Angreifer-Modellierung für Intelligente Stromnetze

Bachelorarbeit von

Thomas Weber

an der Fakultät für Informatik
Institut für Programmstrukturen und Datenorganisation (IPD)

Erstgutachter:	Prof. Dr. Ralf H. Reussner
Zweitgutachter:	Prof. Dr.-Ing. Anne Koziolek
Betreuender Mitarbeiter:	M.Sc. Maximilian Walter
Zweiter betreuender Mitarbeiter:	Dipl.-Inform. Misha Strittmatter

03. Juni 2019 – 03. Oktober 2019

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, 02.10.2019

. . . . . .

(Thomas Weber)

Zusammenfassung

Durch den Umstieg auf erneuerbare Energien und die damit einhergehende Dezentralisierung sowie die immer weiter fortschreitende Digitalisierung des Stromnetzes ergeben sich neue Herausforderungen für den Betrieb eines Stromnetzes. Eine dieser Herausforderungen sind die deutlich erweiterten Angriffsmöglichkeiten, die sich durch den verstärkten Einsatz von Intelligenten Stromzählern und Geräten des *Internet der Dinge* und deren maßgeblichem Beitrag zur Stromverteilung ergeben. Um diese Angriffsmöglichkeiten in Analysen abbilden zu können, wird in dieser Bachelorarbeit eine Erweiterung der bestehenden Analyse von Angriffen auf Intelligente Stromnetze aus dem Smart Grid Resilience Framework vorgenommen. Zu diesem Zweck erfolgt eine Transformation des bestehenden Modells in eine Netzwerktopologie, auf welcher dann eine Angreiferanalyse ausgeführt wird. Die Evaluation dieser Angreiferanalyse erfolgt dabei anhand der bereits bestehenden Angreiferanalyse des Smart Grid Resilience Frameworks. Weiterhin wird die Genauigkeit der Transformation sowie die Skalierbarkeit von Transformation und Angreiferanalyse evaluiert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Elektrischer Strom ist ein Pfeiler unserer heutigen digitalen Gesellschaft. Die Sicherstellung einer reibungslosen Stromversorgung hat daher eine hohe Priorität. Eine Möglichkeit für einen Stromausfall stellen eher selten auftretende Störungen der Stromnetze dar. Für Verbraucher in Amerika im Jahr 2016 lag die durchschnittliche Ausfallzeit des Stromnetzes beispielsweise bei ungefähr vier Stunden [1]. Weiterhin sind allerdings auch direkte Angriffe auf ein Stromnetz mittels Schadsoftware denkbar. Waren diese Angriffe früher auf Kraft- und Umspannwerke beschränkt, bietet sich mit der weiter voranschreitenden Digitalisierung eine immer größer werdende Angriffsfläche.

Diese Angriffsfläche setzt sich zum einen aus dem Missbrauchspotenzial der häufiger eingesetzten Intelligenten Stromzähler und zum anderen aus vernetzten Haushaltsgroßgeräten zusammen, was basierend auf der Veröffentlichung von Soltan et al. [16] erläutert wird. Intelligente Stromzähler erhöhen zwar die Anzahl der verfügbaren Daten zur Steuerung des Netzes und verbessern damit die Stromversorgung, können aber im Fall einer Kontrollübernahme mittels Desinformationen die Stabilität des Netzes empfindlich stören. Geräte des *Internet der Dinge* können durch gezieltes Erhöhen oder Verringern des Stromverbrauchs erhebliche Störungen im Netz verursachen, wenn sie in geographischer Nähe und ausreichender Zahl unter die Kontrolle eines Angreifers gebracht worden sind.

Im Rahmen dieser Arbeit wird die zweite Angriffsart nicht umgesetzt, da sie im zugrunde liegenden Modell zur Beschreibung eines Intelligenten Stromnetzes (The Smart Grid Model (SGM)) aus dem Smart Grid Resilience Framework, welches näher in Kapitel 2.2.1 erläutert wird, nicht beschrieben werden kann. Weil es sich aber um eine interessante Erweiterungsmöglichkeit der Angreiferanalyse handelt, ist der Angriff ausführlicher in Kapitel 3.4 beschrieben. Eine entsprechende Erweiterung des Metamodells war auf Grund der Verwendung in anderen Projekten, welche mit den Ergebnissen der Angreiferanalyse arbeiten, nicht möglich. Des Weiteren ist das Ziel des Smart Grid Resilience Frameworks die Angreiferanalyse im Hinblick auf das Stromnetz selbst. Da die in dieser Arbeit vorgestellte Modellierung einen offenen Ansatz nutzt, ließe sich allerdings auch ein solches Szenario mit dem neuen Ansatz abbilden.

