FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Simulation des zeitlichen Störungsverhaltens vollvermaschter Netzwerke am Beispiel kritischer Infrastrukturen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering / Internet Computing

eingereicht von

Christoph Hohenwarter
Matrikelnummer 0326805

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Dipl.-Ing. Dipl.-Ing. Dr.techn. Karl Michael Göschka
Mitwirkung: Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Kurt Höfler

Wien, 9.10.2013

(Unterschrift Verfasser) (Unterschrift Betreuer)

Erklärung zur Verfassung der Arbeit

Verfasser dieser Arbeit:

Christoph Hohenwarter
Unkenberg 56
5091 Unken
Österreich

„Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 9.10.2013, Christoph Hohenwarter

Abstract

The assignment of this diploma thesis was the observation of the characteristics of infrastructure due a disruption by using a computer model. This leaded to the designing of a simulation model, the calculation and presentation of simulation results regarding a graph representing an infrastructure network. Therefore the time-dependent behavior after the occurrence of a disruption is observed.

The examined infrastructure network can be an electric supply network, a water supply network or any other kind of infrastructure network. This diploma thesis observes the simulated behavior of the infrastructure of an Austrian military base after an theoretical incident. The used data in this thesis was delivered by the Austrian Armed Forces. The relationships between the infrastructure elements are modeled using one-dimensional input and output values as there is not made any distinction between the different performance aspects of each infrastructure element.

For the model development a directed graph is used. The links are defined by functions. Also the used data for this thesis are analyzed and the way of collecting this data is discussed in a critical manner.

The major results of this scientific work are the performance charts of the simulated infrastructure of the observed military base during a dysfunction at which assumptions are made of a general nature and those results are interpreted.

Kurzfassung

Aufgabenstellung dieser Diplomarbeit war die Betrachtung des Verhaltens von Infrastruktur beim Auftreten einer Störung anhand eines Computermodells. Daraus ergibt sich die Fragestellung der Simulationsmodellbildung, der Berechnung und der Darstellung der Simulationsergebnisse eines Graphen, welcher ein Netzwerk von Infrastrukturelementen repräsentiert. Diesbezüglich wird das zeitliche Verhalten des Infrastrukturnetzes nach Eintreten einer Störung betrachtet.

Bei solch einem Infrastrukturnetz kann es sich um ein Energieversorgungsnetz, die Wasserversorgung oder um ein ähnliches Netz handeln. In dieser Diplomarbeit wird konkret das Simulationsverhalten der Infrastruktur eines Militärstützpunkts des österreichischen Bundesheeres nach Auftreten einer theoretischen Störung betrachtet, wobei die zugrundeliegenden Daten dieser Studie vom österreichischen Bundesheer geliefert wurden. Hierbei werden die Beziehungen zwischen den Infrastrukturelementen jeweils als eindimensionale Eingangs- und Ausgangswerte in Prozent betrachtet. Auf eine Unterscheidung der Leistungsaspekte der jeweiligen Elemente wird nicht eingegangen.

Das Modell benutzt einen gerichteten Graphen, dessen Kanten durch Funktionen bestimmt werden. Ebenso werden die dieser Diplomarbeit zugrundeliegenden Daten analysiert und deren Erhebung und Art kritisch besprochen.

Zentrales Ergebnis dieser wissenschaftlichen Arbeit sind die Leistungskurven der Infrastruktur des untersuchten Militärstützpunktes unter verschiedensten Störszenarien, wobei auch allgemeine Schlussfolgerungen gezogen werden und die Ergebnisse interpretiert werden.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Teil I

Anfang

KAPITEL 1

Einleitung

Seit Anbeginn der Geschichte versucht der Mensch die Zukunft vorherzusagen. Mit dem Aufkommen der Wissenschaft versuchten die Menschen die Zukunft nicht mehr nur zu erraten. Man begann zukünftige Zustände zu berechnen, sei es nun die Fallgeschwindigkeit eines Apfels in der Physik oder die Position eines Sterns in der Astronomie. Mit der Entwicklung der Computertechnik nahmen die Möglichkeiten enorm zu, solche Berechnungen durchführen zu können. Dies führte zu der Idee, aus vielen kleinen Prozessen, das große Ganze schrittweise zu berechnen. Die Computersimulation war geboren [16].

Im Allgemeinen will man meist die Veränderung von Objekten über die Zeit nach einer Störung von außen simulieren. Ein Beispiel für den Einsatz der Computersimulation, in dieser Art, ist die Berechnung der Materialverformung bei Autos im Falle eines Unfalls [18]. Ebenso war es seit jeher in der Architektur von Bedeutung, wie sich ein Gebäude bezüglich äußerer Störungen verhält. Das allgemein bekannteste Forschungsgebiet ist hierbei die Simulation des Verhaltens von Gebäuden bei Erdbeben [20]. Ein weiteres interessantes Gebiet der Simulation ist das Verhalten von Elektrizitätsnetzen beim Ausfall von Leitungen oder Kraftwerken [28]. Eine solche Simulation betrachtet das Zusammenspiel mehrerer eigenständiger Komponenten.

Von besonderem Interesse für die Allgemeinheit ist hierbei die Simulation der Auswirkungen von Störungen auf die Infrastruktur [29], da solche Störungen meist viele Menschen betreffen und im Extremfall die Stabilität eines Staates gefährden können, wie Plünderungen und Ähnliches bei längeren Stromausfällen gezeigt haben [24]. Forschungsarbeiten gab es hierzu nach dem 11. September 2001 zur Genüge, im speziellen in den Vereinigten Staaten von Amerika [26][28][21]. Hierbei wurden Simulationsmodelle eingeführt, welche die Abhängigkeiten verschiedenster Infrastrukturbereiche zueinander betrachteten. Ebenso wurde die Wechselwirkung zwischen der Infrastruktur und der Wirtschaft betrachtet, wobei ein Infrastrukturausfall sich auf die Wirtschaft auswirkt, und ein Ausfall der Wirtschaft wiederum auf die Infrastruktur wirkt. Dies wird in [21] behandelt. Im Verlauf dieser Forschungswelle im Bereich der Simulation von

Störungen der Infrastruktur wurde ebenfalls ein Framework entwickelt, welches die Simulation der genannten Abhängigkeit von verschiedenen Bereichen der Infrastruktur bei Störungen erlaubt, siehe [7].

Nun stellt sich bei jeder Simulation die Frage, auf welchen Daten basiert sie und wie wurden diese Daten erhoben. Im besonderen ist von Bedeutung, in welcher Form liegen diese Daten vor und wie können sie in das Modell der Simulation einfließen. Mit dieser Problematik beschäftigt sich diese Diplomarbeit eingehend, wobei es das definierte Ziel ist, eine prototypische Software zu entwickeln, welche die Simulation des Verhaltens eines Infrastrukturnetzwerks erlaubt, bei welchem eine Störung bei einem Elemente aufgetreten ist.

Als Beispieldaten für die Simulation bei dieser Diplomarbeit dienen Infrastrukturnetzwerkinformationen, welche vom österreichischen Bundesheer zur Verfügung gestellt wurden. Aus Geheimhaltungsgründen wurden diese Daten in verschlüsselter Form bereitgestellt.

Die Simulation zielt dabei darauf ab, die theoretischen Probleme eines Infrastrukturnetzwerks bei einem möglichen Notfall vor dem Eintreten des Ereignisses zu identifizieren. Es wird somit die Stufe *Planning* der Phase *Pre-Event* aus dem Paper [5] betrachtet.

Diese Diplomarbeit beschäftigt sich mit der Verarbeitung der verfügbaren Daten und der Computersimulation des Störungsverhaltens von Infrastrukturnetzwerken mittels selbstentwickelter Software. Eine genauere Ausformulierung der Aufgabenstellung dieser Diplomarbeit findet sich im Kapitel 2.

KAPITEL 2

Aufgabenstellung

Die Hauptaufgabe dieser Diplomarbeit war es, eine Anwendung zu entwickeln, welche das zeitliche Verhalten eines Infrastrukturnetzwerks nach einer Störung berechnet. Grundlegender Teil der Aufgabenstellung war, dies auf Basis der zur Verfügung gestellten Daten zu erreichen.

Eine weitere Anforderung dieser Diplomarbeit war, die Oberfläche der Anwendung so zu entwickeln, dass sie einfach zu bedienen sei. Die Anwendung sollte für wenig geübte Benutzer benutzbar sein.

Teil der Aufgabenstellung war ebenfalls, den zur Verfügung gestellten Datensatz in den initialen Zustand der Anwendung zu integrieren.

Ebenso wurde gefordert, die Ergebnisse der Simulation in übersichtlicher Form für den Laien verständlich darzustellen. Die genaue Form der Darstellung wurde nicht definiert.

Ein weiterer Anforderungsaspekt war, dass es sich bei der geforderten Software um eine lokale Desktopanwendung handeln sollte. Die Anwendung sollte nicht webbasierend sein. Dies wurde aufgrund von Sicherheitsbedenken bezüglich der verwendeten Daten verlangt.

Ein wichtiger Punkt des Anforderungsprofils war das gewünschte Feature, wonach es möglich sein sollte, das zu simulierende Netzwerk nach Belieben um Knoten erweitern zu können und die Möglichkeit zu haben, diese auch entfernen zu können. Ebenso war gefordert, dem Netzwerk Knotenfunktionen hinzufügen zu können.

Eine Einschränkung der Aufgabenstellung, welche gefordert wurde, war, das Extrapolieren der Knotenfunktionen nicht einzubauen. Dies sollte, aufgrund der nicht abzuschätzenden Auswirkungen bei der Simulation, unterlassen werden.

Eine Randbedingung der Diplomarbeit war es, bei der Entwicklung der Anwendung, nur Bibliotheken zu verwenden, welche kostenlos verfügbar sind. Ausgenommen hiervon war die Verwendung von *Matlab*.

Des weiteren wurde implizit gefordert, das die Simulation in polynomineller Laufzeit durchgeführt werden sollte. Dies wurde bezüglich der Anzahl der Knoten des Netzwerks gewünscht.

KAPITEL 3

Fragestellung der Diplomarbeit

Allgemein formuliert beschäftigt sich diese Diplomarbeit mit der folgenden Frage:

Wie verhält sich ein beliebiges Infrastrukturnetzwerk, wenn eine Störung auftritt?

Aus der Struktur der Daten und der Gesamtfragestellung ergeben sich folgende Teilfragen:

- Wie sind die zur Verfügung stehenden Daten zu interpretieren?
- Wie bringt man die verfügbaren Daten in eine für eine Simulation passende Form?
- Welches grundlegende Modell wählt man?
- Welches Simulationsmodell wählt man?
- Wie lässt sich ein solcher Simulator praktisch umsetzen?
- Wie stellt man die Ergebnisse dar?
- Wie sind die Ergebnisse zu interpretieren?

Diese hier angeführten Fragen wurden im Zuge dieser Diplomarbeit eingehend behandelt und ausgearbeitet.

KAPITEL 4

Kontext der Arbeit

Die Arbeit beschäftigt sich mit der Krisenprävention, wobei die weitere Ausbreitung einer theoretisch bereits geschehenen Katastrophe betrachtet wird. Es gilt die Schwachstellen, beziehungsweise unvorhergesehene Auswirkungen beim Auftreten eines Problems in der Zukunft, im Bereich der Infrastruktur zu erkennen und nach Möglichkeit schon vor dem tatsächlichen Auftreten geeignet vorzusorgen. Es ist somit eine praktische Anwendung der Theorien aus dem Paper [5], jedoch sind etwaige Lösungen für identifizierte Schwachstellen nicht wirklich Teil dieser Ausarbeitung und werden nur am Rande behandelt.

Man darf diese Forschungsarbeit auch kontextfrei sehen. Es würde sich jedes beliebige Netzwerk simulieren lassen, von welchem sich die Daten in der gleichen Art erheben lassen, wie sie vorliegen. Dies heißt, wenn es sich um ein Netzwerk handelt, dessen Relationen sich eindimensional als Prozente über der Zeit erheben lassen, so kann man dieses Netzwerk mittels der erstellten Software simulieren.

KAPITEL 5

Grundlagen und Stand der Technik

5.1 Stand der Technik

Wie in der Einführung bereits erwähnt, wurde zum Thema der Vorhersage der Effekte bei Störungen der Infrastruktur in den Vereinigten Staaten viel geforscht. Dies wurde wahrscheinlich durch Ängste aufgrund des 11. Septembers 2001 vorangetrieben.

Die Arbeit von Rinaldi et al. [26] betrachtet hierbei die Frage der Wechselwirkung verschiedener Infrastrukturnetze zueinander. Es wird dabei auf die Abhängigkeiten zwischen dem Stromnetz, der Wasserversorgung, dem Transportnetz, dem Gasnetz, dem Telekommunikationsnetz und weiteren eingegangen. Dabei wird ein abzusehender *Dominoeffekt* untersucht. Unter dem *Dominoeffekt* versteht man nach [8]:

> „durch ein Ereignis ausgelöste Folge von weiteren gleichartigen oder ähnlichen Ereignissen“

In diesem Fall handelt es sich um eine Kettenreaktion, bei der der Ausfall eines Systems den Ausfall weiterer nach sich zieht, wobei ein System nach dem anderen versagt.

Die Wasserversorgung hängt, aufgrund der notwendigen Wasserpumpen, in hohem Ausmaß vom Stromnetz ab. Das Stromnetz hängt wiederum vom Gasnetz ab, da Strom oftmals mittels Gas erzeugt wird. Das Gasnetz hängt wiederum vom Transportnetz ab, da das Gas oft per Schiff oder ähnlichem zum Gasnetz transportiert werden muss. Das Transportnetz hängt wiederum vom Telekommunikationsnetz ab, da für den Transport, zur Synchronisierung der verschiedenen Transportmittel, Kommunikation notwendig ist [26]. Wie man unschwer erkennen kann, kann der Ausfall eines Infrastrukturnetzwerks zu einem kaskadierenden Umfallen der gesamten Infrastruktur führen.

Eine Arbeit, welche sich mit der Entwicklung eines Frameworks für die Simulation der Wechselwirkungen zwischen Infrastrukturnetzwerken beschäftigt, ist das Paper [7]. Diese Ausarbeitung betrachtet das Simulationsframework *Critical Infrastructure Modeling System (CIMS©)*. Dieses wurde am Idaho National Laboratory (INL) - USA entwickelt.

Es wird hierbei auf das Paper [26] verwiesen, welches vier verschiedene Arten der Beziehungen zwischen den Infrastrukturnetzwerken unterscheidet:

- Physikalische Wechselwirkung
- Informationsabhängigkeit
- Geographische Wechselwirkung
- Logische Wechselwirkung

Die *Physikalische Wechselwirkung* beschreibt eine Wechselwirkung, bei welcher ein tatsächlicher Güteraustausch stattfindet.

Die *Informationsabhängigkeit* beschreibt eine Abhängigkeit, bei welcher kein Materialaustausch stattfindet, jedoch ein Infrastrukturnetzwerk vom anderen von dessen Informationen abhängig ist.

Eine *geographische Wechselwirkung* besteht, wenn ein Umweltereignis die voneinander abhängigen Infrastrukturnetzwerke gleichzeitig in ihrem Zustand verändern kann. Eine solche Abhängigkeit ist meist bei geographischer Nähe gegeben.

Eine *logische Wechselwirkung* tritt dann auf, wenn eine Wechselwirkung zwischen den Infrastrukturnetzwerken besteht, welche weder physikalisch, noch informationstechnisch, noch geographisch ist. Dies mag zum Beispiel ein Gesetz aus der Vergangenheit sein, welches sich auf die betreffenden Infrastrukturnetzwerke ausgewirkt hat.

In dieser Diplomarbeit kann, aufgrund der vorliegenden Daten, nicht explizit zwischen den verschiedenen Beziehungen zwischen den Infrastrukturelementen unterschieden werden. Die Betrachtungsweise der Beziehungen zwischen den Infrastrukturelementen darf als Mischform aller vier Typen von Wechselwirkungen gesehen werden. Der *Dominoeffekt* spielt in dieser Ausarbeitung eine Rolle, wobei dieser nicht zwischen verschiedenen Infrastrukturnetzwerken auftritt, sondern zwischen den Elementen eines Netzwerks.

5.2 Grundlagen des Modells

Netzwerkgraph

Als Grundlage für das Modell wird ein gerichteter Graph ohne Mehrfachkanten benutzt [6]. Diese Diplomarbeit beschäftigt sich mit dem zeitabhängigen Verhalten eines Netzwerks, dessen

Knotenpunkte zueinander Abhängigkeiten aufweisen. Hierbei handelt es sich um ein Netzwerk im Sinne der Graphentheorie.

Knoten

Die Knoten des Graphen repräsentieren hierbei Ressourcen und Infrastrukturelemente. Dies bedeutet, ein Knoten in unserem Graph kann ein Infrastrukturelement oder eine Ressource sein. Hierbei wird jedoch ausschließlich aus theoretischen Gründen unterschieden, um verschiedene Ansätze zu erklären. In der mathematischen Berechnung sind solche Knoten gleichwertig und werden gleich behandelt. Es besteht kein Unterschied in der Art der Berechnung bei unterschiedlichen Knotentypen.

Infrastrukturelement

Unter einem *Infrastrukturelement* versteht man eine physische Einrichtung, welche eine Leistung erbringt, welche für andere Knoten des Graphen von Relevanz ist, wodurch diese Knoten miteinander in Abhängigkeit stehen.

Beispiele für Infrastrukturelemente:

- Elektrizitätskraftwerk
- Ölraffinerie
- Krankenhaus
- Militärische Einrichtung
- Wasserwerk
- Klärwerk

Ressource

Des Weiteren benutzen wir den Begriff *Ressource*. Eine Ressource beschreibt eine Leistung, welche nicht einem konkreten Infrastrukturelement zugewiesen wird. Eine solche Leistung kann beispielsweise ein Rohstoff sein oder eine Dienstleistung.

Beispiele für Ressourcen:

- Wasser
- Strom
- Telekommunikation
- Gas

- Öl

- Lebensmittel

Hierbei entspricht ein Infrastrukturelement oder eine Ressource jeweils einem Knoten im Graphen, beide sind einander gleichgestellt.

Wie der Anwender den Graph modelliert, bleibt ihm überlassen, beziehungsweise hängt in hohem Ausmaß von den zugrundeliegenden Daten ab. Dies bedeutet, wenn keine genaueren Daten über allgemeine Ressourcen existieren, so kann man beispielsweise die Versorgung mit Wasser oder Strom durch Ressourcenknoten modellieren. Dies bedeutet, dass diese Knoten dann Quellen sind und somit nur ausgehende, jedoch keine eingehenden Kanten besitzen. Diese Kanten werden weiter unten in diesem Dokument noch genauer erklärt. Somit ersetzt man die Abhängigkeit eines Infrastrukturelements von einem anderen Infrastrukturelement durch die Abhängigkeit von einem allgemeinen Ressourcenknoten. Als Beispiel sei hier folgendes angeführt: Der Knoten *Krankenhaus* steht in Abhängigkeit zum Knoten *Wasserwerk*. Diese Abhängigkeit des *Krankenhauses* zum *Wasserwerk* wird nun ersetzt durch eine Abhängigkeit vom Knoten *Wasser*.

Die Modellierung des Netzwerks mit Ressourcen-Knoten, anstatt mit Infrastrukturelement-Knoten ermöglicht die Simulation eines allgemeinen Ressourcenausfalls. Hier wäre als Musterbeispiel ein Versagen der Stromversorgung anzuführen. Da das Stromnetz im Allgemeinen zusammenhängend ist, wirkt sich eine Störung in der Stromproduktion auf das ganze Netz aus. Daher empfiehlt es sich, Strom als Ressourcen-Knoten zu modellieren, welcher nun mit allen stromverbrauchenden Infrastrukturelementen verbunden ist. Siehe Abbildung 5.1.

Durch eine solche Modellierung geht jedoch das räumliche Lokalitätsprinzip verloren. Dies bedeutet, wenn man zum Beispiel ein Versagen der Stromversorgung, aufgrund einer Netzunterbrechung modellieren will, würde sich diese normalerweise nur lokal auswirken. Diese Netzunterbrechung würde sich nur auf einen Teil der Infrastruktur auswirken. Dies ist nicht mittels eines Ressourcen-Knotens wie *Strom* möglich. Um die Lokalität einer Störung zu simulieren, muss man eine Ressource mittels mehrerer Infrastrukturelemente modellieren. Siehe Abbildung 5.2.

Graphische Darstellung

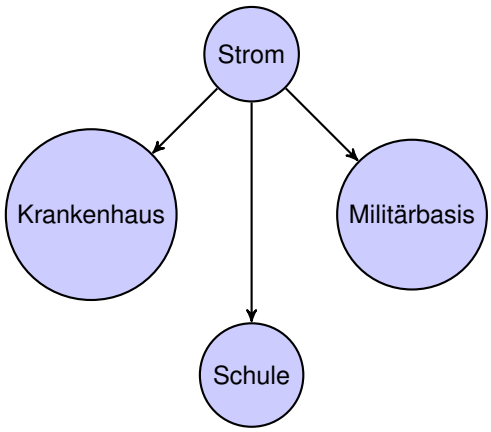

Abbildung 5.1: Strom als Ressource

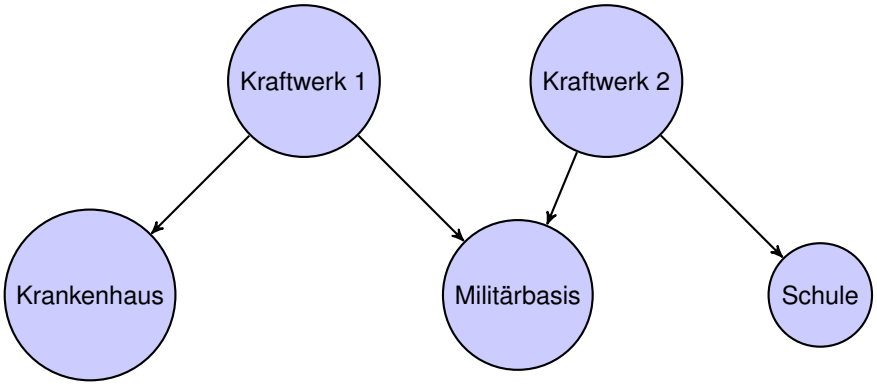

Abbildung 5.2: Modell mittels Infrastrukturelementen für die Stromversorgung

Kanten

Die Kanten in den Graphen 5.1 und 5.2 stellen die Abhängigkeiten der Knoten voneinander dar. Prinzipiell ist das Netzwerk vollvermascht. Jeder Knoten steht mit jedem anderen Knoten in Relation, jedoch werden alle Relationen, welche für die Berechnung keine Bedeutung haben, nicht dargestellt.

