzuschicken. Jeder Knoten konsumiert dabei beim Passieren der Nachricht seinen jeweiligen Wert aus einer vor-

definierten Datenstruktur. Das bedeutet, dass nach Verarbeitung des letzten Knotens alle Datenstrukturen in der Nachricht leer sind, d. h. alle Werte gesetzt bzw. konsumiert wurden.

Die Größe bzw. Anzahl der Nachrichten basiert auf der Annahme eines einfachen Netzes ohne Verzweigungen, wie in Abbildung 28 dargestellt. Diese beiden Fälle gelten im Fall des Starts der Simulation und damit bei der Berechnung des ersten Simulationsschrittes. Das heißt es wird mit diesem Ansatz eine Steady-State-Analyse zu einem bestimmten Zeitpunkt *t* (Simulationsstart, gegebene Parameter) durchgeführt. Die berechneten Resultate können dann bei Beendigung der Simulation in eine Datei ausgegeben werden. Die Notwendigkeit der Information der Knoten über die neu berechneten Werte ergibt sich aus dem Problem, dass das Netz in Zukunft mit den neu berechneten Werten, auch noch für andere DAVIC-Komponenten herangezogen werden soll. Mit dem soeben beschriebenen Ansatz ergeben sich die folgenden Probleme bzw. Herausforderungen:

- Aufwand Implementierung matrizenbasiertes Verfahren
- Monte-Carlo-Simulation (wiederholte Steady-State-Simulation mit veränderten Parametern)
- Größe bzw. Anzahl der Nachrichten (Laufzeit bzw. Speicher)
- Anwendbarkeit auf Verteilnetze

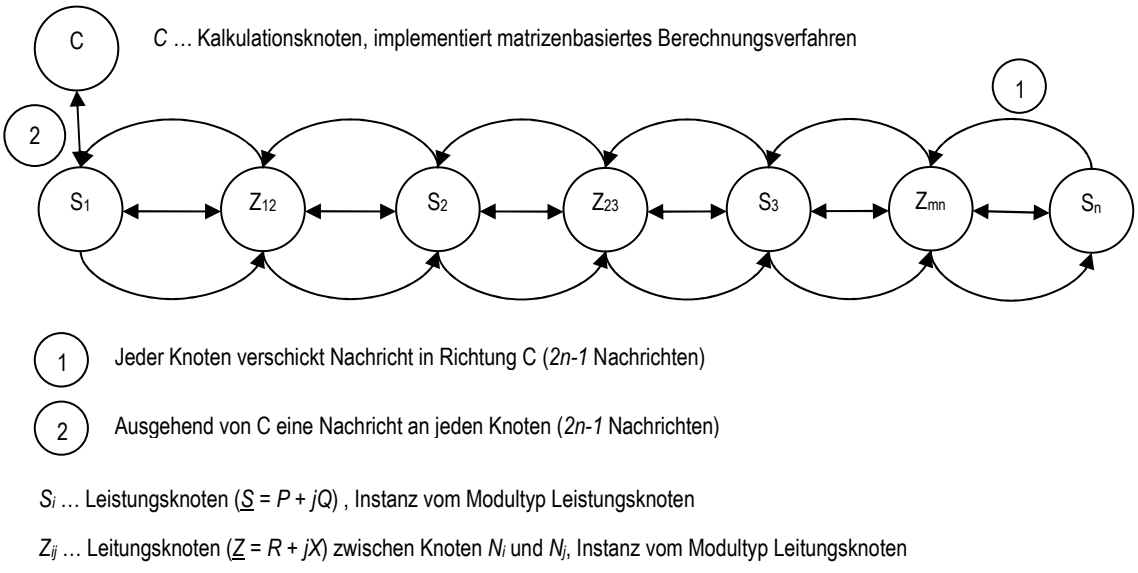


Abbildung 29 – matrizenbasierte Berechnung in OMNeT++ (Fall1)

Der Aufwand für die Implementierung der Berechnung im Kalkulationsknoten ist in die Eruierung der Struktur des Netzes, dem Aufbau der Knotengleichungen und Berechnung derselben zu unterteilen. Ein Vorberechnung und reine Berechnung der z. B. bereits von Hand vorbereiteten Gleichungen erscheint nicht sinnvoll, da die Information aus dem Netz bezogen werden muss. Deshalb ist der Aufwand hier, ohne auf die Implementierungsdetails unter Berücksichtigung von vorhandenen Klassen und Datenstrukturen in C++ einzugehen, als sehr aufwändig einzustufen, da für die Berechnung

ein algebraischer Gleichungslöser implementiert werden müsste. Kommerzielle Produkte wie MATLAB [4] bzw. MAPLE [5] scheinen für diesen Fall besser geeignet.

Bei einer Parameteränderung von Knoten muss die Berechnung von Neuem durchgeführt werden, was zumindest die Neuberechnung der bereits erstellten Knotengleichung erforderlich macht. In Abhängigkeit der Wahl bezüglich Eruierung und Propagierung der Werte jedes Knotens, wird entweder das zu simulierende Netz mit Nachrichten geflutet oder aber je nach Anzahl der Knoten die Größe der Nachrichten maßgeblich beeinflusst (ein Wert pro Knoten).

3.2 Ladder Iterative Technique in OMNeT++ realisiert

Deshalb erscheint basierend auf den Abschnitten 1.3.4, 1.5.1 und 2.3 die Ladder Iterative Technique auch hinsichtlich der Eigenschaften von elektrischen Verteilnetzen die bessere Wahl zu sein. Im Folgenden soll auf die konzeptionelle Abbildung dieses Verfahrens in OMNeT++ eingegangen wer-

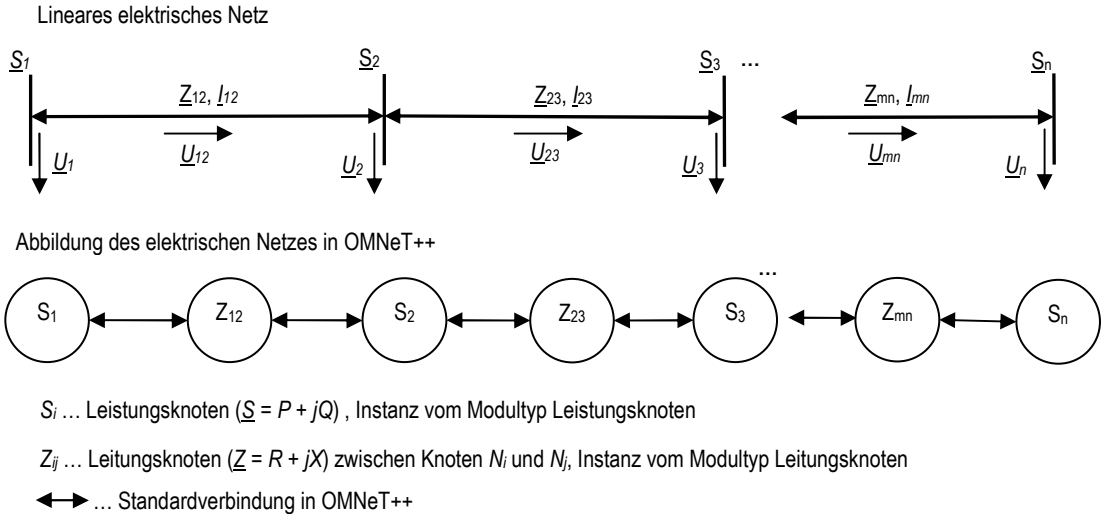


Abbildung 30 – Ladder Iterative Technique (OMNeT++)

den. Gegeben sei nochmals das in Abbildung 30 einfache lineare elektrische Netz mit *n* Leistungsknoten *S* und *n*-1 Leitungsknoten *Z*. Damit ergibt sich für die Berechnung des Stroms an einem Endknoten – hat keinen Nachfolger – $\underline{I}_n = (\underline{S}_n / \underline{U}_n)^*$ und für den Leitungsstrom $\underline{I}_{mn} = \underline{I}_n$. Für innere Knoten, ergibt sich $\underline{I}_m = (\underline{S}_m / \underline{U}_m)^* + \underline{I}_n$, also die Summe aus dem Strom, welcher durch die Leistung und die Spannung am Knoten S_m bestimmt wird und dem Wert des Nachfolgerknotens. Die Spannung auf der Leitung Z_{ij} welche die Knoten S_i und S_j $(i \neq j)$ miteinander verbinden, wird aus der Impedanz $\underline{Z}_{ij}$ und dem Strom $\underline{I}_j$ berechnet. Das bedeutet, dass der Strom auf einer Leitung die die Knoten S_i und S_j miteinander verbindet dem Strom der im Knoten S_j vorhanden ist, entspricht. Für die Berechnung der Knotenströme und Knotenspannungen werden die Kirchhoffschen Regeln herangezogen, siehe auch Abschnitt 1.3.4. Eine zentrale Berechnung an einem Knoten und Propagierung der Ergebnisse an alle Knoten im Netz ist aufgrund der Probleme die auch in Abschnitt 3.1 erwähnt wurden, zu vermeiden. Nachdem jeder Knoten in OMNeT++ [3] auf eingehende Nachrichten reagieren kann,

liegt es nahe, mit einem geeigneten Nachrichtenmodell bzw. der Implementierung eines Zustandsautomaten in den Knoten, den Algorithmus verteilt über das Netz zu implementieren. Damit erzeugt jeder Knoten aus seiner Sicht und in Abhängigkeit der eingehenden Nachrichten einen bestimmten Zustand und Wert und kann diesen in das Netz über Nachrichten weitergeben. Andere Knoten erhalten diese Nachrichten mit den für sie relevanten Werten und führen ebenso die notwendigen Änderungen durch und generieren neue Nachrichten.

Das Verfahren der Ladder Iterative Technique besteht im Wesentlichen aus drei Schritten:

1. Initialisierung
2. Rückwärtsschritt
3. Vorwärtsschritt

Diese Schritte sind in der Implementierung mittels Nachrichten umzusetzen. Wie schon in Abschnitt 1.5 gezeigt, existieren in einem elektrischen Netz verschiedene Komponenten. In dieser Arbeit bzw. für die Implementierung wird aus Gründen des Umfangs der Schwerpunkt vor allem auf die Leistungsabnahme (Lasten), Leistungseinspeisung (Generator) und die Übertragungsleitungen gelegt. Als Modultypen in OMNeT++ [3] können demnach

- Leistungsknoten
- Leitungsknoten

identifiziert werden (siehe auch Abbildung 30). Jede Instanz dieser beiden Knotentypen (Modultypen) reagiert auf verschiedene Nachrichten. Um nun die am jeweiligen Knoten notwendige Berechnung durchführen zu können, ist eine Bestimmung des erhaltenen Parameters, sowie die Richtung aus der die Nachricht kommt, notwendig. In Abhängigkeit dieser Information kann der Zustand in dem vom Knoten implementierten Zustandsautomaten geändert werden bzw. notwendige Aktionen generiert werden. Abbildung 31 zeigt diesen Zusammenhang.

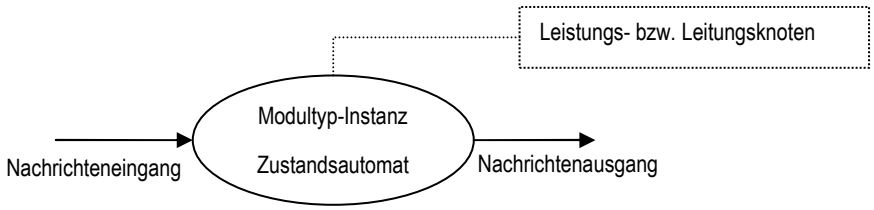


Abbildung 31 – Zustandsautomat (Netzkomponente)

Die Information bezüglich Richtung kann entweder als Inhalt der Nachricht, z. B. als Nachrichtentyp oder aber aufgrund des *gates* auf dem die Nachricht einlangt bestimmt werden. Es soll angenommen werden, dass für jeden Schritt des Verfahrens ein Typ von Nachricht generiert werden soll. Diese Nachrichten beinhalten neben der Information über die Richtung mit der sich die Nachricht durch das Netz bewegen soll, auch die Daten die zwischen den verschiedenen Modultyp-Instanzen respektive Knoten ausgetauscht werden sollen. Als relevante Daten für das elektrische Netz und dessen Modellierung als Kommunikationsgraph können die folgenden Größen festgelegt werden:

- komplexer Strom $\underline{I}$
- komplexe Spannung $\underline{U}$
- komplexe Scheinleistung $\underline{S}$
- Impedanz $\underline{Z}$

Diese Daten sind je nach Lage und Verarbeitung im Netz als Parameter der Modultypen bzw. als Felder in den Nachrichten zu hinterlegen. Als Parameter der Modultyp-Instanzen (Knoten) sollen die Scheinleistung an den Leistungsknoten und die Impedanz an den Leitungsknoten festgelegt werden. Die Werte für den Strom und die Spannung kann von den Knoten berechnet werden und in den Nachrichten als Datenfeld-Wert übermittelt werden. Die Nachrichtentypen können nach den Abarbeitungsschritten des Verfahrens kategorisiert werden:

- Init-Nachricht
- Forward-Nachricht
- Backward-Nachricht

Die Init-Nachricht initialisiert alle Knoten zu Beginn der Simulation. Die Forward- bzw. Backward-Nachrichten dienen der Berechnung der Ströme und Spannungen auf den Leitungen in Anfangs- bzw. Endrichtung (Blätter) des Netzes. Zur Abschätzung des Berechnungsaufwandes des Verfahrens hinsichtlich Nachrichtenanzahl bzw. Berechnungsdauer kann wie folgt vorgegangen werden. Im Initialschritt wird eine Nachricht erzeugt, die in das Netzwerk hineingeschickt wird und am Ende vernichtet werden kann. Alle Knoten sind dann initialisiert. Eine weitere Nachricht ist notwendig um das Netz in Richtung Beginn des Netzes zu durchlaufen. Sofern weitere Iterationen fällig werden – siehe Bedingung am Startknoten in Abschnitt 1.3.4 – ist eine weitere Iteration notwendig. Für jede weitere Iteration werden zwei weitere Nachrichten erzeugt, die allerdings am jeweiligen Ende des Netzes – Anfang oder Ende – zerstört werden können. Das bedeutet, dass in einem linearen Netzwerk wie in Abbildung 30 gezeigt, zwei Nachrichten fix sind und sich der Rest aus $2n$ Nachrichten für n Iterationsschritte zusammensetzt. Die Gesamtanzahl an erzeugten Nachrichten liegt demnach bei Gesamtanzahl $2 + 2n$. Wesentlich und im Unterschied zu dem Ansatz aus 3.1 ist, dass allerdings pro Iteration in einem linearen Zweig jeweils nur eine Nachricht unterwegs ist und diese an den Knoten modifiziert bzw. gelöscht und durch eine andere Nachricht ersetzt werden kann. Weiters liegt der Vorteil des Verfahrens neben der Eignung für elektrische Verteilnetze in der Möglichkeit, die Berechnung des Algorithmus durch die beteiligten Komponenten respektive Knoten verteilt durchführen zu lassen, und auch darin, dass keine zentrale Datenstrukturen (Matrizen o. ä.) notwendig sind.

Im dem nun folgenden Kapitel wird auf die konkrete Umsetzung der Ladder Iterative Technique eingegangen. Dabei wird im Besonderen auf die Implementierung der Netzstruktur, der Nachrichten- und Knotentypen eingegangen. Weiters wird der Algorithmus und dessen Abbildung in einen Zustandsautomaten aus der Sicht der Knoten wie auch im Hinblick auf das Gesamtsystem gegeben. Abschließend wird die Einbettung in das Projekt DAVIC und die Ergebnisse des Verfahrens auf Basis von Beispielnetzen erläutert.

3.3 Netz- und Nachrichtendefinition

Wie schon in Abschnitt 2.6.1 erwähnt, wird die Netzstruktur über die Definition von Modultypen und deren Verbindungen erstellt. In diesem Abschnitt wird gezeigt, wie ein einfaches elektrisches Verteilnetz über die in Abschnitt 2.6.2 erwähnten Modultypen modelliert werden kann. Die beiden Knotentypen werden wie folgt als *Node* und *Line*, also für die Leistungseinspeisung und -abnahme

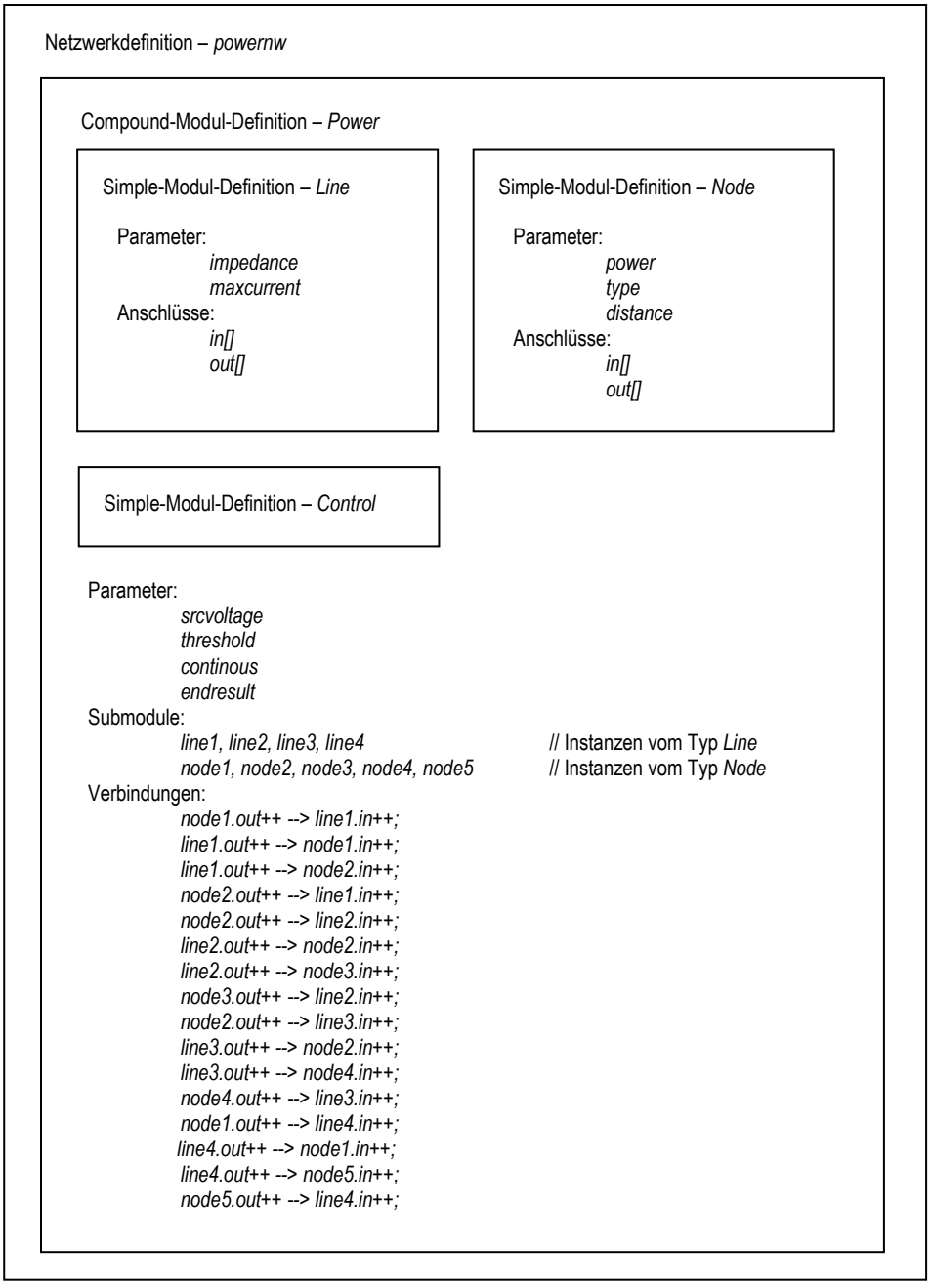


Abbildung 32 – elektrisches Verteilnetz in OMNeT++ (Beispielnetz)

sowie die Eigenschaften der Übertragungsleitungen, bezeichnet. Abbildung 32 zeigt den konzeptionellen Aufbau der Simple- und Compound-Module sowie die Einbettung in das Netzwerk *powernw*, für das Beispielnetz aus Abbildung 4. Das Netzwerk *powernw* kann dann im Framework simuliert werden. Das Compound-Modul *Power* stellt die Verbindungen zwischen den Simple-Modulen her. Für die genaue Syntax der Definitionen sei auf Abschnitt 2.6 und auf [1] verwiesen. Die Leistungsknoten sind als Instanzen des Modultyps *Node* mit *node1* bis *node5*, die Leitungsknoten als Instanzen des Modultyps *Line* mit *line1* bis *line4* bezeichnet. Die Parameter *impedance* und *maxcurrent* des Typs *Line* dienen der Angabe des komplexen Widerstands, der Parameter *maxurrent* der Angabe das maximalen Stroms, der auf der Leitung anliegen darf, jeweils als Zeichenkette in der Form *z = real imaginär*. Die Parameter *power, type* und *distance* des Typs *Node* dienen der Angabe der Leistung, des Typs und der Distanz zum Startknoten (Slack-Knoten). Die Leistung ist als komplexer Wert mit positivem oder negativem Vorzeichen, d. h. je nach Lastfall – Abnahme oder Einspeisung – in Zeichenkettenform *z = real imaginär* anzugeben. Der Typ dient der Angabe, ob es sich um einen Start- oder Endknoten bzw. um einen inneren Knoten des Netzes handelt. Die Codierung wird über eine Zeichenkette codiert welche die Einträge START, INNER oder TERMINAL enthält. Der Parameter für die Distanz enthält die Entfernung jedes Leistungsknotens vom Startknoten (Slack-Knoten) in der Einheit Meter m. Der Startknoten enthält für diesen Eintrag den Wert 0. Das Simple-Modul *Control* enthält keine Parameter und hat auch keine Verbindungen zu den anderen Knotentypen *Line* und *Node*. Das Modul dient der Protokollierung der Ergebnisse nach einem fertiggestellten Simulationslauf. Dadurch, dass es als eigener Modultyp definiert ist, hat es Zugriff auf die strukturellen Eigenschaften aller Modultyp-Instanzen einer Simulationsanwendung. Im Abschnitt 3.6wird noch näher auf die Funktionsweise eingegangen. Die Parameter *srcvoltage, threshold, continous* und *endresult* dienen der Angabe der Spannung am Slack-Knoten, des Schwellwerts für die Berechnung der Differenzbedingung des Verfahrens und der Angabe ob eine laufende und oder nur abschließende Protokollierung der zu berechnenden Werte jedes Knotens durchgeführt werden soll. Eine detaillierte Erläuterung des Ablaufs wird in den Abschnitten 3.4 und 3.5 gegeben.

Für jede Verbindung zwischen den Knoten des Beispielnetzes aus Abbildung 4 müssen im Fall einer einfachen Verbindung zwischen *Node*- und *Line*-Typen zwei Verbindungen für den Hin- und Rücktransport einer Nachricht vorgesehen werden, da Hin- und Rückverbindungen für einen Kommunikationsknoten in OMNeT++ [3] nicht möglich sind [1]. Wie in Abbildung 32 ersichtlich, werden die Verbindungen zwischen den verschiedenen Instanzen der Knotentypen mittels *gate arrays* implementiert. Dies ist an den eckigen Klammern ersichtlich. Dies bedeutet, dass die Indizes der Verbindungen automatisch nach der Verbindungsreihenfolge der Knoten zugewiesen werden. Um eine eindeutige Zuweisung zu gewährleisten und damit einen über das gesamte Netzwerk und damit pro Knotentyp-Instanz eindeutige Richtung für den Nachrichtenfluss gewährleisten zu können, ist folgende Annahme bei der Modellierung getroffen worden. Der Index 0 jedes *gate arrays* zeigt immer in Richtung Anfang des Netzes, es sei denn es handelt sich um den Anfangsknoten (Startknoten) des Netzes. Alle weiteren Indizes zeigen in das Netz hinein. Im Fall eines einfachen linearen Netzes gilt damit für den Index 1, dass dieser immer zum Nachfolger zeigt. Handelt sich um einen Verteilknoten, so zeigen die Indizes 1 bis *n* zu den *n* Nachfolgern des Verteilknotens. Weiters sei angenommen, dass immer nur ein Leitungsknoten zwischen zwei Leistungsknoten platziert werden kann. Dies bedingt, dass sich am Anfang eines Netzes und am Ende eines Leitungsabschnittes (Ast) eines

Netzes, immer ein Leistungsknoten (*Node*) befinden muss. Wie schon erwähnt, kann es in einem Netz auch Verteilknoten geben. Verteilknoten sind dadurch charakterisiert, dass diese mehrere Nachfolgeknoten haben können. Als Nachfolgeknoten kommen dabei nur Leitungsknoten in Frage, welche damit einen neuen Ast an das Netz anschließen, an dem dann wieder Leistungsknoten und Leitungsknoten in abwechselnder Reihenfolge, unter Berücksichtigung der erwähnten Bedingungen, vorkommen können. Damit lässt sich die Modellierung eines elektrischen Netzes (Verteilnetzes) wie folgt zusammenfassend charakterisieren:

- Leistungsknoten am Anfang und Ende jedes Leitungsabschnittes
- Startknoten dient als Slack-Knoten (Spannungswert vorgegeben)
- ein Leitungsknoten sitzt immer zwischen zwei Leistungsknoten
- Verteilknoten hat einen Knotengrad > 2, ein innerer Knoten hat einen Knotengrad = 2, die Endknoten eines Netzes haben Knotengrad = 1, ein Startknoten kann auch ein Verteilknoten sein (mehrere Versorgungslinien)
- Anschluss mit Index = 0 eines *gate arrays* zeigt immer in Richtung Anfang des Netzes (Startknoten)
- auf einen Leistungsknoten muss immer ein Leitungsknoten folgen, es sei denn es handelt sich um einen Endknoten

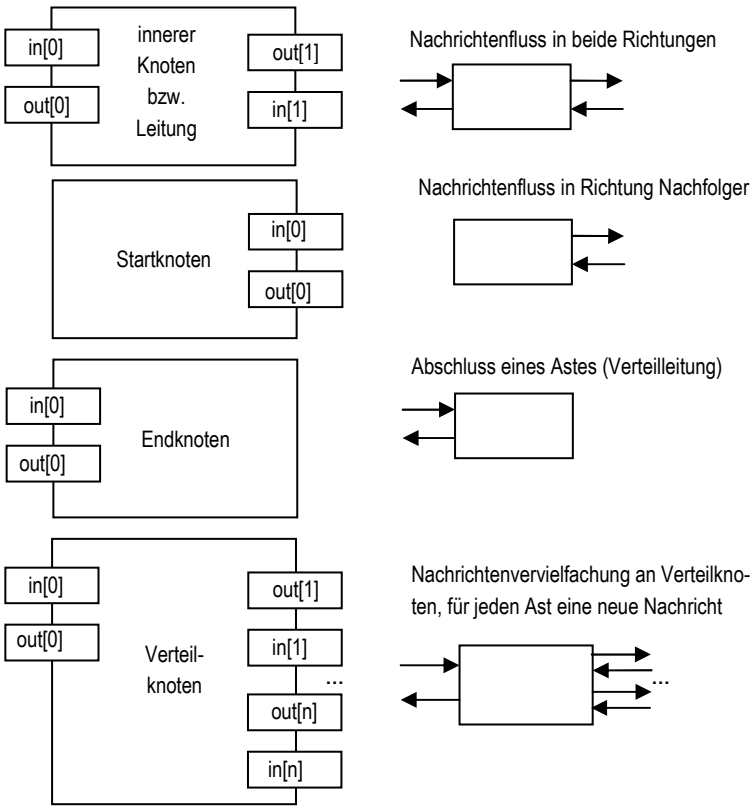


Abbildung 33 – Nachrichtenfluss Knotentypen

Der Zusammenhang der Kriterien sowie die Annahmen, die bei der Modellierung eines elektrischen Verteilnetzes im Zuge der Implementierung in OMNeT++ [3] getroffen wurden, werden nochmals in Abbildung 33 veranschaulicht und erläutert. Wie in der Abbildung ersichtlich, sind die Anschlüsse die in Richtung Startknoten des Netzes zeigen, immer dem Index 0 zugeordnet. Damit kann der Nachrichtenfluss abhängig vom Typ der Nachricht, dem Zustand in dem sich die Instanz des Modultyps befindet und dem Index des *gate arrays* eindeutig festgelegt werden. Im Fall eines inneren Knotens bzw. einer Verbindungsleitung (*Line*) kommt es zu einem Nachrichtenfluss in beide Richtungen. Das bedeutet, dass die eingehenden Nachrichten verarbeitet werden und je nach Zustand des Knotens und in Abhängigkeit vom Typ der Nachricht weitergeschickt bzw. die Nachricht gelöscht und eine neue Nachricht mit einem anderen Typ weitergeschickt wird. Im Fall des Startknotens wird eine Init-Nachricht verschickt bzw. die Backward-Nachrichten verarbeitet und neue Forward-Nachrichten generiert und neu verschickt. Der Nachrichtenfluss am Startknoten findet also immer nur in Richtung der Endknoten der angeschlossenen Verteiläste des Netzes statt. Die Nachrichten am Endknoten werden dort ebenso verarbeitet und der Rückwärtsschritt des Verfahrens eingeleitet. Die eingehenden Nachrichten werden zerstört und eine neue Nachrichtung in Richtung des Startknotens des Netzes verschickt. Für Verteilknoten gilt dasselbe wie für innere Knoten, mit dem Unterschied das mehrere Äste an dem *gate array* ab Index 1 angeschlossen werden können. Die eingehenden Nachrichten bzw. neu zu erstellenden Nachrichten müssen an diesen Knotentypen also für die Anzahl der angeschlossenen Äste vervielfacht werden und an jeden dieser Äste verschickt werden. Eine Besonderheit ergibt sich für eingehende Nachrichten aus den angeschlossenen Ästen, bei der Verarbeitung der Backward-Nachrichten. Auf diesen Fall wird in der Behandlung der Implementierung der Klasse *Node* in Abschnitt 3.4.2 eingegangen. Auch an Startknoten können mehrere Leitungen angeschlossenen werden, sie stellen somit Verteilknoten, allerdings ohne Rückleitung, dar.