Die bisherige Modellierung des SGM ist ein Kompromiss zwischen den drei damit verfolgten Projekten. Diese sind die Analyse eines Angriffs auf das Stromnetz, der Berechnung und Steuerung des Stromflusses während das Netz aktiv ist und die Verwaltung und der Schutz kritischer Infrastruktur. Die Modellierung enthält daher nur die von allen drei Projekten benötigten Informationen, auch wenn für die einzelnen Projekte weitere Informationen hilfreich wären. Aus diesem Grund wird in dieser Arbeit ein erweitertes Modell erstellt, welches aus dem SGM durch eine Transformation generiert wird und sich beliebig mit weiteren Informationen anreichern lässt. Auf dem so erstellten Modell kann dann eine

Angreiferanalyse ausgeführt werden. Um die durch die Analyse erlangten Informationen auch in den anderen Projekten nutzen zu können, erfolgt die Speicherung der Ergebnisse der Angreiferanalyse in einem bereits vorhandenen Modell, dem *SmartgridOutput*-Modell, welches näher in Kapitel 2.2.1.3 erläutert wird.

Das mithilfe der Transformation aus dem SGM erstellte Modell liegt im Palladio Komponentenmodell (PCM) vor. Eine Beschreibung dazu ist in Kapitel 2.2.2 zu finden. Das PCM-Modell kann allerdings im Gegensatz zum SGM-Modell nun beliebig um weitere Architekturinformationen erweitert werden, da im Metamodell des PCM bereits Komponenten vorgesehen sind. Eine Anpassung des Metamodells, welche für das SGM notwendig wäre, um dieselben zusätzlichen Information abbilden zu können, entfällt so. Ein Beispiel für eine solche Erweiterung ist die genauere Modellierung von Kontrollzentren im Netzwerk. Im SGM ist ein Kontrollzentrum ein einzelner Knoten, im generierten PCM-Modell lässt er sich automatisch oder manuell um Informationen, wie die darin verwendeten Computer oder die darauf laufende Software, erweitern.

1.1 Beitrag

Um den Einfluss der weiteren Komponenten der einzelnen Elemente der Topologie nutzen zu können wird zunächst eine Transformation entworfen, welche aus einer Instanz eines SGM-Modells eine Instanz des PCM-Modells erzeugt. Diese Instanz kann dann manuell oder automatisch in einem weiteren Schritt an die konkreten Gegebenheiten im untersuchten Netz angepasst werden.

Als nächster Schritt erfolgt die Entwicklung einer angepassten Angreiferanalyse , welche auf der erstellten Topologie arbeitet. Diese Analyse benutzt die bestehenden Ein- und Ausgabeformate und zusätzlich das generierte PCM-Modell. Die Verwendung der bestehenden Modelle ermöglicht die Nutzung der gewonnen Daten in den anderen Projekten.

Zuletzt wird noch eine Transformation von einem PCM-Modell zurück in ein SGM-Modell implementiert, welche einerseits die Überprüfung der ersten Transformation ermöglicht und andererseits Änderungen der Topologie im PCM-Modell für alle Projekte verwendbar macht.

1.2 Aufbau

Die Bachelorarbeit ist in 7 Kapitel aufgeteilt. Kapitel 1 beschäftigt sich mit einem Überblick über die Arbeit sowie deren Motivation.

Kapitel 2 enthält Erläuterungen der in dieser Arbeit verwendeten Modelle und Konzepte. Zunächst werden die Entwicklungen im Stromnetz beschrieben, welche eine Erweiterung der Topologie gegenüber dem SGM nötig machen. Danach erfolgt eine kurze Beschreibung der verwendeten Modelle aus dem Smart Grid Resilience Framework und dem Palladio Komponentenmodell. Abschließend werden bestehende Vorarbeiten zur Angreiferanalyse erläutert.