Für die Berechnung sind alle Relationen bedeutungslos, welche keine Veränderungen bei benachbarten Knoten erzeugen. Dies heißt, wenn die Funktion von *Knoten A* unabhängig von *Knoten B* ist, so ist keine Kante im Graphen eingezeichnet. Egal welchen Output der Knoten *B* liefert, er beeinflusst nicht den Output von *A*. Wenn das gleiche auch für *A* gilt, wenn der Out-

put von *B* unabhängig vom Output von *A* ist, so ist keine Kante im Graphen eingezeichnet. Die Knoten A und B sind voneinander unabhängig.

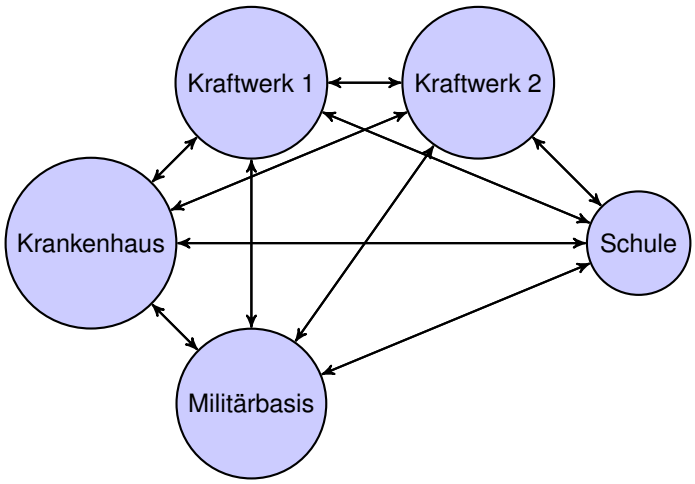

Abbildung 5.3: Vollvermaschte Darstellung des Graphen 5.2

Eine jede Relation zwischen zwei Knoten kann als Funktionsgraph dargestellt werden. Die Relationen des Netzwerks können durch einen Funktionsgraph im dreidimensionalen Raum dargestellt werden. Hierbei hängt der Output des Knoten vom Input und der Zeit ab. Die Einheit des Inputs und des Outputs wird hierbei in Prozent angegeben. Dies bedeutet, dass nur die Ausgabeleistung eines Knotens im Gesamten betrachtet wird. Es wird nicht zwischen verschiedenen Output-Leistungen eines Knotens unterschieden.

Da die Beziehungen zwischen einzelnen Input-Größen eines Knotens nicht feststellbar sind, wäre eine Berechnung mit Relationsgraphen mit verschiedenen Einheiten nicht möglich. Würde man verschiedene Dimensionen in der Berechnung vermischen, so würden die Ergebnisse ihre Vergleichbarkeit verlieren. Daher ist die Betrachtung des Knoten-Outputs als Prozentgröße zielführend.

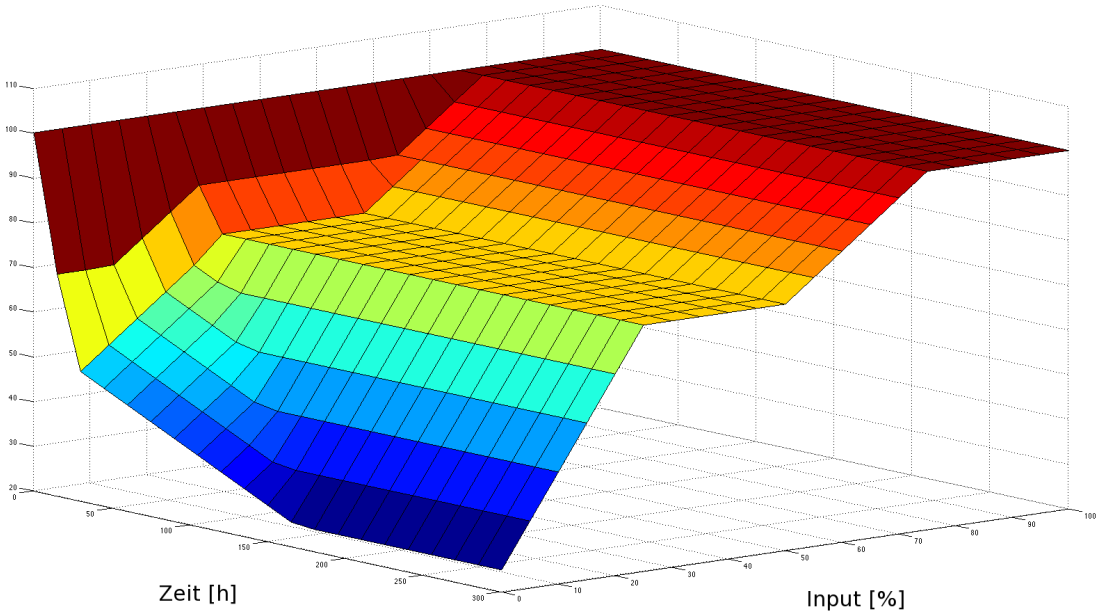

Abbildung 5.4: Graphische Darstellung der Relation zwischen externer IKT-Versorgung und dem Knoten B73

Die Abbildung 5.4 zeigt die Relation zwischen der Ressource *externe IKT-Versorgung* und dem Infrastrukturelement *B73*. Der Graph wurde mittels *Matlab* erstellt. Hierbei werden die Werte linear interpoliert. Wie in der Graphik 5.4 ersichtlich, zeigt eine Achse die Zeit, wobei hier die Einheit in Stunden ausgedrückt wird. Die andere Achse stellt den Input des Knoten *B73* in Prozent dar, welcher dem Output des Knoten *externe IKT-Versorgung* entspricht. Die vertikale Achse liefert die Outputwerte in Prozent des Knotens *B73*.

5.3 Ausgangsdaten

Eigenschaften der Daten

Die vollständigen Daten können dieser Diplomarbeit nicht hinzugefügt werden, da sie der militärischen Geheimhaltung unterliegen. Sie wurden vom österreichischen Bundesheer erhoben und beschreiben militärische Infrastruktur.

Ein Beispieldatensatz:

Input [%]	1 Stunde	1 Tag	7 Tage	30 Tage
100	100	100	100	100
75	100	100	100	100
50	100	75	75	75
25	100	75	75	75
0	100	50	50	50

Tabelle 5.1: Beispieldaten *externe Stromversorgung* $\rightarrow$ *B73*

Die Daten der Tabelle 5.1 sind wie folgt zu interpretieren. Wenn der Knoten *externe Stromversorgung* eine Leistung von 100% erbringt, so liefert der Knoten *B73* nach 7 Tagen Input des Knotens *externe Stromversorgung* einen Output von 100%.
Wie in der Tabelle 5.1 ersichtlich, würde der Knoten *B73* nach 7 Tagen einen Output von 75% liefern, wenn er während diesen 7 Tagen vom Knoten *externe Stromversorgung* nur einen Input von 50% bekommen hätte.

Die Daten liefern also jeweils den Output eines Knotens in Abhängigkeit von *einem* (!) anderen Knoten. Es wird also nicht die Wechselwirkung zwischen den Relationen der einzelnen Knoten betrachtet.

Die kleinste zeitliche Einheit in unserem Modell ist die Stunde. Die zur Verfügung gestellten Daten liefern somit Werte mit dem folgenden Zeitraster:

1 Stunde, 24 Stunden, 168 Stunden und 720 Stunden.

B73 bezeichnet einen Knoten im Netzwerk in verschlüsselter Form. Alle Daten von Knoten, welche keine Ressourcen sind, sondern Infrastrukturelemente, wurden in verschlüsselter Form zur Verfügung gestellt. Dies dient der militärischen Geheimhaltung. Es darf dabei davon ausgegangen werden, dass ein Knoten wie *B73* eine militärische Organisationseinheit beschreibt. Hierbei ist jedoch über die Größenordnung der Knotenpunkte nichts bekannt.

Mögliche Probleme bei der Erhebung der Daten

Die Daten wurden vom österreichischen Bundesheer zur Verfügung gestellt. Über die Art der Erhebung ist dem Autor dieser Diplomarbeit nichts bekannt.
Daher wird man in diesem Teil nur über mögliche Probleme bei der Erhebung spekulieren. Dies soll darlegen, dass die Simulation eines solchen Infrastrukturnetzwerks sich als schwierig gestaltet, da die Erhebung der notwendigen Grunddaten bisweilen unmöglich sein kann.

Grundsätzlich muss man sich vor Augen halten, dass die Vereinfachung der betrachteten Informationen prinzipiell schon dazu führen kann, dass die Simulation Ergebnisse liefert, die nicht der vermeintlichen Realität entsprechen. Den Input und Output der Knoten nur in Prozent zu betrachten vernachlässigt die eigentliche Mehrdimensionalität des Outputs eines Knotens.

Es ist davon auszugehen, dass die Daten nach Einschätzungen von Experten erhoben wurden. Diese Werte mögen nach bestem Wissen und Gewissen abgeschätzt worden sein, jedoch sind solche Werte im Allgemeinen am Ende nur eine Näherung. Wie groß die Abweichung der für diese Diplomarbeit verwendeten Daten ist, lässt sich nicht abschätzen, da, wie bereits erwähnt, keine Informationen über die genaue Erhebung der Daten zur Verfügung stehen.

Im Allgemeinen darf man annehmen, dass die verwendeten Daten sich im Laufe der Zeit verändern werden. Ebenso wird es möglich sein, viele der benutzten Kenngrößen durch organisatorische Maßnahmen zu verändern.
Beispielsweise, um die Abhängigkeit von der Stromversorgung mindestens temporär einzuschränken, wäre die Möglichkeit der Installation von Notstromeinrichtungen, beispielsweise Dieselgeneratoren, welche die externe Stromversorgung bei Eintreten einer Störung, ersetzen. Diese Generatoren können die externe Stromversorgung vollständig oder auch nur partiell ersetzen. Wenn nun ebenso die Versorgung mit Treibstoff nicht funktioniert, so würden natürlich nach dem Verbrauch des Treibstoffs die Dieselgeneratoren ausfallen, und somit die Stromversorgung wiederum zusammenbrechen.

Man kann sehen, dass eine Veränderung einzelner Teilaspekte eines Knotens sich auf andere Kennwerte des Knotens auswirken kann. Daher müsste man nach der Veränderung einer Eigenschaft eines Knotens, auch alle anderen Kennwerte wiederholt betrachten, und die Aktualität dieser Kennwert überprüfen.

An diesem einfachen Beispiel ist ersichtlich, dass Relationen zwischen den verschiedenen Parametern eines Knotens bestehen können. Diese Relationen zu erfassen und realitätsnahe festzustellen, stellt ein praktisches Problem dar. Jedoch war die Erhebung dieser Daten nicht Teil dieser Diplomarbeit. Die Korrektheit der verwendeten Daten wurde als gegeben angenommen.

Um tatsächlich korrekte Daten für ein Infrastrukturelement zu erhalten, müsste man eigentlich den Ausfall der betreffenden Ressourcen physisch simulieren. Hierbei betrachtet man das Infrastrukturelement wie einen elektrischen Schaltkreis, dessen Struktur man nicht kennt, eine sogenannte Blackbox. Nun veränderter man die äußeren Parameter und misst den entsprechenden Output-Wert.

Wenn nun der Knoten, wie im allgemeinen Fall anzunehmen, über einen mehrdimensionalen Leistungvektor verfügt, so müsste man diese verschiedenen Werte, um einen, wie bei unseren Daten, eindimensionalen, prozentuellen Outputwert zu erhalten, mit ihren jeweiligen Gewichtungsfaktoren aufsummieren. In einem solchen Fall stellt natürlich die Festlegung der einzelnen

Gewichtungsfaktoren wiederum eine Wissenschaft für sich da. Dementsprechend gewinnen die erhobenen Daten dadurch an Unschärfe.

Daher würde es sich wahrscheinlich anbieten, in der Praxis die erhobenen Werte des Leistungsvektors eines Knotens zu gleichen Teilen in den Endwert einfließen zu lassen. Dabei würde dann jedoch die Wahl der ausgewählten Leistungswerte eine Rolle spielen. Wenn man nicht alle Dimension der Leistung eines Knotens in den Endwert miteinbezieht, würde die Auswahl der Leistungsdimensionen den errechneten Wert beeinflussen.

Das besagte Messen der Kenngrößen ist jedoch in der Praxis meist nicht möglich. Die auftretenden Einschränkungen wären meist zu gravierend, als dassß sie eine solche Erhebung rechtfertigen würden. Man würde die meist auftretenden Einschränkungen nicht in Kauf nehmen wollen, um solche Daten zu erheben, da die Datenerhebung selbst ja schon dem Ernstfall gleichkommt.

Ein Beispiel für so eine versuchte Störfallsimulation ist das Atomunglück von Tschernobyl [1] . Wie allgemein bekannt ist, ist diese Störfallsimulation nicht wie erwünscht verlaufen. Es kam zu unerwünschten Nebenwirkungen wie einem Supergau.

Ebenso könnte es zu unerwünschten Effekten führen, wenn man zum Beispiel ein Infrastrukturelement wie ein Krankenhaus untersuchen wollen würde. Wenn man eine Messung durchführen wollen würde, wie vorher beschrieben, so wären die möglichen Konsequenzen ethisch nicht vertretbar. Ein Beispielszenario wäre, dem zu untersuchenden Krankenhaus die Stromzufuhr abzudrehen. Für die meisten Krankenhäuser gilt, dass sie dann auf das Notstromsystem umschalten würden. Dies bedeutet im Allgemeinen, die Dieselgeneratoren werden angefahren.
Wenn man nun eine solche Messung durchführt, und die Stromversorgung mindert beziehungsweise einstellt, und das Notfallsystem nicht wie erwartet reagiert, so könnten dabei Menschen zu Schaden kommen. Dies wäre im Allgemeinen höchst unethisch. Alleine die bloße mutwillige Gefährdung wäre aus ethischer Sicht schon äußerst bedenklich.

Somit ist eine messtechnische Betrachtung eines Infrastrukturelements nur dann gerechtfertigt, wenn sich eine Gefährdung von Leib und Leben der betroffenen Personen ausschließen lässt. Des weiteren muss man natürlich eine allgemeine Kosten-Nutzen-Rechnung aufstellen. Es sei zu beachten, in wie weit die möglichen Erkenntnisse aus der Messung der Kenngrößen des Infrastrukturlements die darauf aufbauenden Simulationen verbessert.

Dies führt zu der Kernfrage, wie groß die Unterschiede wären, zwischen den Werten, die man aufgrund von Expertenwissen und Erfahrungswerten erhebt, und den Werten, die man bei einer solchen Messung erhalten würde. Daher stellt sich die Frage, wie präzise die Zahlen sind, die durch das Expertenwissen erhoben wurden. Der erste Schritt der Überlegung wäre, ob man eine Messung durchführen will, welche Güte die Zahlen des Expertenwissens an den Tag legen. Sollte dies nicht abschätzbar sein, so sind die möglichen Risiken einer solchen praktischen Störungssimulation festzustellen.

Oder anders formuliert: Welche möglichen Konsequenzen können im schlimmsten Fall eintreten?

Wie das Beispiel Tschernobyl [1] gezeigt hat, können die Konsequenzen eines solch fehlgeschlagenen Tests gravierend sein, deshalb sollte das Für und Wieder einer solchen Messung gründlich abgewogen werden.

Ein Messvorgang ist nichts anderes als die praktische Simulation einer Störung, eine künstliche Reduktion der Eingangswerte eines Knotens. Dieser Fall sollte natürlich vermieden werden, und kann nur gerechtfertigt werden, durch signifikant verbesserte Daten, welche für eine spätere Simulation mit hoher Wahrscheinlichkeit von Relevanz sind.

Brandschutz- und Katastrophenübungen entsprechen diesem praktischen Simulieren einer Störung, wobei bei diesen Übungen der Ablauf bei einem solchen Notfallszenario bereits vorgegeben ist.

Wie kann so eine Messung aussehen?

Bei einer solchen Messung würde man einen Eingangswert des Knotens linear über die Zeit verändern und die Auswirkungen auf den Ausgangswert messen. Hierbei stellt sich die Frage, wie genau man den Leistungswert, den Output eines Knotens quantisieren kann.

Sei es nun möglich, die Output-Werte eines Knotens zu quantisieren und die Input-Werte gezielt zu steuern, so lässt sich wie bei einem gewöhnlichen elektrischen System, eine Messkurve aufzeichnen.
Als Beispiel senken wir die Stromversorgung zum Zeitpunk t=0 von 100% auf 75%. Danach zeichnen wir den weiteren Verlauf des Output-Werts über die Zeit auf. Dies würde im allgemeinen Fall so lange geschehen, bis sich der Output des Knotens auf einen konstanten Wert eingeschwungen hat, wobei für die weiteren Berechnungen in dieser Diplomarbeit von einem monoton fallenden System ausgegangen wird. Dies ist jedoch für die Datenerhebung nicht relevant. Bei der Erhebung muss lediglich bei der Erfassung eines Messpunktes ein stabiles Niveau erreicht worden sein, wodurch ein dahingehender Messfehler minimiert werden kann.

Prinzipiell muss hier festgehalten werden, dass man über Daten der einzelnen Knoten über den jeweiligen Zeitraum verfügen sollte, über welchen man Berechnungen mit der Simulation durchführen will. In der Simulation selbst ist keine Extrapolation der Daten vorgesehen.
Dies bedeutet, dass der Knoten mit dem geringsten Informationsstand das Maximum der zeitlichen Simulation bestimmt. Wenn nun zum Beispiel ein Knoten über Daten verfügt mit dem höchsten Zeitwert von 240 Stunden, dies entspricht 10 Tagen, so wird die Simulation nur bis zum Wert von 240 Stunden Berechnungsergebnisse liefern.
Daher stellt sich die Frage, wie weit in die Zukunft man rechnen will. Wenn über diese zukünftigen Zeitpunkte keine Informationen der Knotenpunkte vorliegen, so ist auch eine Simulation

der Abhängigkeiten nicht sinnvoll, weil das berechnete Ergebnis mit hoher Wahrscheinlichkeit sehr von der Realität abweichen würde.

Bei den zur Verfügung gestellten Daten, verfügt diese Diplomarbeit über ein Zeitfenster von 720 Stunden. Dies entspricht 30 Tagen. Dies bedeutet, es ist die Simulation des Netzwerkverhaltens für einen Zeitraum von 30 Tagen möglich. Würde man Berechnung über diesen Zeitraum hinaus anstellen, so würde man keine Ergebnisse bekommen, da die Daten nicht extrapoliert werden. Man müsste weitere Daten erheben, um Berechnungen über die 30-Tage-Grenze hinaus anzustellen.

Teil II

Hauptteil

KAPITEL 6

Softwaredesign

6.1 Softwarestruktur

Prinzipiell kann man die entstandene Software in die folgenden fünf Bereiche unterteilen:

- Dateneingabe
- Externe Konfiguration
- Simulationskonfiguration
- Berechnung
- Darstellung der Simulationsergebnisse

Dateneingabe

Die Dateneingabe beschreibt jenen Abschnitt der Software, welcher die Erfassung der Daten ermöglicht. Hierbei werden die unter 5.3 beschriebenen Daten in die Datenbank eingefügt. Dem System können beliebig viele Knoten hinzugefügt werden. Ebenso kann der Zeitraster für alle Knoten verändert werden. Das zu berechnende Netzwerk ist somit flexibel erweiterbar. Dies entspricht einer Grundanforderung an die Software.

Die Daten wurden mittels dem im Kapitel 9 beschriebenen Entwicklungstool aus den zur Verfügung gestellten *Microsoft* Excel-Dateien ausgelesen und in ein Format zur weiteren Verarbeitung gebracht. Bei der erstmaligen Ausführung der Anwendung werden diese Daten in die Datenbank eingefügt. Dies könnte man sozusagen als Installationsvorgang bezeichnen, wobei es kein eigens ausführbares Installationsprogramm gibt.

Externe Konfiguration

Die externe Konfiguration besteht aus einer Datei namens *config.properties*, welche sich im Ausführungsverzeichnis der Anwendung befindet. Diese Datei wird beim erstmaligen Ausführen der Anwendung erstellt. Die Datei *config.properties* kann mittels eines gewöhnlichen Texteditors bearbeitet werden, und bietet die folgenden Konfigurationsmöglichkeiten. Der Wert nach dem Gleichheitszeichen beschreibt den Standardwert nach der Installation.

log level=*OFF* Der Log-Level setzt fest, wie viel Logging-Informationen von der Anwendung gespeichert werden. Diese werden im Ausführungsverzeichnis in der Datei *log.html* gespeichert. Hierbei wird HTML als Format genutzt. Die Log-Datei kann mittels eines gewöhnlichen Browsers betrachtet werden.

log level kann die folgenden Werte annehmen:

- OFF
- FATAL
- ERROR
- WARN
- TRACE
- ALL

Hierbei ist die Liste in der Menge der geloggten Daten aufsteigend. Für den Produktivbetrieb sollte man das Logging ausschalten. Es empfiehlt sich nur zu loggen, wenn Probleme bei der Ausführung der Anwendung auftreten. Dann kann mit Hilfe der aufgezeichneten Informationen das Problem festgestellt werden.
Ein Problem, welches sich so eruieren lassen sollte, wären zum Beispiel mangelhafte Daten, welche der Software es nicht ermöglichen, die notwendigen mathematischen Funktionen ausreichend zu interpolieren. Dies würde nicht zu einem Fehler im Programm führen. Die Berechnungen würden für alle Punkte durchgeführt werden, für welche eine Berechnung möglich ist. Wenn Punkte der Kurve nicht berechnet werden können, so werden diese am Ende einfach nicht in den Ergebniskurven dargestellt.
Wenn nun solche Kurvenwerte nicht berechnet werden können, so kann die Ursache dafür den Log-Dateien entnommen werden.

show datapoints=*FALSE* Diese Option bestimmt, ob die berechneten Werte im Ergebnisgraphen in der Kurve als Datenpunkte angezeigt werden sollen. Damit kann zwischen den berechneten Ergebniswerten und den in der Ergebniskurve interpolierten Werten unterschieden werden. Wenn die Datenpunkte nicht explizit eingezeichnet werden, kann der Benutzer nicht zwischen berechneten Kurvenpunkten und interpolierten Kurvenpunkten unterscheiden.