Für die Implementierung des Verfahren wurden, wie schon erwähnt, drei Nachrichtentypen eingeführt. Die Abbildung 34 zeigt die drei verschiedenen Typen und Felder der Nachrichten. Das Feld *dir* wird benötigt um in der Nachricht die Richtungsinformation mitzugeben. Die Richtungsinformation wird über die Einträge eines Aufzählungstyps codiert. Die Einträge lauten auf INIT (1), FORWARD (2) und BACKWARD (3). Jeder Knoten der die Nachricht erhält kann nun aufgrund dieser

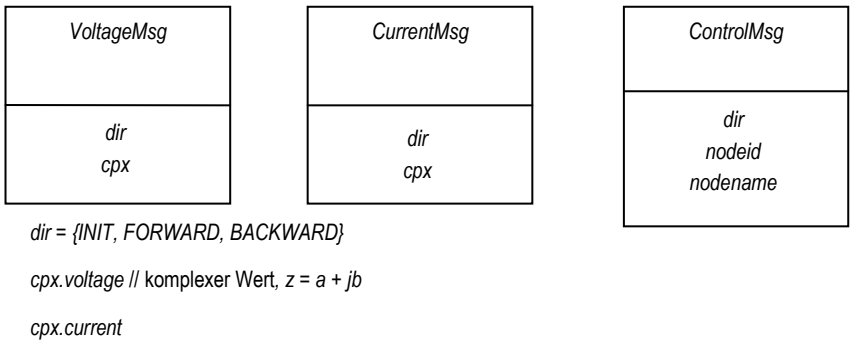


Abbildung 34 – Nachrichtentypen im Verteilnetz

Information und des Zustands in dem sich der Knoten befindet, die Nachricht entsprechend weiterverarbeiten, d. h. zerstören oder in die angegebene Richtung innerhalb des Netzes weiterschicken.

Das Feld *cpx* enthält eine Struktur mit komplexen Zahlenwerten. Diese Werten umfassen Einträge für die Spannung und den Strom und sind sowohl in der Spannungsnachricht (*VoltageMsg*) und Stromnachricht (*CurrentMsg*) vorhanden. Die Kontrollnachricht (*ControlMsg*) umfasst ebenso das Feld *dir* und zusätzlich die Felder *nodeid* und *nodename*. Die zusätzlichen Felder dienen der Übermittlung der Knoten-ID sowie des Knotennamens. Die Kontrollnachricht wird unter anderem dazu verwendet, den Simulationsablauf hinsichtlich Protokollierung der errechneten Werte zu steuern, also auch die Steuerung für eine Monte-Carlo-Simulation zu unterstützen.

Damit ergeben sich in Summe sechs verschiedene Nachrichtentypen. Drei für die Spannungsnachrichten (Init-, Forward-, Backward-Spannungsnachricht), eine für die Stromnachricht (Backward-Stromnachricht), eine weitere für die Kontrollnachricht (Backward-Kontrollnachricht) und eine für eine *self message* (Timer).

3.4 Leistungsknoten (*Node*)

In diesem Abschnitt wird auf die Implementierung der Leistungsknoten eingegangen. Die wesentlichen Methoden des Moduls werden erläutert und die Reaktion des implementierten Zustandsautomaten mittels Zustands- und Ablaufdiagrammen dargestellt. Das Modul leitet sich, wie alle vom Anwender des Frameworks definierten Knotentypen, vom Modul *cSimpleModule* ab (siehe auch Abbildung 26). Jeder Leistungsknoten im elektrischen Verteilnetz wird als Instanz dieser Klasse definiert und implementiert damit die von dieser Klasse definierten Eigenschaften und Methoden. Die Simulation wird von einem bestimmten Knoten dieses Typs gestartet. Jede Instanz generiert Spannungs- (*VoltageMsg*), Strom- (*CurrentMsg*) und Kontrollnachrichten (*ControlMsg*). Verarbeitet werden Nachrichten des Typs Spannungs- und Kontrollnachricht sowie die vom Knoten an sich selbst versendeten *self messages*. Berechnet werden die Ströme für die Vorgängerknoten, im Zuge des Rückwärtsschrittes des Verfahrens, und mittels Nachrichtentransport übermittelt. Weiters wird

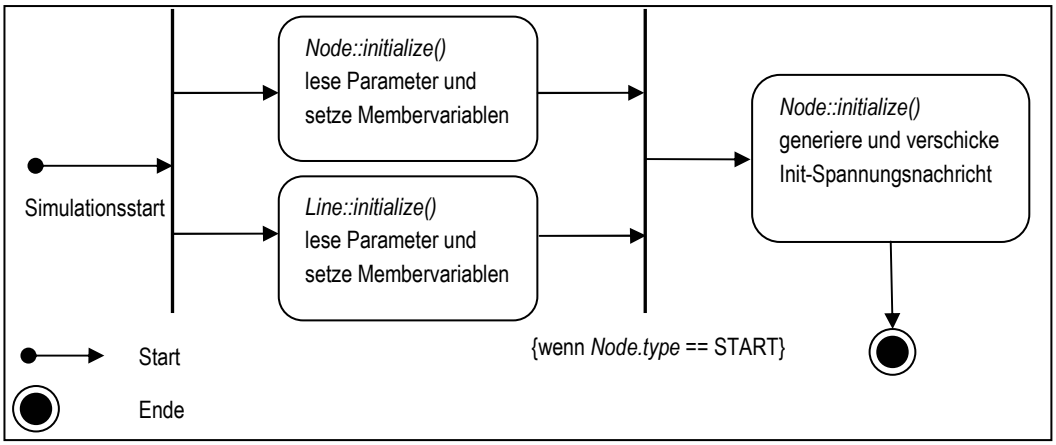


Abbildung 35 – Ablauf des Simulationsstarts

an bestimmten Knoten die Differenzbedingung für einen etwaigen Neustart eines Berechnungsdurchlaufes ausgewertet. Aufgrund der Strukturinformation des Netzes in der NED-Datei (siehe

Abbildung 32), besitzt jede Instanz der Klasse und damit jeder Leistungsknoten im betreffenden Netz, die Parameter *power*, *type* und *distance*. Der Parameter *type* beschreibt dabei den Typ des Knotens, also ob es sich um den Startknoten (*type* = START) oder um einen inneren (*type* = INNER) bzw. Endknoten (*type* = TERMINAL) handelt. Nach der Initialisierung der Anwendung durch das Framework wird der Simulationslauf vom Startknoten gestartet, in dem eine Init-Spannungsnachricht generiert und verschickt wird. Damit wird die gesamte Simulation in Gang gesetzt. Abbildung 35 zeigt diesen Sachverhalt in Form eines Aktivitätsdiagramms. In der Abbildung ist erkennbar, dass nach dem Start die Methode *initialize()* jeder Modultyp-Instanz, also von allen *Node*- und *Line*-Instanzen aufgerufen werden. Diese Methode wird nur einmal bei Start der Simulation aufgerufen und dient dem Anstoßen der Simulation in dem zumindest eine Nachricht generiert und verschickt wird (Eintrag im Finite Event Set). Nachdem im Verfahren der Ladder Iterative Technique ohnehin eine Initialisierung aller Leistungsknoten mit der gegebenen Spannung am Slack-Knoten (Startknoten) notwendig ist, ist die Implementierung hierfür in der *initialize()*-Methode des Leistungsknotens mit Start-Eigenschaft am Besten geeignet. Abbildung 36 zeigt die Klasse und alle ihre Membervariablen und Methoden in UML-Darstellung.

3.4.1 Gesamtablauf der Berechnung in der Klasse *Node*

Im Folgenden wird die Methode *handleMessage(cMessage *msg)* beschrieben. Diese Methode ist neben der Methode *initialize()* die zentrale Methode aller Anwendungsklassen respektive Modulimplementierungen. Nachdem die *initialize()*-Methode Startknotens (*type* = START) ausgeführt wurde, wird für den restlichen Lauf einer gesamten Berechnung des Algorithmus nur mehr die Methode *handleMessage(cMessage *msg)* von der Simulation aufgerufen. In der folgenden Abbildung 37 wird in Form eines Zustandsdiagrammes die Abfolge der Zustände und Übergänge des Leistungsknotens *Node* dargestellt. Wie in der Abbildung ersichtlich, wird wie schon erwähnt in allen Knotentypen die Methode *initialize()* aufgerufen, egal ob von dieser eine bestimmte Funktionalität implementiert wird oder nicht. Die Methode *initialize()* wird verwendet, um Variablen zu initialisieren und die Referenzen auf Hilfsklassen herzustellen und wie schon erwähnt im Fall eines Startknotens den Simulationslauf mittels einer Init-Spannungsnachricht anzustoßen. Danach wird die Methode *handleMessage(cMessage *msg)* solange von der Simulationsumgebung aufgerufen bis das Finite Event Set (FES) leer ist, also keine Nachrichten mehr in der Simulation auf eine Verarbeitung warten bzw. im Netzwerk herumgeschickt werden müssen. Bevor die Simulation beendet wird und das Objekt zerstört wird, wird noch die Methode *finish()* aufgerufen. In dieser Methode werden noch abschließende Protokolle bzw. Aufzeichnungen über Knotenwerte durchgeführt. Der Ablauf innerhalb der Methode *handleMessage(cMessage *msg)* kann wie folgt beschrieben werden. Zuerst wird versucht, einen Timer zu starten. Dies passiert mittels des Aufrufs Methode *startTimer()*. Folgend werden die verschiedenen Möglichkeiten von Nachrichten abgearbeitet. Dies startet mit der Prüfung auf eine *self message* – die vom Timer gegebenenfalls im letzten Schritt generiert wurde und zum aktuellen Zeitpunkt eintrifft –, danach wird auf die Typen Kontrollnachricht (*ControlMsg*), Spannungsnachricht (*VoltageMsg*) hin überprüft. Ist das kontinuierliche Aufzeichnen von Knotenwerten eingeschaltet, so wird die Methode *writeVoltage(…)* aufgerufen. Der Methode werden der komplexe Strom und die komplexe Spannung, die Entfernung zum Startknoten in Meter m und die aktuelle

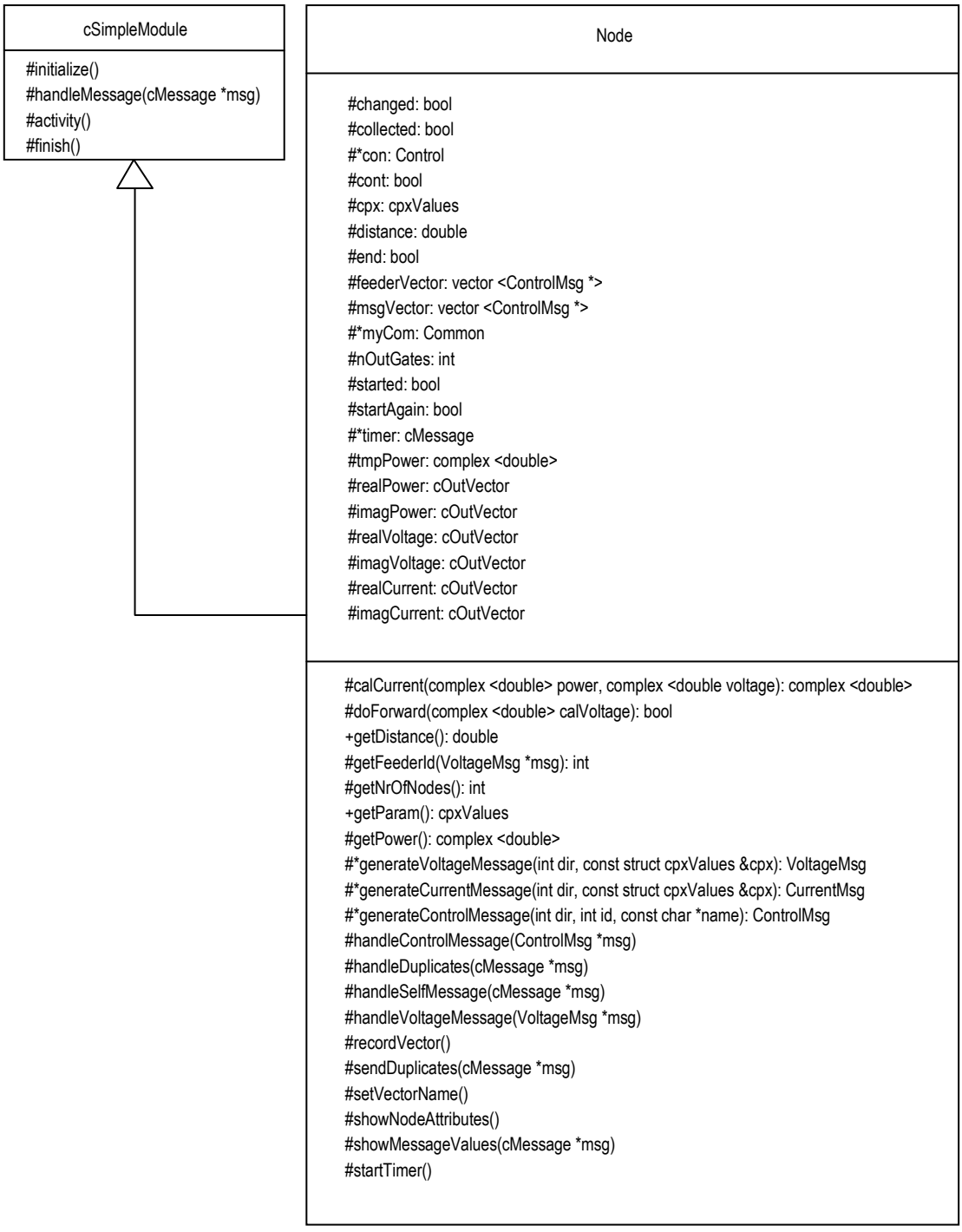


Abbildung 36 – Klassenentwurf des Typs *Node*

Lauf-ID übergeben. Die Lauf-ID beschreibt die Anzahl der bisher vollständig durchgeführten Berechnungsläufe also Simulationsschritte. Zum Abschluss werden noch die Knotenparameter im Graphical User Interface (GUI) von OMNeT++ [3] angezeigt. Nachfolgend soll mit Hilfe der Abbildung 38 die Abfolge der Aktionen dargestellt werden. Die Aufrufe und Abfolge dieser Operationen wird innerhalb der Methode *handleMessage(cMessage *msg)* vollzogen.

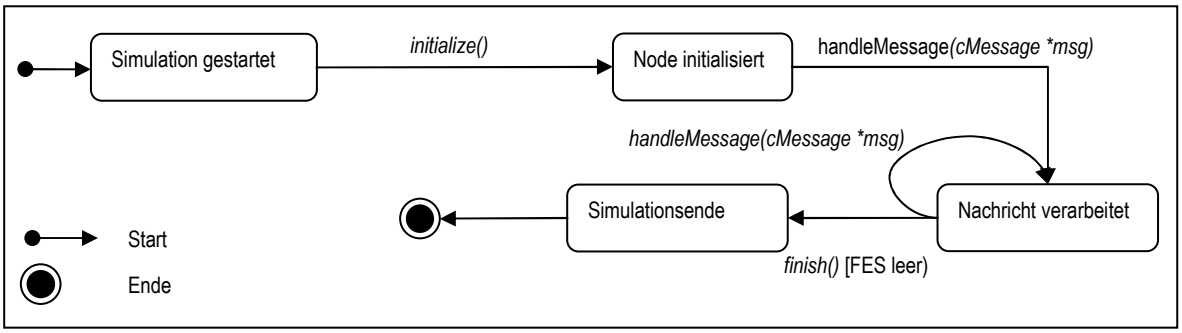


Abbildung 37 – Zustandsautomat des *Node*-Knotens

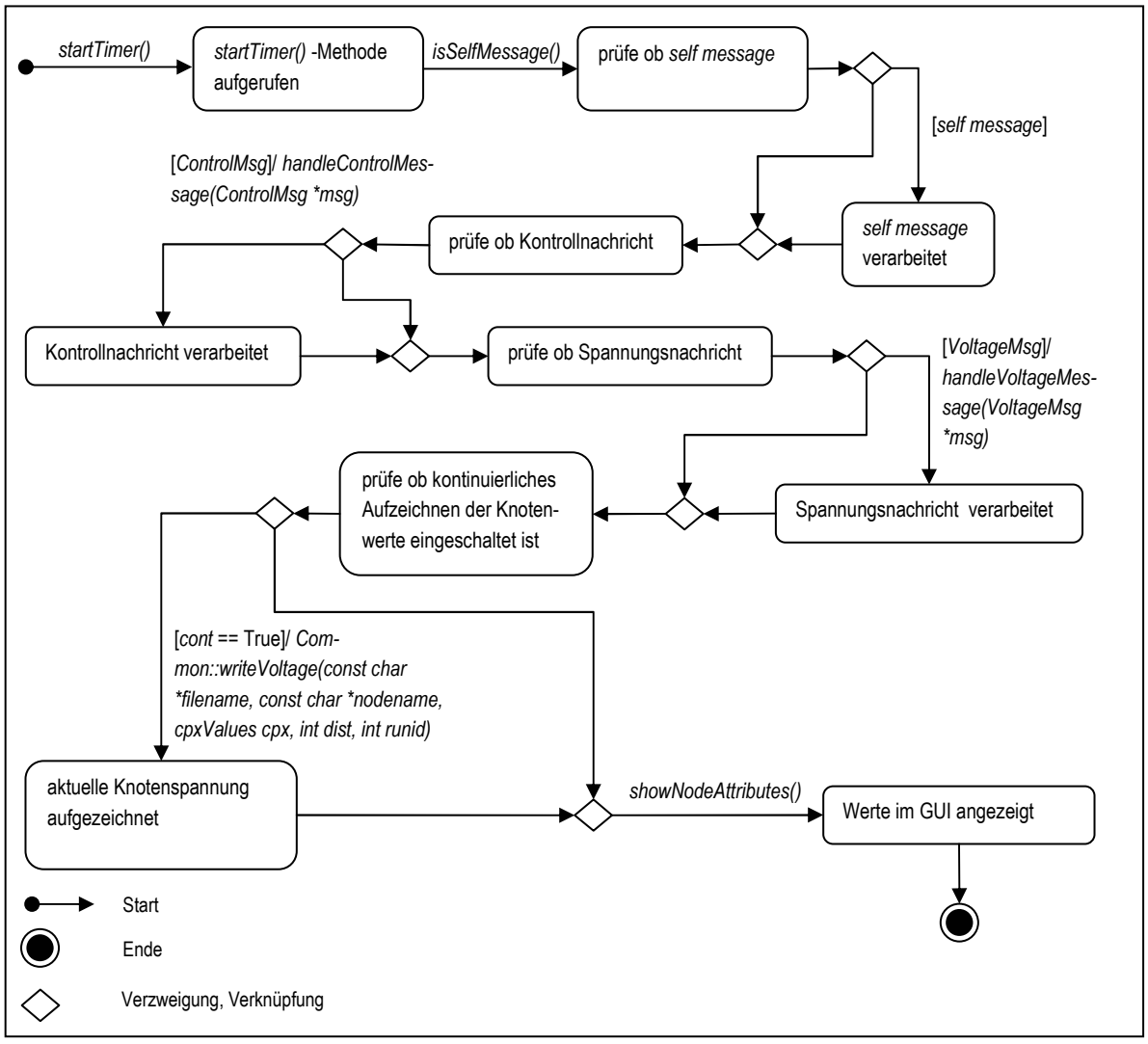


Abbildung 38 – Zustandsautomat des Gesamtablaufs für *Node*-Knoten

Wie in der Abbildung ersichtlich, werden die betreffenden Methoden und Variablen der Klasse *Node* in der Methode *handleMessage(cMessage *msg)* sequentiell nacheinander, je nach Typ der zum Zeitpunkt *t* einlangenden Nachricht überprüft und gegebenenfalls aufgerufen. Je nach Methode ver-

ändert sich dabei auch der Zustand des jeweiligen Knotens. Jeder Leistungsknoten im Netz stellt eine Instanz dieser Klasse dar, die je nach Lage im Netz und je nach Typ der der Nachricht zum Zeitpunkt *t* einen bestimmten Zustand einnimmt und damit unterschiedlich auf die jeweilige Nachricht reagiert. Die Abfolge der Aktionen und Zustände in Abbildung 38 wird für jede eingehende Nachricht an einem Leistungsknoten vom Framework abgearbeitet. Zusammenfassend lassen sich also die Aktionen und Aufgaben des Leistungsknotentyps *Node* im elektrischen Verteilnetz wie folgt beschreiben:

- Timer starten um Leistungsänderung zu initiieren
- *self message* (vom Timer generiert) verarbeiten und Wert zufällig ändern
- Kontrollnachricht (*ControlMsg*) verarbeiten um Wissen über Änderung der Leistung eines Knotens im Netz zu erhalten (nur am Startknoten)
- Spannungsnachricht (*VoltageMsg*) verarbeiten, gilt für alle Knotentypen und alle Richtungstypen von Nachrichten

3.4.2 Detailablauf der Berechnung in der Klasse *Node*

Im folgenden Abschnitt wird auf die Details der Bearbeitung innerhalb der im Abschnitt 3.4.1 beschriebenen Schritte der zentralen Methoden *initialize()* und *handleMessage(cMessage *msg)* der Klasse *Node* näher eingegangen.

Um den Knoten zu initialisieren wird wie in Abbildung 37 dargestellt die Methode *initialize()* aufgerufen. Im Fall des *Node*-Knotens werden hier alle Variablen und eventuelle Zeiger initialisiert. Weiters werden die Parameter für die Leistung und die Quellenspannung des Slack-Knotens aus der INI-Datei gelesen und am Knoten gesetzt. Daraus folgt, dass jeder Knoten nach dem Start der Simulation einen bestimmten Leistungswert anhand der Definition in der INI-Datei zugewiesen bekommt und für den Startknoten weiters die Quellenspannung und der Schwellwert der für die Abbruchbedingung des Verfahren wesentlich ist, definiert ist. Nach der Initialisierung wird von der *Node*-Instanz auf eingehende Nachrichten gewartet.

Als erster Schritt in der Behandlung der eingehenden Nachrichten wird versucht einen Timer zu starten. Der Timer soll zufälligerweise die Generierung einer *self message* bewirken. Damit wird die zufällige Änderung der Leistung eines *Node*-Knotens während des Ablauf der Berechnung (Simulation) nachgebildet. Der Timer wird gestartet, wenn eine aus einer diskreten Gleichverteilung gezogene Zufallszahl im Bereich zwischen 1 und der Gesamtanzahl *n* der im Netz verfügbaren Knoten, der Knoten-ID des betreffenden Knoten entspricht. Weiters wird überprüft, ob dieser Knoten bereits eine *self message* generiert hat, und wenn dies der Fall ist, der Timer nicht ausgeführt. Damit wird ein mehrfaches Starten des Timers vermieden. Kann der Timer gestartet werden, so wird eine *self message* geplant und bei Erreichen der Zeitbedingung versendet. In der Abbildung 39 wird der Ablauf der Aktionen und Bedingungen mit Hilfe eines Aktivitätsdiagramms erläutert.