Verwandte Arbeiten werden im Kapitel 3 erörtert. Dabei werden zunächst Quellen für eine Angreiferanalyse in einem PCM-Modell untersucht. Weiterhin wird herausgearbeitet, inwieweit sie sich zur Beschreibung von Angreiferanalysen in Intelligenten Stromnetzen eignen. Danach werden verwandte Arbeiten aus dem Bereich der Modellierung von Intelligenten Stromnetzen hinsichtlich der darin beschriebenen Erweiterungen einer einfachen Topologie und ihrer Verwendbarkeit für diese Arbeit untersucht. Abschließend erfolgt noch eine Beschreibung zweier Angriffe auf Stromnetze.

Kapitel 4 beschäftigt sich mit den Überlegungen zur Umsetzung der Transformationen und der Wahl der einzelnen Komponenten für das transformierte PCM-Modell. Außerdem werden die genutzten Angreifermodelle und eine Fallstudie beschrieben, die den Nutzen der neuen Modellierung darlegt.

In Kapitel 5 wird die Implementierung der Transformationen und der Angreifermodellierung dargelegt. Des Weiteren erfolgt eine kurze Beschreibung der Benutzung der entwickelten Werkzeuge und eine Erläuterung des Entwicklungsprozesses.

Kapitel 6 umfasst die Evaluation der erarbeiteten Werkzeuge. Die beiden Transformationen werden gemeinsam anhand der generierten Modelle evaluiert, wobei das nach der zweiten Transformation generierte Modell semantisch identisch mit dem Ursprungsmodell sein muss. Weiterhin wird die erstellte Angreiferanalyse mit der bereits bestehenden verglichen. Hierbei sollten beide Analysen identische Ergebnisse liefern. Außerdem erfolgt eine Evaluation der Werkzeuge hinsichtlich ihrer Skalierbarkeit.

Kapitel 7 schließt die Arbeit mit einer Zusammenfassung und einem Ausblick auf mögliche Erweiterungen ab.

2 Grundlagen

Die Erläuterung der Grundlagen beginnt mit den Eigenschaften des aktuellen Stromnetzes und dessen Weiterentwicklung. Diese werden anhand der Veröffentlichungen von Lakervi [11] und Brown [2] vorgestellt. Die langfristige Entwicklung soll dabei zum Intelligenten Stromnetz führen, das anhand der Überblicksarbeit von Fang et al. [7] eingeführt wird.

Nach der Darstellung der Grundlagen von Stromnetzen erfolgt eine Erläuterung der verwendeten Modelle zur Abstraktion des Stromnetzes. Zunächst wird das Smart Grid Resilience Framework beschrieben, das unter anderem ein Modell zur Darstellung Intelligenter Stromnetze bietet. Neben diesem Modell, das im Kapitel 2.2.1.1 erläutert wird, werden die beiden anderen aus diesem Projekt verwendeten Modelle, eines für die Eingabedaten und eines für die Ausgabedaten der Analyse, vorgestellt. Danach erfolgt eine Erläuterung der für die Arbeit verwendeten Teile des Palladio Komponentenmodells anhand der beiden Veröffentlichungen von Reussner et al. [14] und [15]. Besonders von Bedeutung ist hierbei die Unterteilung des Palladio Komponentenmodells in fünf Modelle, die in den Unterkapiteln vorgestellt werden. Abschließend werden bestehende Vorarbeiten erläutert, die unter anderem die existierende Angreiferanalyse beinhalten.

2.1 Stromnetz

Der Zweck eines Stromnetzes besteht darin, Verbraucher und Erzeuger von Strom zu verbinden. Das aktuelle Stromnetz besteht hauptsächlich aus wenigen großen Energieproduzenten, beispielsweise Kohle-, Gas-, oder Wasserkraftwerken. Der von diesen produzierte Strom gelangt über Einspeisenetze in das Hochspannungsnetz. Dieses dient der Verbindung über lange Strecken, da der Verlust beim Transport mit hoher Spannung im Vergleich zu niedriger Spannung geringer ist. Über Umspannwerke erfolgt eine Transformation auf Mittel- oder Niederspannung, die für die Netze der Verbraucher geeignet sind. Eine Kommunikation zwischen Produzent und Konsument findet im Allgemeinen nicht statt, weswegen sich die Produktion nicht am tatsächlichen Verbrauch orientiert, sondern lediglich mit einer Vorhersage des Verbrauchs arbeitet. Direkt angepasst wird die Produktion also nicht an den Verbrauch, sondern an den Zustand des Netzes. [11][2]