Führt man die Berechnung jedoch mit kleinem Δt (siehe 7.5) durch, so empfiehlt es sich nicht, die errechneten Kurvenpunkte explizit zu zeichnen, da sich sonst eine Überlappung der gezeichneten Quadrate ergibt und die Kurve optisch nicht mehr ansprechend wirkt. Ebenso darf bei kleinem Δt angenommen werden, dass die interpolierten Werte der Ergebniskurve den realen Werten sich annähern und die Abweichung gegen Null geht.
Wenn man Δt auf Eins setzt, so wird die Ergebniskurve nicht mehr interpoliert und es finden sich keine interpolierten Punkte mehr im Graphen, sondern nur mehr berechnete Punkte.

hide matlab=*TRUE* Diese Option bestimmt, ob die Berechnungen der Simulation, welche von der Anwendung mittels Matlab durchgeführt werden, mittels minimiertem Matlab gemacht werden sollen. Die Möglichkeit Matlab sichtbar von der Anwendung starten zu lassen dient wiederum der Fehlerfindung. Sollte ein Fehler auftreten, so kann dieser in Matlab sichtbar sein. Für den Produktivbetrieb ist es nicht notwendig Matlab sichtbar auszuführen. Mit der Standardeinstellung erfährt der Anwender nichts vom Gebrauch der *Matlab*-Engine. Diese wird vollständig im Hintergrund ausgeführt.

close matlab=*TRUE* Diese Option bestimmt, ob Matlab nach der Berechnung geschlossen werden soll. Dies ist wiederum ein Mittel, um Probleme bei der Berechnung festzustellen. Diese Option ist meist hilfreich in Kombination mit der vorher genannten Option *hide matlab*. Wenn ein Fehler bei der Berechnung auftritt, welcher in einer Fehlermeldung in der Matlab-Umgebung resultiert, dann kann man diesen Fehler am Besten analysieren, wenn man in der weiter geöffneten Matlab-Umgebung verschiedene Befehle manuell testet. So lässt sich dann der Ursprung einer Fehlermeldung genauer erheben. Dies ist jedoch tendenziell nur fortgeschrittenen Anwendern zu empfehlen, da zu diesem Zweck Kenntnisse der Matlab-Befehlsstruktur notwendig sind und meist auch erweiterte mathematische Kenntnisse benötigt werden.

spline interpol=*FALSE* Diese Option legt fest, wie die resultierenden Kurven der Simulation gezeichnet werden sollen. Als Initialeinstellung wird der auszugebende Graph mit linearer Interpolation gezeichnet. Dies bedeutet, die errechneten Funktionspunkte werden jeweils mit einer einfachen Linie verbunden, um so eine stetige Ergebniskurve zu erhalten. Wird *spline interpol* auf *TRUE* gesetzt, so werden die errechneten Kurven mittels Spline-Interpolation [17] vervollständigt.

Simulationskonfiguration

Im Abschnitt der *Simulationskonfiguration* werden die Eckdaten der Simulation bestimmt. Hierzu gehört die Wahl eines initialen Störungsknotens. Dies ist der Knoten, welcher eine Störung aufweist, und dahingehend nicht 100% Leistung bringt. Ebenso wird bei der Simulationskonfiguration der initiale Leistungslevel dieses nicht voll funktionierenden Knotens festgelegt. Dieser Initialoutputwert kann nun zwischen 0% und 99% liegen und kann in 1%-Stufen eingestellt werden. 0% entspricht dahingehend einem Knoten, welcher keine Leistung von Anfang an erbringt. Dies entspricht einem Totalausfall.

Ebenso kann Δt bei der Berechnung eingestellt werden. Hierbei ist eine zeitliche Auflösung von einer Stunde als Minimum vorgesehen und ein Maximum von 50 Stunden. Diese Einstellung wirkt sich massiv auf die notwendige Rechenzeit aus. Hierbei ist eine Granularität von einer Stunde meist nicht zu empfehlen, da dies meist zu langen Rechenzeiten führt. Ebenso ist dies nicht unbedingt notwendig, da bei der Darstellung der Daten die fehlenden Werte der Berechnung interpoliert werden.

Berechnung

Für die Berechnung werden zuerst die Daten aus der Datenbank gelesen. Diese werden dann in eine brauchbare Form gebracht, um in der Anwendung intern weiter verarbeitet werden zu können. Dies heißt, die Daten werden in eine Funktion überführt. Dann wird die Berechnung unter Berücksichtigung der Simulationskonfiguration durchgeführt. Anschließend werden die Simulationsergebnisse in ein Datenformat konvertiert, welches die darauf folgende Darstellung ermöglicht. Die Berechnung selbst wird noch ausführlich in 8.2 erklärt.

Darstellung der Simulationsergebnisse

Nach der Berechnung werden die Simulationsergebnisse präsentiert. Diese werden in Form von zweidimensionalen Graphen über den zeitlichen Verlauf dargestellt. Hierbei ist auf der horizontalen Achse die Zeit aufgetragen und auf der vertikalen Achse der Output des jeweiligen Knotens. Die Kurven werden interpoliert. Wie bereits beschrieben, können die betreffenden Kurven auf zwei verschiedene Arten interpoliert werden. Entweder mittels linearer Interpolation oder mittels kubischer Spline-Interpolation [17]. Dies hängt von der externen Konfiguration ab.

6.2 Klassendiagramme

Die UML-Diagramme (Unified Modeling Language) wurden mittels Visual Paradigm erstellt [23]. Sie entsprechen dem UML 2.0-Standard [15]. Alle Klassen und Klassenmethoden wurden in der Anwendung in Englisch benannt.

Plain Old Java Objects

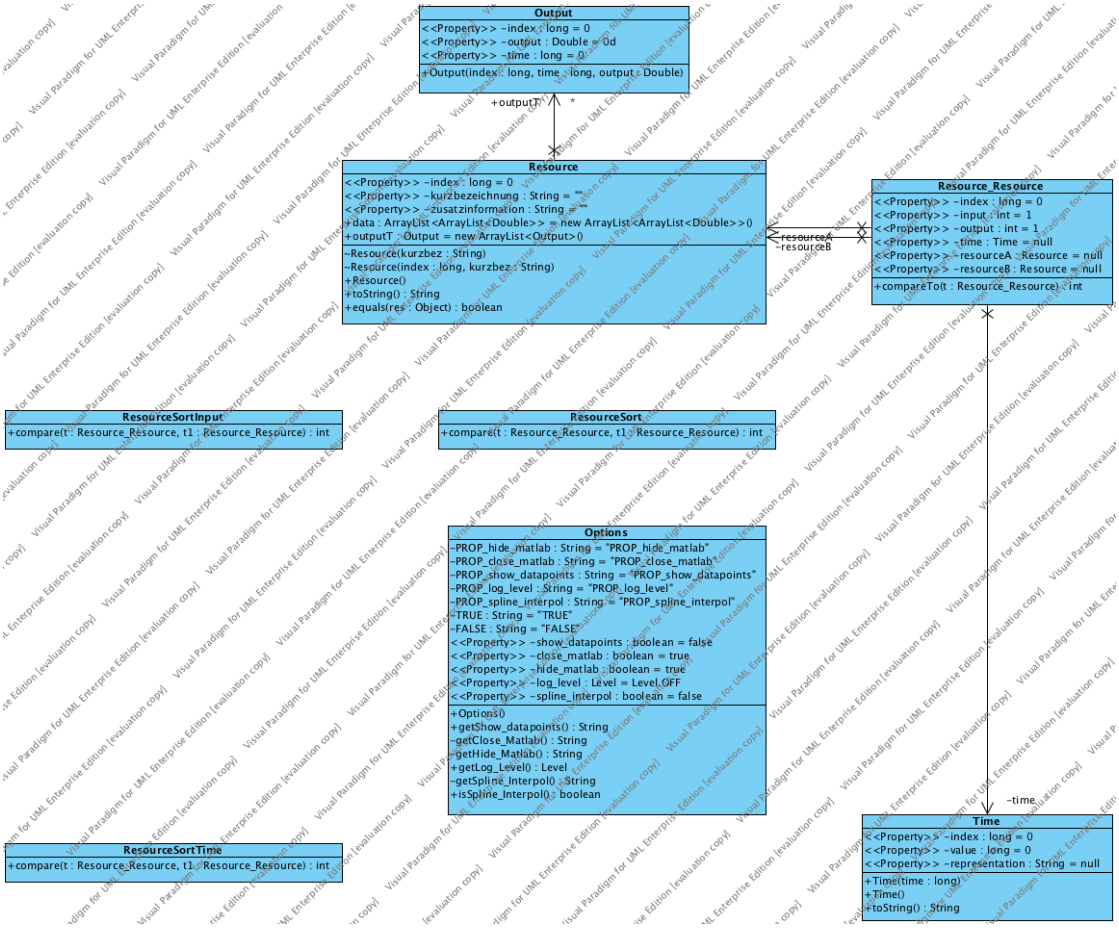

Abbildung 6.1: Klassendiagramm der *Plain Old Java Objects (POJOs)*

Dieses Klassendiagramm zeigt die sogenannten *Plain Old Java Objects (POJOs)*. Diese stellen die einfachen Objekte dar, welche für den Datentransfer innerhalb der Software genutzt werden.

Resource Die *Resource*-Klasse ist hierbei äquivalent zu den in 5.2 genannten Knoten beziehungsweise Infrastrukturelementen. Dahingehend sind die Begriffe *Instrakuturelement*, *Knoten* und *Ressource (Klassenname: Resource (engl.))* Synonyme füreinander und sind im Allgemeinen in dieser Diplomarbeit als gleichwertig anzusehen. Die Arrayliste *data* stellt hierbei eine Liste von Listen dar, welche die Werte beinhaltet, die während der Simulation als Zwischenergebnisse auftreten. Die Klasse *Resource* referenziert die *Output*-Klasse. Diese wird im folgenden beschrieben.

Resource_Resource Die *Resource_Resource*-Klasse beinhaltet zwei *Resource*-Objekte mit der Eigenschaft *Time*. Es werden mit dieser Klasse somit zwei Ressourcen in Relation gesetzt unter der Zeiteigenschaft. Eine ArrayList von *Resource_Resource*-Objekten stellt somit die klassische N:M-Tabelle einer relationalen Datenbank dar, mit der Erweiterung um die Eigenschaft des Zeitwerts als *Time*-Objekt.

Time Die Klasse *Time* modelliert Zeitpunkte in der Umgebung. Hierbei wird jeder Zeitpunkt in der Einheit von Stunden angegeben. Ebenso wird jeder Zeitpunkt, welcher in der Datenbank persistiert wurde, durch eine Zeichenkette dargestellt. Als Beispiel seien hier 24 Stunden angeführt, welche als "1 Tag"dargestellt werden. Die vorab gegebenen Zeitwerte und ihre zugehörigen Zeichenketten sind unter 5.3 angegeben.

Output Ein Objekt der *Output*-Klasse repräsentiert einen errechneten Ergebniswert einer Ressource zu einem bestimmten Zeitpunkt. Hierbei wurde aus Gründen der Laufzeitoptimierung, und um den Kernalgorithmus 8.2 einfach zu halten, auf die Verknüpfung mit der *Time*-Klasse verzichtet.

Ebenso sind in diesem Klassendiagramm die drei Hilfsklassen *ResourceSort*, *ResourceSortInput* und *ResourceSortTime* enthalten.

ResourceSortTime Die Klasse *ResourceSortTime* stellt eine Implementierung des Interface *java.util.Comparator* zur Verfügung. Diese Klasse wird benutzt um eine Sortierung von Objekten des Typs *Resource_Resource* durchzuführen. Mittels der Klasse *ResourceSortTime* kann eine ArrayList [22] nach der Zeit sortiert werden, wobei die Klasse *Time* von *Resource_Resource* referenziert wird.

ResourceSortInput Die Klasse *ResourceSortInput* stellt ebenfalls eine Sortiermöglichkeit für eine ArrayList [22] von *Resource_Resource*-Objekten zur Verfügung. Hierbei werden die Objekte jedoch nach dem Inputwert sortiert. Der Inputwert entspricht dem Leistungswert in % einer eingehenden Kante in einen Knoten. Hierbei wird in der Anwendung eine ArrayList von *Resource_Resource*-Objekten sortiert, welche mit den Daten 5.3 gefüllt wurde. Hierbei geht es somit nicht um ArrayListen, welche die interpolierten Daten darstellen, sondern nur um die Grunddatensortierung.

ResourceSort Die Klasse *ResourceSort* stellt eine Sortiermöglichkeit für ArrayListen zur Verfügung, wobei hierbei eine ArrayList von *Resource_Resource*-Objekten primär nach der Zeit sortiert wird und in zweiter Dimension nach dem Inputwert, wenn die zwei zu vergleichenden *Resource_Resource*-Objekte den gleichen Zeitwert aufweisen. Hierbei werden die Objekte bei gleichem Zeitwert nach dem Inputwert aufsteigend gereiht.

Die Klasse *Options* enthält alle in 6.1 beschriebenen Konfigurationsparameter und stellt diese in der Anwendung dann als Objekt zur Verfügung. Hierbei benutzt die Klasse *Options* die Java-Klasse *java.util.Properties* um die Optionen zu verarbeiten.

Controller-Klassen

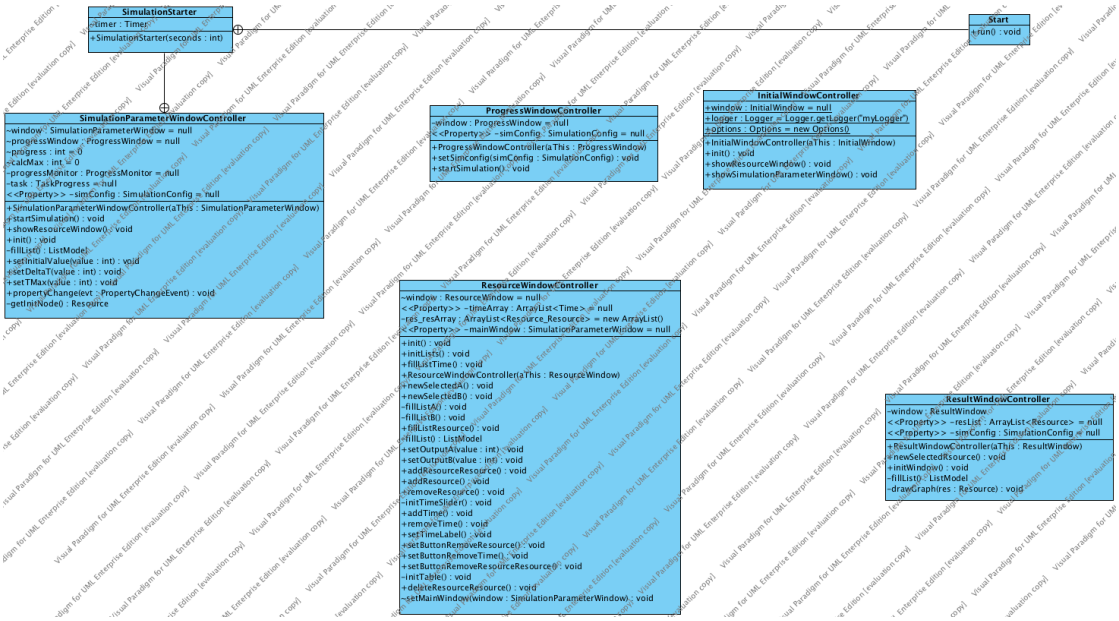

Abbildung 6.2: Klassendiagramm der Controller-Klassen

Die Klassen des Pakets *Controller* enthält alle Klassen, welche die Funktionalität der graphischen Oberfläche implementieren. Diese Klassen enthalten die Logik der Anwendung. Durch die *Controller*-Klassen wird eine Trennung der graphischen Oberfläche von den Logik-Abläufen erreicht. Dadurch lassen sich Änderungen in der graphischen Oberfläche einfacher durchführen. Die Events der Oberfläche rufen hierbei die Methoden in der *Controller*-Klasse auf. Diese Methoden greifen nun wiederum mittels der *Data Access Objects (DAOs)* auf die Datenbank zu oder auf andere persistierte Daten.

Durch diese Trennung der Oberflächenklassen, welche meist durch *JFrame*-Objekte des *Swing-Frameworks* repräsentiert werden, ist es möglich, Steuerelemente der Oberfläche auszutauschen, und dabei die Logik der *Controller*-Klassen nicht verändern zu müssen. Ebenfalls wird dadurch erreicht, das sich Änderungen in der Datenbankstruktur nicht auf den Code in der Oberfläche auswirken.

Jedes Fenster verfügt jeweils über eine eigene *Controller*-Klasse, wobei die *Controller*-Klassen jeweils nach dem Namen des Fensters und dem Suffix *Controller* benannt wurden.

Simulation-Klassen

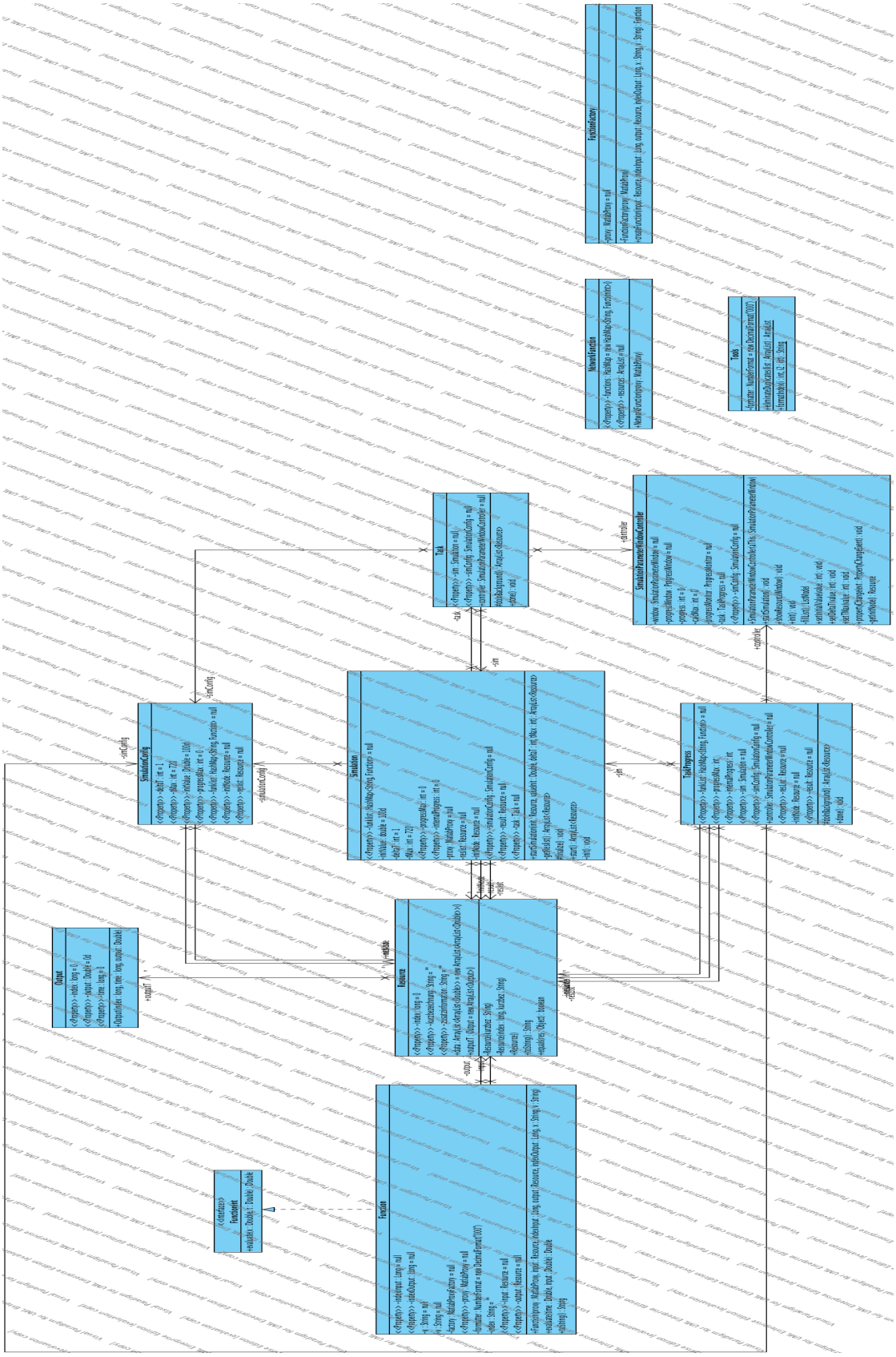

Abbildung 6.3: Klassendiagramm der Klassen des Pakets *simulation*

Die Abbildung des Pakets *Simulation* enthält die für die Berechnungen notwendigen Klassen. Ebenso sind verschieden *Plain Old Java Objects* enthalten, welche von den *Simulations*-Klassen benutzt werden.

Simulation Die Klasse *Simulation* beinhaltet den Algorithmus zur Simulation. Ebenso ist diese Klasse mit den notwendigen Datenstrukturen verknüpft welche für die Berechnung notwendig sind. Der Algorithmus wird im Kapitel 8.2 noch ausführlich erklärt.

SimulationConfig Die Klasse *SimulationConfig* beinhaltet die notwendigen Parameter für die Simulation. Diese Parameter werden unter 6.1 beschrieben.

FunctionInt stellt das Interface für die Funktionen dar. Es fordert die Methode *evaluate*, welche das Ergebnis der mathematischen Funkion aus dem Graph mittels *Matlab* liefert. Diese Methode entspricht der *Matlab*-Methode *evaluate*. Eine Beispielkurve einer solchen Funktionsevaluierung wird in 5.4 dargestellt.

Function stellt die Implementierung des Interface *FunctionInt* dar. Funktionen werden hierbei von der Klasse *FunctionFactory* erzeugt. Dies ist sinnvoll um alle Funktionen mit dem gleichen MatlabProxy zu verbinden. Der MatlabProxy stellt hierbei die Verbindung der *Java Virtual Machine* mit *Matlab* dar.