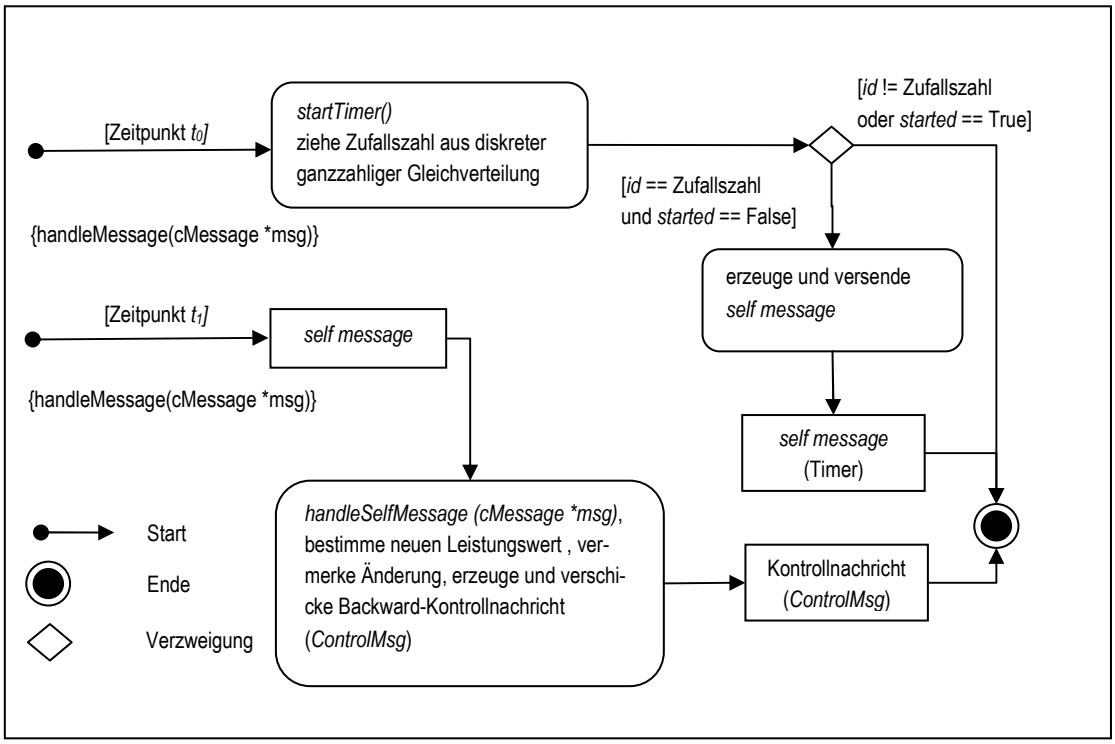


Abbildung 39 – Verarbeitung und Erzeugung einer *self message* (Timer)

Wie in der Abbildung zu erkennen ist, werden sowohl der Aufruf der Methode *startTimer()* sowie die Behandlung der etwaigen *self message* von der zentralen Methode *handleMessage(cMessage *msg),* die in jeder Klasse respektive in jedem Modul das vom Anwender implementiert wird enthalten ist, aufgerufen. Das bedeutet, dass die Methode *startTimer()* jedesmal beim Durchlaufen der Methode *handleMessage(cMessage *msg)* aufgerufen wird.

Als zweiter Schritt wird die Behandlung der, im gegebenenfalls letzten Schritt (Zeitpunkt t_0), erzeugten *self message* durchgeführt (siehe Abbildung 39). Dazu wird die Methode *handleSelfMessage(…)* aufgerufen. Zuerst ein zufälliger neuer Leistungswert generiert und vermerkt, dass sich der Wert des Knotens geändert hat. Der neue Wert für die daraus notwendige Neuberechnung, wird bis zum Neustart des Gesamtverfahrens zwischengespeichert. Weiters wird eine Backward-Kontrollnachricht (*ControlMsg*) erzeugt und in Richtung des Startknotens versendet.

Als dritter Schritt der Nachrichtenbehandlung am *Node*-Knotentyp wird eine eventuell zu behandelnde Kontrollnachricht (*ControlMsg*) verarbeitet. Für die Bearbeitung einer Kontrollnachricht an *Node*-Knoten sind zwei Fälle zu unterscheiden. Handelt sich um einen inneren Knoten, dann wird die Kontrollnachricht einfach in Richtung des Startknotens, also auf das *gate* mit Index = 0 (siehe Abbildung 33), versendet bzw. weitergeleitet. Für den zweiten Fall des Startknotens, wird an diesem vermerkt, dass sich zumindest ein Leistungswert im Netz geändert hat. Weiters wird die Kontrollnachricht in einer entsprechenden Datenstruktur gespeichert, da sonst die Simulation aufgrund eines leeren FES beendet wird. Die folgende Abbildung 40 zeigt den Zusammenhang des Ablaufes mit Hilfe eines Aktivitätsdiagramms.

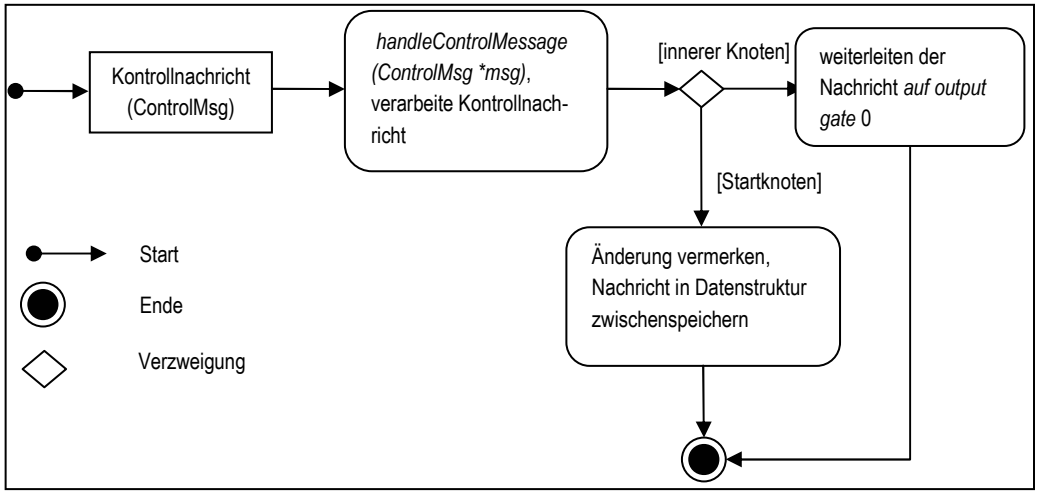


Abbildung 40 – Verarbeitung einer Kontrollnachricht

Im vierten Schritt der Nachrichtenbehandlung werden die unterschiedlichen Typen von Spannungsnachrichten (*VoltageMsg*) unter Berücksichtigung des Knotentyps (Parameter *type*), also ob es sich um den Startknoten, inneren bzw. Verteilknoten oder um einen Endknoten handelt, behandelt.

Im Fall des Startknotens des Netzes muss nur die Behandlung einer Backward-Spannungsnachricht berücksichtigt werden. Zu diesem Zweck wird am Startknoten die Spannung aus der Nachricht ausgelesen und am Knoten gesetzt. Folgend wird die Differenz zwischen dem in der Nachricht übermittelten und neu berechneten und gesetzten Spannungswert und der am Startknoten gegebenen Spannung, die durch den Parameter *srcvoltage* bestimmt worden ist ist, berechnet. Diese Differenz wird mit dem Schwellwert, der durch den Parameter *threshold* bestimmt worden ist ist, verglichen.

Ist die Differenz größer dem gegebenen Schwellwert, ist also eine Neuberechnung notwendig, so wird die Spannung am Knoten auf den Wert des Parameters *srcvoltage* zurückgesetzt. Danach wird der Wert in der Spannungsnachricht neu gesetzt und Richtung der Nachricht geändert, so dass daraus eine Forward-Spannungsnachricht entsteht. Die Forward-Spannungsnachricht wird dann an den Anschluss (*gate*) mit dem Index des Anschlusses von dem die Nachricht eingelangt ist, verschickt. Damit wird der am Startknoten betreffende Ast neu berechnet, also der Vorwärtsschritt der Ladder Iterative Technique für diesen Ast durchgeführt.

Ist keine Neuberechnung notwendig, also die Differenz der Spannungswerte kleiner dem Schwellwert, so wird folgend je nach Parametrisierung in der INI-Datei, der aktuelle Wert der Spannung in eine Datei weggeschrieben. Weiters kann in der INI-Datei festgelegt werden, ob am Ende einer abgeschlossenen Simulation die Endergebnisse für alle Knoten in einer gesonderten Datei erfasst werden sollen. Ist der entsprechende Parameter gesetzt, so wird der im Vorfeld erfasste Index des Anschlusses (*gates*) auf dem die Nachricht eingelangt ist, in einer Datenstruktur eingetragen. Sofern die Anzahl der Elemente in dieser Struktur mit der Anzahl der *output gates* die in Richtung der Endknoten des Netzes zeigen – Startknoten befindet sich bereits am Anfang des Netzes – übereinstimmen (alle Äste abgearbeitet), kann mit der Erfassung der Resultate für alle Knoten begonnen werden. Die Aufzeichnung übernimmt die Klasse *Common*, auf die später noch eingegangen wird. Der entsprechenden Methode *collectData(…)* der Klasse wird unter anderem die Lauf-ID der Simulation über-

geben. Die Lauf-ID bezeichnet einen ganzzahligen numerischen Wert der die Anzahl der totalen bisher durchgeführten Simulationen beschreibt. Das heißt, dass für ein Netz mit mehreren Ästen, die

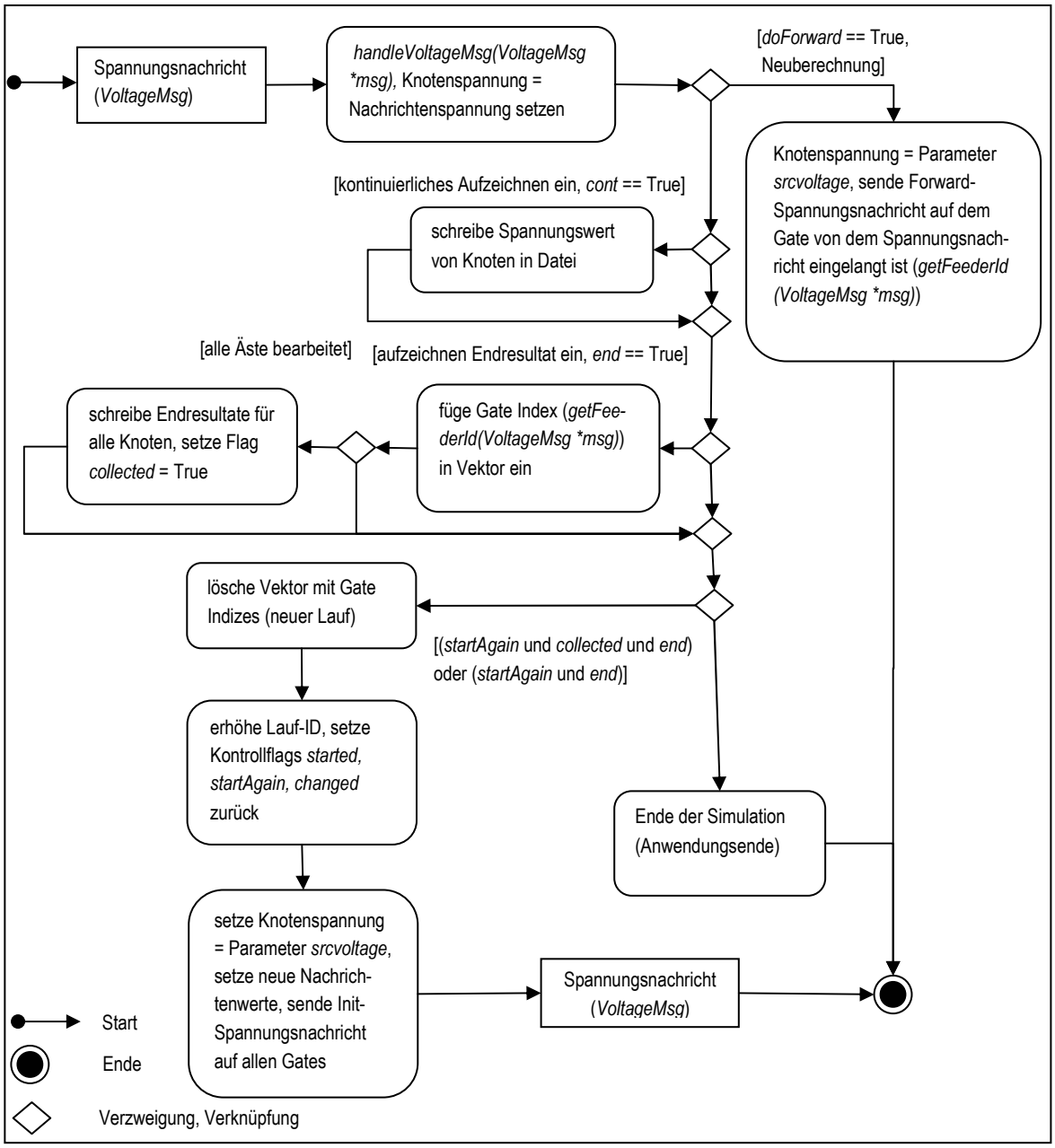


Abbildung 41 – Verarbeitung einer Backward-Spannungsnachricht am Startknoten

an den Startknoten (Slack-Knoten) angeschlossen sind, einerseits mehrere Berechnungs- respektive Simulationsschritte (Vorwärts- bzw. Rückwärtsschritte der Ladder Iterative Technique) für jeden Ast notwendig sein können, andererseits kann es wie schon erwähnt in den einzelnen Knoten jedes Astes zu einer Änderung der Leistungsparameter kommen (Monte-Carlo-Simulation). Daraus folgt, dass nach einer Berechnung aller Äste und damit Berechnung aller Werte des Netzes, eine Neuberechnung des gesamten Netzes mit den geänderten Parametern notwendig ist, was durch die Lauf-ID

gekennzeichnet wird. Danach wird überprüft ob die Simulation neu gestartet werden soll (Leistungsänderungen sind aufgetreten), die Daten geschrieben wurden und die Protokollierung für die Enderfassung eingeschaltet ist oder nur neu gestartet werden soll und die Protokollierung ausgeschaltet ist. Trifft beides nicht zu, kann die Simulation beendet werden.

Andernfalls wird im Fall einer eingeschalteten Endprotokollierung die Datenstruktur mit den Indizes der Anschlüsse (*gates*) – Indizes die die Äste am Startknoten markieren – und ein Flag das das erfolgreiche Schreiben anzeigt, gelöscht. Im Anschluss daran wird die Lauf-ID erhöht und das Flag für den Neustart zurückgesetzt.

Abschließend wird der Wert für die Spannung auf den Wert des Parameters *srcvoltage* zurückgesetzt und eine Init-Spannungsnachricht generiert und versendet. Das bedeutet, dass ein völlig neuer Simulationslauf mit der Abfolge der Aktionen Initialisierung, Rückwärts- und Vorwärtsschritt beginnen kann. Die Nachricht wird an alle angeschlossenen Äste des Startknotens versendet. Abbildung 41 zeigt die Abfolge der Aktionen bei der Behandlung einer Backward-Spannungsnachricht aus der Sicht des Startknotens anhand eines Aktivitätsdiagramms.

Im Fall der Nachrichtenverarbeitung an einem inneren Knoten ist zu unterscheiden ob es sich um einen einfachen inneren Knoten (nur ein Nachfolger) oder um einen Verteilknoten (mit mehreren angeschlossenen Leitungen) handelt. Weiters müssen an inneren Knoten Init-, Forward- und Backward-Spannungsnachrichten berücksichtigt werden.

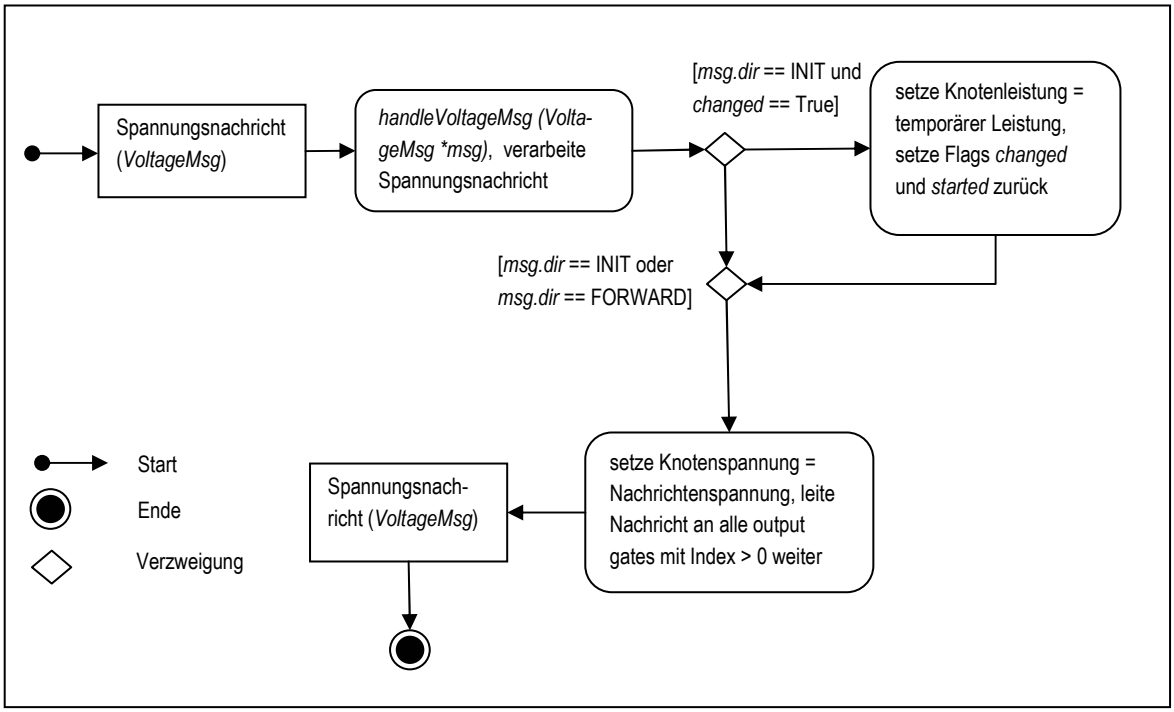


Abbildung 42 – Init- und Forward-Spannungsnachricht am inneren Knoten

Im Fall einer Init-Spannungsnachricht werden an einem einfachen inneren Knoten die folgenden Aktionen durchgeführt (siehe Abbildung 42). Zuerst wird festgestellt, ob sich während der bisherigen Abarbeitung des Algorithmus der Leistungswert des Knotens geändert hat. Hat sich der Wert

geändert, so wird der Leistungswert auf den in der Zwischenzeit zwischengespeicherten temporären Wert gesetzt, und damit die Leistungsänderung am Knoten vollzogen. Weiters werden die entsprechenden Flags zurückgesetzt. Darauf folgend wird sowohl im Fall einer Init- als auch einer Forward-Nachricht der neue Spannungswert aus der Nachricht gelesen und am Knoten gesetzt und die Nachricht an den bzw. die nachfolgenden Leitungsknoten versendet.

Im Fall einer Backward-Nachricht werden an einem einfachen inneren Knoten die folgenden Aktionen durchgeführt. Zuerst wird die Spannung am Knoten auf die Spannung in der Nachricht gesetzt, und der Strom basierend auf der Leistung des Knotens und der erhaltenen Spannung berechnet. Weiters wird zu diesem berechneten Strom, der Strom vom Nachfolgerknoten, der von diesem in der Nachricht ebenso übermittelt wurde, hinzuaddiert. Damit wird die Knotenstromregel erfüllt. Im Anschluss daran wird die erhaltene Spannungsnachricht (*VoltageMsg*) zerstört. Eine Stromnachricht (*CurrentMsg*) wird erzeugt und die aus der Verarbeitung der Spannungsnachricht übernommenen Werte (Spannungen) bzw. basierend auf diesen Werten neu berechneten Werten (Ströme) in eine Stromnachricht (*CurrentMsg*) verpackt und versendet. Die folgenden Aktivitätsdiagramme in Abbildung 43 zeigt die Behandlung einer Backward-Spannungsnachricht für einen einfachen inneren Knoten.

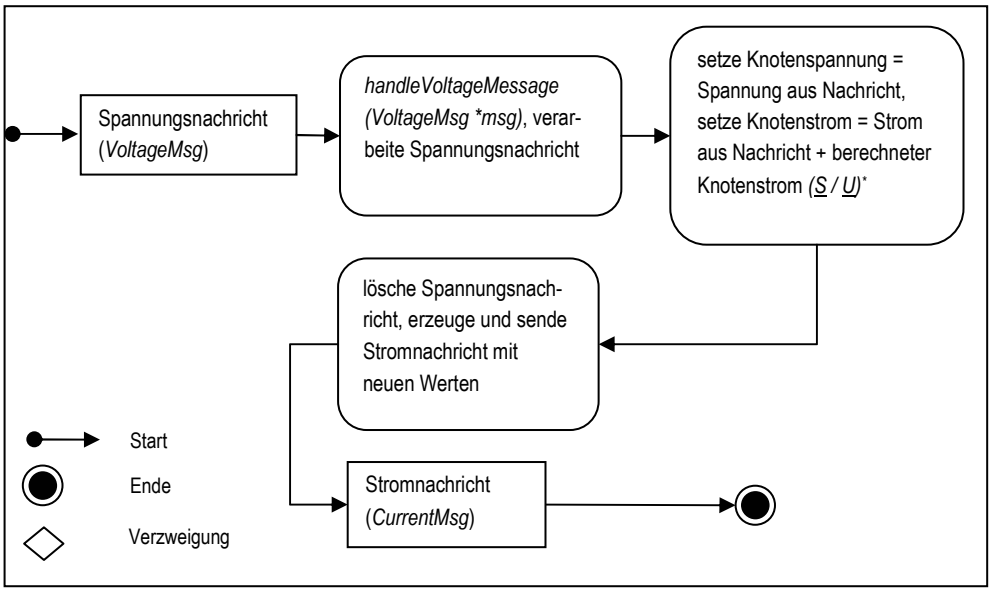


Abbildung 43 – Backward-Spannungsnachricht am inneren Knoten

Im Fall der Verarbeitung einer Spannungsnachricht an einem Verteilknoten müssen mehrere Dinge berücksichtigt werden. Trifft eine Init- bzw. Forward-Spannungsnachricht an einem Verteilknoten ein, so muss zusätzlich zu den Aktionen die bei einem einfachen inneren Knoten durchgeführt werden, die Nachricht an alle angeschlossenen Leitungen respektive nachfolgenden *Node*-Knoten verschickt werden. Das bedeutet, dass die Nachricht *n-1* mal dupliziert werden muss um an *n* Anschlüsse, die in Richtung der Nachfolgeknoten zeigen – also Indizes der *gates* größer 1 aufweisen – , versendet werden zu können. Im Fall der Forward- bzw. Backward-Spannungsnachrichten müssen die jeweiligen Spannungen am verarbeitenden Knoten bzw. die Spannung für den Vorgängerknoten und den Strom am verarbeitenden Knoten gemäß der Verarbeitung an einem inneren Knoten (Kirchhoff-

sche Gesetze) berechnet werden. Spezielle Beachtung verdient dabei, die Verarbeitung der Backward-Spannungsnachrichten, da hier aufgrund der Kirchhoffschen Knotenstromregel die Ströme aus den umliegenden Knoten aufsummiert werden müssen und eine gültige Spannung für die Berechnung des Spannungsfalls der auf der Vorgängerleitung auftritt, berechnet werden muss. In Abbildung 44 wird ein Ausschnitt eines elektrischen Verteilnetzes mit einem Verteilknoten und drei daran angeschlossenen Leitungen und Knoten und das dazu äquivalente Modell mit den Modulen und den Anschlüssen dargestellt. In der Abbildung ist ersichtlich, dass beim Rückwärtsschritt des Verfahrens, also der Berechnung der Ströme in den Knoten, in diesem Fall auf die Abarbeitung aller angeschlossenen drei Leitungen gewartet werden muss um den korrekten Strom im Verteilknoten $node_i$ (Knotentyp-Instanz von *Node*) zu berechnen.