2.1.1 Veränderungen im Stromnetz

Das Stromnetz verändert sich kontinuierlich und passt sich an neue Randbedingungen an. Dazu gehört einerseits die weiter fortschreitende Digitalisierung des Netzes und andererseits die Dezentralisierung der Produktion. Außerdem führt der verstärkte Einsatz erneuerbarer Energien zu Produktionsschwankungen, die entweder durch erhöhten Ver-

brauch oder eine Drosselung der Produktion in herkömmlichen Kraftwerken ausgeglichen werden müssen.

Einen großen Beitrag zur Verbesserung des Netzes erhoffen sich die Energieerzeuger dabei von den Intelligenten Stromzählern, die durch ein Gesetz verpflichtend bis 2032 in allen Haushalten mit einem Verbrauch über 6000 kWh eingebaut sein müssen [5]. Dadurch steigt die Menge der verfügbaren Daten deutlich an, da die Intelligenten Stromzähler den Verbrauch in Zeiteinheiten von 15 Minuten erfassen. Im Gegensatz dazu ist die bisherige Erfassung üblicherweise lediglich einmal jährlich. Mit den so erhobenen Daten lässt sich das Stromnetz dann genauer steuern. [2]

Eine weitere Änderung liegt in der Dezentralisierung der Stromproduktion, die nicht mehr nur aus einigen wenigen großen Produzenten besteht, sondern beispielsweise um Solaranlagen auf Hausdächern oder einzelne Windrädern erweitert wird. Ein ausgedehntes Hochspannungsnetz ist allerdings weiterhin erforderlich, da auch bei erneuerbaren Energien Produzent und Verbraucher häufig weit voneinander entfernt sind. Ein Beispiel dafür sind die Windparks in der Nord- und Ostsee, während die großen Stromabnehmer in Bayern oder Baden-Württemberg liegen. [2]

Neben diesen Änderungen, die vor allem das Netz selbst betreffen, erfolgt auch eine Veränderung der eingesetzten Endgeräte. Diese, beispielsweise Haushaltsgroßgeräte wie Trockner oder Waschmaschinen, werden immer häufiger mit intelligenten Funktionen wie einer Zeitsteuerung ausgestattet. Damit können die Geräte unabhängig von der Anwesenheit des Eigentümers aktiviert werden. Dies kann, wie in der Veröffentlichung von Soltan et al. [16] dargelegt, durchaus einen großen Einfluss auf den Verbrauch haben. Ein zu starker Anstieg des Verbrauchs kann dabei die Stabilität des Stromnetzes und damit die Versorgungssicherheit gefährden.

2.1.2 Intelligentes Stromnetz

Ein sogenanntes Intelligentes Stromnetz basiert, wie in der Überblicksarbeit von Fang et al. [7] beschrieben, auf bidirektionalem Fluss von Daten und Energie in einem stark vernetzten Stromnetz. Die Steuerung von Produktion und Verbrauch ist dabei eng miteinander verknüpft, was eine große Menge an zu erhebenden Daten erfordert. Diese werden über Intelligente Stromzähler erhoben und ermöglichen es, die Produktion dynamisch an den aktuellen Verbrauch anzupassen. Das stellt einen Fortschritt gegenüber der rein auf Vorhersagen basierenden Produktion bisher dar. Neben dieser Art der Anpassung ist allerdings auch die umgekehrte Art sinnvoll, beispielsweise kann der Verbrauch bei einer erhöhten Stromproduktion ebenfalls erhöht werden. Ein Beispiel dafür wäre die intelligente Steuerung der Ladeelektronik von Elektroautos, die eine Ladung nur vornimmt, wenn entsprechend Strom eingespeist wird.