FunctionFactory erzeugt Instanzen der Klasse *Function*. Hierbei sind alle Funktionen welche aus der gleichen Fabrik kommen, mit der gleichen Instanz von *Matlab* verbunden. Eine solche Instanz wird bei jedem Simulationsaufruf neu erstellt. Es besteht somit die Möglichkeit, mehrere Simulationen gleichzeitig laufen zu lassen. Dies ist jedoch nur dann sinnvoll, wenn der dazu benutzte Rechner über mehrere Rechenkerne verfügt, da ansonsten die Berechnungen wiederum serialisiert ausgeführt werden.

NetworkFunction repräsentiert die Menge aller Funktionen und somit die vollständige mathematische Beschreibung des Netzwerks. Ein Objekt der Klasse *NetworkFunction* baut das mathematische Modell direkt bei der Initialisierung auf.

Tools Die Klasse *Tools* stellt Hilfsmethoden zur Verfügung, wie beispielsweise die Zeichenketten-Formatierung für die Funktionsbenennung für die Kommunikation mit *Matlab*.

Die Klassen *Output* und *Resource* wurden schon unter 6.2 beschrieben.

Window-Klassen

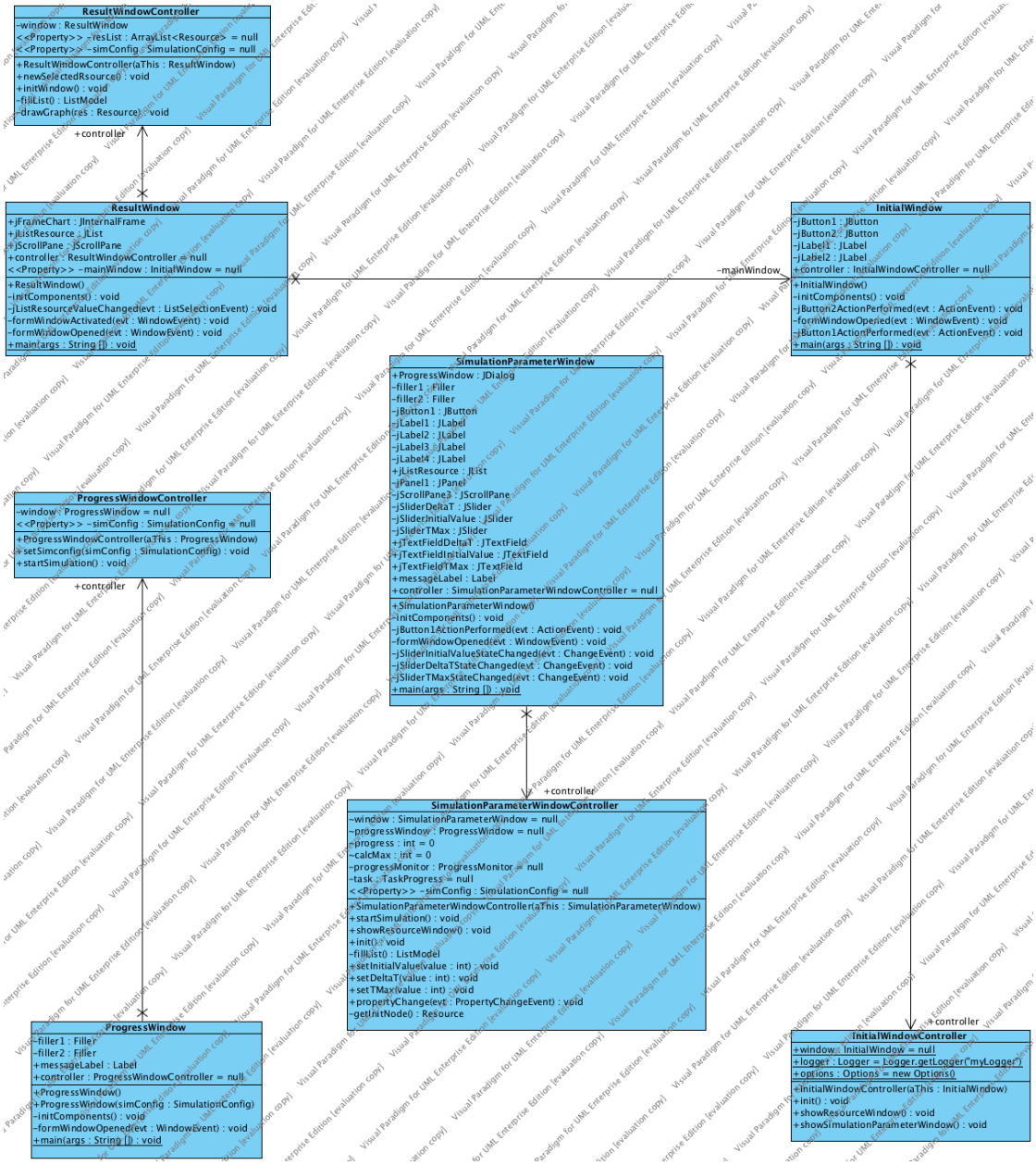

Abbildung 6.4: Klassendiagramm der Klassen des Paketes *window*

Die *Window*-Klassen stellen die graphische Oberfläche der Anwendung dar. Jede *Window*-Klasse ist abgeleitet von der Java-Klasse *javax.swing.JFrame*. Es wurde somit das *Swing-Framework* benutzt, da es über mehr Features als das *AWT-Framework* (*Abstract Window Toolkit*) verfügt.

Als Nachteil des *Swing-Frameworks* gegenüber dem *AWT-Framework* darf die niedrigere Geschwindigkeit von *Swing* genannt werden [9].

Jede *Window*-Klasse steht hierbei in Verbindung mit der dazugehörenden *Controller*-Klasse. Die *Controller*-Klassen beinhalten hierbei die zugehörige Logik der Fenster und bearbeiten die Benutzerinteraktionen. Dies wurde bereits im Kapitel 6.2 erläutert.

InitialWindow Diese Klasse stellt das erste Fenster der graphischen Benutzeroberfläche dar. Von dieser Klasse aus verzweigt die Anwendung zu allen anderen Fenstern. Events in dieser Klasse führen hierbei Methoden in der Klasse *InitialWindowController* aus.

ResultWindow Diese Klasse beschreibt das Fenster der graphischen Oberfläche, welches die Ergebnisse der Simulation darstellt. Hierbei werden Graphen benutzt und die Ergebnisse in Form von Kurven repräsentiert. Events in dieser Klasse führen hierbei Methoden in der Klasse *ResultWindowController* aus.

SimulationParameterWindow Diese Klasse beschreibt das Fenster der graphischen Oberfläche, welches die notwendigen Parameter für die Simulation aufnimmt. Events in dieser Klasse führen hierbei Methoden in der Klasse *SimulationParameterWindowController* aus.

ProgressWindow Diese Klasse beschreibt das Fenster der graphischen Oberfläche, welches anzeigt, dass die Berechnungen der Simulation gerade ausgeführt werden. Events in dieser Klasse führen hierbei Methoden in der Klasse *ProgressWindowController* aus.

Die in der Abbildung 6.4 zu sehenden Klassen *ResultWindowController*, *InitialWindowController*, *SimulationParameterWindowController* und *ProgressWindowController* wurden schon in 6.2 erläutert.

Database-Klassen

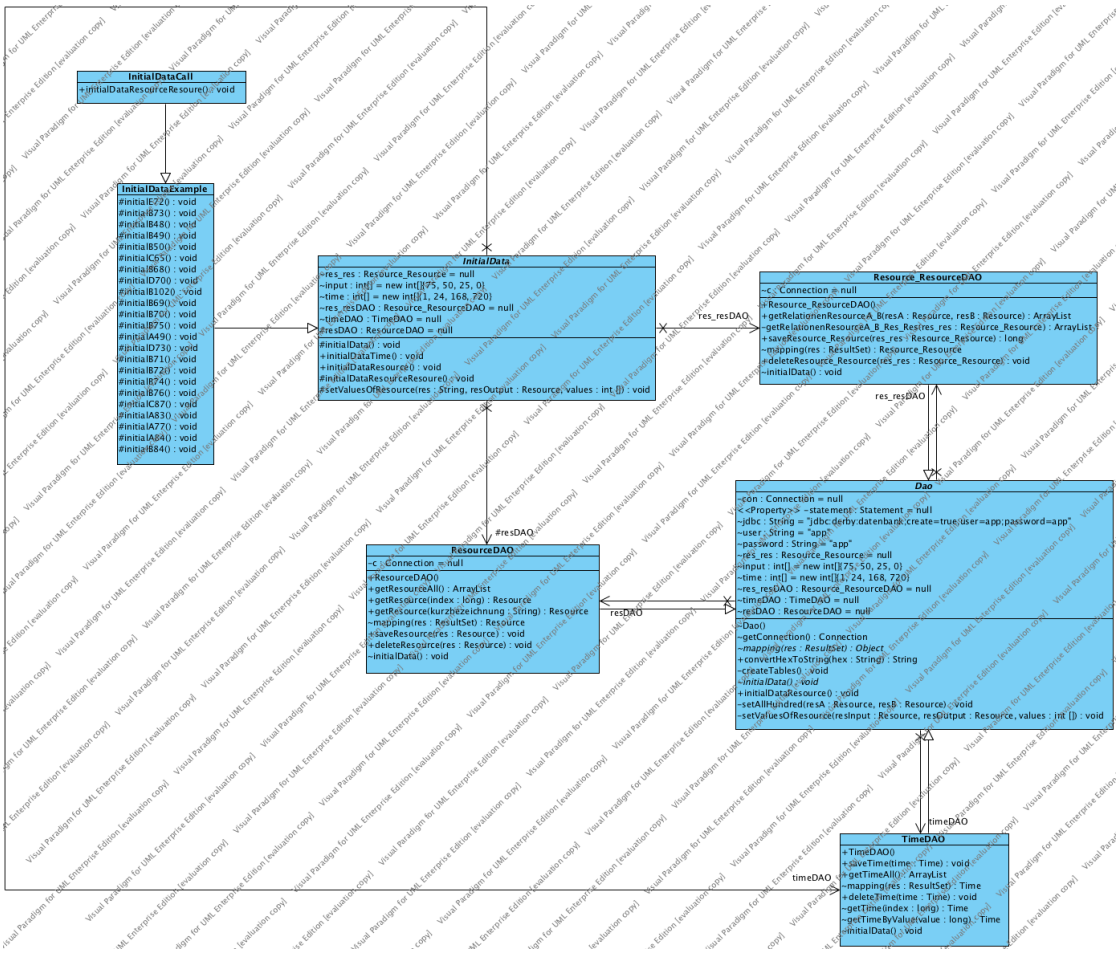

Abbildung 6.5: Klassendiagramm der *Database*-Klassen

Die Klassen dieses Pakets sind die für die Persistierung der in 6.2 beschriebenen *Plain Old Java Objects (POJOs)* notwendig. Dieses Paket enthält alle für den Datenbank zugriff notwendigen Klassen und Methoden und ist somit die Verbindung der *Controller*-Klassen zur Datenbank.

Dao Die *DAO-Klasse* stellt die zentrale Klasse des *Database*-Paketes dar. *DAO* steht für *Data Access Object*. Diese Klasse ist vom Typ *abstract* und definiert die notwendigen Methoden für den Datenbankzugriff aller abgeleiteten Klassen. Hierbei werden von den ableiteten Klassen die sogenannten *CRUD-Operationen* implementiert. CRUD ist ein Acronym und wird meist mit den folgenden Wörtern gleichgesetzt: *C*reate, *R*ead, *U*pdate und *D*elete [27]. Alle abgeleiteten Klassen von *DAO* verfügen über die gleiche Datenbankanbindung. Die Datenbankverbindungszeichenkette ist im Code eingetragen und wird zur Kompilierungszeit festgelegt.

Die Datenbankverbindungszeichenkette ist wie folgt:

jdbc:derby:datenbank;create=true;user=app;password=app

Diese Datenbankverbindungszeichenkette legt fest, dass als Datenbankschnittstelle JDBC (*Java Database Connectivity*) benutzt wird (*jdbc*). Es wird eine *Derby*-Datenbank der *Apache Foundation* benutzt (*derby*). Der Name der Datenbank ist *datenbank* und diese wird im Arbeitsverzeichnis der Anwendung erstellt (*datenbank*). Das Arbeitsverzeichnis ist im Produktionsmodus, sofern es nicht anders definiert wurde bei der Ausführung durch *java*, das Verzeichnis in welchem sich die JAR-Datei befindet. Wenn die Datenbank noch nicht existiert, so wird sie automatisch erstellt (*create=true*). Hierbei wird als Benutzername *app* und als Passwort *app* benutzt (*user=app;password=app*).

TimeDAO Diese Klasse wird abgeleitet von der Klasse *Dao* und stellt somit Methoden für das Hinzufügen, Lesen, Ändern und Löschen von Datensätzen der Klasse *Time* zur Verfügung.

ResourceDAO Diese Klasse wird von der Klasse *Dao* abgeleitet und stellt somit Methoden für das Hinzufügen, Lesen, Ändern und Löschen von Datensätzen der Klasse *Resource* zur Verfügung. Mit Objekten dieser Klasse können somit Knoten in unserem Netz verändert, hinzugefügt oder gelöscht werden, wie sie unter 5.2 beschrieben wurden.

Resource_ResourceDAO Diese Klasse wird von der Klasse *Dao* abgeleitet und stellt somit Methoden für das Hinzufügen, Lesen, Ändern und Löschen von Datensätzen der Klasse *Resource_Resource* zur Verfügung. Mit Objekten dieser Klasse können somit die Kanten in unserem Netz verändert, hinzugefügt und entfernt werden, wie sie unter 5.2 beschrieben wurden.

InitialData Diese Klasse stellt die grundlegenden Methoden zur Verfügung, um die Datenbank bei der erstmaligen Ausführung der Anwendung mit den Ausgangsdaten dieser Diplomarbeit zu füllen.

InitialDataExample Diese Klasse erbt von der Klasse *InitialData*. Die Klasse enthält die Implementierungen welche die initialen Daten der Datenbank hinzufügen. Diese Klasse wurde automatisiert mittels des *Critical Infstructure Tools* (9) erstellt. Inhalt dieser Klasse sind somit hauptsächlich die Ausgangsdaten und die notwendigen Methoden zum Einfügen.

InitialDataCall Diese Klasse erbt von der Klasse *InitialDataExample*, und enthält eine Methode zum Aufruf aller Methoden der Klasse *InitialDataExample*, welche zum Einfügen der Ausgangsdaten in die Datenbank dienen. Diese Klasse wird ebenfalls mittels des *Critical Infstructure Tools* (9) erzeugt.

6.3 Die Datenbank

Als Datenbankkonzept wurde eine relationale Datenbank gewählt, welche auf *SQL* basiert, der Structured Query Language. SQL ist ein international anerkannter Standard.

Als konkrete Implementierung des *Datenbank Management Systems (DBMS)* wurde *Derby* benutzt. Hierbei handelt es sich um eine Datenbank-Engine des Open Source-Datenbankprojekts *Apache Derby*. *Derby* ist eine ausschließlich auf Java basierende Datenbankentwicklung [14].

Hierbei fand der *Embedded Driver* Anwendung als *JDBC*-Treiber (*Java DataBase Connectivity*). Mit diesem Treiber wird eine Datenbank im Dateisystem angesprochen, welche sich nur für eine lokale Anwendung eignet. Es wird hierbei kein Server angesprochen, sondern mittels des Treibers direkt, gemäß dem SQL-Standard, auf die Datenbank zugegriffen.

Die Datenbank wird beim erstmaligen Ausführen der Anwendung im Arbeitsverzeichnis der Anwendung erstellt. Es wird ein Ordner mit dem Namen *datenbank* erstellt. In diesem befinden sich alle für die Datenbank notwendigen Dateien. Von besonderem Interesse sind hierbei die Dateien *db.lck*, *dbex.lck* und *derby.log* von Bedeutung.

db.lck, dbex.lck Diese Dateien stellen sogenannte Lock-Dateien dar. Sie werden vom Datenbanktreiber beim Zugriff auf die Datenbank erstellt. Bei einer Datenbank im *Embedded*-Modus darf nur jeweils ein Treiber auf die Datenbank zugreifen. Dies ist so vorgesehen, da im *Embedded*-Modus kein Server vorhanden ist, welcher den kongruenten Zugriff mehrerer Clients regelt, daher würden sich beim Parallelzugriff auf eine *Embedded*-Datenbank möglicherweise unbestimmte Zustände ergeben. Um dies zu verhindern, ist jeweils nur der Zugriff eines Treibers zur gleichen Zeit gestattet. Dies wird mittels der *Lock*-Dateien sichergestellt. Beim Verbinden mit der Datenbank überprüft der *Embedded*-Treiber das Vorhandensein der beiden *Lock*-Dateien *db.lck* und *dbex.lck*. Ist eine der beiden Dateien vorhanden, so gibt der Datenbanktreiber eine Fehlermeldung an die Anwendung zurück und stellt keine Verbindung zur Datenbank her.
Wurde die Verbindung zur Datenbank in einem normalen Programmablauf erwartungsgemäß geschlossen, so werden die genannten *Lock*-Dateien vom Datenbanktreiber gelöscht.

Sollte es, wider Erwarten, nicht zu einem planmässigen Anwendungsende kommen, sonder die Anwendung unerwarteter Weise auf nicht vorgesehenem Weg abgebrochen werden, beispielsweise im Fall eines Stromausfalls, so werden diese *Lock*-Dateien vom Treiber nicht gelöscht. In so einem Fall müssen die Dateien vom Benutzer manuell gelöscht werden, um eine weitere Ausführung der Anwendung mit der zuvor benutzten Datenbank zu ermöglichen.

Eine weitere Konsequenz, welche sich aus den beschriebenen Zugriffseinschränkungen für den Datenbanktreiber ergibt, ist die Tatsache, dass ein mehrmaliges Ausführen der Anwendung mit der gleichen Datenbank nicht möglich ist. Dies bedeutet, das Ausführen der Anwendung mit gleichem Arbeitsverzeichnis und somit gleichem Datenbankverzeichnis und somit

gleicher Datenbank ist nur einmal zur gleichen Zeit möglich. Bei wiederholtem Ausführen der *gleichen* Anwendung würde dies zu einer Fehlermeldung führen.

Es ist jedoch möglich, die Anwendung ein weiteres Mal auszuführen, wenn hierzu verschiedene Datenbanken benutzt werden, wenn also die Anwendung in verschiedenen Verzeichnissen ausgeführt wird. Der Vorteil einer solchen parallelen Ausführung hält sich jedoch für die Berechnungen selbst in Grenzen, da *Matlab* selbst Berechnungen nur im *Single-Threaded-Modus* ausführt, sofern keine speziellen Bibliotheken für die Parallelisierung benutzt werden. Daher würde dies zu einem Warten auf das Ausführungsende einer sich in Berechnung befindenden Kalkulation führen, und nicht zu zeitgleichen Berechnungen. Daher ist das mehrfache Ausführen der Anwendung auf einer Maschine im Allgemeinen nicht zu empfehlen und bringt keine Vorteile in der Berechnungsgeschwindigkeit.

derby.log Die Datei *derby.log* enthält alle Fehlermeldungen, welche die Datenbank direkt betreffen. Ein versuchter paralleler Zugriff, welcher mittels der genannten *Lock*-Dateien verhindert wurde, wird ebenfalls mittels dieser Log-Datei aufgezeichnet, sowie andere Probleme in Bezug auf die Datenbank. Die Einträge in der Datei *derby.log* werden vom Datenbanktreiber erstellt, und nicht direkt von der Anwendung selbst, daher sind Informationen zu den Fehlermeldungen in dieser Datei beim Datenbankhersteller selbst zu finden [14].

Das Entity Relationship-Diagramm

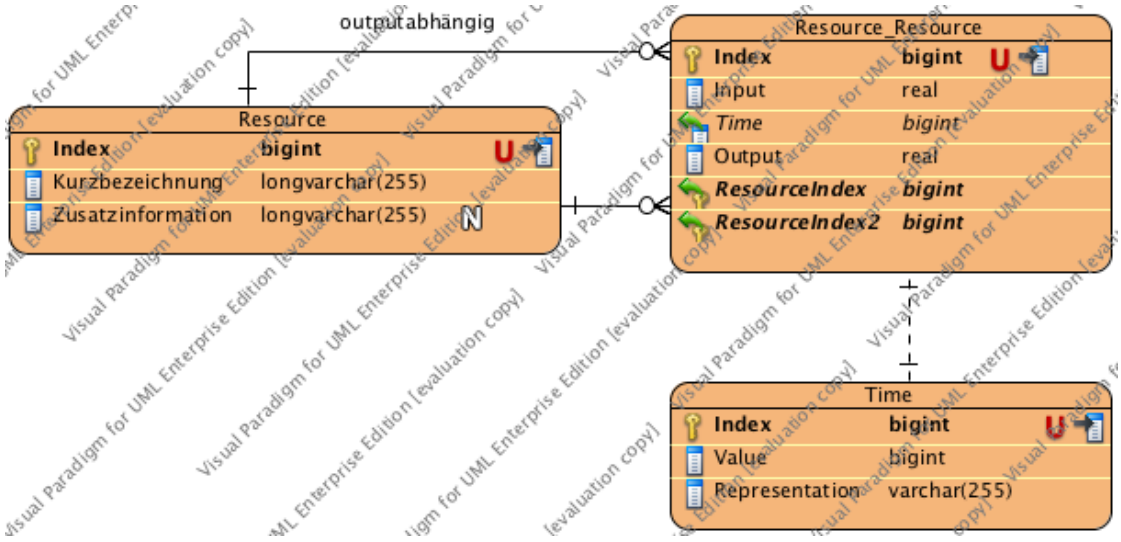

Abbildung 6.6: Entity Relationship Diagramm

Wie in diesem Entity Relationship-Diagramm, kurz ER-Diagramm genannt, ersichtlich ist, besteht die Datenbank aus drei Tabellen, welche zueinander durch Datenbank-Constraints in

Relation stehen. Im Diagramm sind diese Relationen durch die Verbindungslinien zwischen den einzelnen Tabellen gekennzeichnet.