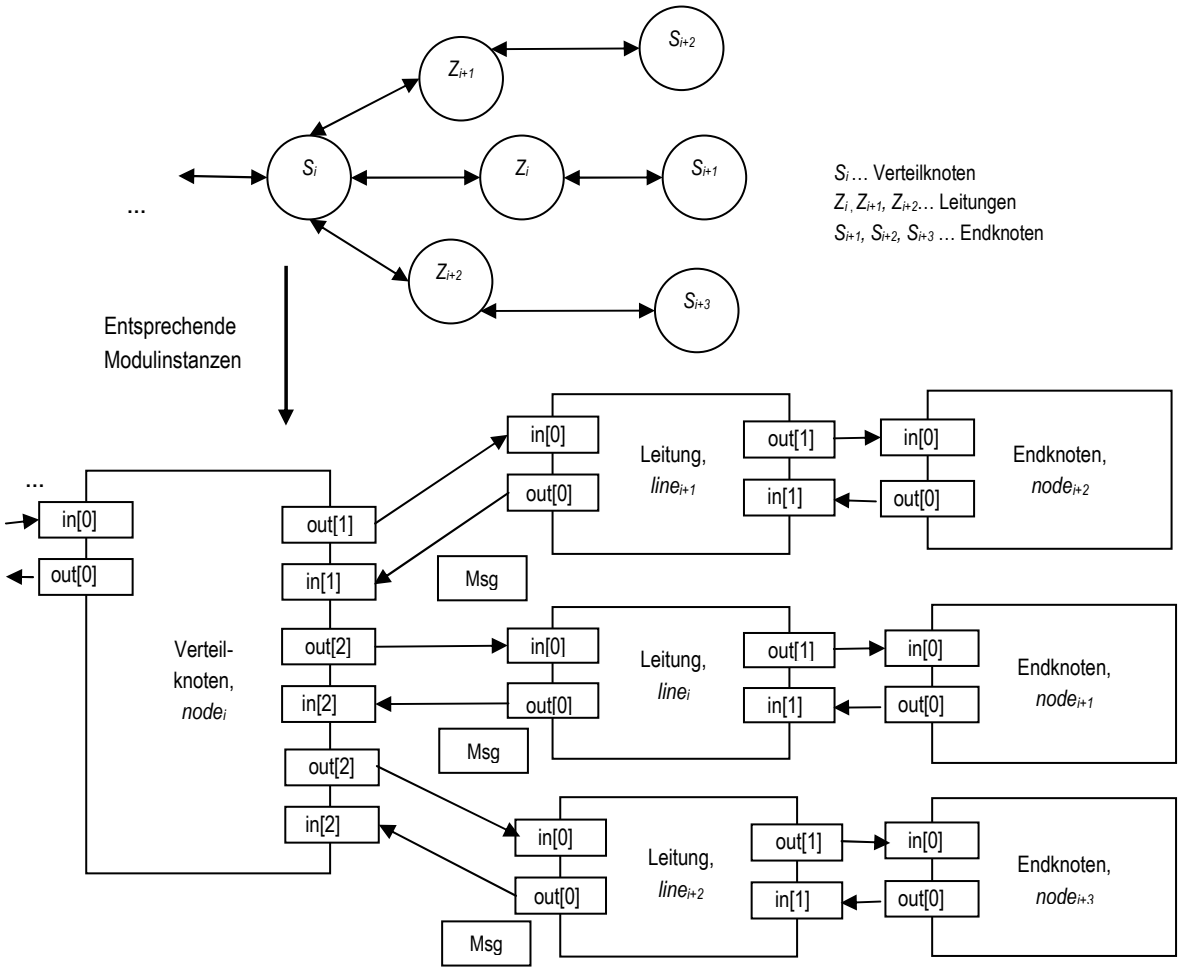


Abbildung 44 – Anschlüsse am Verteilknoten

Wie auch schon in Abbildung 6 gezeigt wird der Strom eines Knotens durch die gegebene Leistung und der anliegenden Spannung berechnet. Allerdings sind bei der Rückrechnung auch die Ströme des Nachfolgeknotens zu berücksichtigen. Die in den Nachfolgerknoten berechneten Ströme werden dort in einer Stromnachricht verpackt und an den davor liegenden Leitungsknoten verschickt und von dort an den davor liegenden aktuell zu betrachtenden Knoten weitergeleitet. Der Knoten emp-

fängt nun die Nachricht und kann den Stromwert auslesen und zu dem am Knoten durch die Leistung und Spannung gegebenen Strom am Knoten dazuaddieren. Die empfangene Spannungsnachricht kann danach gelöscht werden. Damit wird die Kirchhoffsche Knotenstromregel erfüllt. Für den Fall des Verteilknotens $node_i$ aus Abbildung 44 bedeutet dies, dass alle Ströme von den Knoten $node_{i+1}$, $node_{i+2}$ und $node_{i+3}$ zum berechneten Strom von dem Verteilknoten $node_i$ addiert werden müssen. Der Strom ergibt sich damit zu $\underline{I}_i = (\underline{S}_i / \underline{U}_i)^* + \underline{I}_{i+1} + \underline{I}_{i+2} + \underline{I}_{i+3}$. Um dies bewerkstelligen zu können, ist es notwendig, alle drei Nachrichten die von den Leitungen $line_i$, $line_{i+1}$ und $line_{i+2}$ kommen, zu verarbeiten. Nachdem die Leitungsabschnitte nicht wie in Abbildung 44 gezeigt, auch unterschiedlich lang sein können, ist es wie schon erwähnt notwendig zu warten bis der Nachrichtenrücktransport aus allen an den Verteilknoten angeschlossenen Teilabschnitten erledigt ist, um den Gesamtstrom am Verteilknoten berechnen zu können. Nach Abarbeitung aller Nachrichten ist weiters auch notwendig, die Spannung $\underline{U}_i$ am Verteilknoten $node_i$ zu bestimmen. Nachdem im Rückwärtsschritt nicht davon ausgegangen werden kann, dass die Impedanzen der an den Verteilknoten angeschlossenen Leitungen und auch die Ströme an den unmittelbaren Nachfolgeknoten nicht gleich sind – diese werden aufgrund der an den Knoten gegebenen anliegenden Spannung und der gegebenen Leistung berechnet – , ist mit unterschiedlichen Spannungswerten auf den Leitungen zu rechnen. Deshalb ist es notwendig einen bestimmten Spannungswert für die Berechnung des Stromes am Verteilknoten heranzuziehen. Gemäß [Ker07] ist dabei der Spannungswert des letzten berechneten Astes heranzuziehen. In der für diese Arbeit herangezogenen Implementierung wurde dies

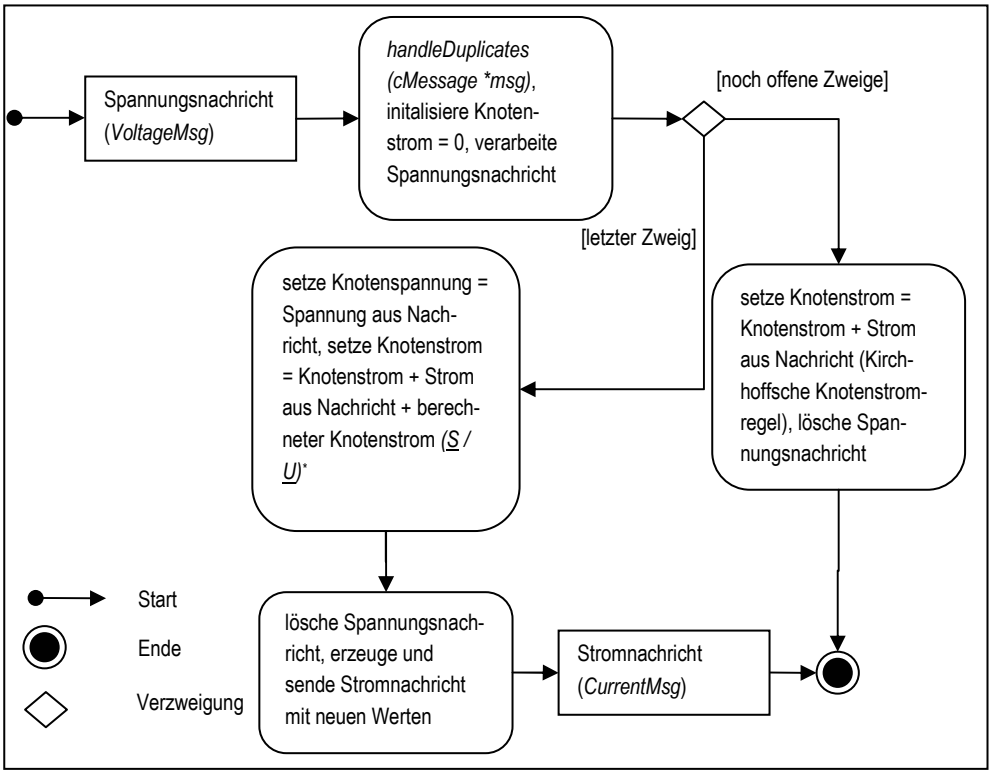


Abbildung 45 – Backward-Spannungsnachricht am Verteilknoten

gemäß dieser Empfehlung gelöst (siehe auch Abschnitt 1.3.4). Ist der Strom und die Spannung am

Verteilknoten mittels der beschriebenen Methode berechnet, wird die empfangene Spannungsnachricht gelöscht und anstatt dessen eine Stromnachricht erstellt, in die die soeben neu berechneten und gelesenen Werte eingetragen werden. Diese Stromnachricht (*CurrentMsg*) wird dann an das *output gate* 0, also in Richtung der davorliegenden Leitung bzw. Knoten, versendet. In Abbildung 45 wird anhand eines Aktivitätsdiagramms der Ablauf an einem Verteilknoten in Bezug auf die Nachrichtenverarbeitung einer Backward-Spannungsnachricht dargestellt.

Für den Fall der Verarbeitung einer Spannungsnachricht an einem Endknotens müssen ebenso die Nachrichtentypen Init-Nachricht und Forward-Nachricht betrachtet werden. Im Fall einer Init-Nachricht wird auch hier überprüft ob sich der Leistungswert des Knotens geändert hat und dieser dem in der Zwischenzeit gespeicherten temporären Wert gleichgesetzt. Weiters werden die entsprechenden Flags zurückgesetzt, wenn eine Änderung vollzogen wurde. Darauf folgend wird sowohl im Fall einer Init- als auch einer Forward-Nachricht die Knotenspannung gleich der Nachrichtenspannung gesetzt. An den Endknoten kommt es dann allerdings zur Beendigung des Vorwärtsschrittes des Verfahrens, deshalb muss nun basierend auf den nun soeben gesetzten neuen Spannungswerten, der Strom am Knoten neu berechnet werden. Die Spannungsnachricht wird daraufhin gelöscht und anstatt dessen, eine Backward-Stromnachricht (*CurrentMsg*) erzeugt und an das *output gate* 0 gesendet. Die Nachricht wird also in Richtung Anfangsknoten des Netzes gesendet. Das folgende Aktivitätsdiagramm in Abbildung 46 zeigt den Ablauf der Nachrichtenbearbeitung an den Endknoten des Netzes.

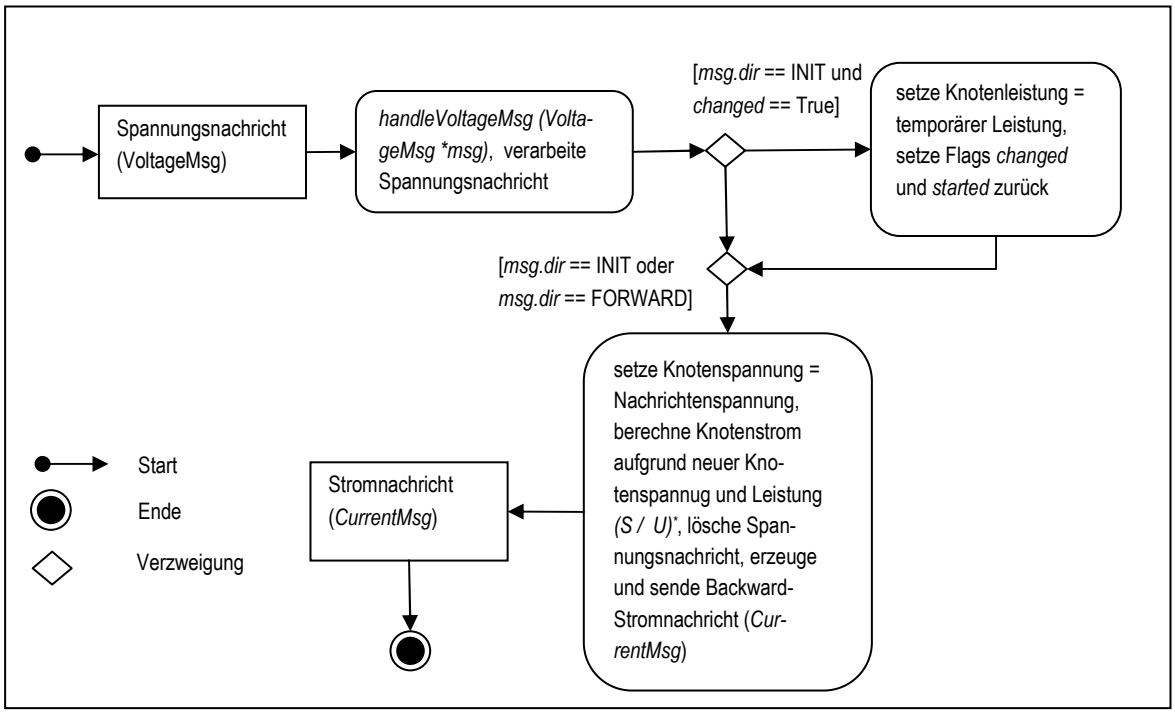


Abbildung 46 – Init- und Forward-Spannungsnachricht am Endknoten

3.5 Leitungsknoten (*Line*)

Im folgenden Abschnitt wird die Implementierung des Leitungsknotens (*Line*) erläutert. Die wesentlichen Methoden des Moduls werden erläutert und die Reaktion des implementierten Zustandsautomaten mittels Zustands- und Ablaufdiagrammen dargestellt. Das Modul leitet sich, wie alle vom Anwender des Frameworks definierten Knotentypen, ebenso vom Modul *cSimpleModule* ab. Jeder Leitungsknoten im elektrischen Verteilnetz wird als Instanz dieser Klasse definiert und implementiert damit die von dieser Klasse definierten Eigenschaften und Methoden. Jede Instanz generiert Spannungsnachrichten (*VoltageMsg*). Verarbeitet werden Nachrichten des Typs Spannungs- (*VoltageMsg*), Strom- (*CurrentMsg*) und Kontrollnachricht (*ControlMsg*). Am Leitungsknoten werden die Spannungsfälle für die vorhergehenden und nachfolgenden Leitungsknoten (*Node*) berechnet. Weiters werden in den Leitungsknoten je nach aktuell ausgeführtem Abarbeitungsschritt des Algorithmus – Rückwärts- bzw. Vorwärtsschritt – die Kirchhoffsche Maschenregel zur Berechnung der Spannung für den Nachfolge- bzw. Vorgänger-Leistungsknoten (*Node*) berechnet. Aufgrund der Strukturinformation des Netzes in der NED-Datei (siehe Abbildung 32), besitzt jede Instanz der Klasse und damit jeder Leistungsknoten im betreffenden Netz, die Parameter *impedance* und *maxcurrent.*

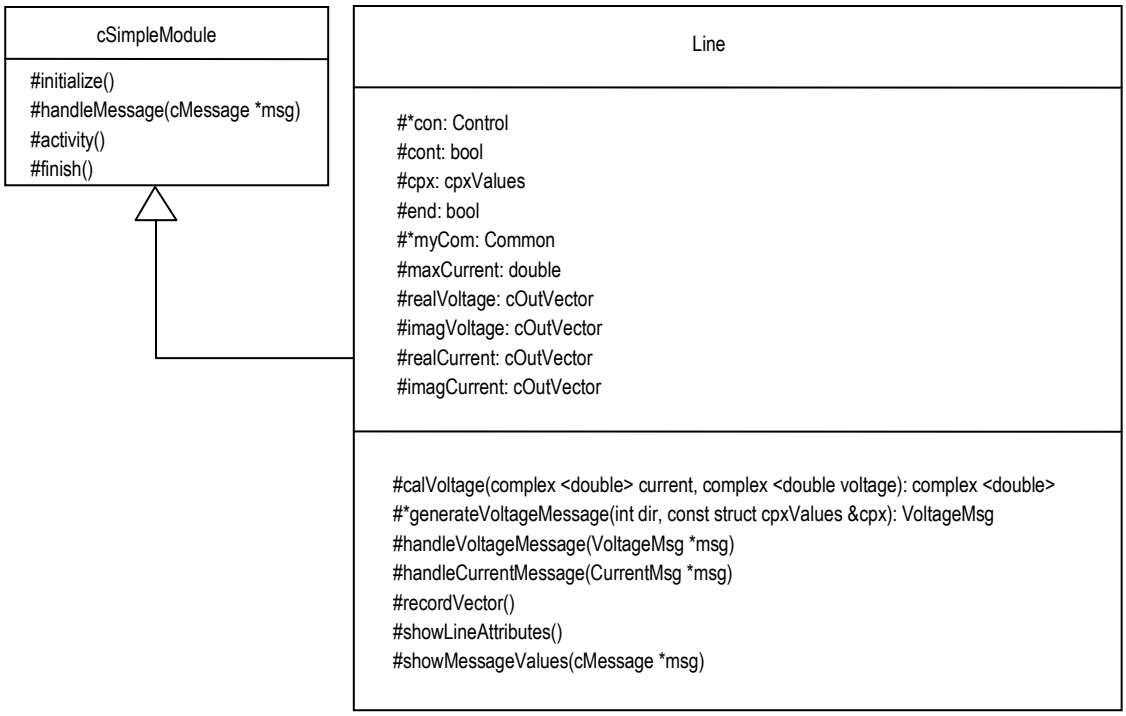


Abbildung 47 – Klassenentwurf des Typs *Line*

Der Parameter *impedance* beschreibt dabei den komplexen Widerstand $\underline{Z}$ des Leitungsknotens (*Line*) in der Einheit Ohm Ω. Der Parameter *maxcurrent* gibt den maximalen Strom in der Einheit Ampere A an, der die Leitung durchfließen darf. In Abbildung 47 werden alle Methoden und Variablen der Klasse *Line* in UML-Darstellung dargestellt.

3.5.1 Gesamtablauf der Berechnung in der Klasse *Line*

Wie auch schon in Abschnitt 3.4.1 für den Fall des Leistungsknoten *Node* erläutert, wird auch im Fall des *Line*-Knotens die Bearbeitung der eingehenden Nachrichten durch die Methode *handleMessage(cMessage *msg)* bewerkstelligt. In der *initialize()*-Methode der *Line*-Klasse wird lediglich der Parameter für die Impedanz eingelesen und am Knoten als Eigenschaft gesetzt. Damit werden allen Leitungen nach dem der Start der Simulation die komplexen Widerstandswerte, die in der INI-Datei angegeben werden, der jeweiligen Knoteninstanz zugewiesen. Im Gegensatz zum Leitungsknoten (*Node*) wird hier keine Nachricht versendet, da bereits der *Node*-Knotentyp als Bootstrap-Mechanismus – aus der Sicht des Startknotens – fungiert. Nach der Initialisierung wird mittels der Methode *handleMessage(cMessage *msg)* auf eingehende Nachrichten gewartet. Dies geschieht

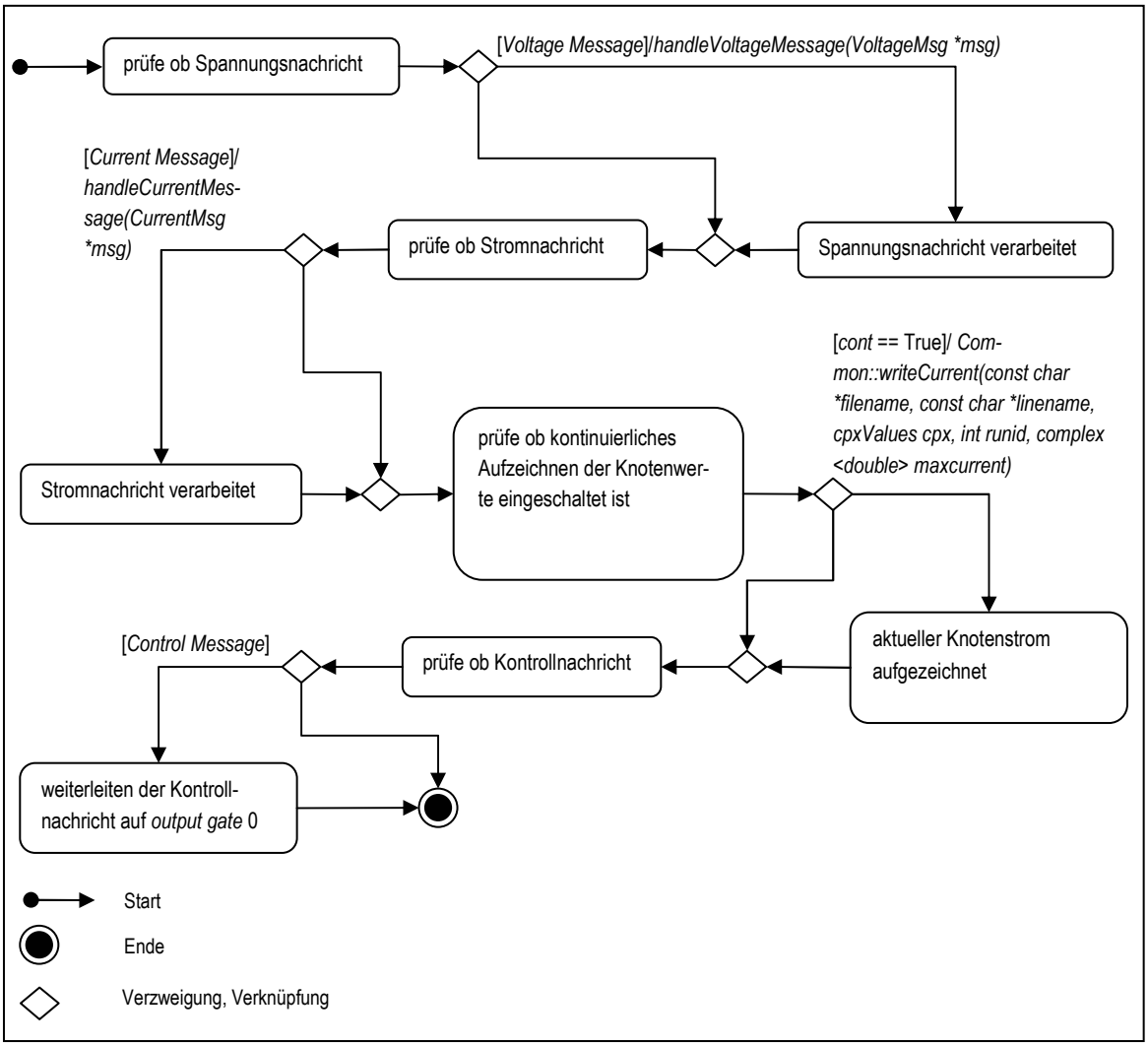


Abbildung 48 – Zustandsautomat des Gesamtablaufs für *Line*-Knoten

analog zur Abbildung 37 die den Zustandsautomat für den *Node*-Knoten beschreibt. In Abbildung 48 wird in Form eines Zustandsautomaten die Abfolge der Aktionen und Zustände innerhalb der

Methode *handleMessage(cMessage *msg)* beschrieben. Der gesamte Ablauf für den Leitungsknoten kann wie folgt kurz beschrieben werden. Es wird in sequentieller Reihenfolge überprüft ob es sich bei der ankommenden Nachricht um eine Spannungsnachricht (*VoltageMsg*), eine Stromnachricht (*CurrentMsg*) oder um eine Kontrollnachricht (*ControlMsg*) handelt. Zusammenfassend lassen sich also die Aktionen und Aufgaben des Leitungsknotentyps *Line* im elektrischen Verteilnetz wie folgt beschreiben:

- Spannungsnachricht (*VoltageMsg*) verarbeiten, Kirchhoffsche Maschenregel umsetzen, Spannungswert für Nachfolge-Leistungsknoten berechnen (Vorwärtsschritt)
- Stromnachricht (*CurrentMsg*) verarbeiten, Kirchhoffsche Maschenregel umsetzen, Spannungsfall auf Leitung berechnen
- Kontrollnachricht (*ControlMsg*) verarbeiten, Nachricht an Vorgänger-Leistungsknoten weiterleiten

3.5.2 Detailablauf der Berechnung in der Klasse *Line*

Im folgenden Abschnitt wird auf die Details der Bearbeitung innerhalb der im Abschnitt 3.5.1 beschriebenen Schritte der zentralen Methode *handleMessage(cMessage *msg)* der Klasse *Line* näher eingegangen.

Als erster Schritt der Behandlung von Nachrichten wird der Fall einer Spannungsnachricht (*VoltageMsg*) bearbeitet. Für die Klasse *Line* kommen hier Init- und Forward-Spannungsnachrichten in Frage. Ein einlangende Init-Spannungsnachricht wird einfach an den Nachfolge-Leistungsknoten weitergeleitet, da diese wie schon erwähnt nur im Initialschritt des Verfahrens, der Initialisierung der Leistungsknoten mit der vorgegebenen Spannung am Startknoten dient. Für den Fall einer Forward-Spannungsnachricht wird gemäß der Maschenregel die Spannung für den Nachfolge-Leistungsknoten im Vorwärtsschritt des Verfahrens berechnet. Das bedeutet, dass die Spannung und der Strom aus der Nachricht ausgelesen wird. Der Strom in der Nachricht stammt aus dem vorhergehenden Rückwärtsschritt des Verfahrens und wird über den Nachrichtentransport über die verschiedenen Knotentypen behalten. Danach wird der Spannungsfall auf der Leitung aus dem Strom der Nachricht und der Impedanz der Leitung berechnet. Es ergibt sich somit die Spannung für den Nachfolge-Leitungsknoten zu $\underline{U} = \underline{U}_{VoltageMsg} - (\underline{Z}_{Line}\ \underline{I}_{VoltageMsg})$. Diese Spannung wird in die Nachricht eingetragen und an den Nachfolgeknoten (*Node*) weitergeleitet. In der folgenden Abbildung 49 wird dieser Sachverhalt mit Hilfe eines Aktivitätsdiagramms dargestellt. Wie in der Abbildung zu erkennen ist, wird jede Init-Spannungsnachricht einfach weitergeleitet, somit an das *output gate* 1 der Leitung geschickt.