Durch die starke Vernetzung des Stromnetzes lassen sich so einzelne Lastspitzen vermeiden und insgesamt wird eine gleichmäßigere Auslastung des Netzes erreicht. Außerdem ist es möglich, den Strompreis dynamisch über Angebot und Nachfrage zu berechnen. Ein dynamischer Strompreis ermöglicht beispielsweise das bessere Ausnutzen einer Überproduktion durch Windkraftanlagen. Diese führt zu einer Preissenkung, wodurch sich der

Stromkauf lohnt. Der Strom wird dann entweder direkt verbraucht oder in Akkus günstig gespeichert, um ihn dann bei Knappheit teurer zu verkaufen. Für die Speicherung bieten sich dabei die dezentralen Akkus in Elektroautos oder auch in Häusern mit eigener Stromproduktion an. Dadurch wird die Infrastruktur entlastet und es entfällt die Notwendigkeit großer Energiespeicher.

Interessant für diese Arbeit ist dabei vor allem die Topologie der kleineren Mittel- und Niederspannungsnetze, insbesondere welche Teilnehmer im Netz miteinander kommunizieren und welche Auswirkungen ein Angriff auf ein solches Stromnetz haben kann. [7]

2.2 Genutzte Modelle

Diese Arbeit behandelt die Analyse von Angriffen auf Intelligente Stromnetze. Um diese Angriffe zu analysieren, werden die benötigten Komponenten im SGM beziehungsweise im PCM entworfen. Daher erfolgt hier eine kurze Einführung in die beiden verwendeten Projekte und die in ihnen enthaltenen Modelle. Weitere Informationen zu den Transformationen und der Analyse sind in Kapitel 4 erläutert.

2.2.1 Smart Grid Resilience Framework

Das Smart Grid Resilience Framework, das näher in [13] beschrieben ist, beschäftigt sich mit der Beschreibung von Intelligenten Stromnetzen und verschiedenen Analysen auf diesen. Ein Intelligentes Stromnetz wird dabei im SGM-Modell beschrieben, das im nächsten Kapitel erläutert wird. Darin lässt sich die Topologie eines Stromnetzes, auf bestimmte Komponenten abstrahiert, abbilden.

Des Weiteren enthält das Projekt ein Modell für die Zustände der Komponenten eines SGM-Modells, also beispielsweise, ob ein enthaltener Intelligenter Stromzähler defekt oder kompromittiert ist. Dieses Modell hat den Namen *SmartgridInput* und ist näher im Kapitel 2.2.1.2 erläutert. Neben dem Eingabemodell gibt es noch ein Modell für die Ausgabe der Angreiferanalyse, das *SmartgridOutput*-Modell, welches näher im Kapitel 2.2.1.3 erläutert wird. Dieses enthält Informationen über den Zustand der einzelnen SGM-Modellelemente nach der Angreiferanalyse , also zum Beispiel, ob ein Intelligenter Stromzähler kompromittiert ist oder nicht.

Außerdem enthält das Projekt eine bestehende Angreifermodellierung, die näher in Kapitel 2.2.1.4 erläutert wird. Die Modellierung enthält einen lokalen und viralen Hacker, die sich auf verschiedene Arten im Netzwerk ausbreiten können.

2.2.1.1 SmartgridTopo

Das *SmartgridTopo*-Modell dient der Beschreibung der Topologie eines Intelligenten Stromnetzes. Dazu sind verschiedene Komponenten modelliert, die zusammen ein Intelligentes Stromnetz bilden. Diese Komponenten sind Intelligente Stromzähler (Smart Meter), Kontrollzentren (Control Center), Netzwerkknoten (Network Node) und Stromnetzknoten

(Power Grid Node). Die Intelligenten Stromzähler enthalten die Angabe des von ihnen angeforderten Stroms als Ganzzahl. Außerdem besitzt jede der Komponenten einen eindeutigen Identifikator.

Neben den genannten Komponenten gibt es noch drei mögliche Arten, diese untereinander zu verbinden. Diese sind eine physische (PhysicalConnection), eine logische (Logical Communication) oder eine Stromverbindung (Power Connection). Außerdem enthält das Modell noch Zwischensysteme für die logische Kommunikation. Diese sind generische Netzwerkcontroller (Generic Controller) und Komponenten für die Kommunikation zwischen Intelligenten Stromnetzen (Inter Com).