Wie die Abbildung 6.6 zeigt, besteht der *Primary-Key* bei allen *Entitäten* (Tabellen) ausschließlich aus einem Index vom Typ *bigint*. Dieser Index wird beim Einfügen eines Datensatzes vom Datenbanksystem automatisch generiert und ist eine positive ganze Zahl. Auf Grund dieser Eigenschaft des Datenbankdesigns, wurde die Möglichkeit ausgeschlossen, einen Datenbankfehler zu provozieren, welcher aufgrund eines sich wiederholenden Primärschlüssels auftreten könnte. Aufgrund dieser Eigenschaft ist beispielsweise auch die Benennung einer Ressource mit der gleichen Kurzbezeichnung und der gleichen Zusatzinformation möglich. Die Datenbank kann gleich bezeichnete Ressourcen unterscheiden. Für den Benutzer könnte dies in der Oberfläche der Anwendung jedoch schwierig sein, daher sei die Sinnhaftigkeit einer solchen Gleichbenennung bei den Ressourcen dahingestellt. Die Zusatzinformation für eine Ressource muss nicht angegeben sein.

Als äquivalente Datentypen für Zeichenkette in Java wurden in der Datenbank die Typen *varchar* und *longvarchar* benutzt. *longvarchar* und *varchar* bieten die Möglichkeit, Zeichenketten mit einer Länge von maximal 32.700 Zeichen zu speichern. Beim Typ *longvarchar* ist jedoch die Angabe einer maximalen Zeichenanzahl beim Erstellen der Tabelle nicht verpflichtend.

Der für die *Primary-Keys* benutzte SQL-Datentyp *bigint* entspricht in Java dem Datentyp *long*. *bigint* verfügt über 8 Bytes, und kann ausschließlich ganze Zahlen speichern. Hierbei ist der positive Maximalwert 9223372036854775807, gerundet $9,2 \cdot 10^{18}$ [14].

Für Werte, welche im Fließkomma-Format gespeichert werden, wird der SQL-Datentyp *real* genutzt. Dieser verfügt über 4 Bytes und entspricht dem Java-Datentyp *float*. Der SQL-Datentyp *real* besitzt einen Wertebereich von $-3.402 \cdot 10^{38}$ bis $3.402 \cdot 10^{38}$ wobei der kleinste darstellbare positive Wert $1.175 \cdot 10^{-37}$ ist und der größte darstellbare negative Wert $-1.175 \cdot 10^{-37}$ [14]. Für die Anforderungen dieser Diplomarbeit ist diese Genauigkeit absolut ausreichend.

Wie durch die Namen der Entitäten ersichtlich ist, entspricht eine Tabelle jeweils dem namentlichen Äquivalent als *Plain Old Java Object* der Abbildung 6.1. Ebenso sind die Relationen im ER-Diagramm natürlich ähnlich ausgeprägt wie im Klassendiagramm. Die Klasse *Output* des Klassendiagramms der *POJOs* enthält jedoch kein äquivalentes Gegenüber im ER-Diagramm, da für die prototypische Anwendung dieser Diplomarbeit keine Speicherung der Simulationsergebnisse vorgesehen war.

Die Entität *Resource* steht in doppelter Relation zur Entität *Resource_Resource*, wobei ein jeder Eintrag in der Tabelle wiederum in Relation zu einer *Time*-Entität steht. Die *Time*-Entitäten stellen somit Attribute der Relationen verschiedener *Resource*-Entitäten zueinander dar. Um jedoch Datenduplikate und somit Speicherplatz zu sparen, wurden diese Relationsattribute in eigene Entitäten ausgelagert.

6.4 Genutzte Softwarebibliotheken

JFreeChart

JFreeChart[19] ist eine vollständig auf Java basierende Graphikbibliothek zur Darstellung von zwei- und dreidimensionalen Graphen. Die Bibliothek wird hierbei als freie Software nach der *GNU Lesser General Public Licence (LGPL)* [10] vertrieben. Freie Software bedeutet, es steht den Benutzern frei, die Software zu verändern, zu verbreiten, auch zu verkaufen und weiteres. Hierbei muss jedoch immer der Quellcode zugänglich gemacht werden. Für diese Diplomarbeit wurde *JFreeChart* in der Version 1.0.14 benutzt.

JFreeChart ist abhängig von der Softwarebibliothek *JCommon*. *JCommon* stellt Funktionalitäten wie Textverarbeitungsfunktionen, Logging-Funktionen, Benutzerschnittstellenfunktionen, Funktionen zur Datenserialisierung und weitere Hilfsfunktionen zur Verfügung. *JCommon* wird ebenfalls unter der *GNU Lesser General Public Licence (LGPL)* [10] vertrieben. *JCommon* wurde in der Version 1.0.17 verwendet.

Die Softwarebibliothek *JFreeChart* wurde im Speziellen in diesem Projekt für die Darstellung der Simulationsergebnisse benutzt. Die Berechnungsergebnisse werden hierbei in einem zweidimensionalen Liniendiagramm dargestellt, wobei die Kurven interpoliert werden. Dies kann mittels linearer Interpolation geschehen oder auch mittels kubischer Spline-Interpolation [17].

log4j

log4j[12] ist eines der bekanntesten Logging-Utilitys in der Javawelt. Es wurde von der Apache Software Foundation entwickelt und wird unter der Apache License, Version 2.0 vertrieben. [11]. *log4j* wurde für die Diplomarbeit in der Version 1.2.17 benutzt.

log4j beinhaltet verschiedene Logging-Pattern und ermöglicht somit Logging in verschiedenen Komplexitätsstufen. In der Anwendung wurde das Logging mittels HTML-Ausgabe implementiert. Die Anwendung erstellt eine Log-Datei im ausführenden Verzeichnis und loggt dort etwaige Informationen in die Datei *log.html* im HTML-Format. Der Log-Level kann bei *log4j* durch eine externe Konfigurationsdatei ohne eine Änderung des Sourcecodes verändert werden. Dies wurde im Abschnitt 6.1 beschrieben.

matlabcontrol

matlabcontrol[3] ist ein *Java Application Programmer Interface (API)* zur Steuerung von *Matlab* mittels Java. Es wurde in dieser Diplomarbeit in der Version 4.0.0 verwendet.

matlabcontrol ermöglicht vollständigen Zugriff auf die *Matlab*-Engine. Dies bedeutet, es ist mittels dieser Schnittstelle möglich, beliebige *Matlab*-Befehlen mittels einer Java-Anwendung auszuführen. Dies entspricht dem Vorgehen eines Benutzers bei der manuellen Eingabe von

Befehlen in *Matlab* in der gewohnten *Matlab*-Oberfläche. Ebenso ermöglicht dieses Interface das Öffnen und Schließen von *Matlab*.

Des Weiteren lassen sich Parameter für Matlab setzen, die zum Beispiel das Öffnen von *Matlab* im versteckten Modus oder im minimierten Modus ermöglichen. In der Anwendung wird *Matlab* in der Standardeinstellung unsichtbar gestartet, daher ist die Benutzung der *Matlab*-Engine für den Benutzer nicht sichtbar und daher unerheblich. Jedoch ist es für die Anwendung zwingend erforderlich, dass *Matlab* in der Rechenumgebung installiert wurde, um das Funktionieren der Software zu ermöglichen. Die Anwendung wurde für die *Matlab-Version R2012a 7.14.0.739* entwickelt. Für andere Versionen kann die Funktionsfähigkeit nicht garantiert werden. Im speziellen gilt diese Einschränkung der Funktionszusage für ältere Versionen von *Matlab*. Ebenfalls ist ein Funktionieren der Software mit neueren Versionen von *Matlab* nicht zu garantieren, da manche benutzten *Matlab*-Befehle von *Mathworks*, der Herstellerfirma von *Matlab*, als überholt für kommende Versionen gesetzt wurde, und daher in neueren Versionen möglicherweise nicht mehr unterstützt werden.

6.5 Entwicklungstools

Visual Paradigm

Für die Modellierung der Software, insbesondere der Klassendiagramme 6.2, wurde die Softwaresuite *Visual Paradigm for UML* [23] in der Version 10.0 mit einer temporären Enterprise-Lizenz genutzt. Die Wahl fiel auf *Visual Paradigm for UML* aufgrund der ausgeprägten Round-Trip-Engineering-Unterstützung. Hierbei kann auf Grundlage des Modells, zum Beispiel eines Klassendiagramms, der entsprechende Java-Code generiert werden. Ebenso kann auf Basis von Java-Klassen ein übereinstimmendes UML-Diagramm erzeugt werden.

Visual Paradigm for UML wird dabei vom Hersteller wie folgt beschrieben [23]:

> „Visual Paradigm for UML (VP-UML) is a UML design tool and UML CASE tool designed to aid software development. VP-UML supports key modeling standards such as Unified Modeling Language (UML) 2.4, SoaML, SysML, ERD, DFD, BPMN 2.0, ArchiMate 2.0, etc. It supports software development teams in requirements capturing, software planning (use case analysis), code engineering, class modeling, data modeling, etc.“

Netbeans

Nach mehreren Versuchen, die Entwicklungsumgebung *Eclipse* für die Entwicklung der Anwendung zu nutzen, entschied man sich schlussendlich für die Entwicklungsumgebung *Netbeans* (Version 7.1.1). *Eclipse* zeigte sich zu instabil bezüglich der graphischen Entwicklungsoberfläche zum Erstellen einer GUI. Der GUI-Editor der *Eclipse*-Version *Indigo* erwies sich als sehr fehlerbehaftet.

Netbeans stellte sich noch in weiteren Bereichen als vorteilhaft heraus. *Netbeans* inkludiert die Unterstützung eines *SVN*-Systems ohne die Notwendigkeit ein Plug-in oder ähnliches installieren zu müssen. Ebenso enthält es eine integrierte Oberfläche zur Serveradministration für Datenbankserver und Anwendungsserver. Des Weiteren werden graphische Möglichkeiten zur direkten Datenbankmanipulation zur Verfügung gestellt. Im Gesamten erwies sich *Netbeans* als geeigneter.

Matlab

Matlab wurde nicht nur als mathematische Engine für die Anwendung genutzt, wie unter 6.4 beschrieben, sondern auch für die Erstellung von Graphiken und Ähnlichem für die schriftliche Ausarbeitung. Siehe Abbildung 5.4.

TexShop

TexShop ist ein Editor für das Betriebsystem *Mac OS X*, welcher die Erstellung von Dokumenten mittels *LaTex* ermöglicht. Dieses Dokument wurde mit *TexShop* und *LaTex* erstellt. *LaTex* ist ein Textsetzungssystem, welches auf *Tex* basiert. *LaTex* benutzt dabei Makros, welche *Tex* benutzen [25].

KAPITEL 7

Die Anwendung

7.1 Initialfenster

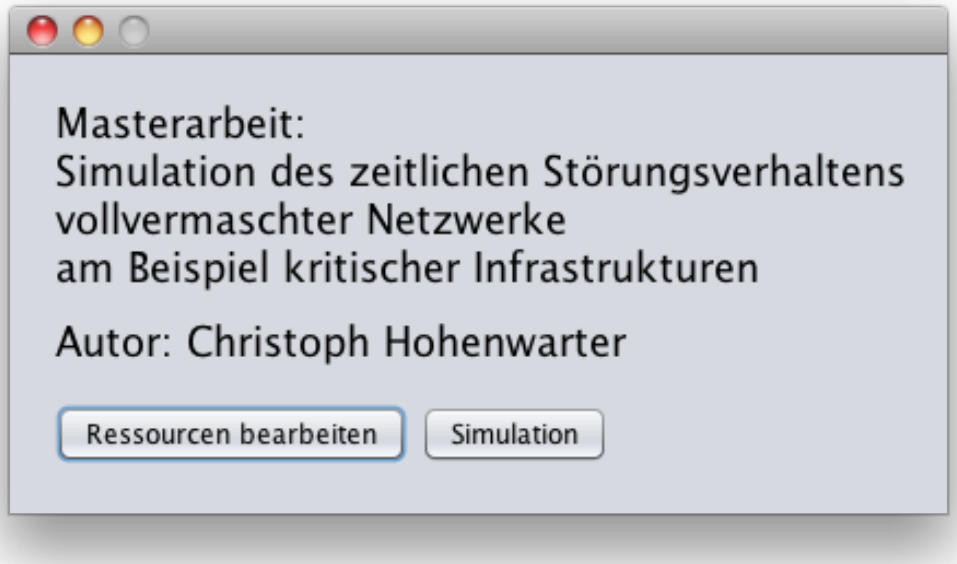

Abbildung 7.1: Initialfenster der Anwendung

Das initiale Fenster stellt die primäre Verzweigung der Anwendung dar. Von diesem Fenster aus gelangt man zu allen weiteren Fenstern. Wobei die Anwendung hierbei über zwei Hauptzweige verfügt. Der erste Zweig ermöglicht das Editieren der Ressourcendaten, sowie der Punkte in

der Zeitachse. Hierbei werden die Daten der Datenbank bearbeitet. Der zweite Zweig verfolgt die Berechnung und die Darstellung der Simulationsergebnisse. Wie offensichtlich sein sollte, gelangt man mittels des Buttons *Ressourcen bearbeiten* zum Teil der Oberfläche zum Bearbeiten der Informationen in der Datenbank und über den Button *Simulation* zum Teil, welcher mittels der eingegebenen Daten die Berechnungen durchführt.

7.2 Fenster der Ressourcenrelationen

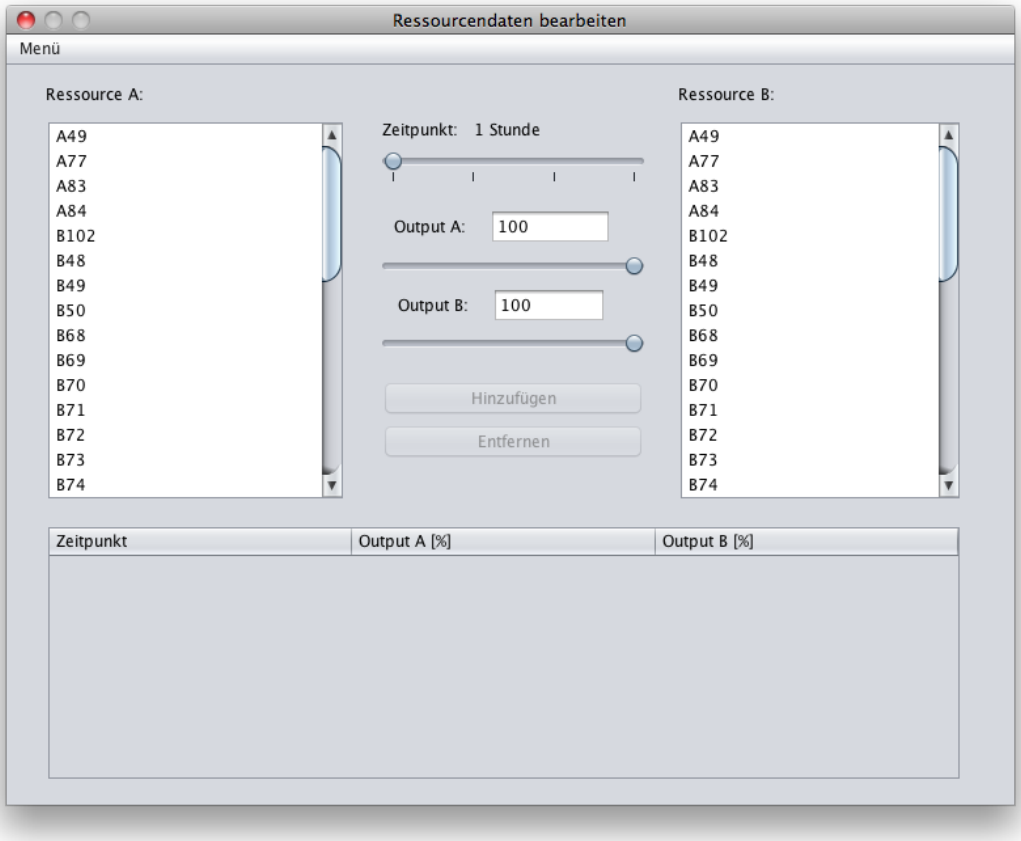

Abbildung 7.2: Fenster der Relationen zwischen den Ressourcen

In diesem Fenster können die entsprechenden Punkte der Funktionen zwischen den einzelnen Ressourcen bearbeitet werden. Hierbei bestimmt der Benutzer zuerst die *eingehende* Ressource, die Ressource *A* und als nächstes die *ausgehende* Ressource, die Ressource *B*. Danach wird ein Zeitpunkt im Graphen bestimmt für welchen ein Funktionspunkt festgelegt werden kann. Es wird somit ein Punkt in der Ebene der Relation zwischen zwei Ressourcen bestimmt. Dies ent-

spricht der Festlegung eines Punktes in einem Graphen, wie zum Beispiel der in der Abbildung 5.4 dargestellte.

Beim Design dieser Oberfläche wurde auf eine vollständige Bedienbarkeit mittels Maus Wert gelegt. Dies wurde durch die Benutzung von *Slidern* erreicht. Dies macht die Bedienung der Oberfläche langsamer, jedoch erspart es dem Benutzer die Bedienung der Tastatur. Dies ist im Allgemeinen für nicht besonders versierte Nutzer von Vorteil.

Im unteren Teil des Fensters werden die betreffenden Relationen dargestellt. Ebenso gelangt man von diesem Fenster aus zu weiteren Fenstern, welche das Hinzufügen und Entfernen von Ressourcen ermöglichen, sowie die Manipulation des Zeitrasters. Zu diesen Fenstern gelangt man mittels des Menüs am oberen Rand des Fensters.

7.3 Fenster zur Ressourcenbearbeitung

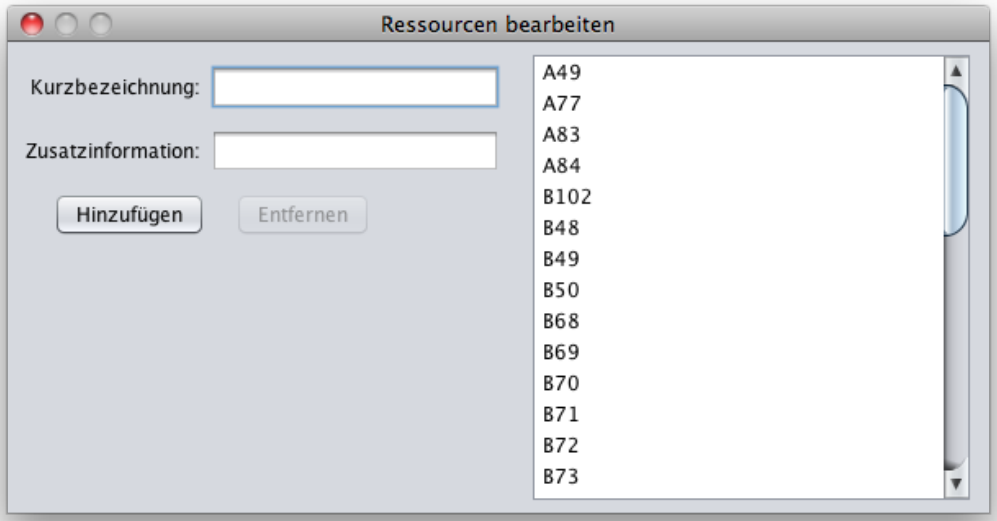

Abbildung 7.3: Fenster zum Bearbeiten der Ressourcen

In diesem Fenster können Ressourcen dem Netzwerk hinzugefügt werden. Ebenso können Ressourcen entfernt werden. Dieses Fenster dient also dazu, das Netzwerk um Knoten zu erweitern oder zu vermindern. Jeder Knoten bekommt hierbei eine Kurzbezeichnung. Ebenso kann für jeden Knoten auch Zusatzinformation angegeben werden. Jedoch ist die Angabe einer zusätzlichen Zeichenkette, zur Beschreibung des Knotens, nicht zwingend erforderlich. Wenn eine

Ressource entfernt wurde, so werden auch alle zugehörigen Relationen zu anderen Ressourcen aus der Datenbank entfernt.

7.4 Fenster zur Zeitpunktbearbeitung

Abbildung 7.4: Fenster zum Bearbeiten der Zeitwerte

Hier können Zeitpunkte hinzugefügt werden, welche für die Definition der Relationen zwischen den Ressourcen notwendig sind. Es können nur Funktionswerte für Zeitpunkte definiert werden, die im System angelegt wurden und dahingehend in diesem Fenster zu sehen sind. Die in diesem Fenster definierten Zeitpunkte bestimmen somit den zeitlichen Raster der Funktionen zwischen den Ressourcen. Hierbei wird zum einen der *Zeitpunkt* festgelegt, welcher als Zahlenwert angegeben wird und die Einheit *Stunden* hat, sowie bekommt der Zeitpunkt eine Zeichenkette als Identifikation. Diese muss nicht, wie in der Abbildung 7.4 ersichtlich, durch größere Zeiteinheiten bestimmt sein. Es ist auch möglich, als Bezeichnungen einfache Zeichenketten wie *Zeitpunkt 1*, *Zeitpunkt 2*, *Zeitpunkt 3* usw. zu benutzen. Dies bietet sich im speziellen dann an,

wenn die gewählten Zeitpunkte keine anders lautende Bezeichnung haben, um beispielsweise einen Funktionswert der 27. Stunde einzutragen.