Als zweiter Schritt der Nachrichtenbehandlung wird eine etwaige Stromnachricht (*CurrentMsg*) bearbeitet. In diesem Fall wird ebenso wie bei einer Forward-Spannungsnachricht die Maschenregel an der Leitung, allerdings für den Vorgänger-Leitungsknoten (*Node*) umgesetzt. Aus der Stromnachricht die vom Nachfolge-Leistungsknoten des Leitungsknoten eintrifft, werden wieder die Spannungs- und Stromwerte ausgelesen. Die Spannung stammt in diesem Fall aus dem davor ausgeführten Vorwärtsschritt des Verfahrens. Danach wird wieder der Spannungsfall auf der Leitung aus dem

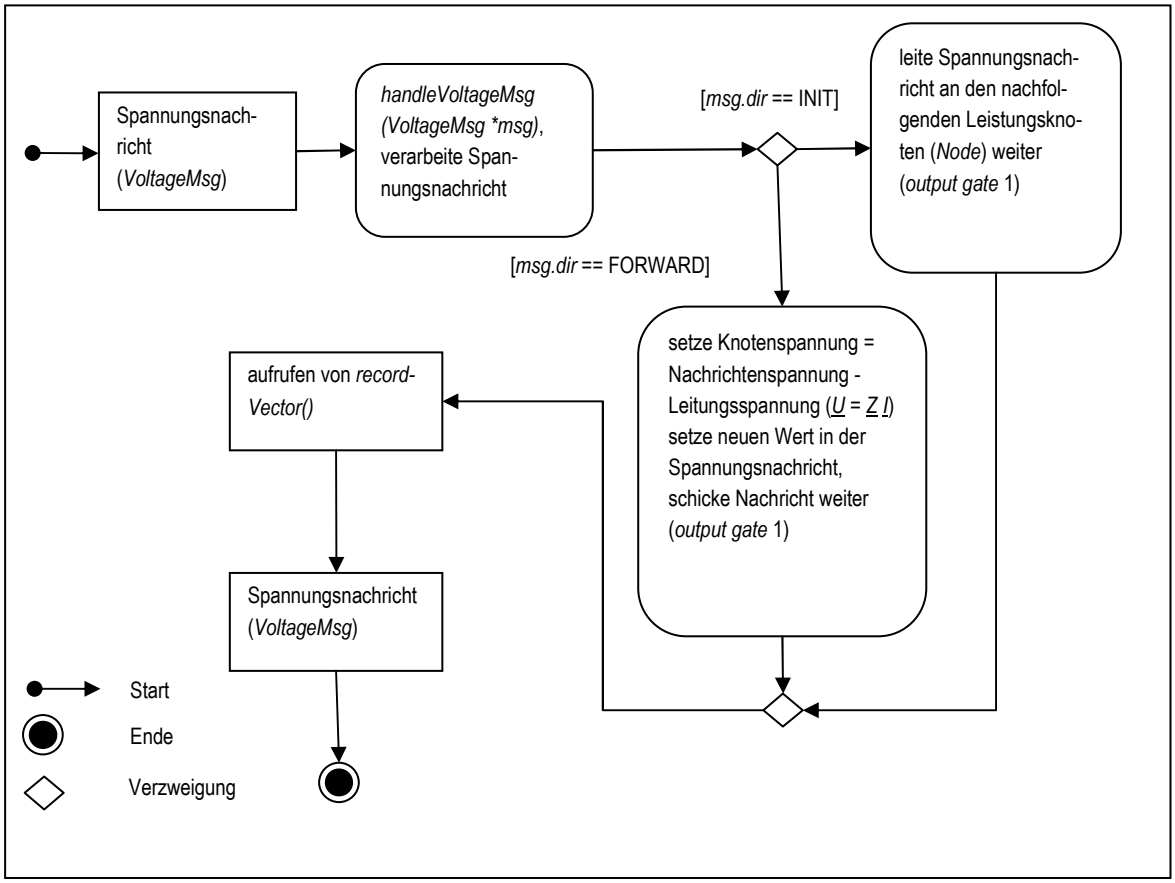


Abbildung 49 – Init- und Forward-Spannungsnachricht am Leitungsknoten

Strom der Nachricht und der Impedanz berechnet. Aus der Spannung des Nachfolgeknotens, die in der Nachricht enthalten ist, kann nun gemäß der Maschenregel die Spannung für den Vorgänger-

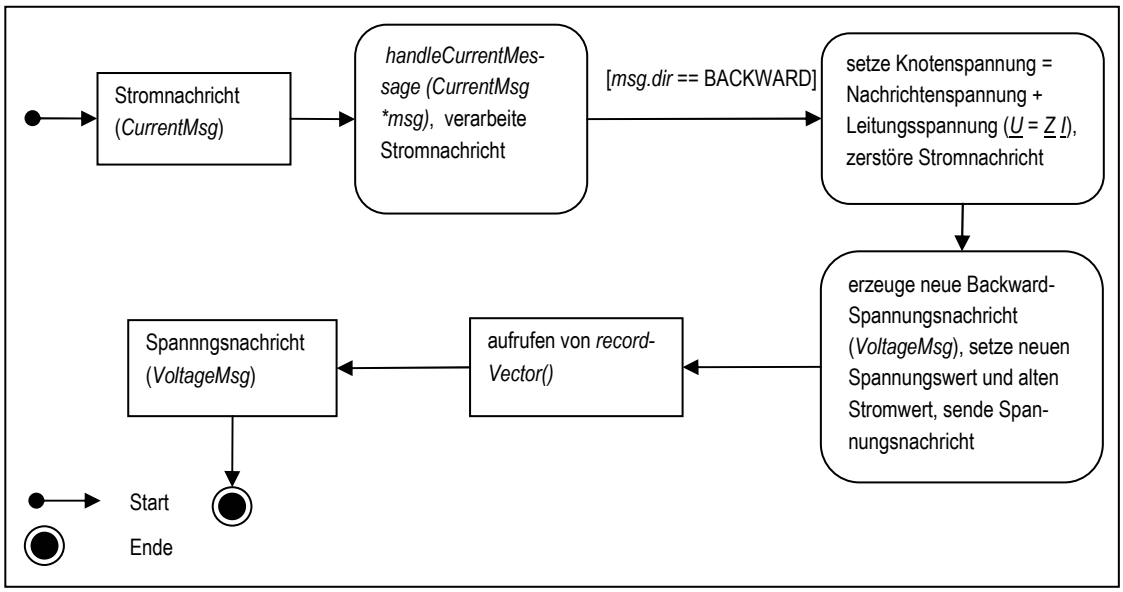


Abbildung 50 – Backward-Spannungsnachricht am Leitungsknoten

Leistungsknoten zu $\underline{U} = \underline{U}_{VoltageMsg} + (\underline{Z}_{Line}\ \underline{I}_{VoltageMsg})$ berechnet werden. Dieser Spannungswert wird in die Nachricht eingetragen und an das *output gate* 0, also in Richtung des Startknotens bzw. des Vorgänger-Leistungsknotens, geschickt. In Abbildung 50 wird die Abfolge der Schritte in Form eines Aktivitätsdiagramms dargestellt.

3.6 Hilfsklassen *Common* und *Control*

Für die Implementierung des Lastflussverfahrens in OMNeT++[3] wurden neben den Klassen zur Implementierung des Verhaltens der Leistungsknoten (*Node*) und Leitungsknoten (*Line*) auch noch die Klassen *Common* und *Control* entwickelt.

Die folgende Abbildung 51 zeigt die beiden Klassen mit den Methoden und Variablen in UML-Darstellung. Wie in der Abbildung zu erkennen ist, leitet sich nur die Klasse *Control* wieder von der Basisklasse des Frameworks, *cSimpleModule* ab. Im Fall der *Control*-Klasse wird dies benötigt, da diese auch als eigener Modultyp in der NED-Datei hinterlegt ist. Die Klasse respektive Modultyp besitzt jedoch keinerlei Verbindung zu den Anwendungsmodulen (Klassen) *Node* und *Line*, da von der Klasse keine Nachrichten verarbeitet werden müssen, sondern nur eine globale Sicht auf die Simulationsverhältnisse hergestellt wird.

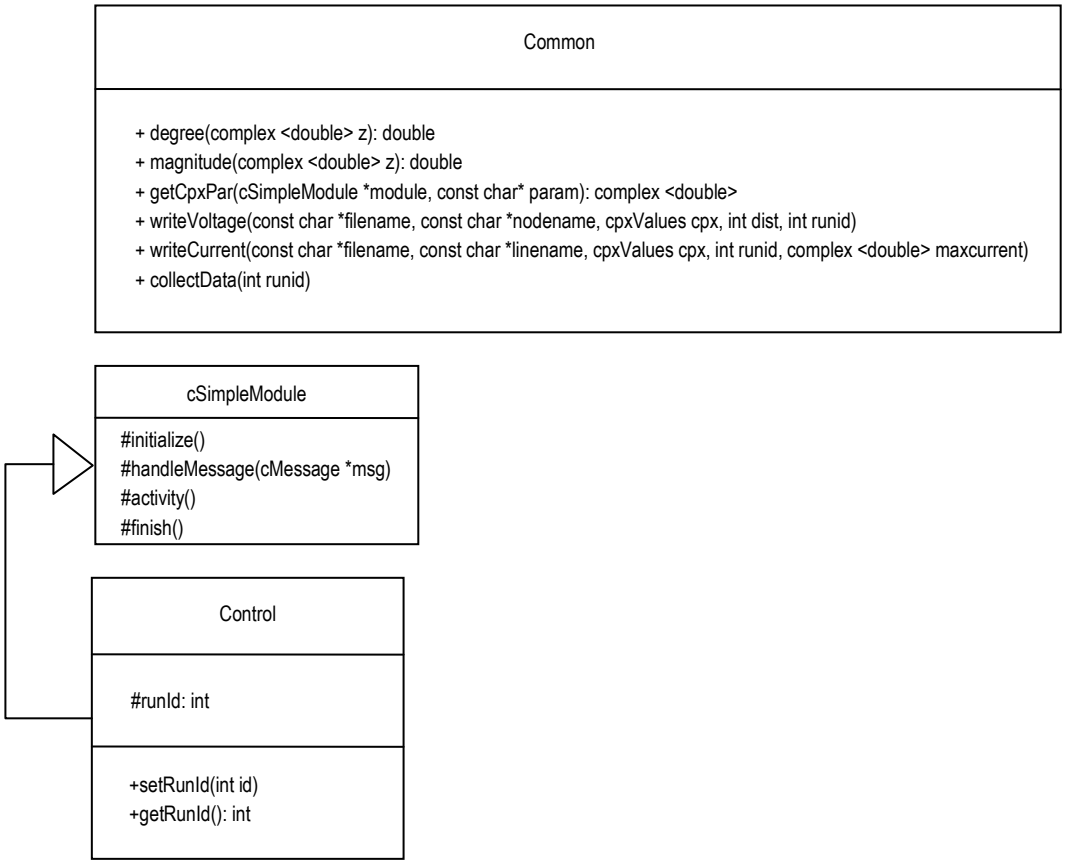


Abbildung 51 – Klassenentwurf *Common*- und *Control*-Klasse

Die Klasse *Control* dient der Verwaltung von in der Simulation, bezüglich aller Instanzen von allen in der Anwendung verfügbaren Knotentypen, verfügbaren globalen Variablen. Dies ist notwendig da es in einer OMNeT++-Anwendung [3] sonst nicht möglich ist, globale Variablen auf Modulebene zu implementieren, da jede Klasse die vom Anwender implementiert wird, nur als Vorlage für die Anzahl an Instanzen respektive in der Simulation verfügbaren Knoten dient. Das bedeutet, dass eine Variable die als global in einer Klasse definiert wird, nur in der jeweiligen Instanz global ist und es deshalb sonst keine Möglichkeit gibt, eine globale Sicht aus der Perspektive der Simulationsumgebung herzustellen (siehe auch [1]). Die Klasse beinhaltet eine Variable zur Speicherung der schon bekannten Lauf-ID welche über entsprechende öffentliche Methoden zugänglich ist. Über die beiden Methoden kann nun jede Anwendungsklasse wie *Node* und *Line* eine Referenz (Zeiger) über die Methode *Simulation::moduleByPath* [1] herstellen und auf das globale Modul (Klasse) zugreifen und die Lauf-ID setzen bzw. auslesen. Damit besitzt die Simulation während der Abarbeitung jeweils Kenntnis über die bisher aktuelle Anzahl an totalen Simulationsläufen (abgeschlossene Ladder-Iterative-Technique-Läufe).

Die *Common*-Klasse stellt nur eine einfache C++-Klasse dar. Die Klasse *Common* stellt Methoden zur Verfügung die von den anderen Klassen gemeinsam benutzt werden können. Dazu gehören die Methoden *degree(…), magnitude(…), getCpxPar(…), writeVoltage(…), writeCurrent(…)* und *collectData()*. Die Methode *degree(…)* berechnet den Winkel der Polardarstellung einer komplexen Zahl. Die Methode berechnet *magnitude(…)* den Betrag der Polardarstellung. Die Methode *getCpxPar(…)* liest den komplexen Wert bestehend aus Real- und Imaginärteil in Zeichenkettenform aus der INI-Datei bzw. der NED-Datei der Anwendung und wandelt diesen in eine komplexe Zahl, die vom Programm verarbeitet werden kann um. Die Methoden *writeVoltage(…)* und *writeCurrent(…)* führen die schon beschriebenen Protokollierung der Spannungswerte der Leistungsknoten und Stromwert der Leitungsknoten in eine Datei durch. Die Methode *collectData()* ruft die Methode *writeVoltage(…)* auf, wobei sie nur die Leistungsknotentypen berücksichtigt. Dies wird durch die *cTopology*-Klasse des Frameworks bewerkstelligt, mit der Strukturinformationen über das Netz eruiert werden können. Der Methode wird die aktuelle Lauf-ID übergeben. Im Speziellen werden für jede Leistungsknoten-Instanz die zutreffenden Werte wie Knotenname, Leistung und Distanz zum Startknoten ermittelt und an die Methode *writeVoltage(…)* übergeben. Damit kann am Ende eines Simulationslaufes das Endergebnis für jeden Knoten in der Datei weggeschrieben werden. Die Klasse *Common* ist im Gegensatz zur Klasse *Control* nicht als eigener Knotentyp in der NED-Datei hinterlegt (siehe auch Abbildung 32), sondern wird von den Klassen *Node* und *Line* durch Zeiger referenziert. Es wird zwischen den Klassen also eine einfache Assoziation hergestellt.

3.7 Zusammenspiel aller Komponenten

In diesem Abschnitt wird das Zusammenspiel aller Komponenten nochmals erläutert, mit dem Ziel den Aufbau des Gesamtsystems anzugeben. Wie in den letzten Abschnitten 3.4 und 3.5 aus der Perspektive der jeweiligen Knotentypen ersichtlich, können die Nachrichtentypen und deren Verarbeitungsrichtung wie folgt charakterisiert werden. Der Leistungsknotentyp *(Node*) verarbeitet die folgenden Nachrichtentypen:

- Backward-, Init- und Forward-Spannungsnachricht (*VoltageMsg*), an Verteilknoten müssen diese im Vorwärts- und Rückwärtsschritt gesondert behandelt werden
- *self message*, Leistungsparameter soll zufällig geändert werden
- Kontrollnachricht (*ControlMsg*), Startknoten weiß über Änderung Bescheid, kann Neustart initiieren

Der Leitungsknotentyp (*Line*) verarbeitet die Nachrichtentypen:

- Backward-Stromnachricht (*CurrentMsg*)
- Init- und Forward-Spannungsnachricht (*VoltageMsg*)
- Kontrollnachricht (*ControlMsg*)

3.7.1 Nachrichtenfluss

Im folgenden Abschnitt wird zusammenfassend und aus der Sicht des Gesamtablaufs der Schritte des Verfahrens der Ladder Iterative Technique, der Nachrichtenfluss zwischen den Knoten dargestellt. In Abbildung 52 wird die Abfolge der Nachrichten zwischen den verschiedenen Knotentypen-Instanzen (Modultypen) $node_i$ und $line_i$ in Form des Beispielnetzes aus Abbildung 4 und basierend auf der Modellierung von Abbildung 32 für die Initialisierungsphase der Ladder Iterative Technique dargestellt.

Bevor der Nachrichtenfluss zum Laufen kommt, soll angenommen werden, dass als Knotentypen-Instanzen ($node_i$ und $line_i$) mit den zu ihrem Knotentyp gehörenden Parameter initialisiert wurden (Leistung und Impedanz). Wie in der Abbildung zu erkennen ist, werden am Startknoten $node_1$ zwei Init-Spannungsnachrichten ($vmsg_1$ und $vmsg_2$) erzeugt und in das Netz, welches in dem Beispielnetz aus zwei Teilästen besteht, hineingeschickt. Die Init-Spannungsnachricht wird mit dem Wert des Parameters *srcvoltage* initialisiert (entspricht der Spannung am Startknoten). Die Leitungsknoten $line_i$ schicken diese einfach weiter, die Leitungsknoten $node_i$ setzen jeweils ihren Spannungswert ($node_i.cpx.voltage$) auf den Spannungswert der Nachricht ($vmsg_i.cpx.voltage$). Am Verteilknoten $node_2$ muss die Nachricht $vmsg_1$ dupliziert werden, da hier ein weiterer Ast angeschlossen ist. Somit kursieren im Netz insgesamt drei Init-Spannungsnachrichten (*VoltageMsg*). Die Nachrichtenverabeitung kann im Detail in Abschnitt 3.4 und 3.5 für die Knotentypen *Node* und *Line* nachvollzogen werden. Als Endergebnis des Initialisierungsschritts sind alle Knoten mit der Spannung, die am Startknoten anliegt, initialisiert. Weiters wird an den Endknoten $node_3$, $node_4$ und $node_5$ nach Empfang der Init-Spannungsnachrichten ($vmsg_1$, $vmsg_3$ und $vmsg_2$) der Strom aus der empfangenen Spannung und der gegebenen Leistung des Knotens berechnet ($(node_i.cpx.power / node_i.cpx.voltage)^*$), die Init-Spannungsnachricht zerstört und anstatt dessen eine Backward-Stromnachricht (*CurrentMsg*) erzeugt und versendet. Die Abarbeitung der Backward-Nachrichten bzw. der Austausch bzw. Bearbeitung der Nachrichten an den Knoten im Zuge des Rückwärtsschritts des Verfahrens wird in der folgenden Abbildung 53 näher erläutert. Wie in der Abbildung zu erkennen ist, werden von den Knoten $node_3$, $node_4$ und $node_5$ die drei Backward-Stromnachrichten $cmsg_1$, $cmsg_2$ und $cmsg_3$ erzeugt und in Richtung des Startknotens $node_1$ versendet. Die Spannungs-

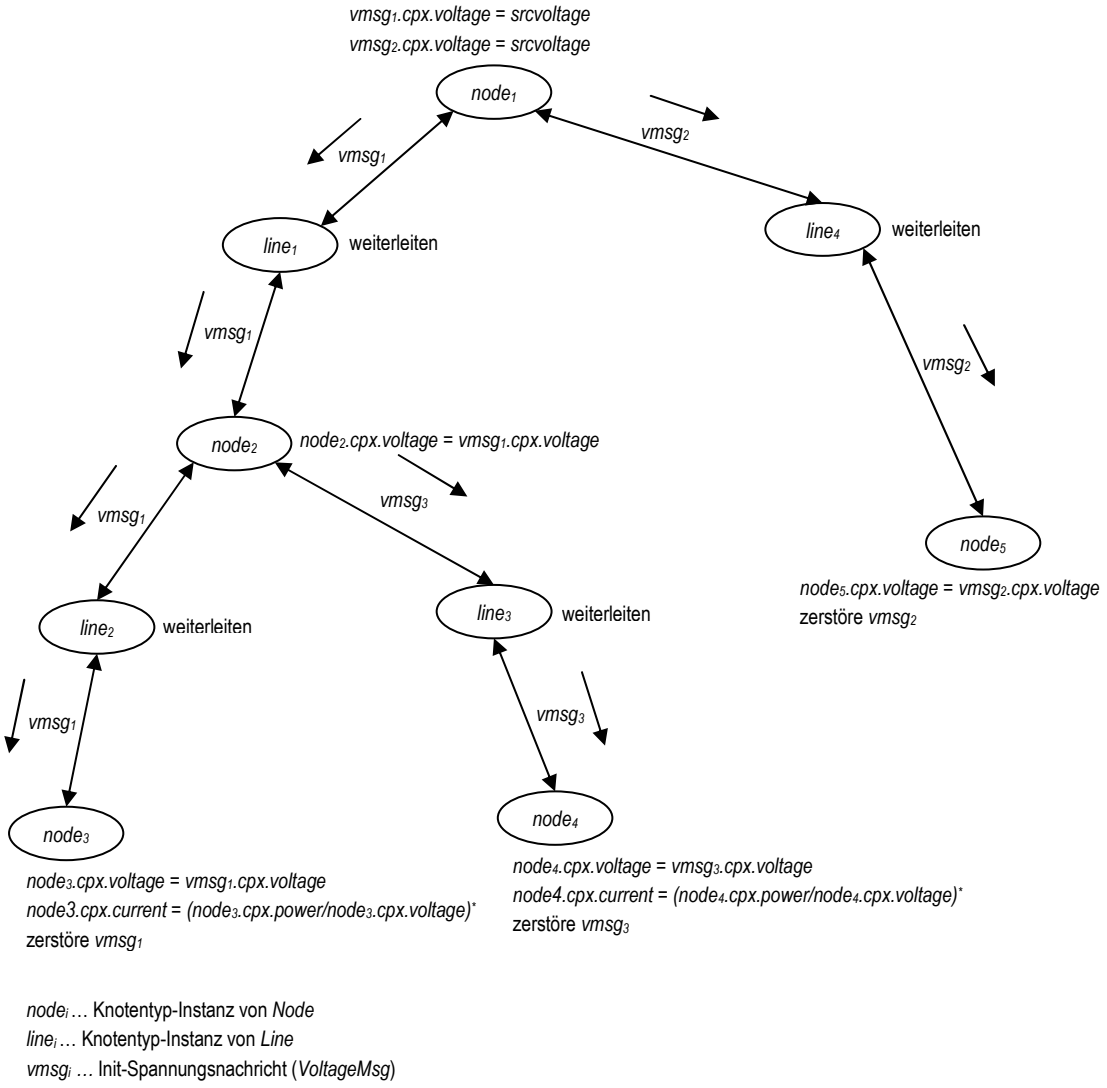


Abbildung 52 – Nachrichtenfluss Initialisierungsphase

und Stromwerte der Nachrichten entsprechen den Werten an den Knoten, also in diesem Fall der der von der Initialisierungsphase herrührenden Spannung des Knoten $node_1$ und der an den Endknoten durch diese Spannung und dem jeweiligen Knoten zugeordnete Leistung definierten bzw. zu errechnendem Strom. An den Leitungsknoten $line_2$, $line_3$ und $line_4$ werden nun die Stromnachrichten empfangen und die durch die Kirchhoffsche Maschenregel definierten Spannungsfälle auf den Leitungen bzw. die Spannungen für die Knoten $node_2$ und – für den rechten Verteilast für den Vergleich mit dem Startknoten – $node_1$, berechnet. Danach werden die Stromnachrichten gelöscht und anstatt dessen die Spannungsnachrichten $vmsg_1$, $vmsg_2$ und $vmsg_4$ erzeugt und mit den neuen Spannungswerten befüllt. Die Stromwerte ändern sich nicht. Danach werden die Nachrichten versendet. Für den rechten Verteilast kann nun am Startknoten $node_1$ bereits die Differenz zwischen der gegeben Spannung der Spannung aus der Nachricht $vmsg_4$ berechnet und mit dem gegebenen Schwellwert verglichen

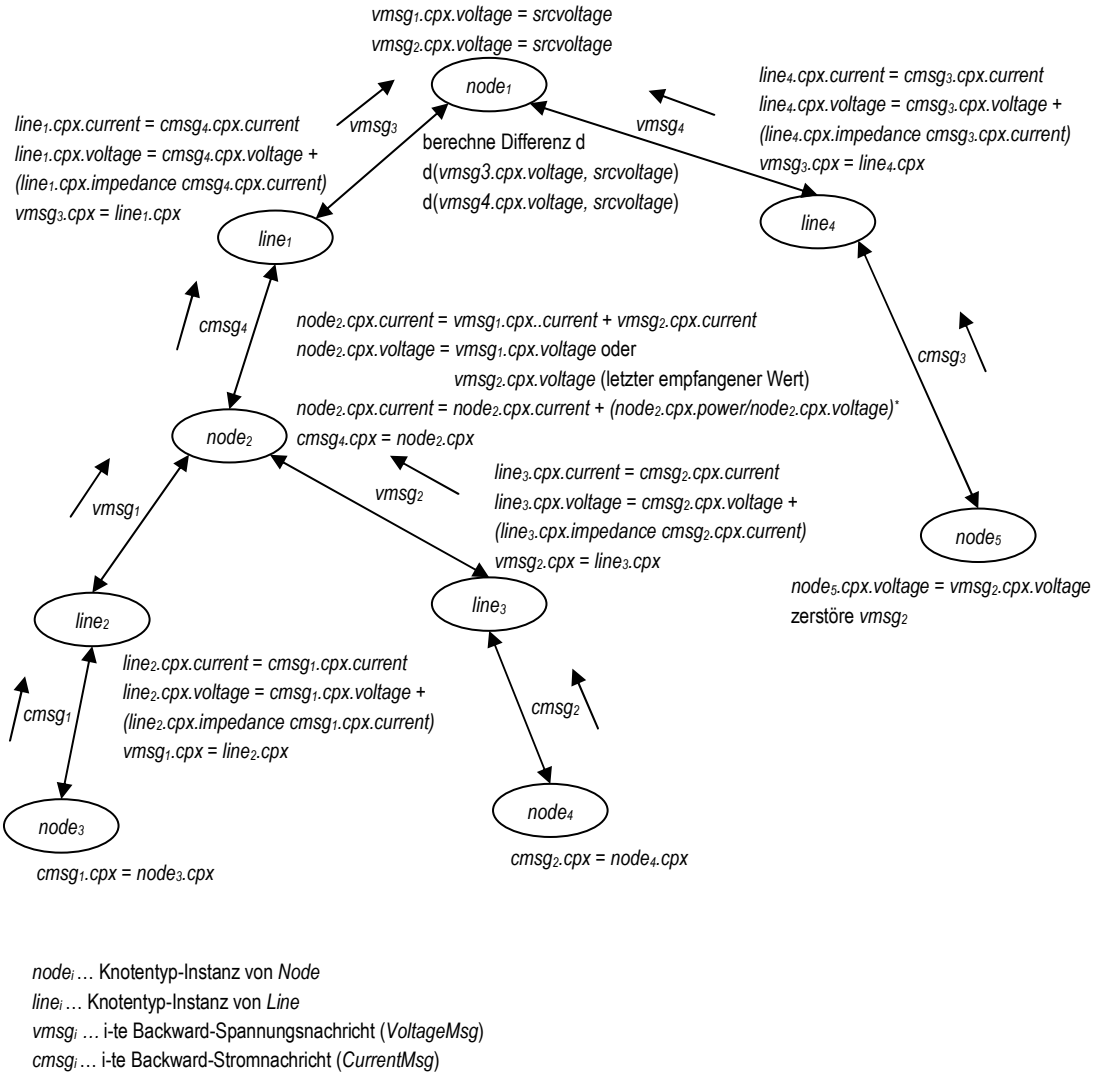


Abbildung 53 – Nachrichtenfluss beim Rückwärtsschritt

werden. Ist die Differenz größer dem Schwellwert so wird ein weiterer Vorwärtsschritt vom Startknoten initiiert. Für den linken Teilast muss nun am inneren bzw. Verteilknoten $node_2$ zuerst die Kirchhoffsche Knotenstromregel eingehalten werden. Es müssen also die Ströme von Knoten $node_3$ und $node_4$ berücksichtigt werden. Das bedeutet, dass die Ströme aus den Nachrichten $cmsg_2$ und $cmsg_1$ gelesen und am Knoten $node_2$ aufsummiert werden. Je nachdem, welcher Teilast zuerst abgearbeitet wird, wird dann die Spannung für den Knoten $node_2$ gesetzt und daraus der Strom aufgrund der gegebenen Leistung am Knoten berechnet. Die Nachrichten $vmsg_1$ und $vmsg_2$ werden nach Abarbeitung zerstört. Das bedeutet, dass wie an einem einfachen inneren Knoten eine neue Stromnachricht $cmsg_4$ in Richtung des Leitungsknoten $line_1$ versendet wird. Nach der Berechnung des Spannungsfalls auf *line1* und der Berücksichtigung der Maschenregel und der damit verbundenen Generierung der Nachricht $vmsg_3$ kann der Vergleich am Startknoten für diesen Spannungswert durchge-

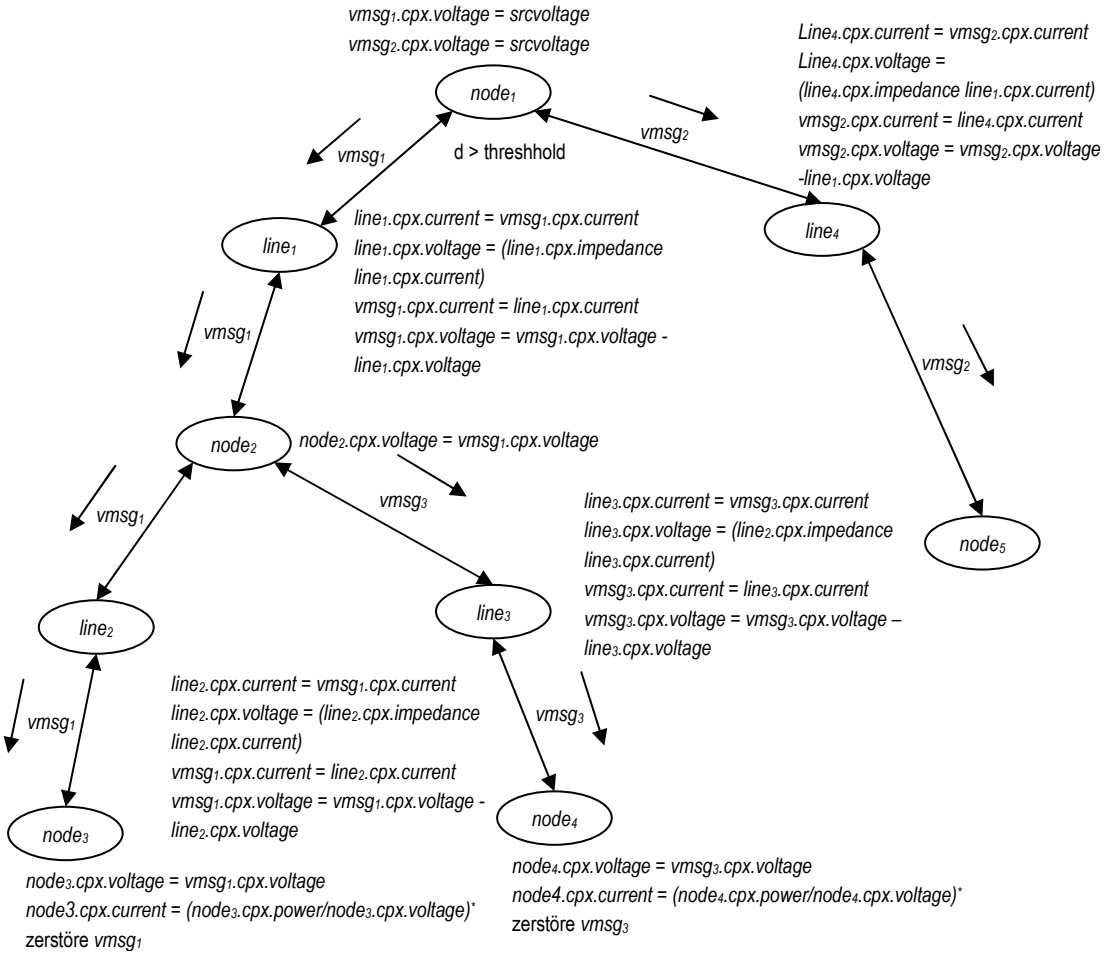


Abbildung 54 – Nachrichtenfluss beim Vorwärtsschritt

führt werden. Von dem schon erwähnten Vergleich hängt es ab ob dann der linke Teilast, vom Startknoten *node1* aus gesehen, neu mittels eines Vorwärtsschrittes berechnet werden muss. Die Abbildung 54 zeigt den Nachrichtenfluss für unser Beispielnetz im Zuge der Abarbeitung des Vorwärtsschrittes des Verfahrens. Sofern ein weiterer Vorwärtsschritt notwendig ist, wird die empfangene Spannungsnachricht erneut mit dem Spannungswert des Startknotens initialisiert, die Richtung geändert und erneut an den zu berechnenden Teilast des Netzes verschickt. In unserem Beispiel werden also nochmals die Nachrichten $vmsg_1$ und $vmsg_2$ verschickt. Die Leitungsknoten $line_1$ und $line_4$ erhalten diese Nachrichten und berechnen nun wieder basierend auf der Kirchhoffschen Maschenregel einerseits den Spannungsfall auf den Leitungen und andererseits die Spannung für die Nachfolgeknoten $node_2$ und $node_5$. Die daraus resultierenden Werte werden in den Nachrichten $vmsg_1$ und $vmsg_2$ gesetzt und weitergeschickt. Die Spannungsfälle werden im Vorwärtsschritt also aufgrund der Ströme die im vorhergegangenen Rückwärtsschritt an den Knoten berechnet wurden, berechnet. Nachdem es sich beim Knoten $node_2$ um einen Verteilknoten handelt, muss die Nachricht $vmsg_1$ dupliziert werden, es kommt also die Nachricht $vmsg_3$ hinzu. Wird im Zuge der Abarbeitung der Nachrichten bzw. Abfolge von Leistungs- und Leitungsknoten ein Leistungs-Endknoten erreicht, so wird an diesem erneut mit dem Rückwärtsschritt (siehe Abbildung 53) begonnen.

3.7.2 Komponenten

Im folgenden Abschnitt wird nochmals ein Überblick über die Komponenten des Gesamtsystems gegeben. Die folgende Abbildung 55 zeigt ein UML-Analysediagramm mit den Klassen der Anwendung.

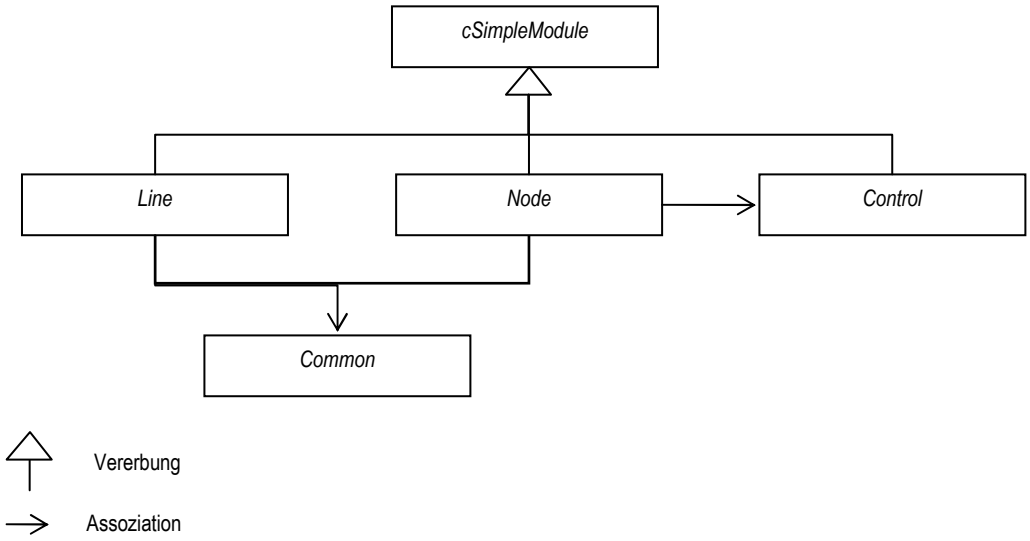


Abbildung 55 – Komponten-Klassendiagramm

Wie in der Abbildung ersichtlich ist, leiten sich die Klassen *Line*, *Node* und *Control* von der Basisklasse *cSimpleModule* ab, die alle notwendigen Funktionalitäten für die Nachrichtenverarbeitung des Frameworks zur Verfügung stellt. Die Klasse *Node* besitzt weiters eine Assoziation in Richtung der Klasse *Control*, welche für die globale Sicht auf die Simulation zuständig ist. Die Hilfsklasse *Common* wird von den Klassen *Line* und *Node* gleichermaßen referenziert. In der folgenden Abbildung 56 wird das Zusammenspiel der notwendigen Dateien zusammenfassend dargestellt.

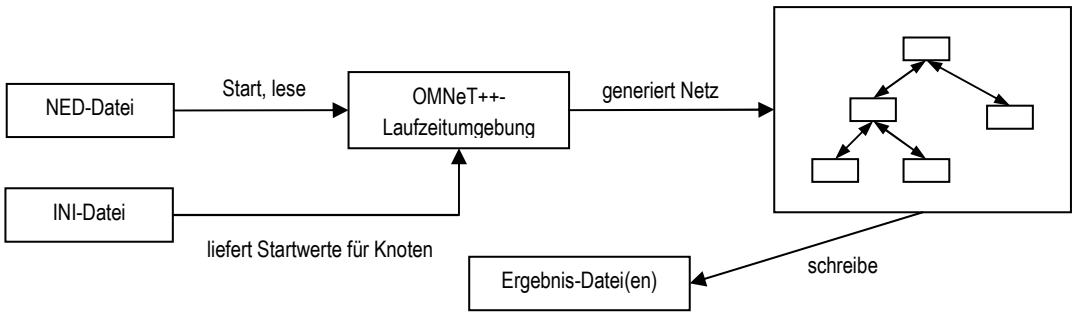


Abbildung 56 - Komponentendateien

Die NED-Datei in der die Beschreibung der Topologie des Netzes vorhanden ist, wird also beim Programmstart gelesen und von der Laufzeitumgebung in ein entsprechendes elektrisches Netz umgewandelt. Parallel dazu wird die INI-Datei gelesen, in der die Startwerte für die Knotentyp-Instanzen respektive Knotenparameter des Netzes als Startwerte vorgegeben werden können. Unmit-

telbar nach Aufbau des Netzes wird die Initialisierung der einzelnen Knoteninstanzen vorgenommen. Im konkreten Fall beginnt also der Startknoten (Leistungsknoten) im elektrischen Netz mit dem Versand der Init-Spannungsnachricht. Alle anderen Knoten warten dann auf die eingehenden Nachrichten. Während bzw. am Ende der Simulation (FES leer), werden die entsprechenden Ergebnisdateien geschrieben.

4. Ergebnisse und Diskussion

In diesem Kapitel wird auf die Ergebnisse der Arbeit näher eingegangen. Dazu wird zuerst auf die Korrektheit der Lastflussrechnung und im weiteren auf die umgesetzten Anforderungen bzw. deren Erfüllung näher eingegangen. Weiters werden alternative Implementierungsmöglichkeiten beleuchtet und der gegenwärtigen Implementierung gegenübergestellt.

4.1 Ergebnisse der Berechnung

In Kapitel 3 wurden der Lösungsansatz und die Implementierung der gewählten Lösung eingehend beleuchtet. Im diesem Abschnitt wird auf die Simulationsergebnisse in Bezug auf die Berechnung des Lastflusses eingegangen. Dazu wird unser Beispielnetz aus Abbildung 32 herangezogen. Das Netz besteht aus fünf verschiedenen Leistungs- und vier verschiedenen Leitungsknoten. An den Leistungsknoten sind Verbraucher und Dezentrale Erzeugungsanlagen angeschlossen. Die Leistung wird als Summenleistung in *PQ*-Form modelliert (siehe Abschnitt 2.1). Die Simulation wurde basierend auf den gegebenen Startwerten aus Abbildung 4 und Tabelle 1 durchgeführt. Die Ergebnisse lassen sich wie folgt in Tabelle 3 darstellen.

Tabelle 3 – Simulationsergebnisse des Beispielnetzes

Knoten	Entfernung vom Startknoten [km]	Spannung-Realteil [kV]	Spannung-Imaginärteil [kVar]
$node_1$ *(N1)*	0	30	0
$node_2$ *(N2)*	10	28,69	0,30
$node_3$ *(N3)*	18	28,51	0,44
$node_4$ *(N4)*	20	28,51	0,32
$node_5$ *(N5)*	25	30,68	0,42

Die Knoten $node_2$, $node_3$ und $node_4$ formieren dabei den linken Teilast (Leitung1) des Netzes, der an den Startknoten (Slack-Knoten) $node_1$ angeschlossen ist. Der rechte Teilast (Leitung2) besteht nur aus einem Leistungsknoten $node_5$ (siehe Abbildung 57).

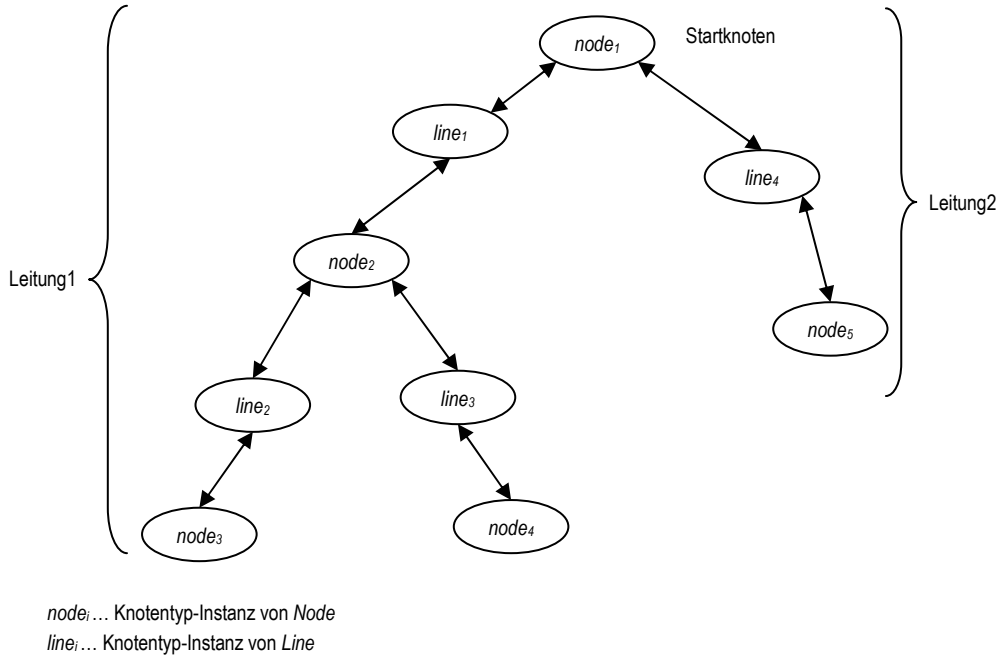


Abbildung 57 – Leitung1 und Leitung2 im Beispielnetz

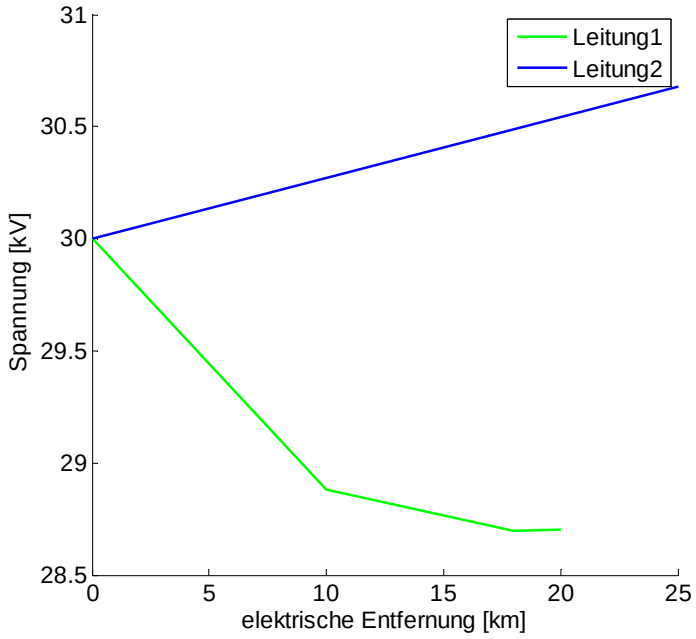


Abbildung 58 – Spannungsverlauf und elektrische Entfernung

Die Verbindungsknoten für die Leitungen $line_1$ bis $line_4$ sind hier nicht aufgeführt, da im Folgenden vor allem auf die Spannungsverläufe an den Leistungsknoten eingegangen werden soll. In der

Abbildung 58 wird der Spannungsverlauf in Bezug auf die elektrische Entfernung vom Startknoten gezeigt. Wie in der Abbildung zu erkennen ist, kommt es auf Leitung1 zu einem stetigen Abfall der Spannungen mit zunehmender Entfernung vom Startknoten (*node1*). Im rechten Teilast hingegen (Leitung2) kommt es zu einem Anstieg, da wie auch in Tabelle 1 und Abbildung 4 dargestellt, die Leistung die von der Dezentralen Erzeugungsanlage DEA_5 in das Netz eingebracht wird, nicht von etwaigen Lasten kompensiert wird. Die Simulationsergebnisse sind mit denen des kommerziellen Pakets DIgSILENT Power Factory [7] vergleichbar. Es wurden noch weitere Simulationen mit einfachen Verteilnetzen durchgeführt, auch in deren Fall kam es zu keinen nennenswerten Abweichungen – diese liegen zwischen 0,16% und 0,20% bei einer Referenzspannung von 30 kV für das Beispielnetz – in Bezug auf die Referenzlösungen. Dazu sei als weiterer Vergleich auch nochmals auf die Ergebnisse der Berechnung aus der Gauß-Seidel-Iteration, welches in MATLAB [4] berechnet wurde, und dessen Ergebnisse in Tabelle 2 verwiesen. Die Differenzen liegen auch hier für das Beispielnetz in einer Größenordnung von 0,1% bis 0,16%. Als Gründe für die Abweichungen scheinen die einfachen Modelle hinsichtlich der Leistungsknoten und der Leitungsknoten ausschlaggebend zu sein. Das Paket DIgSILENT Power Factory [7] besitzt für das Beispielnetz genaue Leitungsdaten hinsichtlich Leitungstyp und Leistungsfaktor. In der gegenwärtigen Implementierung werden diese Daten, vor der Verwendung im Modell, in die Form $\underline{Z} = R + jX$ umgewandelt, da nur diese Darstellung vom Algorithmus verarbeitet werden kann. Aufgrund der ähnlichen Abweichungen zu dem Paket DIgSILENT Power Factory [7] und zu den Ergebnissen der Gauß-Seidel-Iteration wird abgeleitet, dass die Berechnung des Lastflusses korrekt vom Programm umgesetzt bzw. berechnet wird. Die detaillierten Berechnungsschritte wurden ja bereits in Abschnitt 3.4, 3.5 und 3.7.1 erläutert.

4.2 Technische Modellbildung

Im folgenden Abschnitt wird auf die Umsetzung des Lastflussanalyse-Verfahrens bezüglich der Modellbildung eingegangen. Bezüglich der Umsetzung des Lastflussanalyse-Verfahrens wurde ja schon im letzten Abschnitt die Richtigkeit der Berechnung erläutert. Das Modell für die Lasten und Leitungen unterstützt zur Zeit die Abbildung eines einphasigen Systems. Wie schon im Abschnitt 2.3 erläutert, kommt es vor allem im Bereich von Niederspannungsnetzen zu unsymmetrischen bzw. unbalancierten Lastfällen. Dies bedeutet, dass bei der Analyse alle drei Phasen eines Verteilnetzes zu betrachten sind. Im vorliegenden Fall wurden Netze der Mittelspannungsebene analysiert, wo es aufgrund der zu erwartenden Lastsymmetrie ausreicht, eine Phase als Referenz für die restlichen verbleibenden Phasen zu betrachten. Die entsprechenden Modelle bzw. die Implementierung in den Modulen kann in Zukunft durch die Modelle die in Kersting et al. [Ker07] beschrieben sind, erweitert werden. Verbindungen zwischen Bussen (Knoten) eines Verteilnetzes können durch Leitungen, Transformatoren bzw. Umschalter [Ker07] verbunden werden und werden daher auch als Serienkomponenten bezeichnet. In der vorliegenden Implementierung wurde aus Gründen des Umfangs nur die Leitungen (Leitungsknoten *Line*) durch ein einfaches Modell ($\underline{Z} = R + jX$) nachgebildet. Die Busse eines Verteilnetzes beinhalten statische Lasten, Generatoren, verteilte Lasten und Kapazitäten [Ker07]. In der vorliegenden Implementierung wurde nur das statische Lastmodell in Form von *PQ*-Knoten implementiert. Die Leistungsknoten geben also eine bestimmte konstante Leistung mit Wirk- und Blindleistungsanteil vor. Für Generatoren die im Verteilnetz auch als dezentrale Erzeu-

gungsanlage (DEA) bezeichnet werden, wird ein konstantes Lastmodell mit negativem Wirkanteil herangezogen [Ker06].

Im Abschnitt 1.5.1 über die verwandten Arbeiten hinsichtlich Modellbildung, wurde auf einige Implementierungen zur Analyse von Verteilnetzen unter besonderer Bezugnahme auf die technischen Grundlagen von elektrischen Verteilnetzen eingegangen. Diese Arbeiten beschreiben bezüglich der Methodik die Umsetzung der Ladder Iterative Technique [Ker07] und wenden diese auf verschiedene Verteilnetze an. Thomson et al. [Tho03] kommt dabei zum Schluss, dass ein konstantes Lastmodell aufgrund der hohen stochastischen Natur der Lasten bei der Simulation und aufgrund der daraus folgenden nur sehr kurzlebigen Gültigkeit der Ergebnisse, der Realität unzureichend Rechnung trägt. In der vorliegenden Implementierung wurde deshalb ein Ansatz einer einfachen Monte-Carlo-Simulation implementiert. Das bedeutet, dass während der Berechnung des Netzes die Leistungsknoten (*Node*) zufällig ihren Leistungswert innerhalb eines vorgegebenen Wertes ändern können (siehe auch Abschnitt 3.4.2). Die meisten der erwähnten Arbeiten bilden das Verfahren in MATLAB [4] ab. Nur in der Arbeit von [Mok99] wird erwähnt, dass das System mittels der Sprache C++ als eigenständige Anwendung – also wie im vorliegenden Fall mittels OMNeT++ [3] – entwickelt wurde. Als Voraussetzung für alle Systeme wird erwähnt, dass das Verteilnetz eine radiale Struktur aufweisen muss. Die in dieser Arbeit umgesetzte Implementierung ist zur Zeit ebenso auf rein radiale Strukturen ausgelegt. Das von [Shi88] umgesetzte Verfahren, welches auch schwach vermaschte Systeme auf rein radiale Strukturen umwandeln bzw. berechnen kann, wird gegenwärtig nicht unterstützt. Alle Arbeiten implementieren ein Drei-Phasen-Modell, da in diesen Arbeiten vor allem Analysen im Niederspannungsbereich durchgeführt wurden. Die Erweiterung ist in der Implementierung dieser Diplomarbeit möglich aber gegenwärtig noch nicht umgesetzt. Die Berechnung beschränkt sich auf eine Einphasen-Darstellung der Werte, welche allerdings für die analysierten Verteilnetze im Mittelspannungsbereich ausreichend ist, da in diesem Fall nur sehr geringe Unsymmetrien zu erwarten sind.

4.3 Softwaremäßige Abbildung

Im folgenden Abschnitt wird auf die Abbildung des Verfahrens in OMNeT++ [3] im Vergleich zu den gewählten Verfahren aus Abschnitt 1.5.2 und auf die Vorteile und Nachteile der Implementierung näher eingegangen. Weiters werden die Erweiterungsmöglichkeiten der gewählten Lösung beschrieben. Elektrische Verteilnetze können als Graphenmodell aufgefasst werden [Bou98]. Die Arbeit von Bouchard et al. [Bou98] erwähnt dabei das Abstract-Factory- [Fre04], Composite- [Fre04] und Iterator-Entwurfsmuster [Fre04] als mögliche Varianten, die zur Implementierung einer Anwendung zur Analyse von Verteilnetzen herangezogen werden können. Das Abstract-Factory-Entwurfsmuster dient der Implementierung einer „*Schnittstelle um Familien von in Verbindung stehenden oder abhängigen Objekten erzeugen zu können, ohne ihre konkreten Klassen zu spezifizieren*“ [Fre04, S 156]. Das Composite-Entwurfsmuster erlaubt es „*Objekte in Baumstrukturen anzuordnen um Teil-Ganzes-Beziehungen herzustellen. Für Klienten bedeutet dies, dass diese die Teile-als auch zusammengesetzten Komponenten gleich behandeln können*“ [Fre04, S 356]. Das Iterator-Muster erlaubt es „*auf die Elemente von zusammengesetzten Objekten sequentiell ohne Kenntnis*

über die innere Struktur der Objekte zugreifen zu können“ [Fre04, S 336]. Für die Implementierung einer Anwendung zur Lastflussanalyse von elektrischen Verteilnetzen wie in dieser Arbeit durchgeführt, ergeben sich die folgenden Möglichkeiten [Bou98].

- Abstract-Factory-Muster für die Entwicklung der Komponenten heranziehen
- Composite-Muster für die Abbildung der Relationen zwischen den Komponenten, also der Topologie bzw. der Netzstruktur heranziehen
- Iterator-Muster für die Traversierung der Objekte des Netzes benutzen

Diese Schritte bzw. Herangehensweise wird auch in der Arbeit von Bouchard et al. [Bou98] empfohlen. Für die vorliegende Arbeit bzw. Implementierung wurde davon aus mehreren Gründen kein Gebrauch gemacht. Erstens war die Anwendung unter dem Framework OMNeT++ [3] zu entwickeln, welches die schon im Abschnitt 2.6 erwähnte Vorgehensweise zur Definition bzw. Implementierung einer Simulation notwendig macht. Die Anwendung des Abstract-Factory-Musters innerhalb des Frameworks kann nicht sinnvoll angewendet werden. Grund dafür ist der Umstand, dass jede Klasse die ein bestimmtes Verhalten implementiert von der (abstrakten) Basisklasse *cSimpleModule* abgeleitet werden muss, wobei die eigentliche Funktionalität in den Methodenrümpfen der Basisklasse *initialize()* und *handleMessage(cMessage *msg)* zu hinterlegen ist. Weitere Abstraktionsebenen wie diese für alle Entwurfsmuster vorgesehen sind – Komposition wird bei Entwurfsmustern immer der Möglichkeit von Vererbung vorgezogen – sind hier zwar prinzipiell möglich, erschweren dann aber die Verwendungen von notwendigen Funktionalitäten bzw. Methoden, die sinnvollerweise vom Framework vorgegeben werden. Als Beispiel sei hier der Zugriff auf die Anschlüsse (*gates*) bzw. Namen und Indizes von Modulen genannt.

Die Struktur bzw. Topologie des elektrischen Verteilnetzes wird, wie schon erwähnt, aufgrund von sogenannten NED-Dateien beschrieben. Diese werden von Hand oder über ein vom Framework bereitgestelltes Werkzeug (Anm. Drag and Drop), erzeugt. Daraus folgt, dass das Composite-Muster hier ebenso nicht sinnvoll angewandt werden kann, da der Aufbau des Netzes beim Start der Anwendung vom Framework aufgrund der gegebenen NED-Datei aufgebaut wird. Das Framework stellt also bereits selbst die Abhängigkeiten zwischen den Modulen bzw. eventuelle Teil-Ganzes-Beziehungen her (vgl. Definition Compound-Modul, Simple-Modul, Netzwerk, siehe Abschnitt 2.6.1).

Das Iterator-Muster zum Durchlaufen der Objektstruktur wird vom Framework schon durch die Definition von Methoden innerhalb der Klasse *cSimpleModule* bzw. von Parent- bzw. Hilfsklassen des Frameworks nachgebildet [2] und braucht deshalb nicht gesondert implementiert werden.

Das bedeutet, dass die von u. a. von Bouchard et al. [Bou98] sinnvollen Entwurfsmuster im konkreten Fall bedingt durch gegebenen Funktionalität des Frameworks OMNeT++ [3] aufgrund der soeben beschriebenen Begründungen nicht umgesetzt wurden.