7.5 Fenster der Simulationsparameter

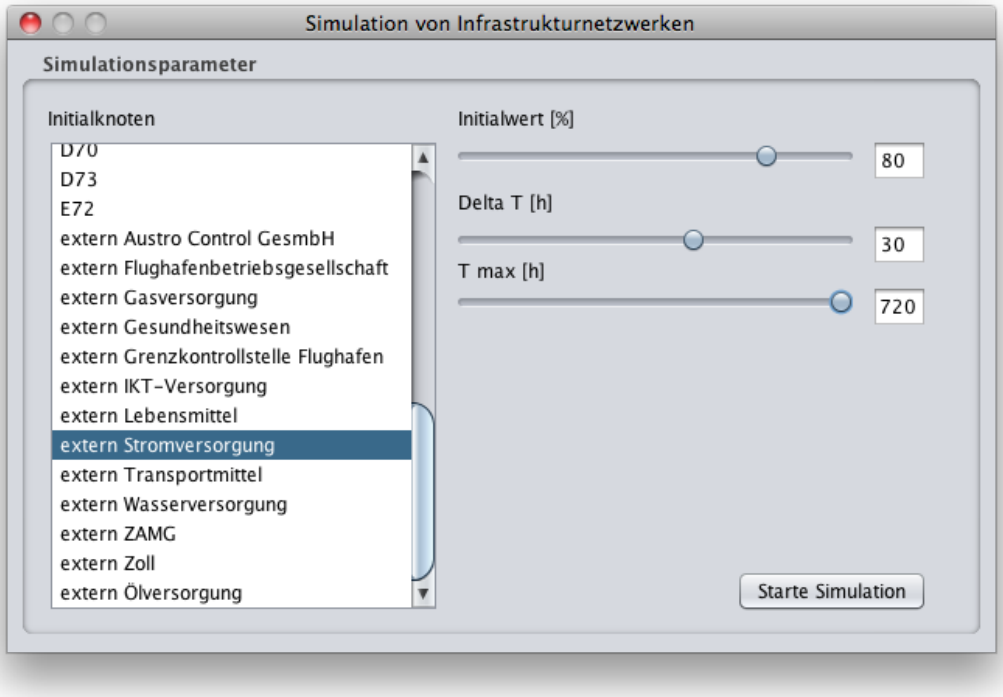

Abbildung 7.5: Fenster zum Bearbeiten der Simulationsparameter

In diesem Fenster werden die in 8.2 benutzten Parameter vom Benutzer konfiguriert. Als erstes kann hier der Knoten definiert werden, welcher die Störung aufweist. Dieser wird anhand der Liste ausgewählt. Hierbei kann es sich, wie in 5.2 beschrieben, um eine Ressource oder ein Infrastrukturelement handeln. Diese sind gleichwertig. In diesem Beispiel wurde als Störquelle die *externe Stromversorgung* gewählt. Für diese gilt dann der rechts daneben eingestellte Initialwert, wie in diesem Beispiel von 80%. Diese bedeutet, es wird in diesem Fall eine Reduktion der externen Stromversorgung um 20% berechnet. Des Weiteren ist ersichtlich, dass wir eine Genauigkeit von einem Berechnungspunkt alle 30 Stunden bestimmt haben. Dies ist relativ ungenau, jedoch reduziert dies die notwendige Rechenzeit erheblich. Die notwendigen Funktionswerte zur Darstellung der Ergebnisse werden interpoliert. In diesem Beispiel wird die maximale Zeitspanne berechnet. Es wird der Verlauf der Netzwerkknoten bis zum Zeitpunkt von 720 Stunden berechnet. Dies entspricht dem maximalen Definitionsraum der Funktionen des Netzwerks auf

Grundlage der vorliegenden Daten. Die Relationen der Knoten wurden bis maximal 720 Stunden definiert.

7.6 Fenster *Die Berechnung wurde gestartet*

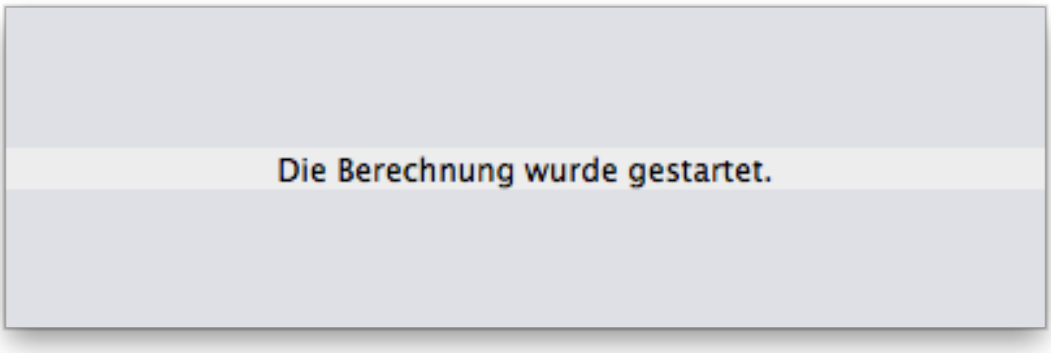

Abbildung 7.6: Berechnung gestartet

Dieses Fenster zeigt, dass die Berechnung gestartet wurde und mittels *Matlab* durchgeführt wird. Solange diese Einblendung sichtbar ist, werden Berechnungen durchgeführt. Diese laufen gemäß der Standardeinstellungen im Hintergrund in *Matlab* ab, können jedoch durch die in 6.1 beschriebenen Einstellungen sichtbar gemacht werden. Wenn die Sichtbarkeit von *Matlab* konfiguriert wurde, so wird die Oberfläche angezeigt, und man sieht die Eingaben der *Matlab*-Shell, als würden die Daten von einem Benutzer manuell eingegeben werden. Durch diese Sichtbarkeit der Eingaben kann man bei Bedarf mögliche Fehler, aufgrund der Meldungen von *Matlab*, feststellen. Wie dies aussieht, wenn *Matlab* durch die Anwendung gesteuert wird, kann man in der Abbildung 7.7 betrachten.

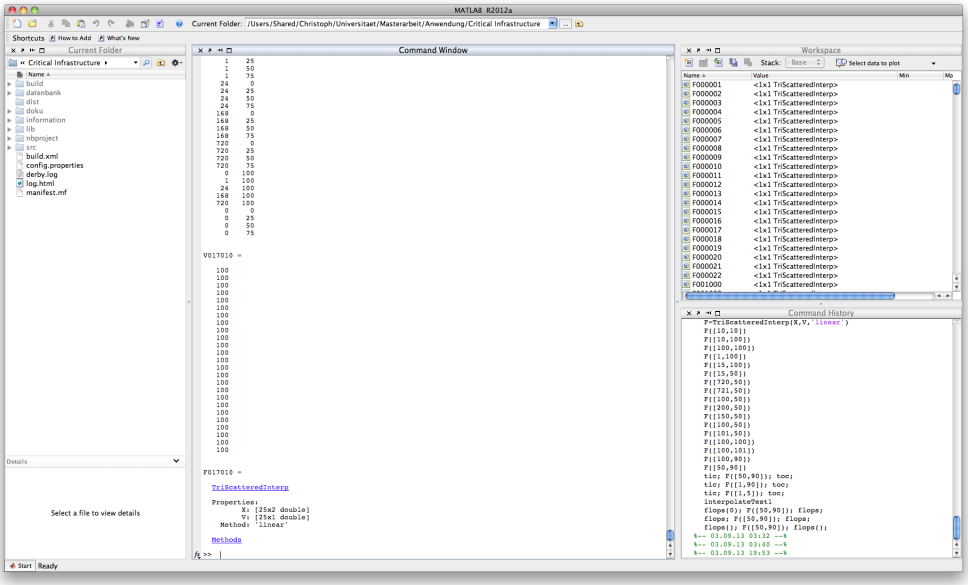

Abbildung 7.7: Matlab im sichtbaren Modus während der Berechnung

7.7 Ergebnisfenster

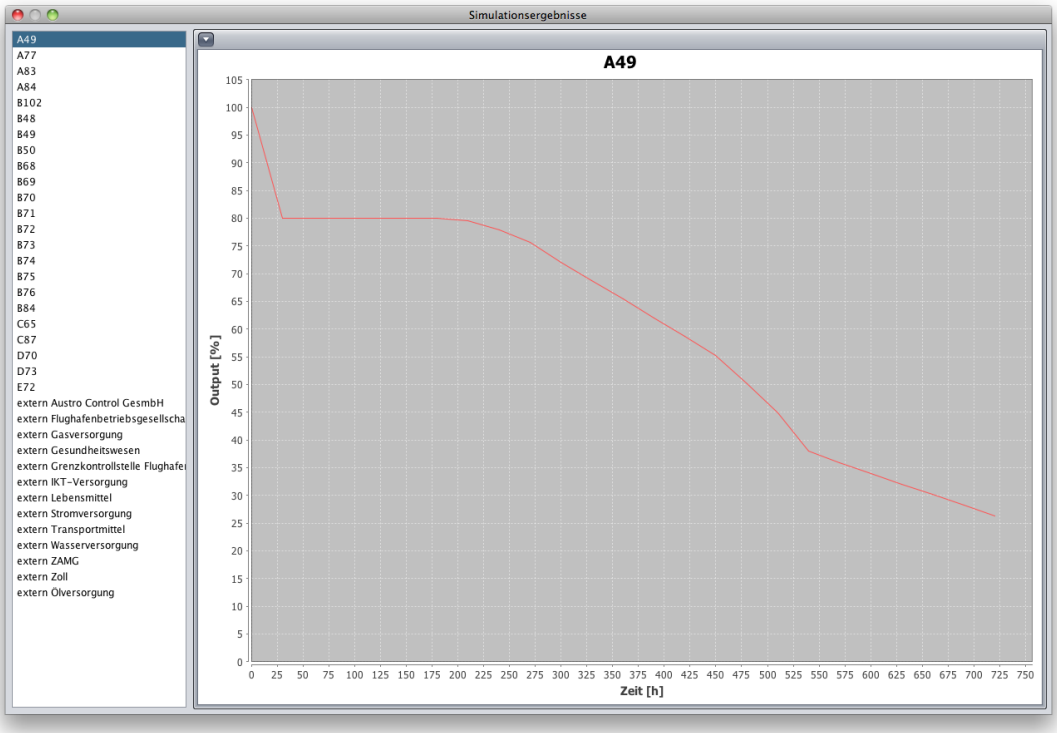

Abbildung 7.8: Ergebnis der Ressource A49

In diesem Fenster können die Ergebnisse der Simulation betrachtet werden. Hierbei wird die Leistung, der sogenannte Output, der einzelnen Knoten als zweidimensionale Kurve dargestellt. Hierbei stellt die horizontale Achse die Zeit dar. In vertikaler Richtung ist die Leistung des Knotens in Prozent aufgetragen. In der Auswahlliste am linken Rand kann der Benutzer den Knoten auswählen, welchen er betrachten möchte.

KAPITEL 8

Die Simulation

8.1 Das Simulationsmodell

Das *Simulationsmodell* beschreibt das Gedankenmodell, welches benutzt wurde, um die Berechnungen durchzuführen. Dieses Simulationsmodell basiert auf den in 5.2 festgelegten Grundlagen und findet seine Umsetzung in dem in 8.2 dargestellten Code.

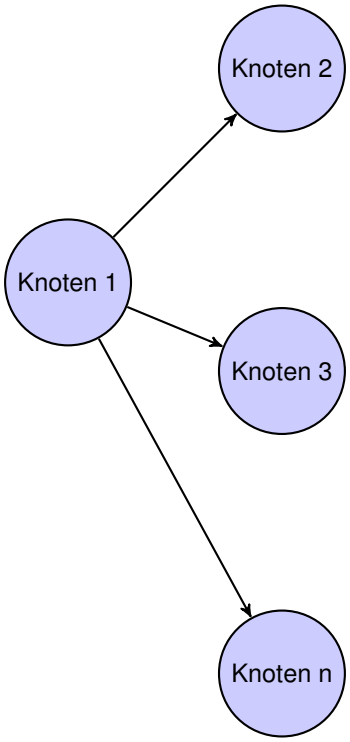

Abbildung 8.1: Initialschritt der Simulation für den Zeitpunkt t_1

Die Abbildung 8.1 stellt den ersten Schritt in der Simulation dar. Hierbei ist der *Knoten 1* der Knoten, welcher die initiale Störung aufweist. Dies soll heißen, dieser Knoten verfügt über einen von 100% abweichenden Output zum Zeitpunkt *t=0*.

Die „Störungswelle“ breitet sich somit vom Initialstörungsknoten aus und durchdringt prinzipiell das ganze Netzwerk, da hier ein vollvermaschtes Netzwerk betrachtet wird, jedoch hängt die Ausbreitung natürlich von den jeweiligen Funktionen der Knoten ab. Wenn Knoten keine Abhängigkeit zueinander aufweisen, so wird sich dementsprechend die „Störungswelle“ nicht weiter in die betreffende Richtung ausbreiten.

Rechentechnisch unterscheidet sich der erste Schritt der Simulation nicht von allen weiteren Schritten. Die Berechnung des ersten Zeitpunkts nach $t_0 = 0$, wobei gilt $t_1 = 1 \cdot \Delta t = \Delta t$, wird nach dem gleichen Muster ausgeführt wie die Berechnung aller weiterer Zeitpunkte. Allgemein gilt somit $t_i = i \cdot \Delta t$, wobei i die Berechnungsschritte in der Zeitachse beschreibt und Δt die Zeitsprünge in Stunden auf der Zeitachse. Der erste Berechnungsschritt wird hierbei für die Knoten $K' = \{(k,l)|k,l \in K, k \neq l\}$, wobei K die Menge aller Knoten im Netzwerk darstellt, ausgeführt. Es werden also alle Verknüpfungen der Knoten berechnet, jedoch darf ein Knoten nicht von sich selbst abhängen. Diese Knotenmenge wird mit K' bezeichnet.

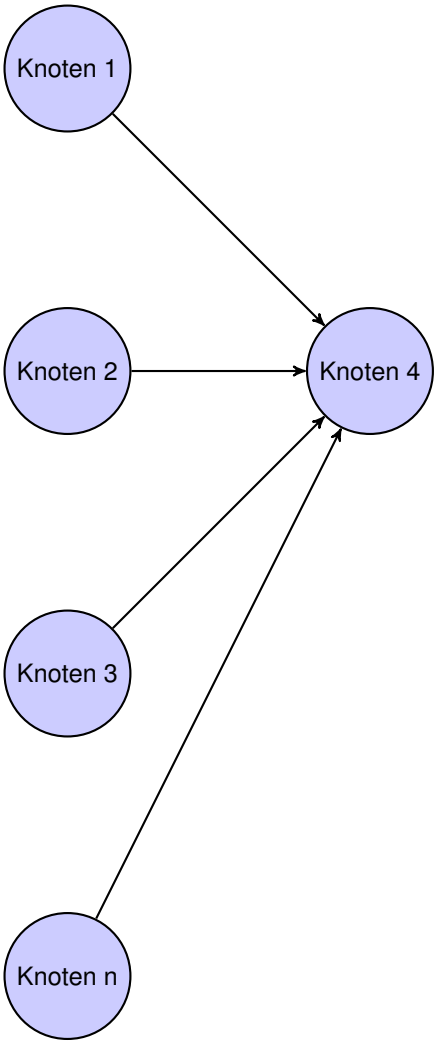

Abbildung 8.2: Grundschritt der Simulation

Der Grundschritt der Simulation, beschreibt den Rechenschritt, bei welchem aus allen Eingangswerten eines Knotens, in unserem Beispiel in der Abbildung 8.1 der *Knoten 4*, im Allgemeinen hier *Knoten i* genannt, der kleinste Eingangswert aller anderen Knoten des vorhergehenden Zeitpunktes errechnet wird. Es wird somit die stärkste Abhängigkeit festgestellt.

Mathematisch lässt sich dies wie folgt beschreiben:

$$Output_{i,t} = min(\{f_i(j, t-1) | j \neq i \wedge i, j \in K\})$$

t Der Zeitpunkt, für welchen der Output des Knotens i berechnet werden soll. t ist hierbei ein Element der Menge T, welche alle zu berechnenden Zeitpunkte darstellt. T eine geordnete

Liste von Zeitpunkten, wobei gilt: $t_0 \rightarrow t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_n$. In Worten: Auf den Zeitpunkt t_0 folgt der Zeitpunkt t_1. Auf den Zeitpunkt t_1 folgt der Zeitpunkt t_2 usw.. Auf den Zeitpunkt t_m folgt der Zeitpunkt t_{m+1}, wobei t_n den letzten Zeitpunkt in der Menge darstellt. Dies bedeutet, der in der Formel beschriebene Zeitpunkt $t-1$ ist der vorhergehende Zeitpunkt der Berechnung von t.

Es gilt:

$$t_{m+1} = t_m + \Delta t$$

Es ist somit der Zeitpunkt t_{m+1} der Nachfolger des Zeitpunkts t_m. Für m gilt: $m \in N$.

Δt beschreibt die zeitliche Granularität der Berechnung. Diese wirkt sich auf die Berechnungsdauer aus und wird unter 8.3 noch genauer behandelt.

i Der Knoten i ist der Knoten, für welchen der Output berechnet werden soll. In der Abbildung 8.1 ist dies der *Knoten 4*. Der Knoten i wird für die Funktionsmenge $f_i(j, t-1)$ festgehalten. Nur der Knoten j variiert.

$f_i(j, t-1)$ stellt die Funktionen dar, welche die Relationen zwischen den Knoten i und j definieren, wobei die Knoten j in der Graphik 8.1 den Knoten *1*, *2*, *3* usw. bis *n* entsprechen. Ausgenommen für j ist der *Knoten 4*. *Knoten 4* entspricht hierbei dem Knoten i. Die Funktionen wie folgt zu betrachten: Alle Knoten j stellen einen eingehenden Knoten vom Knoten i dar, wobei aufgrund dieser Eingangswerte der Knoten j durch die Funktionen $f_i(j, t-1)$ für alle Knoten j aus K der Output des Knotens i zum vorhergehenden Zeitpunkt $t-1$ das Ergebnis der Funktionsmenge ist. Die genannten Funktionen entsprechen somit den im Graphen eingezeichneten gerichteten Kanten.

K stellt die Menge aller Knoten im Netzwerk dar, wobei gemäß der Graphik 8.1 gilt:

$$n = \mid K \mid$$

In Worten: Der Index n entspricht der Anzahl der Elemente der Menge der Knoten des Netzwerks K.

$j \neq i$ Die Bedingung $j \neq i$ beschreibt, das keine Funktionsschleifen im Graphen enthalten sein dürfen. Dies bedeutet, ein Knoten kann keine Auswirkungen auf sich selbst haben. Daraus folgt, wenn sich die Eingangswerte des Knoten i über einen Berechnungsschritt nicht verändern, sich keiner der anderen Knoten auf den Knoten i in einem Rechenschritt auswirkt, so wird der Knoten i seine Leistungsfähigkeit, seinen Outputwert beibehalten. Der Output des Infrastrukturelements wird sich ohne äußere Einwirkung nicht verringern. Ein Knoten, ein Infrastrukturelement, kann nicht an sich selbst „zugrunde“ gehen.

Würde man jedoch die Möglichkeit des Steigens des Outputs eines Knoten in Betracht ziehen, so müsste eine solche Selbstbeeinflussung eines Knotens in das Modell aufgenommen werden, damit sich Knoten der Simulation ähnlich verhalten könnte wie folgende Beispiele anderer „Systeme“. Wenn zum Beispiel ein Mensch unzureichend Wasser zur Verfügung gestellt bekommen würde, so würde er, ab einem gewissen Zeitpunkt, daran zugrunde gehen, auch wenn man die Menge an zugeführtem Wasser wieder auf Normalniveau setzen würde. Ein anderes Beispiel wäre auch das Außer-Kontrolle-Geraten eines Atomkraftwerks.

8.2 Der Algorithmus zur Berechnung

In diesem Abschnitt wird der wichtigste Teil der Anwendung erklärt. Es handelt sich hierbei um den Programmcode zur schlussendlichen Simulation der Infrastrukturnetzwerke. Dieser Programmcode führt die Berechnungen durch, welche in 8.1 beschrieben wurden.

Die Übergabeparameter

Resource init

Dieser Parameter stellt den initialen Knoten dar, bei welchem eine Störung aufgetreten ist. *init* ist vom Typ *Resource* und kann ein Infrastrukturelement oder eine Ressource sein. Wie unter (5.2) beschrieben, sind Infrastrukturelemente und Ressourcen im Modell gleichwertig.

Von diesem initialen Knoten startet sozusagen die Ausbreitungswelle der Störung. Als bildliches Beispiel mag der Wurf eines Steins ins Wasser dienen. Wobei sich in unserem modellhaften See andere Steine befinden. Es breitet sich sozusagen die erste Welle von der Eintrittsstelle des Steins aus, wobei die Welle im Weiteren von den anderen Steinen im See reflektiert wird und ebenfalls zurückkommen kann zum initial geworfenen Stein. Hierbei wurde in dieser Diplomarbeit nur die Fragestellung eines initial gestörten Knotens behandelt. Es ist somit nur möglich einen Stein in den See zu werfen. In dieser Diplomarbeit wurde das Konzept mehrerer gestörter Knoten zum Zeitpunkt Null nicht behandelt. Es kann somit nur die initiale Störung eines Knotens simuliert werden, nicht jedoch die Simulation der Störung mehrerer Knoten zum Zeitpunkt Null. Daher verfügt die Simulationsfunktion *startSimulation* auch nur über eine Ressource als Übergabeparameter.

Double valueInit

Dieser Parameter setzt den Startwert des gestörten Knotens. Dieser Wert wird als Datentyp *Double* übergeben. Der Übergabewert muss zwischen 0 und 100 liegen, wobei ein Übergabewert von 100 nicht sinnvoll ist, da in diesem Fall sich keine Veränderungen im Netzwerk abzeichnen würde, da bei voller Leistungsfähigkeit aller Knoten das Netzwerk stabil bei 100 % Leistung bei allen Knoten bleibt. Der Algorithmus würde in diesem Fall zwar seine Berechnungen normal durchführen, jedoch wäre das Ergebnis natürlich nichtig, da alle Knoten weiterhin volle Leistung bringen und nicht beeinträchtigt werden.

int deltaT

Dieser Parameter bestimmt die Schrittweite in der Berechnung auf der Zeitachse. Dies bedeutet, bei kleinem *deltaT* werden mehr Punkte in Abhängigkeit von der Zeitachse berechnet. Daher müssen im Ergebnisgraphen dann weniger Punkte der Kurven interpoliert werden. Ebenso sinkt die Gesamtabweichung der gezeichneten Ergebniskurven vom realen Verlauf der Kurven. Die errechneten Kurven sind somit bei kleinerem *deltaT* genauer.

Jedoch wirkt sich *deltaT* auf die Rechendauer aus. Bei kleinerem *deltaT* steigt die notwendige Rechenzeit, wobei sie bei höherem *deltaT* sinkt. Somit ist *deltaT* ein Mittel um zwischen Rechenaufwand und Genauigkeit der Berechnung abzuwägen. *deltaT* kann vom Benutzer vor der Berechnung eingestellt werden und verfügt über einen Wertebereich von 1 bis 50, wobei nur ganze Zahlen einzustellen sind, weil die höchste Granularität der Berechnung auf der Zeitachse einen Wert von einer Stunde aufweist. Die höchste Rechengenauigkeit in zeitlicher Hinsicht beträgt somit eine Stunde. Bei einer Auflösung von einer Stunde werden im Ergebnisgraphen keine Werte mehr interpoliert, sondern ausschließlich die berechneten Werte angezeigt. Jedoch bedeutet ein *deltaT* von einer Stunde auch einen maximalen Rechenaufwand. Die Frage der Rechenzeit wird im Abschnitt 8.3 noch ausführlich behandelt.

int tMax

Dieser Parameter beschreibt die maximale Zeit, für welche die Kurven berechnet werden. *tMax* hat hierbei einen Wertebereich von 1 bis 720 Stunden. 720 Stunden entsprechen 30 Tagen. *tMax* kann hierbei mit einem Abstand von einer Stunde bestimmt werden. *tMax* hat einen starken Einfluss auf die zu erwartende Rechenzeit und kann vom Benutzer mittels der Anwendungsoberfläche bestimmt werden. Der Einfluss auf die Berechnungsdauer wird im Abschnitt 8.3 dargelegt.

```
1  /**
2  * @param init The specific resource node where the simulation wave starts
3  * @param valueInit The initial output-level of the inital node
4  * @param deltaT The step value in time for the simulation, deltaT is of
5  * dimension [h]
6  * @param tMax The end in time to calculate to, tMas is of dimension [h]
7  */
8
9  public ArrayList<Resource> startSimulation(Resource init, Double valueInit,
       int deltaT, int tMax)
10 {
11
12   int t = 0;
13   int i3 = 0;
14   double epsilon=0.1d; // Epsilon area near 0
15
16   // Define initial values for t=0
17   for (int i = 0; i < getReslist().size(); i++) {
18     ((Resource) getReslist().get(i)).data.add(new ArrayList<Double>());
19     if (init.equals(getReslist().get(i))) {
20       getReslist().get(i).outputT.add(new Output(i3, t, valueInit));
21     } else {
```

```
22        getReslist().get(i).outputT.add(new Output(i3, t, 100d));
23      }
24    }
25
26    t = t + deltaT;
27    while (t <= tMax) {
28      i3++;
29      for (int i = 0; i < getReslist().size(); i++) {
30        for (int i2 = 0; i2 < getReslist().size(); i2++) {
31          if (i != i2) {
32
33            Double successor = getReslist().get(i2).outputT.get(i3 -
                  1).getOutput();
34
35            Double value = null;
36            if ((Function) getFunklist().get(Tools.formatIndex(i2, i)) !=
                  null) {
37
38            // Calculated with t
39            value = ((Function) getFunklist().get(Tools.formatIndex(i2,
                  i))).evaluate(new Double(t), new Double(successor));
40
41            // If the calculated value gets into the epsilon area of 0, its
                  set to zero to avoid numerical issues (NaN)
42            if(value<epsilon)
43            {
44            value=0d;
45            }
46
47            if (Double.isNaN(value)) {
48              controller.InitialWindowController.logger.warn("Funktion
                    "+((Function) getFunklist().get(Tools.formatIndex(i2,
                    i))).toString()
49              +" mit t="+t + " und input= "+successor+"% konnte nicht
                    ausgewertet werden. (NaN)");
50            }
51
52            } else {
53            // no function is defined, so there is no influence by this node,
54            // so its fully connected => output is 1 related to node i2
55            value = 100.0d;
56            }
57
58          // Test if the value for t is set
59          try {
60            ((Resource) getReslist().get(i)).data.get(i3);
61          } catch (IndexOutOfBoundsException e) {
62            ((Resource) getReslist().get(i)).data.add(new ArrayList<Double>());
63          } finally {
64            ((Resource) getReslist().get(i)).data.get(i3).add(new
                  Double(value));
65          }
66        }
```

```
67      } // Inner node loop
68
69      ArrayList<Double> kiT = getReslist().get(i).data.get(i3);
70      // Successor is used for calculation of next value => monoton falling
71      kiT.add(new Double(getReslist().get(i).outputT.get(i3 - 1).getOutput()));
72      Double min = 100.0; // implicit max = 1.0
73      for (int i4 = 0; i4 < kiT.size(); i4++) {
74        min = Math.min(min, kiT.get(i4).doubleValue());
75      }
76      getReslist().get(i).outputT.add(new Output(i3, t, new Double(min)));
77
78      }// Outer node loop
79      t = t + deltaT;
80    } // Time step loop
81
82    controller.InitialWindowController.logger.info("Simulation finished");
83    // Simulation finished
84    return getReslist();
85  }
```

Codeerklärung

Zeile 12-14:

Hierbei handelt es sich um Hilfsvariablen für den Schleifendurchlauf, beziehungsweise den Startzeitpunkt 0. *epsilon* beschreibt den Zahlenbereich um Null. Berechnungen, welche gegen Null gehen, beziehungsweise sich um den Nullpunkt bewegen, werden auf Null gesetzt. Um ein numerisches Schwingen, im Bereich um Null, zu vermeiden wurde *epsilon* eingeführt. Da sich *epsilon* auf das Schwingen der Output-Leistung bezieht, welche eine Auflösung von Eins aufweist, wurde *epsilon* auf den Wert 0.1 gesetzt und liegt damit bei einem Zehntel des nächsthöheren Wertes, welcher als Output errechnet werden kann. Aufgrund der Ergebnisse mehrerer Testläufe hat sich herausgestellt, das 0.1 ein geeigneter Wert für *epsilon* ist.

Zeile 17-24:

In den Zeilen 17 bis 24 wird der Output-Vektor des Netzwerks initialisiert. Hierbei wird der Output-Wert eines jeden Knotens auf 100 % gesetzt, mit Ausnahme des Knotens, welcher der initiale Knoten mit einer Störung ist. Der Knoten mit der Störung erhält im Initial-Output-Vektor den Wert des Parameters *valueInit*.

Zeile 26:

Es wird *t* um *deltaT* erhöht, um so den t-Wert für den ersten Berechnungspunkt zu erhalten.

Zeile 27:

Der Schleifenkopf läuft die Zeitachse ab bis zum gewünschten maximalen Zeitwert.

Zeile 29-30:

Die beiden Schleifenköpfe durchlaufen die Knotenliste jeweils. Somit ergibt sich das kartesische Produkt $K \times K$ innerhalb der Schleifen, wobei K die Menge aller Knoten im Netzwerk ist. Die Knoten K werden in dieser Diplomarbeit auch als Ressourcen bezeichnet (5.2).

Zeile 31:

Diese If-Verzweigung filtert das weiter oben genannte kartesische Produkt $K \times K$. Und so ergibt sich $K' = \{(k,l)|k,l \in K, k \neq l\}$. Also das kartesische Produkt aller Knoten, jedoch ohne der Knotentupel mit sich selbst. Dies ist nicht notwendig, da ein Knoten keine Rückkopplung zu sich selbst haben darf.

Zeile 33:

Es wird der Output-Wert des vorhergehenden Zeitpunkts ausgelesen, also $Output_{i,t-1}$ wobei t der Zeitpunkt des Schleifendurchlaufs ist.

Zeile 36:

Es wird überprüft, ob zwischen den betreffenden Knoten eine Relation besteht. Eine solche besteht, wenn eine Funktion vorhanden ist, welche die Abhängigkeit der beiden Knoten definiert.

Zeile 42-45:

Wenn sich der errechnete Output des Knotens innerhalb der *epsilon*-Umgebung befindet, also $Output < epsilon$, dann wird der Wert auf Null gesetzt, um numerische Konvertierungsprobleme zu verhindern, welche bei sehr kleinen Zahlen auftreten.

Zeile 47-50:

Es wird überprüft, ob der aus der Funktion zweier Knoten errechnete Wert einen verwertbaren Zahlenwert ergibt. Es lässt sich kein Funktionswert ermitteln, wenn keine Interpolation für *Matlab* möglich war. Dies ist dann der Fall, wenn die gegebenen Daten der Relation zweier Knoten nicht ausreichend waren. Dies bedeutet, es war *Matlab* nicht möglich einen Wert für die gegebene Stelle im Graphen zu interpolieren. Der in Abbildung 5.4 gezeigte Graph ist vollständig. Er ist für jeden Punkt der Ebene im Definitionsbereich definiert. Dies muss jedoch nicht so sein. Wenn es nicht möglich ist, jeden Punkt zu interpolieren, so wird dies durch dieses Codefragment abgefangen und als Warnmeldung in die Datei *log.html* geschrieben.

Zeile 55:

Die Knoten stehen zueinander nicht in Relation. Dies bedeutet das Verhalten des einen Knoten beeinflusst nicht das Verhalten des anderen Knoten. Daher wird eine volle Verbindung mit 100% gesetzt. In diesem Schritt wird aus dem möglicherweise nur partiell definierten Netzwerk, ein vollvermaschtes Netzwerk.

Zeile 59-65:

Diese Zeilen lösen kein logisches, sondern nur ein programmiertechnisches Problem. Wenn es sich um den ersten Wert des Lösungsvektors zum Zeitpunk *t* handelt, so wird das Array initialisiert.

Zeile 69-76:

Es wird der minimale Wert aller Funktionen eines Knotens ermittelt, welche zum Zeitpunkt *t-1* berechnet wurden. Somit wird der minimale Output aller Nachbarknoten des aktuellen Knotens des vorhergehenden Zeitpunkts errechnet. Diese Vorgehensweise garantiert eine monoton fallende Ergebniskurve, wobei durch die Zeile 72 nochmals ein Maximalwert von 100% garantiert wird.

Zeile 79:

In dieser Zeile wird der *t*-Wert um den *Delta*-Wert erhöht. Es wird somit der nächste zu berechnende Zeitpunkt festgelegt.

Zeile 82-84:

Das Ende der Simulation wird in die Log-Datei *log.html* eingetragen und die Methode liefert einen Rückgabewert, welcher eine ArrayList von *Resource*-Objekten darstellt, welche jeweils ihren betreffenden Ergebnisvektor als Arraylist enthalten.

8.3 Laufzeitberechnung

Bei genauer Betrachtung des Codes kommt man auf folgende Laufzeitsumme:

Genaue Laufzeitberechnung

$$\sum = n \cdot t_1 + t_2 + \rho \cdot \frac{t_{max}}{\Delta t} \cdot (n^2 - n) \cdot t_3 + \frac{1}{\rho} \cdot \frac{t_{max}}{\Delta t} \cdot (n^2 - n) \cdot t_4 + \frac{t_{max}}{\Delta t} \cdot (n^2 - n) \cdot t_5 + \frac{t_{max}}{\Delta t} \cdot n \cdot t_6 + \frac{t_{max}}{\Delta t} \cdot n^2 \cdot t_7 + \frac{t_{max}}{\Delta t} \cdot t_8 + t_9$$

n Anzahl der Knoten im Netzwerk. Diese ist gleich der Anzahl, der an die Methode übergebenen *Resource*-Objekten.

t_1 Laufzeit der Zeilen 18 bis 23. Diese ist konstant und im Allgemeinen zu vernachlässigen.

t_2 Laufzeit der Zeile 26. Diese ist konstant und im Allgemeinen zu vernachlässigen.

t_{max} Dies ist der maximale Zeitwert, bis zu welchem die Berechnung durchgeführt werden soll. Dieser hat entscheidenden Einfluss auf die Berechnungsdauer.

Δt Dieser Wert beschreibt die Genauigkeit der Berechnung über die Zeitachse und hat entscheidenden Einfluss auf die Berechnungsdauer.

ρ beschreibt die Dichte der Definitionen von Funktionen des Netzwerkgraphen. Eine Dichte von 1 entspricht hierbei einem vollvermaschten Netzwerk, in welchem jede Verbindung zwischen zwei Knoten durch eine Funktion definiert ist, welche mittels Matlab berechnet wird. Dies bedeutet, es gibt keine trivialen Relationen im Netzwerk. Kein Knoten ist unabhängig von den anderen Knoten. ρ ist somit wie folgt definiert: $\rho = \frac{Anzahl\ der\ definierten\ Funktionen}{n^2-n}$

t_3 Laufzeit der Zeilen 39 bis 50. Dieser Code wird $\rho \cdot \frac{t_{max}}{\Delta t} \cdot (n^2 - n)$-mal ausgeführt und beinhaltet den, auf die einzelnen Befehle bezogen, rechenintensivsten Teil. Der rechenintensivsten Code ist hierbei die Evaluierung der Funktionswerte mittels *Matlab*.

t_4 Laufzeit der Zeile 55. Diese wird $\frac{1}{\rho} \cdot \frac{t_{max}}{\Delta t} \cdot (n^2 - n)$-mal aufgerufen. Diese Zeile wird somit im inversen Verhältnis zur Funktionsdichte des Netzwerkgraphen aufgerufen.

t_5 Laufzeit der Zeilen 33 bis 36 und der Zeilen 59 bis 66. Dies beschreibt den Code innerhalb aller Hauptschleifen. Diese werden, in Abhängigkeit von n, gleich oft aufgerufen und können daher zusammengefasst werden. Dieser Code wird $\frac{t_{max}}{\Delta t} \cdot (n^2 - n)$-mal ausgeführt.

t_6 Laufzeit der Zeilen 69 bis 72 und der Zeile 76. Diese werden $\frac{t_{max}}{\Delta t} \cdot n$-mal ausgeführt.

t_7 Laufzeit der Zeile 74. Diese wird $\frac{t_{max}}{\Delta t} \cdot n^2$-mal ausgeführt.

t_8 Laufzeit der Zeilen 79. Diese wird $\frac{t_{max}}{\Delta t}$-mal ausgeführt.

t_9 Laufzeit der Zeile 82 bis 84. Diese werden einmal aufgerufen.

Die weiter oben dargelegte Berechnung der Ausführungszeit lässt sich nun vereinfachen.

Wir vereinfachen nun die oben angeführte Berechnung. Wir vernachlässigen die Einkalkulierung der Rechenzeiten t_2, t_4 und t_9, da ihre tatsächliche Ausführungszeit irrelevant ist im Verhältnis zum restlichen Code. Die Berechnungen der Funktionswerte mittels *Matlab* ist für die Laufzeit entscheidend, und daher ist der Faktor ρ für die praktische Berechnungsdauer von Relevanz. t_5 ist ebenfalls zu vernachlässigen. Nur die Funktionsberechnung mittels *Matlab* ist laufzeittechnisch relevant, da die Berechnung der dreidimensionalen Funktionen bedeutsame Rechenzeit in Anspruch nimmt. Die Berechnung eines Funktionswerts nahm hierbei auf einem *Macintosh, Mac OS X, Version 10.6.8, 2.93 GHz Intel Core 2 Duo, 4 GB 1067 MHz DDR3* durchschnittlich 3 Millisekunden in Anspruch.

Vereinfacht ergibt sich folgende Laufzeit:

$$\sum' = \rho \cdot \frac{t_{max}}{\Delta t} \cdot (n^2 - n) \cdot t_3$$

Wir nehmen an, für den praktischen Fall, dass sich das zu berechnende Netzwerk aus einer größeren Anzahl von Knoten zusammensetzt, für die gilt: $n^2 \gg n$. Dies führt zu $n^2 - n \approx n^2$. Da t_3 nicht von den Parametern der Methode abhängt, ist t_3 ein Rechenschritt mit einer konstanten Laufzeit.

Daraus ergibt sich für die asymptotische Laufzeitberechnung [4], welche mit Θ in der Informatik bezeichnet wird, folgende Formel:

$$\Theta(\rho, t_{max}, \Delta t, n) = \rho \cdot \frac{t_{max}}{\Delta t} \cdot n^2$$

Die als Grundlage für diese Diplomarbeit benutzten Daten stellen ein ρ von Eins dar, da jede Relation zwischen den gegebenen Knoten definiert ist. Ebenso sind Relationen als Funktionen definiert, welche eine Ebene als Funktionskurve mit einem Wert von 100% liefern, und somit voneinander unabhängig sind.

KAPITEL 9

Critical Infrastructure Tool

Beim *Critical Infrastructure Tool* handelt es sich um eine Hilfsanwendung, welche im Zuge der Entwicklung der Hauptanwendung erstellt wurde. Das *Critical Infrastructure Tool* dient hierbei zur Vorbereitung und Umwandlung der Daten in ein für die Anwendung brauchbares Format.

Da es erwünscht war, dass die verfügbaren Daten der Anwendung als Initialwerte zur Verfügung stehen sollte, mussten diese Daten aufbereitet werden.

Die Daten wurden als *Microsoft*-Excel-Tabellen (Dateiendung: xls) zur Verfügung gestellt. Es zeigte sich sehr schnell, dass die gegebenen Daten in ihrem Ausmaß für die manuelle Übertragung in die Anwendungsstruktur bei weitem zu umfangreich waren, daher wurde die Hilfsanwendung *Critical Infrastructure Tool* notwendig. Diese Hilfsanwendung stellt hierbei ein Entwicklungstool dar, welches über keine Benutzeroberfläche verfügt und ausschließlich in der Entwicklungsumgebung 6.5 benutzt und getestet wurde.

Diese Hilfsanwendung überträgt hierbei die in den Excel-Tabellen enthaltenen Daten in die Dateien *InitialDataCall.java* und *InitialDataExample.java*. Diese sind in der Abbildung 6.5 dargestellt. *InitialDataCall.java* enthält hierbei die Methodenköpfe zum Einfügen der einzelnen Ressourcen. *InitialDataExample.java* enthält die tatsächlichen Daten, welche in die Datenbank eingefügt werden. Die Daten werden hierbei mittels der entsprechenden *DAO*-Klasse eingefügt.

Aufgrund von Unregelmässigkeiten in den ursprünglichen Excel-Tabellen, mussten diese zur weiteren Verarbeitung mittels *Critical Infrastructure Tool* zuerst manuell vorbearbeitet werden. Dies war aufgrund vereinzelter Zeilenverschiebungen oder Spaltenverschiebungen in manchen Tabellen notwendig.

Würden sich die Ausgangsdaten der Anwendung ändern, und wollte man diese wiederum in den initialen Datensatz der Anwendung integrieren, so wäre dies mittels der Hilfsanwendung

Critical Infrastructure Tool möglich, jedoch müsste dies wiederum in der Entwicklungsumgebung (6.5) geschehen, da *Critical Infrastructure Tool* nicht für die eigenständige Ausführung konzipiert wurde. Dies war für den prototypischen Betrieb der Software nicht vorgesehen.

Critical Infrastructure Tool verwendet für den Zugriff auf die Excel-Tabellen die Bibliothek *Apache POI - the Java API for Microsoft Documents* in der Version 3.9 [13].

Teil III

Schlussteil

KAPITEL 10

Ergebnisse

10.1 Erwartete Ergebnisse

Im Allgemeinen wurde als Ergebnis bei den einzelnen zeitlichen Verläufen der Knoten ein ähnlicher Kurvenverlauf erwartet, wie ihn die Eingangsdaten aufweisen. Es war anzunehmen, dass die Ergebniskurven den Funktionen der Eingangsdaten ähnlich sein würden. Jedoch durfte man davon ausgehen, dass die resultierenden Kurven stärker fallen würden, als die Funktionen der einzelnen Relationen. Dies war anzunehmen, da aufgrund von Überlagerungen des Simulationsmodells (8.1) eine Verstärkung der absteigenden Funktionen absehbar war.

Eine weiteres erwartetes Ergebnis war die Bildung von sogenannte „Inseln“ im Netzwerk. Man nehme an, es wird der Ausfall eines Infrastrukturelements simuliert. Dieses sei nun angenommenerweise ein Postamt. Nun wird der Ausfall des Postamts möglicherweise den Ausfall weiterer Infrastrukturelemente nach sich ziehen, jedoch werden alle Knoten, welche vom Postamt weder direkt noch indirekt abhängen, weiterhin ihre Funktion erfüllen können und keinen Leistungsabfall aufweisen.

Aufgrund der Vielzahl an vorhandenen Relationen in den Daten, war zu erwarten, dass eine hohe Abhängigkeit der Knoten zueinander besteht. Daher konnte man prognostizieren, dass die Störung eines einzelnen Knoten sich meist auf das gesamte Netzwerk auswirken würde. Dies bedeutet, ein gestörter Knoten wird möglicherweise zum Ausfall vieler anderer Knoten im Netzwerk führen.

10.2 Tatsächliche Ergebnisse

Ergebnisse anhand eines Beispiels

Diese Simulation wurde mit folgenden Parametern durchgeführt:

- Initialknoten: extern Stromversorgung
- Initialwert: 80%
- Δt: 30 Stunden
- T max: 720 Stunden

Es wurde also ein leichter Einbruch der externen Stromversorgung simuliert. Diese Simulation wurde grobkörnig berechnet, um die Auswirkungen auf das Ergebnis zu zeigen. Simuliert wurde der ganze an Daten zur Verfügung stehende Zeitraum.

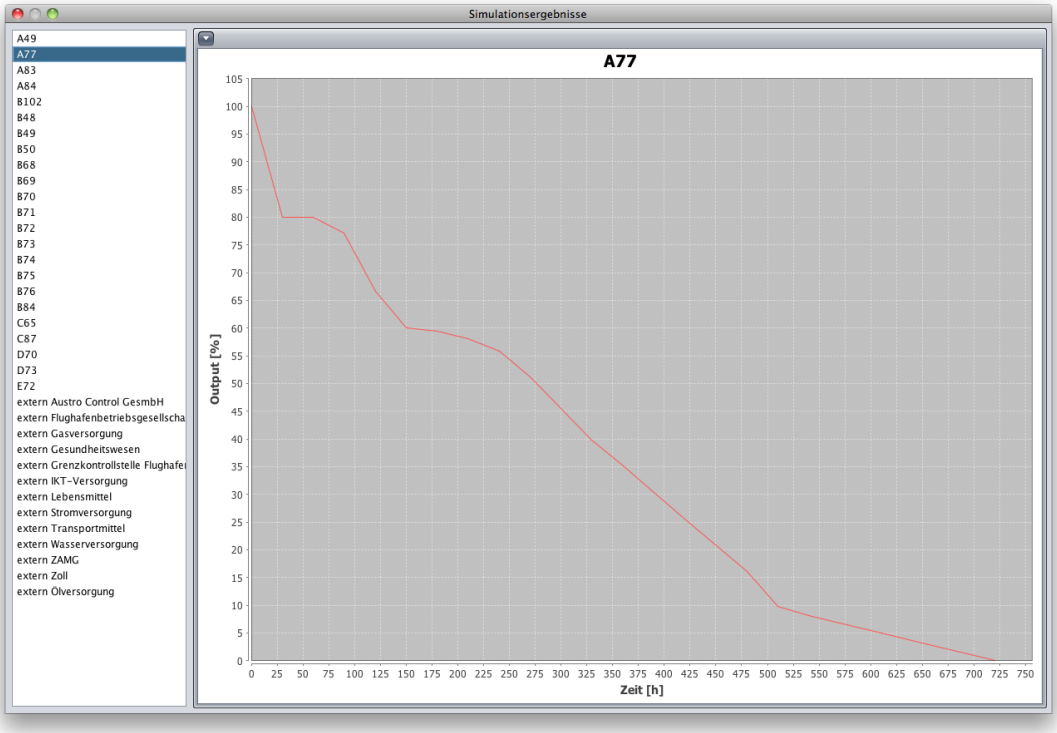

Abbildung 10.1: Ergebnis der Ressource A77

In der Abbildung 10.1 lässt sich am ersten berechneten Punkt direkt die Auswirkung der gewählten Schrittweite Δt erkennen. Dieser erste errechnete Punkt liegt bei 30 Stunden, was genau dem Δt entspricht. Der Teil der Kurve, zwischen dem Startzeitpunkt $t = 0$ und dem Zeitpunkt $t = 30$ wurde linear interpoliert. Hierbei wurde für den Zeitwert von 30 Stunden ein Output von 80 % errechnet.