Selvan et al. [Sel04] schlägt in seiner Arbeit ebenfalls einen objektorientierten Ansatz zur Modellierung von Lastflussanalyse-Verfahren vor. Die grundsätzliche Struktur wurde schon in Abbildung 8 dargestellt. Auch in dieser Arbeit wird davon ausgegangen, eine komplett neue Applikation ohne Zuhilfenahme eines Frameworks zu entwickeln. Das Verteilsystem selbst wird damit aus den Kom-

ponenten Bus, Sammelschiene, Last und Nebenwiderstand modelliert. Busse bestehen aus Root-, Fork- und Terminal-Knoten. Sammelschienen können aus Leitungsabschnitten bestehen die wiederum mit Bussen in Verbindung stehen. Übertragungsleitungen, Transformatoren und Umschalter sind hier spezielle Komponenten vom Typ Leitungsabschnitt. Im Vergleich zu in dieser Arbeit gewählten Implementierung lässt sich feststellen, dass die Busse als Leistungsknoten (*Node*) und die Übertragungsleitungen als Leitungsknoten (*Line*) implementiert wurden. Das bedeutet, dass Lasten nicht als eigenständige Objekte mit einem bestimmten Leistungswert, sondern als Sammellast an einem bestimmten Bus modelliert wurden. An diesen Bussen kann sich, wie schon erwähnt, die Leistung zufällig ändern. Die Leistungsknoten haben aufgrund der Definition in der NED-Datei die Eigenschaft eines Start- (Slack-), Verteil- oder Endknotens. Dies ist mit der Implementierung von Selvan et al. [Sel04] vergleichbar. Assoziationen mussten vorerst zwischen den Klassen nicht abgebildet werden, da das Wissen über die Struktur des Netzes respektive den Knotentypen aus der NED-Datei stammt. Im Vergleich zur Arbeit von Selvan et al. [Sel04] werden die Berechnung der Spannungs- und Stromwerte in jedem Iterationsschritt nicht an einem zentralen Objekt (Sammelschienen-Objekt) sondern an den jeweiligen Knoteninstanzen selbst durchgeführt. Die Weiterleitung der an den jeweiligen Knoten berechneten Werte, erfolgt über den bereits beschriebenen Nachrichtenaustausch.

Li et al. [Li04] schlägt wie schon in Abschnitt 1.5.2 erwähnt, ein Framework zur Berechnung von elektrischen Verteilnetzen vor. In dieser Arbeit wurden vor allem die vorgeschlagenen Algorithmen- (Algorithm Layer) und Komponenten-Schicht (Component Layer) näher erläutert. Die in dieser Arbeit entwickelten Komponenten *Node* und *Line* zur Modellierung von Leistungs- und Leitungsknoten besitzen ebenso wie das Zweiknoten-Modell aus Abbildung 9, mindestens zwei Anschlüsse, wobei der linke Anschluss jeweils näher an der Spannungsebene liegt und die rechten Anschlüsse näher an den Nachfolgern bzw. Verbrauchern (siehe auch Abbildung 33). In Abbildung 12 wurde die konzeptionelle Schichtung der Ebenen mit der Angabe des Grades an Wiederverwendung gezeigt. In der folgenden Abbildung 59 wird im Vergleich dazu, die konzeptionelle Zuordnung der Teile durch OMNeT++ [3] gezeigt.

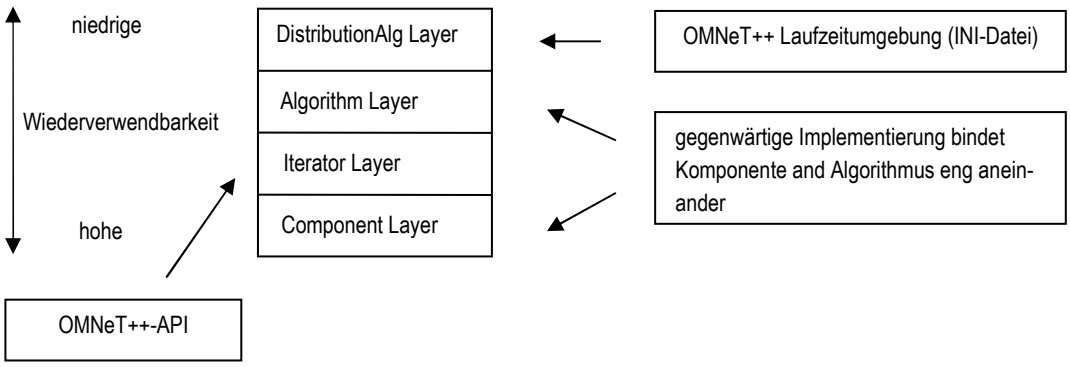


Abbildung 59 – Vergleich Framework Li et al. [Li04] und OMNeT++

Wie in der Abbildung gezeigt, wird in der gegenwärtigen Implementierung des Verfahrens, der Algorithmus – also die Nachrichtenbehandlung – innerhalb der jeweiligen Modultyp-Instanz abgewickelt. Die Komponenten werden über die NED-Datei definiert und miteinander verbunden und die

Aktionen bzw. Reaktionen auf die eingehenden Nachrichten durch die Laufzeitumgebung durch die Kernmethoden *initialize()* und *handleMessage(…)* definiert. Das bedeutet, dass zwischen den zu modellierenden Komponenten und dem Verhalten, welches letztendlich der Berechnung für den jeweilig zu betrachtenden Knoten entspricht – eine enge Kopplung vorherrscht. Das Durchlaufen der Struktur kann durch die schon erwähnten Methoden (u. a. *cTopology*) bewerkstelligt werden. Bezüglich der Möglichkeit die Anwendung zwecks Leistungsvorteil verteilt auf mehreren CPUs durchführen zu lassen, gibt es in OMNeT++ [3] die Möglichkeit dies durch entsprechende Konfigurationseinstellungen in der INI-Datei vorzunehmen (siehe auch [1]). Erwähnt sei das Template-Method-Entwurfsmuster [Fre04] dass sich im Framework OMNeT++ wie in der folgenden Abbildung 60 erläutert, widerspiegelt. Das Muster definiert eine (abstrakte) Oberklasse, welche eine bestimmte

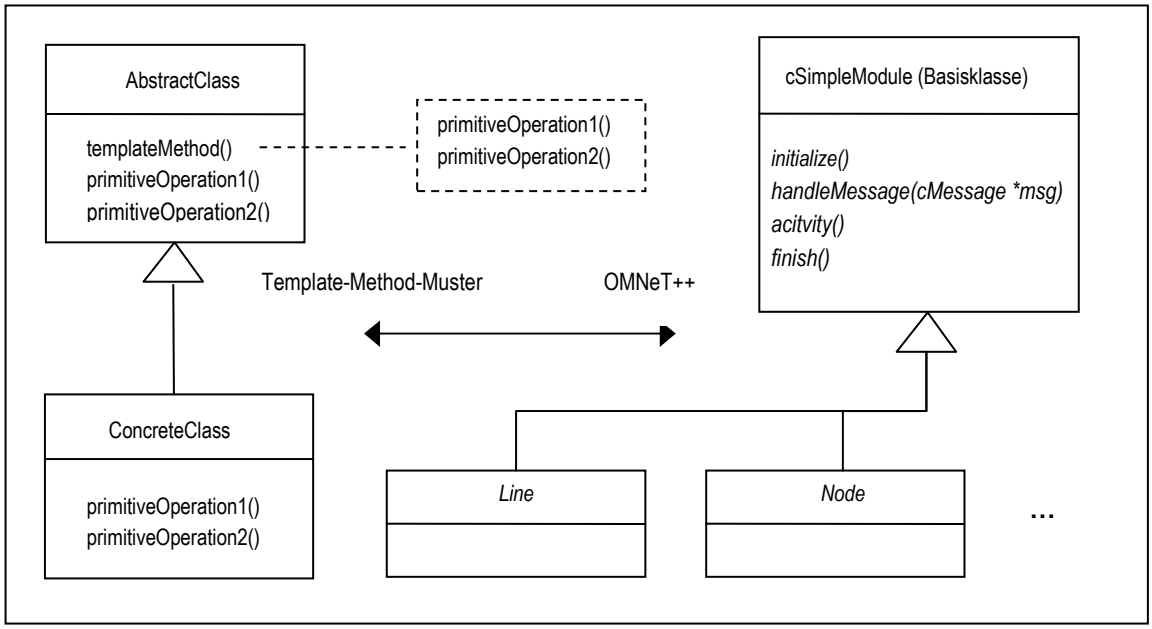


Abbildung 60 – Template-Method-Muster und OMNeT++

Template-Methode definiert, in deren die lokalen aber nicht implementierten (abstrakten) Methoden *primitiveOperation1()* und *primitiveOperation2()* definiert sind. Ein konkrete Klasse kann nun diese Oberklasse ableiten, wobei es die primitiven Operation mit der eigenen gewünschten Funktionalität überschreibt. Die Abfolge der Aufrufe wird allerdings nach wie vor durch die Template-Methode, welche in der abstrakten Basisklasse definiert ist, festgelegt. Damit ergibt sich eine fixe Struktur, wobei die bestimmte Einzelteile die durch primitiven Operationen beschrieben werden, durch eine konkrete Implementierung überschrieben, bzw. an die Wünsche des Anwender angepasst werden können. Im Fall des Frameworks OMNeT++ [3] sind hier die primitiven Operationen die u. a. schon bekannten Methoden *initialize(), handleMessage(…)* usw. Diese werden allerdings von der Laufzeitumgebung in einer ganz bestimmten Reihenfolge aufgerufen bzw. abgearbeitet. Das bedeutet, dass hier aus der Sicht des Frameworks bereits eine bestimmte Flexibilität vorgesehen ist. Im konkreten Anwendungsfall werden die Algorithmen individuell in den primitiven Operationen der An-

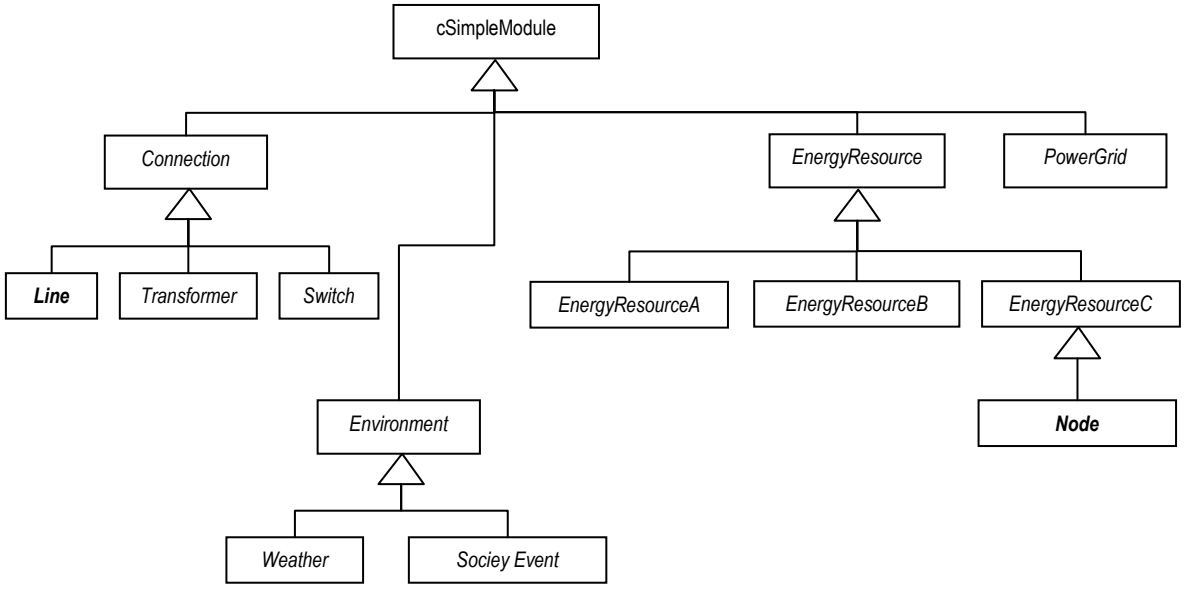


Abbildung 61 - Entwurf Klassenframework für konventionellen Block

wendungsklassen *Line* und *Node* hinterlegt, von wo sie in der schon erwähnten Reihenfolge aufgerufen werden. Der in dieser Arbeit gewählte Ansatz sieht also für jede Komponente, die neu als Modultyp in der NED-Datei hinzugefügt wird, vor, auch eine dementsprechende Modulimplementierung (Algorithmus) durchzuführen. Ein Problem ergibt sich in diesem Fall, wenn verschiedene Modultypen implementiert werden sollen, die einen hohen Grad an ähnlicher Funktionalität bzw. Behandlung von Nachrichten und Algorithmen implementieren. Als Beispiel sei hier z. B. eine Energieressource genannt, die z. B. ein Generator oder eine Last sein kann. Für diesen Fall bietet OMNeT++ [3] die Möglichkeit an, bereits existierende Modulimplementierungen – also die Klassen zu den entsprechenden Einträgen in der NED-Datei – abzuleiten. Damit können in den vererbten Klassen die Funktionen erweitert bzw. redefiniert und damit eine andere bzw. geringfügig andere Funktionalität bereitgestellt werden. Eine Voraussetzung ist hierbei, dass die entsprechenden Algorithmen bzw. Funktionen in Methoden bereitgestellt werden, die von der Template-Methode *handleMessage(…)* aufgerufen werden kann. In Abbildung 61 wird exemplarisch dargestellt, wie ein Modell, das rein auf Vererbung basiert und dem Template-Method-Muster von OMNeT++ [3] Rechnung trägt, aussehen könnte, um den konventionellen Block von DAVIC abbilden zu können. Wie in der Abbildung zu erkennen ist, existieren für dieses Modell die Modultypen *Connection, EnergyResource, PowerGrid* und *Environment*. Diese Typen implementieren die für sie minimale, und für alle davon abgeleiteten Typen, Nachrichtenbearbeitung. Die Nachrichten definieren sich über die Eingangs- bzw. Ausgangsschnittstellen die sich aus der funktionalen bzw. fachlichen Sicht heraus ergeben. Das bedeutet, dass z. B. der Modultyp *EnergyResource* die Werte Frequenz und Spannung als Eingangswert und einen Wert für eine bestimmte Leistung als Ausgabewert produziert. Für diesen konkreten Fall müssten primitive Methoden zur Verarbeitung von Frequenz- und Spannungsnachrichten und für die Generierung von Leistungsnachrichten erstellt werden. Diese könnten dann in den abge-

leiteten Typen *EnergyResourceA, EnergyResourceB, EnergyResourceC* überschrieben bzw. erweitert werden. Die drei Typen wurden im Projekt DAVIC spezifiziert, um ein bestimmtes Verhalten von wiederum von diesen Typen abgeleiteten Ressourcen zu spezifizieren. Nur die Typen B und C erhalten bzw. erhalten und generieren Nachrichten bzw. setzen Aktionen und beeinflussen damit andere Knoten im Netz.

Für Verbindungen (*Connections*) und Umweltfaktoren (*Environment*) könnten ebenso Rohimplementierungen für Standardverhalten bereitgestellt werden und diese von den tatsächlichen Typen erweitert bzw. überschrieben werden. Die bis jetzt entwickelten Typen *Line* und *Node* würden sich vom *Connection*-Typ bzw. *EnergyResource*-Typ ableiten. Wesentlich ist hier anzumerken, dass zwar alle Klassen implementiert sein müssen (Basis und Ableitung), aber nur die tatsächlich verwendeten Komponenten in der NED-Datei vorzusehen bzw. vernetzen sind. Der Vorteil liegt hier also in der Wiederverwendung von Code unter der Voraussetzung, dass sich hinreichend Gemeinsamkeiten zwischen den Typen der Hierarchie finden lassen, und der Möglichkeit einfach neue Komponenten dem Netz hinzufügen zu können. Ein Nachteil der gegenwärtigen Implementierung ist die Abbildung der Nachrichtenbehandlung durch einen Zustandsautomaten mittels üblicher Kontrollstrukturen in den primitiven Methoden (*initialize(), handleMessage(…)*) des Frameworks. Als übliche Kontrollstrukturen werden z. B. *if-then-else*-Strukturen bezeichnet. Das Problem liegt hier in der Schwierigkeit, Änderungen oder neue Anforderung nachträglich dem Code hinzuzufügen, da hier eventuelle unerwünschte Seiteneffekte in der bereits implementierten Nachrichtenbehandlung

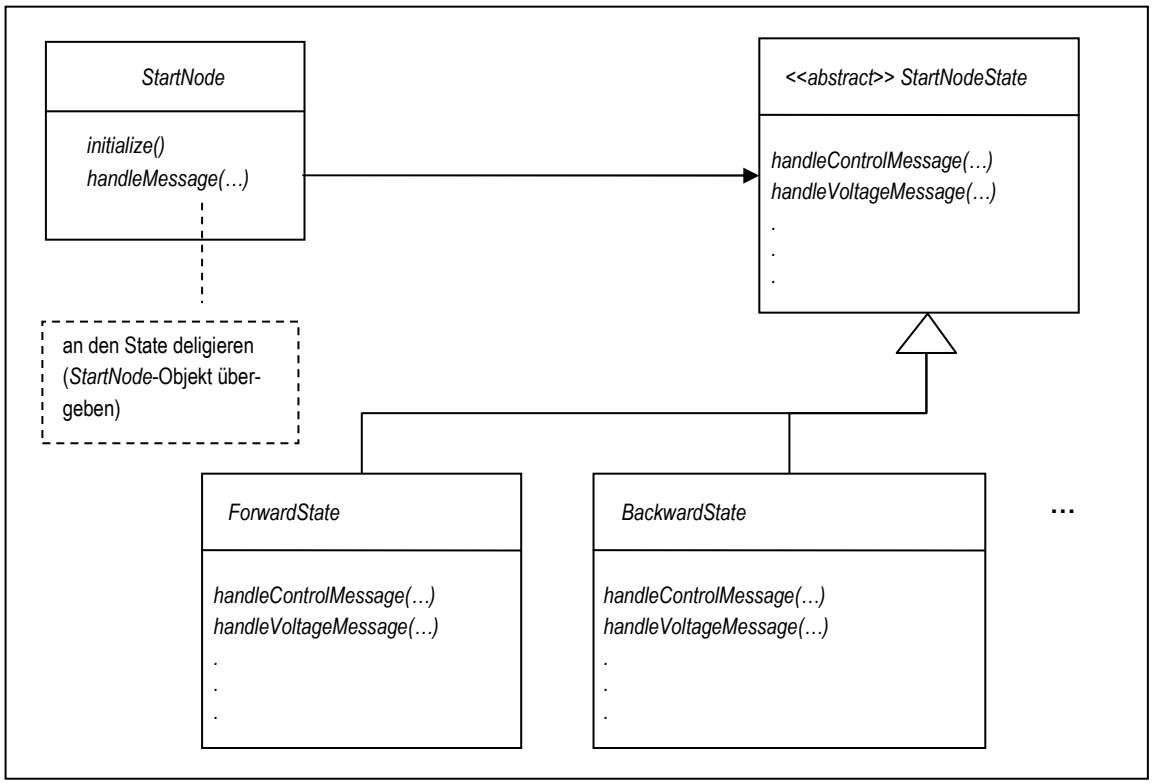


Abbildung 62 – State-Muster für Startknoten

bzw. Implementierung des Algorithmus nicht auszuschließen sind. Auch für die Implementierung eines Zustandsautomaten kann auf Entwurfsmuster zurückgegriffen werden. Das notwendige Muster wird als State-Entwurfsmuster [Fre04] bezeichnet. In der Abbildung 62 wird das Modell des Musters in Form eines UML-Diagrammes für die Implementierung des Startknotens dargestellt. Um die Zuordnung zu den Zuständen übersichtlicher zu machen wird für den *Node*-Knoten kein Parameter für die Zuordnung zum Typ des Knotens – also Startknoten, innerer Knoten oder Endknoten – mehr vorgesehen, sondern die Typen als eigene Modultypen (*StartNode, InnerNode, TerminalNode*) implementiert. Wie in der Abbildung zu erkennen ist, wird die Behandlung der Nachricht auf entsprechende Zustandsklassen (*ForwardState* bzw. *BackwardState)* delegiert. Das bedeutet, dass innerhalb der Methode *handleMessage(…)* zuerst der Typ der Nachricht festgestellt wird und danach die Bearbeitung and die jeweilige Zustandsimplementierung weitergeleitet wird. Im Prinzip ist es auch möglich, von den konkreten Zustandsklassen die Folgezustände bestimmen zu lassen und damit Zustandsübergänge zu realisieren. In der gegenwärtigen Implementierung des Lastfluss-Verfahrens wird dies allerdings nicht benötigt. Die Erleichterung bei diesem Ansatz ist, dass für beliebige Nachrichtentypen die gegebenenfalls zu behandeln sind bzw. Änderungen bei der Berechnung, diese in nur in diesen Zustandsklassen geändert bzw. neue Zustandsklassen hinzugefügt werden müssen. Für konkrete Details dieses Muster betreffend sei auf Freeman et al. [Fre04] verwiesen.

4.4 Laufzeiten und Nachrichtenaufkommen

In diesem Abschnitt wird auf die Laufzeiten und die Beschränkungen die sich durch das Nachrichtenaufkommen in der gegenwärtigen Implementierung ergeben, eingegangen.

In dem gegebenen Beispielnetz aus Abbildung 4 und dessen OMNeT++-Modell [3] aus Abbildung 32 ergaben sich drei bzw. zwei Iterationsschritte für den linken bzw. den rechten Ast des Netzes, bis die Abbruchbedingung (Spannungsdifferenz kleiner Schwellwert) erfüllt wurde. Der Schwellwert wird mit 0,1% der Quellenspannung von 30 kV angenommen. Die Simulation brauchte 5 s bis zur Beendigung, wobei hier insgesamt zwei Total-Simulationsläufe berechnet wurden, da sich Leistungsparameter für einige Knoten geändert haben. Diese Ergebnisse wurde auf einem Pentium-Centrino Rechner mit 1,6 GHz erzielt. Insgesamt wurden 53 Nachrichten während der Total-Simulationsläufe erzeugt. Die Anzahl an erzeugten Nachrichten kann wie folgt abgeschätzt werden. Wie in Abbildung 52 dargestellt werden im Initialisierungsschritt des Verfahrens die Init-Spannungsnachrichten (*VoltageMsg*) vom Startknoten an alle angeschlossenen Äste des Netzes verschickt. Daraus folgt, dass zumindest eine Nachricht generiert wird. Weiters sei der Grad $d_{out}(node_i)$ eines Knotens $node_i$, als jene Anzahl von ausgehenden Kanten bezeichnet, die in Richtung der Blätter des Netzes zeigt. Für die Initialisierungsphase der Ladder Iterative Technique, angewendet auf unser Beispielnetz, ergeben sich daraus die Gesamtanzahl *N* an zu erwartenden Nachrichten zu

$$N = 1 + \sum_{i=1}^{n} (d_{out}(node_i) - 1) \qquad \textbf{(26)}$$

für jeden Iterationsschritt der Initialisierungs- bzw. Vorwärtsschrittphase des Algorithmus, für *n* Knoten. Wie in (26) dargestellt, ist eine Nachricht als fix anzunehmen, da diese vom Startknoten bei der Initialisierung des Verfahrens erzeugt wird (Init-Spannungsnachricht). Die weiteren Nachrichten werden an jedem Knoten, der einen Verzweigungsgrad $d_{out} > 2$ aufweist, für die Anzahl an angeschlossenen Ästen dupliziert. Das bedeutet, dass z. B. im Fall des Knotens $node_1$ welcher einen Grad $d_{out} = 2$ aufweist, eine weitere Nachricht generiert werden muss. Beim Knoten $node_2$ kommt es zu einer weiteren Vervielfachung um eine weitere Nachricht, deshalb werden im Beispielnetz aufgrund der Verteileigenschaft des Startknotens und des Knotens $node_2$, insgesamt drei Nachrichten generiert ($vmsg_1$, $vmsg_2$, $vmsg_3$). Daher ergibt sich die Summe der im Initialisierungs- und Vorwärtsschritt zu erwartenden Nachrichten in (26) aus der Startnachricht plus der Anzahl der angeschlossenen Äste an einem Verteilknoten minus eins für *n* Knoten.

Für den Rückwärtsschritt des Verfahrens verschicken die *m* Endknoten des Netzes nacheinander die *m* Backward-Stromnachrichten (*CurrentMsg*) in Richtung des Startknotens. Mit dem Erreichen eines im Netz vorhandenen Verteilknotens $node_i$ wird diese Anzahl um $d_{out}(node_i)-1$ verringert, bis nur mehr eine Nachricht von jedem Ast des Startknotens an diesem eintrifft.

Die Grenzen der gewählten Implementierung ergeben sich durch die Größe des zu simulierenden Netzes, da die Nachrichtenübermittlung quer durch das gesamte Netz bis zu den Endknoten und die Rückrechnung aus diesen in Richtung des Startknotens durchgeführt werden muss. Aufgrund fehlender Netzdaten mit mehreren hunderten bzw. tausenden Knoten fehlen hier allerdings Messwerte. Netze dieser Größenordnung müssten in diesem Fall in Form einer automatisch generierten NED-Datei erzeugt und simuliert werden, da die manuelle Erstellung derartiger Netzstrukturen nicht praktikabel ist. Dies ist eine Aufgabe für weitere Arbeiten, welche sich in Zukunft mit dieser Thematik auseinandersetzen.

5. Zusammenfassung und Schlussfolgerungen

In diesem Kapitel werden die Arbeit und die Ergebnisse nochmals zusammengefasst und ein Ausblick auf zukünftige Erweiterungen und Arbeiten zu dieser Thematik gegeben. Zuerst werden elektrische Energienetze sowie der Entwicklung in der jüngsten Vergangenheit bezüglich dezentraler Erzeugung und der damit einhergehenden Problemen sowie die daraus für diese Arbeit wesentliche Problemstellung erläutert. Danach werden zusammenfassend die Ergebnisse der Implementierung bewertet. Abschließend wird noch ein kurzer Ausblick auf zukünftige Schritte zur Verbesserung bzw. Erweiterungen gegeben.

5.1 Energienetze – Entwicklung und Problemstellung

Elektrische Energienetze lassen sich in Hoch-, Mittel- und Niederspannungsnetze einteilen. Hochspannungsnetze werden heutzutage üblicherweise zur Energieübertragung über weite Strecken aufgrund deren geringerer Leistungsverluste verwendet. Mittel- und Niederspannungsnetze dienen eher der Verteilung der elektrischen Energie für die Energieverbraucher bzw. Endkunden.