Input [%]	1 Stunde	1 Tag	7 Tage	30 Tage
100	100	100	100	
75	75	75	75	
50	50	50	50	
25	25	25	25	
0	0	0	0	

Tabelle 10.1: Beispieldaten *externe Stromversorgung* $\rightarrow$ *A77*

Wie man mit Hilfe der Tabelle 10.1 abschätzen kann, ist der errechnete Wert von 80% nach 30 Stunden gemäß dem Simulationsmodell 8.1 zutreffend. Da es sich um den ersten errechneten Funktionswert handelt, kann dieser ausschließlich von der Ressource *externe Stromversorgung* abhängen.

Des Weiteren kann man die Knickpunkte der Kurve erkennen, welche jeweils an den errechneten Stellen auftreten, somit also immer bei einem Vielfachen von Δt, jedoch halten sich die Verzerrungen, abgesehen vom ersten Berechnungspunkt, in Grenzen.

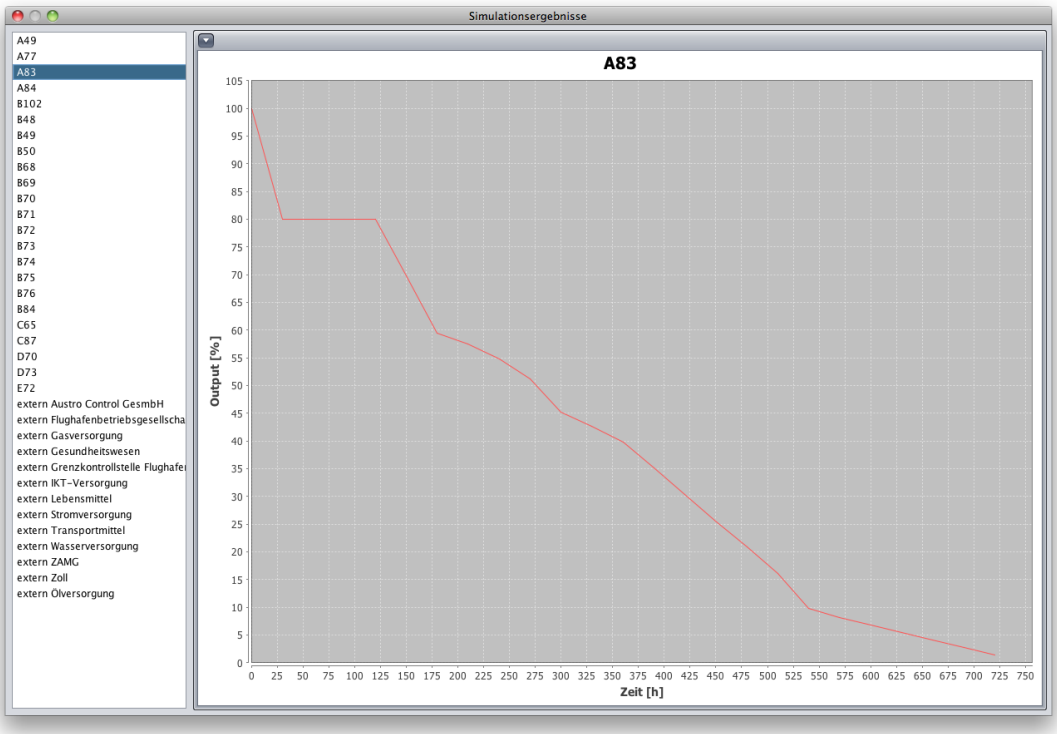

Abbildung 10.2: Ergebnis der Ressource A83

In der Abbildung 10.2 lässt sich das gleiche Verhalten der Kurve in den ersten 30 Stunden wie in der Abbildung 10.1 betrachten. Der Knoten *A83* verfügt über das gleiche Abhängigkeitsverhältnis zur Ressource *externe Stromversorgung*.

Die horizontale Linie zwischen der Stunde 30 und der Stunde 120 lässt sich so erklären, dass der Knoten *A83* in diesem Zeitraum von keinem anderen Knoten beeinflusst wird. Ebenso ist seine direkte Abhängigkeit von *externe Stromversorgung* zwar gegeben, jedoch fällt die Funktion der direkten Abhängigkeit nicht weiter. Erst nach 120 Stunden fällt der Knoten *A83* wieder in die Abhängigkeit von einem anderen Knoten. Dies liegt wahrscheinlich daran, dass ein Eingangsknoten von *A83* einen Scheitelpunkt der Funktion *Knoten x* $\rightarrow$ *Knoten A83* unterschritten hat, und somit den Knoten *A83* wiederum beeinflusst.

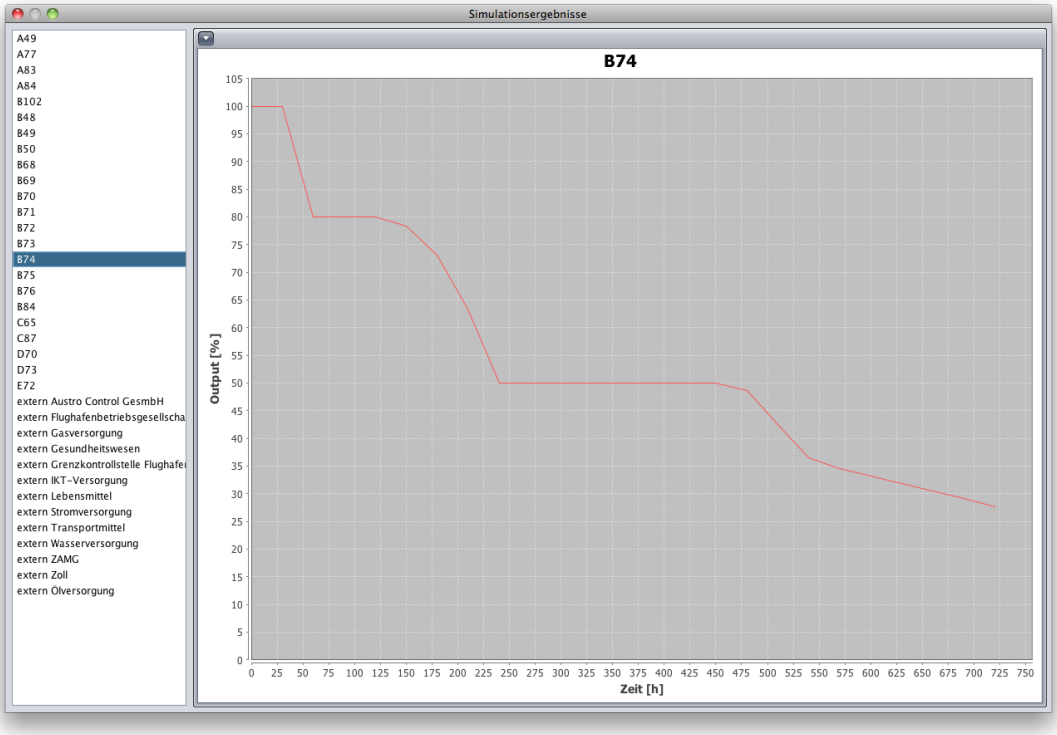

Abbildung 10.3: Ergebnis der Ressource B74

In der Abbildung 10.3 erkennt man, dass die Kurve ähnliche Merkmale aufweist, wie die Kurve in der Abbildung 10.2. Die Phase der Unabhängigkeit zwischen dem Zeitwert 240 und 450 entspricht dem gleichen Verhalten, wie bereits in der Abbildung 10.2, beziehungsweise wie in dieser Kurve selbst in der Zeit von 30 bis 120 Stunden. Man darf davon ausgehen, dass der Grund dafür derselbe ist, die vorhergehende Unabhängigkeit gefolgt vom Unterschreiten eines Scheitelpunktes einer Relation.

Wie an den hier angeführten Kurven ersichtlich ist, entsprechen die Kurven in ihrer Form den im Abschnitt 10.1 beschriebenen erwarteten Ergebnissen. Die berechneten Kurven ähneln in ihrer Form den Querschnitten der Funktionen der Relationen der Knoten. Wenn man Querschnitte des Graphen der Abbildung 5.4 betrachtet, so ähneln die errechneten Kurven diesen Querschnitten. Hierbei darf man die Kurven als Mischformen der Querschnitte eines solchen Graphen über die Zeit und den Input sehen, da ein betreffender Knoten von beiden abhängt. Somit entsprechen die Kurven in diesem Beispiel dem erwarteten Ergebnis.

Die Ergebnisse im Allgemeinen

Die in 10.1 angeführten erwarteten Ergebnisse sind nur teilweise eingetreten. Das Kurvenverhalten entspricht im Allgemeinen dem erwarteten Muster. Die errechneten Kurven haben die vorhergesagte Form. Jedoch hat sich gezeigt, das die angenommene „Inselbildung“ nicht in dem Ausmaß eingetreten ist wie erwartet. Als Beispiel sei hier die Simulation eines kompletten Stromausfalls beschrieben. Es hat sich gezeigt, dass hierbei ein Großteil des Netzwerks gegen Null geht.

Ebenso zeigte sich bei der Simulation des Ausfalls einzelner Infrastrukturelemente, dass das bestehende Modell tendenziell den Totalausfall anstrebt, was auf eine zu starke Abhängigkeit zwischen den einzelnen Infrastrukturelementen hindeutet. Dies bedeutet, dass bereits die Zerstörung eines Infrastrukturelements das Erliegen des kompletten Systems zur Folge haben kann.

KAPITEL 11

Schlussfolgerungen, Fragen und Weiterentwicklungsmöglichkeiten

11.1 Schlussfolgerungen

Es lässt sich schlussfolgern, dass das betrachtete Netzwerk nicht sehr störungsresistent ist und auf unerwartete Ereignisse empfindlich reagiert.

Um den Zerfall des Netzwerks bei Ausfall eines Infrastrukturelements zu vermeiden, bietet sich, wie in der IT üblich, die Verwendung von redundanten Systemen an. Das Verwenden von Redundanzen vermeidet das Problem des *Single Point of Failure*. Unter einem *Single Point of Failure* versteht man, wenn der Ausfall einer Komponente eines Systems den Niedergang des gesamten Systems herbeiführt [2].

11.2 Fragen

Eine bedeutsame Frage, welche sich bei der Ausarbeitung dieser Diplomarbeit gestellt hat, ist inwieweit eine Erhebung der Ausgangsdaten, welche die Grundlage der Simulation bilden, überhaupt möglich ist (siehe 5.3). Diese Frage bleibt wohl unbeantwortet, sofern sie überhaupt zu beantworten ist. Möglicherweise konnten die Urheber der Daten dieser Diplomarbeit dazu erweiterte Erkenntnisse erringen, jedoch gilt bezüglich dieser Diplomarbeit weiterhin die Hypothese, dass eine genaue mathematische Erfassung der Beziehung zwischen zwei Infrastrukturelementen in der Praxis nicht möglich ist, und eine solche Erhebung bestenfalls eine Näherung der Realität darstellen kann.

Eine weitere Frage, welche sich aufdrängt, ist die Frage der tatsächlichen Evaluierbarkeit dieser Simulation. Eigentlich ist diese Simulation nur tatsächlich evaluierbar, wenn ein zuvor simulierter Störfall dann in der simulierten Form eintritt, und man das Verhalten des Systems misst,

sofern die quantitative Messung des Systemverhaltens überhaupt möglich ist. Danach könnte man die gemessenen Werte mit den errechneten Werten der Simulation vergleichen und somit die Korrektheit der zugrunde liegenden Daten und des Simulationsmodells 8.1 nachweisen.

Wie dem vernunftbegabten Menschen schnell klar sein dürfte, ist ein solches Vorgehen im Allgemeinen nicht möglich. Somit ist ein Nachweis der Übereinstimmung des Modells mit der Realität für den allgemeinen Fall nicht möglich und kann nur geschätzt werden.

11.3 Weiterentwicklungsmöglichkeiten

Da es sich bei der implementierten Anwendung, im Rahmen dieser Diplomarbeit, um eine Prototypen-Software handelt, folgt daraus, dass sich viele Möglichkeiten für Weiterentwicklungen und Verbesserungen bieten. Es seien ein paar Vorschläge hier genannt.

Mehrere initial gestörte Knoten

Eine denkbare Erweiterung der Anwendung wäre, die Möglichkeit zu integrieren, mehrere Knoten zu definieren, welche eine Störung zu Beginn der Simulation aufweisen. Dies würde den Fall simulieren, das mehrere Infrastrukturelemente gleichzeitig gestört wurden. Es ist der zeitgleiche Ausfall von mehreren Knoten bei einer Naturkatastrophe denkbar. Daher wäre die Einbindung der Simulationsmöglichkeit eines solchen Szenarios ein sinnvoller Schritt in der Weiterentwicklung der Software.

Speichern der Simulationsergebnisse

Ein weiterer Schritt, um die Anwendung für den praktischen Einsatz zu rüsten, wäre die Implementierung der Möglichkeit des Speicherns der Simulationsergebnisse in der Datenbank, um diese zu einem späteren Zeitpunkt wieder abrufen zu können, und eine erneute Berechnung zu vermeiden. Ebenso wäre der Export der Ergebnisse im Excel-Format ein wünschenswertes Feature.

Entwicklung einer Oberfläche für die Hilfsanwendung *Critical Infrastructure Tool*

Die in Kapitel 9 beschriebene Hilfsanwendung verfügt über keine Benutzeroberfläche und kann daher nur in der Entwicklungsumgebung beziehunsweise in der Befehlseingabe benutzt werden. Es wäre sinnvoll für dieses Tool eine Benutzeroberfläche zu entwickeln. Ebenso ist dieses Tool auf die Benutzung des Zeitrasters von 1, 24, 168 und 720 Stunden begrenzt. Dieses Zeitraster entspricht den zur Verfügung gestellten Daten. Eine Erweiterung, welche das Einlesen von Werten mit beliebigen Zeitpunkten aus einer Excel-Tabelle ermöglichen würde, wäre sinnvoll.

Verbesserung der Darstellung der Ergebnisse

Zur Verbesserung der Darstellung der Ergebnisse könnte man dem Benutzer ermöglichen, die Kurven mehrerer Knoten im Ergebnisgraphen überlagert darzustellen. Dadurch ließe sich das

Verhalten verschiedener Knoten besser vergleichen und man könnte Abhängigkeiten im Netzwerk besser erkennen. Dies würde es erleichtern, Lösungen für allfällige Abhängigkeitsprobleme zu finden.

KAPITEL 12

Zusammenfassung

Im Verlauf dieser Diplomarbeit konnte ein Simulationsmodell entwickelt werden, welches an Hand der zur Verfügung stehenden Daten das Verhalten eines Infrastrukturnetzes unter Störung prognostizieren kann. Basierend auf diesem Modell konnte eine Anwendung erstellt werden, welche die Berechnungen mittels der gegebenen Daten durchführt.

Des Weiteren wurden die errechneten Ergebniskurven kritisch betrachtet und nach intuitiven Gesichtspunkten auf ihre Korrektheit überprüft. Es konnte festgestellt werden, dass die Simulationsergebnisse prinzipiell den erwarteten Ergebnissen entsprachen.

Ebenso wurde die Problematik der Erhebung der Daten für solche Infrastrukturnetze erörtert. Hierbei kam man zu dem Schluss, das im Allgemeinen eine genaue Erhebung solcher beschreibender Daten für ein solches Netzwerk, wie das gegebene, nicht möglich ist. Daraus folgt, die für diese Studie benutzten Daten sind im günstigen Fall als Näherung zu betrachten. Aufgrund dieser Ungenauigkeit der Datenbasis lässt sich folgern, dass eine exakte Vorhersage des Verhaltens des betrachteten Systems im Störungsfall nicht möglich ist. Die statistische Abweichung dieser Näherung wurde nicht bestimmt.

Zusammenfassend darf angemerkt werden, dass die Simulation eines solch komplexen Szenarios wie die Störung der Infrastruktur näherungsweise möglich ist, jedoch die Genauigkeit der Berechnungen nicht zu bestimmen ist.

Literaturverzeichnis

[1] Greenpeace Berlin. 20 Jahre Tschernobyl - Ablauf einer Katastrophe. http://tschernobyl.greenpeace-berlin.de/tagx.html, 26 September, 2013.

[2] Stephen J. Bigelow. Definition single point of failure (SPOF). http://searchdatacenter.techtarget.com/definition/Single-point-of-failure-SPOF, 26 September, 2013.

[3] Google Code. matlabcontrol - A Java API to interact with MATLAB. http://code.google.com/p/matlabcontrol/, 26 September, 2013.

[4] Thomas H Cormen. *Algorithmen: eine Einführung*. Oldenbourg Verlag, 2007.

[5] Michael Joseph Devine, Paul H. Barnes, Cameron Newton, and Ashantha Goonetilleke. A critical analysis of crisis escalation models : understanding stages and severity in infrastructure disturbance. In *9th Annual International Conference of the International Institute for Infrastructure Renewal and Reconstruction : Risk-informed Disaster Management : Planning for Response, Recovery and Resilience*, Queensland University of Technology, Brisbane, QLD, July 2013.

[6] Reinhard Diestel. *Graphentheorie. 4. Auflage*. Springer, 2010.

[7] Donald D. Dudenhoeffer, M.R. Permann, and M. Manic. CIMS: A Framework for Infrastructure Interdependency Modeling and Analysis. In *Simulation Conference, 2006. WSC 06. Proceedings of the Winter*, pages 478–485, 2006.

[8] Dudenverlag. Der Dominoeffekt. http://www.duden.de/rechtschreibung/Dominoeffekt, 26 September, 2013.

[9] Barry Feigenbaum. SWT, Swing or AWT: Which is right for you? http://www.ibm.com/developerworks/grid/library/os-swingswt/, 26 September, 2013.

[10] Free Software Foundation. GNU Lesser General Public Licence (LGPL). http://www.gnu.org/licenses/lgpl.html, 26 September, 2013.

[11] The Apache Software Foundation. Apache License, Version 2.0. http://www.apache.org/licenses/LICENSE-2.0, 26 September, 2013.

[12] The Apache Software Foundation. Apache log4j 1.2. http://logging.apache.org/log4j/1.2/, 26 September, 2013.

[13] The Apache Software Foundation. Apache POI - the Java API for Microsoft Documents. http://poi.apache.org/, 26 September, 2013.

[14] The Apache Software Foundation. Java DB 10.6.2.1 Documentation, Java DB Reference Manual. http://docs.oracle.com/javadb/10.6.2.1/ref/refderby.pdf, 26 September, 2013.

[15] Object Management Group. Unified Modelling Language 2.1.2 Infrastructure Specification. Specification Version 2.1.2, Object Management Group, November 2007.

[16] Stephan Hartmann. The World as a Process: Simulations in the Natural and Social Sciences, August 2005.

[17] Michiel Hazewinkel. *Encyclopaedia of Mathematics - An Updated and Annotated Translation of the Soviet "Mathematical Encyclopaedia*. Springer, Berlin, Heidelberg, 1997. aufl. edition, 1997.

[18] Steven W Kirkpatrick, JW Simons, and TH Antoun. Development and validation of high fidelity vehicle crash simulation models. *SAE transactions*, 109(6):872–881, 2000.

[19] Object Refinery Limited. JFreeChart. http://www.jfree.org/jfreechart/index.html, 26 September, 2013.

[20] XZ Lu, XC Lin, YH Ma, Yi Li, and LP Ye. Numerical simulation for the progressive collapse of concrete building due to earthquake. In *Proc. the 14th World Conference on Earthquake Engineering*, pages 12–17. October, 2008.

[21] Hyeung-Sik J. Min, Walter Beyeler, Theresa Brown, Young Jun Son, and Albert T. Jones. Toward modeling and simulation of critical national infrastructure interdependencies. *IIE Transactions*, 39(1):57–71, January 2007.

[22] Oracle. Java Platform Standard Edition 6. http://docs.oracle.com/javase/6/docs/api/, 26 September, 2013.

[23] Visual Paradigm. Visual Paradigm for UML. http://www.visual-paradigm.com/, 26 September, 2013.

[24] Marc Pitzke. Blackout von 1977: New Yorks dunkelste Nacht). http://www.spiegel.de/panorama/zeitgeschichte/blackout-von-1977-new-yorks-dunkelste-nacht-a-493609.html, 26 September, 2013.

[25] LaTeX project team. An introduction to LaTeX. www.latex-project.org/, 26 September, 2013.

[26] S.M. Rinaldi, J.P. Peerenboom, and T.K. Kelly. Identifying, understanding, and analyzing critical infrastructure interdependencies. *Control Systems, IEEE*, 21(6):11–25, 2001.

[27] V. Stanojevic, S. Vlajic, M. Milic, and M. Ognjanovic. Guidelines for framework development process. In *Software Engineering Conference in Russia (CEE-SECR), 2011 7th Central and Eastern European*, pages 1–9, 2011.

[28] WilliamJ. Tolone, David Wilson, Anita Raja, Wei-ning Xiang, Huili Hao, Stuart Phelps, and E.Wray Johnson. Critical Infrastructure Integration Modeling and Simulation. In *Intelligence and Security Informatics*, volume 3073 of *Lecture Notes in Computer Science*, pages 214–225. Springer Berlin Heidelberg, 2004.

[29] Springer Gabler Verlag. Gabler Wirtschaftslexikon, Stichwort: Infrastruktur. http://wirtschaftslexikon.gabler.de/Archiv/54903/infrastruktur-v9.html, 26 September, 2013.