In den letzten Jahren kommt es aufgrund der zunehmenden Liberalisierung der Energiemärkte sowie des weltweit zunehmenden Bedarfs an elektrischer Energie zur vermehrten dezentralen Erzeugung und Einspeisung von elektrischer Energie. Das bedeutet, dass die ursprüngliche stark hierarchische Struktur von Energienetzen durchbrochen wird, d. h. es kann nicht mehr davon ausgegangen werden, dass die Energieeinspeisung nur von oben (strikte Top-Down-Struktur) durchgeführt wird. Vor allem im Bereich der Verteilnetze der Mittel- und Niederspannungsebenen kommt es verstärkt zu Einspeisungen, die oftmals in infrastrukturell wenig erschlossenen Gebieten angeboten werden können. Damit stehen vor allem die Netzbetreiber vor Problemen hinsichtlich der Leitungskapazitäten. Aufgrund der schon erwähnten historischen stark hierarchisch organisierten Netzstrukturen sind die Leitungen oftmals nicht für Einspeisungen in den Verteilästen der Mittel- und Niederspannungsebene ausgelegt. Die Netzbetreiber stehen damit vor dem Problem, die Leitungen verstärken zu müssen, welches eine sehr kostenintensive Variante darstellt. Wird dieser Weg gewählt, werden potentielle Einspeiseanlagen oftmals wirtschaftlich uninteressant. Deshalb wurden hier in der jüngsten Zeit Forschungsprojekte zum Thema „Aktiver Verteilnetzbetrieb“ [Lug08] initiiert, die sich mit der Thematik der flexiblen Spannungsregelung befassen und Restkapazitäten der Leitungen, die sich

aufgrund von Gleichzeitigkeiten von Erzeugung und Verbrauch ergeben können, zu nutzen und damit kostspielige Leitungsverstärkungen für längere Zeit vermeiden helfen sollen. Um die Auswirkungen von Einspeisungen abschätzen zu können, gibt es die Möglichkeit Lastflussanalysen für elektrische Energienetze durchzuführen. Dabei werden die Komponenten der Netze durch Modelle angenähert und mittels den schon erwähnten entsprechenden Verfahren analysiert bzw. berechnet. Als Ergebnis erhält man die Spannungen an den Knoten (Bussen) bzw. Strömen auf den Leitungen.

Generell gibt es schon lange bewährte Methoden der Netzanalyse vor allem für den Bereich der Übertragungsnetze. Dieses sind vor allem matrizenbasierte Verfahren, wie u. a. in dieser Arbeit erwähnte Verfahren von Gauß-Seidel, Newton-Raphson und Fast Decoupled Load Flow. All diese Verfahren bedienen sich im Prinzip der Abbildung der Knotengleichungen in Matrizenform und Lösung der daraus resultierenden Gleichungssystemen nach den gewünschten Unbekannten. Für den Fall elektrischer Verteilnetze, hat sich in der Vergangenheit herausgestellt, dass die erwähnten matrizenbasierten Verfahren, aufgrund deren radialer Netzstruktur und deren Verhältnis zwischen Wirk- und Blindwiderstand der betreffenden Leitungen, eher ungeeignet sind da diese hier schlecht konvergieren. Aus diesem Grund werden für Verteilnetze Vorwärts-/Rückwärtsschritt-Verfahren verwendet, eines davon ist die Ladder Iterative Technique von Kersting [Ker07]. Die Verfahren funktionieren nur mit radialen Netzwerken, in denen also keine Schleifen vorhanden sind. Als Basis können in diesem Fall die Busse bzw. Knoten des Netzes als Leistungsabnehmer (Verbraucher) bzw. Leistungseinspeiser (Generatoren) in komplexer *PQ*-Form ($\underline{S} = P + jQ$) modelliert werden. Die Leitungen werden als komplexe Impedanzen ($\underline{Z} = R + jX$) modelliert. Für weitere Komponenten eines elektrischen Verteilnetzes wie z. B. Transformatoren, Umschalter etc. können ebenso entsprechende Modelle entwickelt werden (siehe Kersting [Ker07]). Für die Details bezüglich der Ladder Iterative Technique sei auf Abschnitt 1.3.4 verwiesen. Diese Diplomarbeit beschreibt die Implementierung dieses Verfahrens für einfache Leistungs- (*PQ*-Form) und Leitungsknoten (komplexe Impedanz) mit Hilfe eines nachrichtenbasierten Ansatzes.

Der nachrichtenbasierte Ansatz ergibt sich aus der Architektur bzw. Implementierung des Projekts DAVIC, in welches das in dieser Arbeit beschriebene Verfahren in Zukunft eingebunden werden soll. DAVIC stellt ein Framework dar, welches es u. a. ermöglichen soll, Geld-, Kommunikations- und Energieflüsse in elektrischen Energiesystemen darzustellen bzw. analysieren und simulieren zu können. Der prinzipielle Aufbau wurde schon in Abbildung 3 dargestellt und in Abschnitt 1.2 beschrieben. Wie schon erwähnt, wird DAVIC mit Hilfe des Open-Source-Frameworks OMNeT++ [3] entwickelt. OMNeT++ [3] erlaubt es Graphenmodelle aufzubauen, wobei die Knoten des Graphen ein bestimmtes Verhalten aufweisen, das bedeutet, die Knoten reagieren auf eingehende Nachrichten und generieren gegebenenfalls neue Nachrichten oder leiten die empfangenen Nachrichten weiter. Weiters können in den Knoten Zustandsvariablen abgebildet werden, somit besitzt jeder Knoten im Netz zu jedem Zeitpunkt einen bestimmten Zustand. Das implementierte Verfahren soll in Zukunft in den konventionellen Block von DAVIC eingebettet werden können. Dieser konzeptionelle Block umfasst die Bereiche Umwelt, Energienetz und Ressource.

5.2 Bewertung der Implementierung

Im folgenden Abschnitt wird die gewählte Implementierung abschließend beleuchtet und die Ergebnisse sowie die Vor- und Nachteile der Lösung aufgezeigt.

Wie schon in den Abschnitten 3.4 bis 3.7 und dem Kapitel 4 eingehend beschrieben und diskutiert, wurden die grundlegenden Komponenten des Netzes als Modultypen *Node* und *Line* in OMNeT++ [3] abgebildet. Der jeweilige Teil der notwendigen Schritte des Verfahrens der Ladder Iterative Technique ist den Modultypen als Zustandsautomat mit Hilfe von typischen Kontrollstrukturen abgebildet. Das bedeutet, dass der *Line*-Modultyp aufgrund der eingehenden Stromnachrichten im Fall des Rückwärtsschrittes den Spannungsfall für den Vorgängerknoten, und im Fall einer Spannungsnachricht im Fall des Vorwärtsschritt des Verfahrens den Spannungsfall für den Nachfolgerknoten berechnet. Die *Node*-Modultypen berechnen in erster Linie die Ströme aus der gegebenen Leistung und der Spannung die sie aus den Nachrichten von den angeschlossenen Leitungen (*Line*-Modultypen) erhalten. Aufgrund der Berechnung des Testnetzes und der mit anderen Verfahren bzw. Paketen ermittelten Ergebnisse, lässt sich die berechnungstechnische Korrektheit des Verfahrens ableiten.

Hinsichtlich der Erweiterbarkeit um andere Komponenten bzw. der Möglichkeit der Einbindung in DAVIC weist die gegenwärtige Implementierung noch einige Schwächen auf, die bereits im vorigen Kapitel 4 unter Abschnitt 4.3 erläutert wurden.

Zusammenfassend lassen sich die Vorteile der gegenwärtigen Implementierung wie folgt beschreiben. Es ist keine Implementierung in C++ zur Abbildung der Netzstruktur bzw. Traversierung der daran beteiligten Objekte notwendig. Dies resultiert aus der Abbildung in OMNeT++ [3]. Eine rein textuelle Beschreibung der Knoten des Netzes innerhalb der NED-Datei ermöglicht es, die Topologie des Netzes zu erzeugen. Es sind keine Entwurfsmuster wie z. B. Composite-Muster o. ä. für die Implementierung notwendig. Diese Vorgangsweise ist für Netze mit wenigen Knoten praktikabel. Die bislang implementierten Knotentypen zur Modellierung der Leistungseinspeisung (*Node*) bzw. -abnahme und Leitungsmodellierung (*Line*) funktionieren korrekt. Ein Netz welches mit diesen Komponenten aufgebaut wird kann einfach simuliert werden und liefert plausible Ergebnisse. Das Template-Method-Muster welches konzeptionell in OMNeT++ [3] umgesetzt ist, erlaubt es mittels des in Abbildung 61 vorgeschlagenen Frameworks den konventionellen Block von DAVIC möglichst modular und erweiterbar zu implementieren

Die Nachteile der Implementierung können wie folgt zusammengefasst werden. Um komplizierte Kontrollstrukturen zu vermeiden, ist es besser auf Alternativen wie das State-Muster (siehe z. B. Abbildung 62) zurückzugreifen. Für umfangreiche Netze ist die Generierung der Netzstruktur in der NED-Datei nicht praktikabel, deshalb erscheint die Anwendung eines Netzwerkgenerators zur Erzeugung von NED-Dateien sinnvoll. Der Netzwerkgenerator BRITE [6] könnte dafür geeignet sein. Die Laufzeit der Implementierung ist aufgrund fehlender Messreihen für umfangreiche Netze noch nicht abschätzbar. Es ist jedenfalls eine Begrenzung durch OMNeT++ [3] bzw. das resultierende Nachrichtenaufkommen zu erwarten. Die gegenwärtige Implementierung ist noch nicht modular genug, die hohe Bindung zwischen Komponente und Algorithmus erschwert im Moment die Einbindung in DAVIC.

5.3 Ausblick

In Anlehnung an die erwähnten Nachteile der gegenwärtigen Lösung lassen sich zukünftige Schritte wie folgt beschreiben. Die Implementierung des Verhaltens der aktuellen als auch noch zu entwickelnder Knoten könnte mittels des State-Musters versucht werden. Komplizierte Kontrollstrukturen sollten dadurch wegfallen und damit die Erweiterbarkeit bzw. Wartbarkeit der Komponenten im einzelnen und damit auch der gesamten Anwendung bzw. des Frameworks erleichtert werden.

Nachdem die Netzstrukturen, also die Modultypen und deren Verbindungen sowie die Definition von zusammengesetzten Modultypen in der NED-Datei vorgenommen werden muss, liegt zur Zeit eine Beschränkung hinsichtlich der Netzgrößen vor, da eine manuelle Erstellung der Strukturen weder über einen einfachen Editor noch durch das vom Framework OMNeT++ [3] bereitgestellte Werkzeug *gned* praktikabel ist. Als Alternative bieten sich hier z. B. der Open-Source-Netzwerkgenerator ***B**oston University **R**epresentative **I**nternet **T**opology G**e**nerator* (BRITE) [6] zum Generieren von Netzwerkstrukturen an. Für diesen Generator existiert bereits eine Modifikation, so dass nunmehr auch Dateien im NED-Format, welches von OMNeT++ [3] verarbeitet werden kann, erzeugt werden können. Die Implementierung des Generators ist offen gehalten, dass heißt, es können eigene Klassen zur Erzeugung von Strukturen von Komponenten und deren Verbindungen implementiert und dem Generator über standardisierte Schnittstellen hinzugefügt werden. Als zukünftige Arbeit müsste hier also eine Klasse zur Erzeugung von elektrischen Verteilnetzen entwickelt werden. Damit wäre es auch möglich, entsprechend große Netze zu generieren und die gegenwärtige Implementierung des einfachen Verteilnetzes durch die Komponenten *Node* und *Line* bzw. noch zu entwickelnde Komponenten, zu simulieren. Laufzeiten und die Beschränkungen von OMNeT++ [3] hinsichtlich der Aufgabe der Lastflussanalyse könnten damit abgeschätzt werden.

Die Entwicklung eines Frameworks zur Simulation von elektrischen Verteilnetzen, welches auf dem Framework OMNeT++ [3] basiert und auch mit den fachlichen Komponenten von DAVIC besser interagieren kann, ist zu empfehlen. Als fachliche Komponenten werden z. B. die Umweltaspekte, Energieressourcen und das Energienetz des konventionellen Blocks von DAVIC bezeichnet. Die Abbildung dieser fachlichen Bereiche auf konkrete Implementierung ergibt dann z. B. die Komponenten *Connection*, *EnergyResource*, *Environment* und *PowerGrid* (siehe Abbildung 61). Damit wäre z. B. die Kommunikation zu Bereichen der Umwelt möglich bzw. könnten eigenständige Ressourcen – Typen von *EnergyResource* – als einzelne Lasten bzw. Generatoren entwickelt und über *Connections* verbunden werden. Die Modellierung als Sammellast (Gesamtleistung $\underline{S}$ als positiven bzw. negativen Wert) würde demnach wegfallen. Das Konzept eines derartigen Frameworks bzw. Vererbungsmodells welches auf dem in OMNeT++ [3] konzeptionell implementierten Template-Muster basiert, wurde schon in Abbildung 61 dargestellt.

Methodenbeschreibung

Common::collectData: Diese Methode ruft die Methode *Common::writeVoltage* für alle im Netz vorhandenen *Node*-Knoten auf. Die Abschlussprotokollierung kann dadurch realisiert werden.

Common::degree: Die Methode liefert den Winkel zwischen dem Real- und Imaginärteil der, der Methode als Parameter übergebenen komplexen Zahl.

Common::getCpxPar: Die Methode liefert eine komplexe Zahl im C++-Stil zurück. Die Methode liest den in der INI-Datei bzw. NED-Datei als Zeichenkette repräsentierten komplexen Wert, wandelt in eine komplexe Zahl um und retourniert diesen.

Common::magnitude: Die Methode liefert den Betrag der, der Methode als Parameter übergebenen komplexen Zahl.

Common::writeCurrent: Die Methode schreibt den Stromwert, welcher auf den Leitungsknoten auftritt, in eine Textdatei. Als Parameter wird der Dateiname, Knotenname, die komplexe Struktur *cpx*, die Lauf-ID der Simulation und der maximale Strom der Leitung erwartet. Verletzungen bezüglich der Strombelastung werden in der Datei vermerkt.

Common::writeVoltage: Die Methode schreibt den Spannungswert in eine Textdatei. Als Parameter wird der Dateiname, Knotenname, die komplexe Struktur *cpx*, die Distanz zum Startknoten in Meter m und die Lauf-ID der Simulation erwartet.

Control::getRunId: Diese Methode liefert die in der Anwendung global vorhandene Lauf-ID als Integer-Wert zurück.

Control::setRunId: Diese Methode setzt die in der Anwendung global vorhandene Lauf-ID. Als Parameter wird ein Integer-Wert erwartet.

Line::calVoltage: Der Methode werden die Parameter Strom $\underline{I}$ und Spannung $\underline{U}$ übergeben. Zuerst wird der Spannungsfall auf der Leitung mittels der durch den Leitungsparameter bestimmten Impedanz $\underline{Z}$ und des übergebenen Stromes zu $\underline{U}_{Line} = \underline{Z}\,\underline{I}$ berechnet. Dieser Wert wird zur übergebenen Spannung hinzuaddiert und als Ergebnis retourniert und berechnet sich damit zu $\underline{U}_{retourniert} = \underline{U}_{Line} + \underline{U}$.

Line::generateVoltageMessage: Liefert einen Pointer auf eine Spannungsnachricht. Übergeben werden der Methode ein Parameter für die Richtung mit der sich die Nachricht durch das Netz bewegen soll und eine komplexe Struktur die die Werte für Spannung und Strom enthält.

Line::handleVoltageMessage: Die Methode behandelt den Fall einer eingehenden Spannungsnachricht. Dies ist im Vorwärtsschritt des Verfahrens der Fall. Berechnet wird also der Spannungsfall auf der Leitung und die Spannung für den Nachfolge-Leistungsknoten. Der Methode wird ein Pointer auf eine Spannungsnachricht als Parameter übergeben.

Line::handleCurrentMessage: Die Methode behandelt den Fall einer eingehenden Stromnachricht. Dies ist im Rückwärtsschritt des Verfahrens der Fall. Berechnet wird also der Spannungsfall auf der Leitung und die Spannung für den Vorgänger-Leistungsknoten. Der Methode wird ein Pointer auf eine Stromnachricht als Parameter übergeben.

Line::recordVector: Diese Methode dient der Aufzeichnung von Knotenwerten im Vektor-Format von OMNeT++ [3].

Line::showMessageValues: Diese Methode zeigt die aktuellen Werte der Struktur *cpx* in der der Methode als Parameter übergebenen Nachricht (*cMessage*) an.

Line::showLineAttributes: Diese Methode zeigt die aktuellen Werte der Struktur *cpx* des Knotens im GUI von OMNeT++ [3] an.

Node::calCurrent: Berechnet den Strom eines Knotens N_i. Übergeben wird der Methode die komplexe Leistung $\underline{S}_i$ und Spannung $\underline{U}_i$ als komplexer Gleitkommatyp. Aus der Formel $\underline{I}_i = (\underline{S}_i / \underline{U}_i)^*$ wird der komplexe Strom am Knoten N_i berechnet und retourniert.

Node::doForward: Liefert den Wahrheitswert True oder False, je nachdem ob ein neuer Simulationsschritt (Vorwärts- und Rückwärtsschritt) durchgeführt werden soll. Handelt es sich beim Leistungsknoten um den Startknoten, so wird bei Eingang einer Spannungsnachricht die aus dem Netz zurückgeschickt wurde, die Differenz zwischen dem neuen Spannungswert und der gegebenen Quellenspannung berechnet. Diese Differenz wird mit dem gegebenen Schwellwert verglichen. Ist die Differenz kleiner dem Schwellwert wird der Wahrheitswert False ansonsten True zurückgegeben. Der Methode wird der neue Spannungswert übergeben.

Node::getDistance: Liefert die Entfernung des betreffenden Knotens vom Startknoten (Slack-Knoten) in der Einheit Meter m.

Node::getFeederId: Liefert den Index des *input gates* an welchem eine bestimmte Nachricht eingangen ist. Als Parameter der Methode wird ein Pointer auf eine Spannungsnachricht erwartet.

Node::getNrOfNodes: Liefert die Anzahl der Knoten (Leitungs- und Leistungsknoten) im gesamten Verteilnetz. Dies wird durch die Verwendung der *cTopology*-Klasse des Frameworks bewerkstelligt [1].

Node::getParam: Liefert die komplexe Struktur mit den Werten für die Spannung und den Strom des betreffenden Knotens.

Node::getPower: Liefert einen neuen Wert für die Leistung des Knotens zur Laufzeit. Retourniert wird ein komplexer Wert, wobei der Real- und Imaginärteil der Leistung sich zwischen 0 und 1 Megawatt MW bewegt. Für die Berechnung des zufälligen Wertes wird auf die Framework-API von OMNeT++ zurückgegriffen und die Methode *uniform(…)* verwendet, die gleichverteilte Zufallszahlen im gegebenen Bereich liefert [1].

Node::generateVoltageMessage: Liefert Pointer auf eine Spannungsnachricht. Übergeben werden den Methoden ein Parameter für die Richtung mit der sich die Nachricht durch das Netz bewegen soll und eine komplexe Struktur die die Werte für die Spannungs- und Stromwerte enthält.

Node::generateCurrentMessage: Liefert Pointer auf eine Stromnachricht. Übergeben werden den Methoden ein Parameter für die Richtung mit der sich die Nachricht durch das Netz bewegen soll und eine komplexe Struktur die die Werte für die Spannungs- und Stromwerte enthält.

Node::generateControlMessage: Liefert Pointer auf eine Kontrollnachricht mit gegebener Richtung, Knoten-ID und Knotennamen, die als Parameter der Methode übergeben werden.

Node::handleControlMessage: Behandelt den Fall einer eingehenden Kontrollnachricht. Der Methode wird als Parameter ein Pointer auf eine Kontrollnachricht übergeben. Der Ablauf wird in Abschnitt 3.4 und Abschnitt 3.5 beschrieben.

Node::handleDuplicates: Behandelt die Verarbeitung von Nachrichten die aus dem Netz zurückgeschickt werden aus der Sicht eines Verteilknotens. Der Methode wird ein Pointer auf eine OMNeT++-Message (*cMessage*) übergeben. Der konkrete Typ wird innerhalb der Methode festgestellt. Der Ablauf wird in Abschnitt 3.4 beschrieben.

Node::handleSelfMessage: Die Methode behandelt die Verarbeitung einer Nachricht, die vom Knoten selbst verschickt wurde. Der Methode wird als Parameter ein Pointer auf eine OMNeT++-Message (*cMessage*) übergeben. Mit Hilfe dieser Methode wird die Änderung eines Leistungswertes realisiert. Details dazu sind in Abschnitt 3.4 beschrieben.

Node::handleVoltageMessage: Die Methode behandelt den Fall einer eingehenden Spannungsnachricht und implementiert die Basisfunktionalität des Knotens. Für die Details sei auf den Abschnitt 3.4.1 und 3.4.2 verwiesen. Der Methode wird ein Pointer auf eine Spannungsnachricht als Parameter übergeben.

Node::recordVector: Diese Methode dient der Aufzeichnung von Knotenwerten im Vektor-Format von OMNeT++ [3].

Node::startTimer: Diese Methode initiiert das Versenden einer *self message* und initiiert damit die Änderung des Leistungswertes des betreffenden Knoten.

Node::sendDuplicates: Diese Methode versendet die ihr übergebene Nachricht an alle Nachfolgeknoten mit Gate-Array-Index > 0, also in Richtung der Endknoten des Netzwerkes. Als Parameter wird ein Pointer auf eine OMNeT++-Message (*cMessage*) erwartet, der korrekte Nachrichtentyp wird innerhalb der Methode bestimmt bzw. erzeugt.

Node::showNodeAttributes: Diese Methode zeigt die aktuellen Werte der Struktur *cpx* des Knotens im GUI von OMNeT++ [3] an.

Node::showMessageValues: Diese Methode zeigt die aktuellen Werte der Struktur *cpx* in der der Methode als Parameter übergebenen Nachricht (*cMessage*) an.

Node::setVectorName: Diese Methode setzt den Namen für alle Elemente innerhalb des Vektor-Formats von OMNeT++ [3].

Literaturverzeichnis

[Ame89] Van Amerogen, R.A.M.: A GENERAL-PURPOSE VERSION OF THE FAST DECOUPLED LOADFLOW, IEEE Trans., Vol. PWRS-4, May 1989, S 760-770

[Bar95] Baron, G.: Einführung in die Mathematik für Informatiker Band2, Zweite verbesserte Auflage, Springer-Verlag/Wien, 1996, S 180ff

[Bou98] Bouchard, D.E. et al.: REPRESENTING POWER DISTRIBUTION SYSTEMS USING OBJECTS AND PATTERNS, Electrical and Computer Engineering, 1998. IEEE Canadian Conference on. Volume 2, 24-28 May 1998, Page(s):846-849 vol.2

[Fre04] Freeman, E. et al.: Head First Design Patterns, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, S 275ff, S 315ff, S 385ff

[Ham97] Hambley, Allan R., Electrial Engineering Principles & Applications, Prentice-Hall, Inc. 1997, Simon & Schuster / A Viacom Company Upper Saddle River, New Jersey 07458, S 800-801

[Ker76] Kersting, W. H., Medive, D.L.: An application of ladder network theory to the solution of three-phase radial load-flow problems, IEEE Conference Paper, paper presented at the IEEE Winter Power Meeting, New York, 1976

[Ker06] Kersting, W. H. et al.: Recommended Practices for Distribution System Analysis, Power Systems Conference and Exposition, 2006, PSCE '06, 2006 IEEE PES, Oct. 29 2006-Nov. 1 2006, Page(s):499-504

[Ker07] Kersting, W. H.: Distribution System Modeling and Analysis, 2nd ed., 6000 Broken Sound Parkway NW, Suite 300, CRC Press, Taylor & Francis Group, 2007, S 6, S 324-333

[Khu06] Khushalani, S., Schulz, N.: Unbalanced Distribution Power Flow with Distributed Generation, Transmission and Distribution Conference and Exhibition, 2005/2006 IEEE PES, May 21-24, 2006, Page(s):301-306

[Kun94] Kundur, P.: Power System Stability and Control, New York, McGraw-Hill, 1994, S 259-269

[Li04] Li, F, Broadwater, R.P.: Software Framework Concepts for Power Distribution System Analysis, Power Systems, IEEE Transactions on, Volume 19, Issue 2, May 2004, Page(s):948-956

[Lug08] Lugmaier, A. und Brunner, H.: Leitfaden für den Weg zum aktiven Verteilernetz, Bundesministerium für Verkehr, Innovation und Technologie, Berichte aus Energie- und Umweltforschung, 13a/2008, S 1-15

[Mok99] Mok, H.M. et al.: Power Flow Analysis for Balanced and Unbalanced Radial Distribution Systems, The Australasian Universities Power Engineering Conference, Darwin Sept 26 -29 1999

[Pal08] Palensky, P. et al.: A simulation platform for distributed energy optimization algorithms, Lawrence Berkeley National Laboratory, CA, USA, 2008, S 1-3

[Sel04] Selvan, M.P., Swarup, K.S.: Distribution System Load Flow using Object-Oriented Methodology, Power System Technology, 2004. PowerCon 2004. 2004 International Conference on, Volume 2, 21-24 Nov. 2004, Page(s):1168-1173 Vol.2

[Shi88] Shirmohammadi, D. et al.: A COMPENSATION-BASED POWER FLOW METHOD FOR WEAKLY MESHED DISTRIBUTION AND TRANSMISSION NETWORKS, IEEE Transactions on Power Systems, Vol. 3, No. 2, May 1988

[Spr03] Spring, E.: Elektrische Energienetze, Bismarckstraße 33, D-10625 Berlin, VDE VERLAG GMBH, Berlin und Offenbach, 2003, S 11-12, S 16-30, S 156, S 198-202, S 205, S 266-268

[Sto74] Stott, B. and Alsac, O.: FAST DECOUPLED LOAD FLOW, IEEE Trans, Vol. PAS-93, May 1974, S 859-869

[Sto00] Stojanovic, D.P., Korunovic, L.: The analysis of load parameters influence on distribution network calculation results, Electrotechnical Conference, 2000. MELECON 2000. 10th Mediterranean Volume 3, Issue , 29-31 May 2000, Page(s): 903-906 vol.3

[Tho03] Thomson, M. et al.: SECONDARY DISTRIBUTION NETWORK POWER-FLOW ANALYSIS, Proceedings of the IASTED International Conference POWER AND ENERGY SYSTEMS, February 24-26, 2003, Palm Springs, CA, USA

[Tho07] Thomson, M., Infield, David G.: Network Power-Flow Analysis for a High Penetration of Distributed Generation, Power Systems, IEEE Transactions on, Volume 22, Issue 3, Aug. 2007 Page(s):1157-1162

[Tre70] Trevino, C.: Cases of difficult convergence in load-flow problems, IEEE Summer Power Meeting, Los Angeles, 1970

Internet Referenzen

[1] http://www.omnetpp.org/doc/manual/usman.html, OMNeT++ Discrete Event Simulation System, Version 3.2, User Manual, S 1, S 5-7, S 11-18, S 20, S 23, S 37-39, S 44, S 87ff, S 149ff

[2] http://www.omnetpp.org/doc/api/index.html, OMNeT++ API (Application Program Interface)

[3] http://www.omnetpp.org, OMNeT++ Discrete Event Simulation System

[4] http://www.mathworks.com/products/matlab/, MATLAB

[5] http://www.maplesoft.com/Products/Maple/, MAPLE

[6] http://www.cs.bu.edu/BRITE/, BRITE, Boston University Representative Internet Topology Generator

[7] http://www.digsilent.de/, DIgSILENT

[8] http://www.era.co.uk/Services/eracs.asp, ERACS