Bis auf den Stromnetzknoten enthalten alle Komponenten noch eine Möglichkeit zur Speicherung ihres Ortes mittels Längen- und Breitengrad. Ein Beispiel für eine graphische Darstellung eines im SGM beschriebenen Intelligenten Stromnetzes ist in Abbildung 2.1 zu finden. Darauf zu sehen ist die Topologie eines Intelligenten Stromnetzes, das aus einem Kontrollzentrum (grüner Kreis), drei Netzwerkknoten (blaue Quadrate), vier Intelligenten Stromzählern(rote Kreise) und sieben Stromnetzknoten(gelbe Rauten) besteht. Eine Legende für die einzelnen Komponenten ist in der Abbildung 2.2 zu finden.

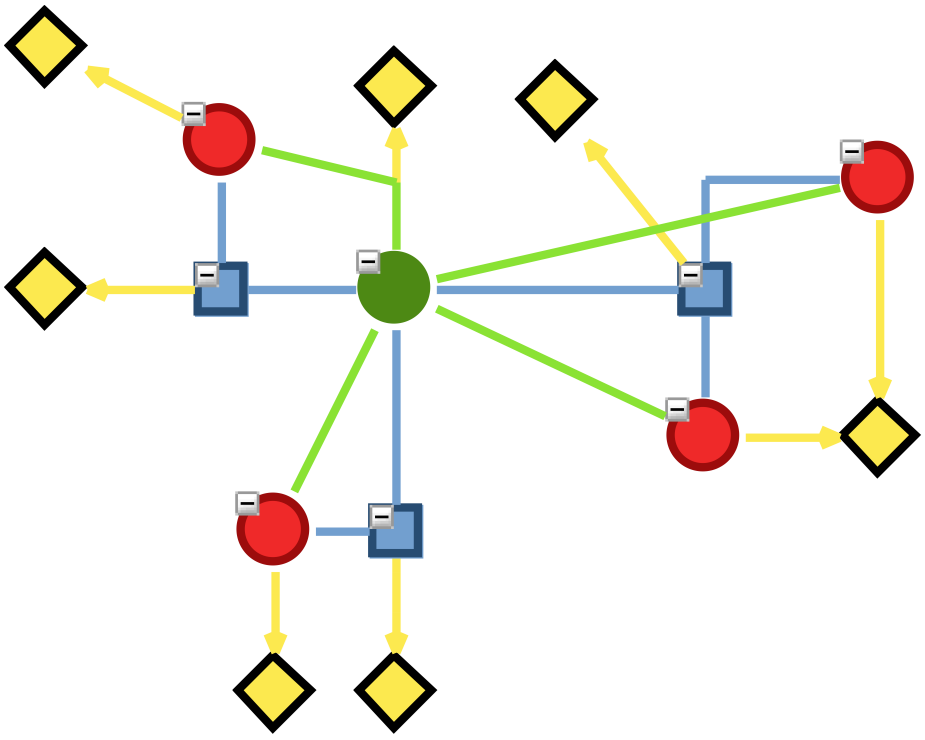

Abbildung 2.1: SGM-Modell aus dem Smart Grid Resilience Framework

Abbildung 2.2: Legende für verwendete SGM-Modelle

2.2.1.2 SmartgridInput

Das *SmartgridInput*-Modell dient der Beschreibung der Zustände der Komponenten eines SGM-Modells. Dafür enthält das Modell für jeden Stromnetzknoten einen *PowerState*, der die Information enthält, ob der Stromnetzknoten mit Strom versorgt ist. Weiterhin enthält das Modell für jede andere Netzwerkkomponente einen *EntityState*, in dem gespeichert ist, ob der Netzwerkknoten kompromittiert oder zerstört ist. Außerdem enthält ein *SmartgridInput*-Modell noch das zugrunde liegende *SmartgridTopo*-Modell.

Um die Daten aus dem Modell nutzen zu können, ist es dementsprechend notwendig, das zugrunde liegende *SmartgridTopo*-Modell zu laden. Ein Beispiel für die Darstellung des Modells ist die mit entsprechenden Beschriftungen versehene Abbildung 2.3. Der Übersichtlichkeit halber sind hierbei die *PowerStates* weggelassen und die Eigenschaft *isDestroyed* mit *des* und die Eigenschaft *isHacked* mit *hac* abgekürzt. Da der bestehende Editor leider nicht funktioniert hat, sind die Beschriftungen aller SGM-Modelle, welche die Daten aus den anderen Modellen enthalten, manuell hinzugefügt worden.

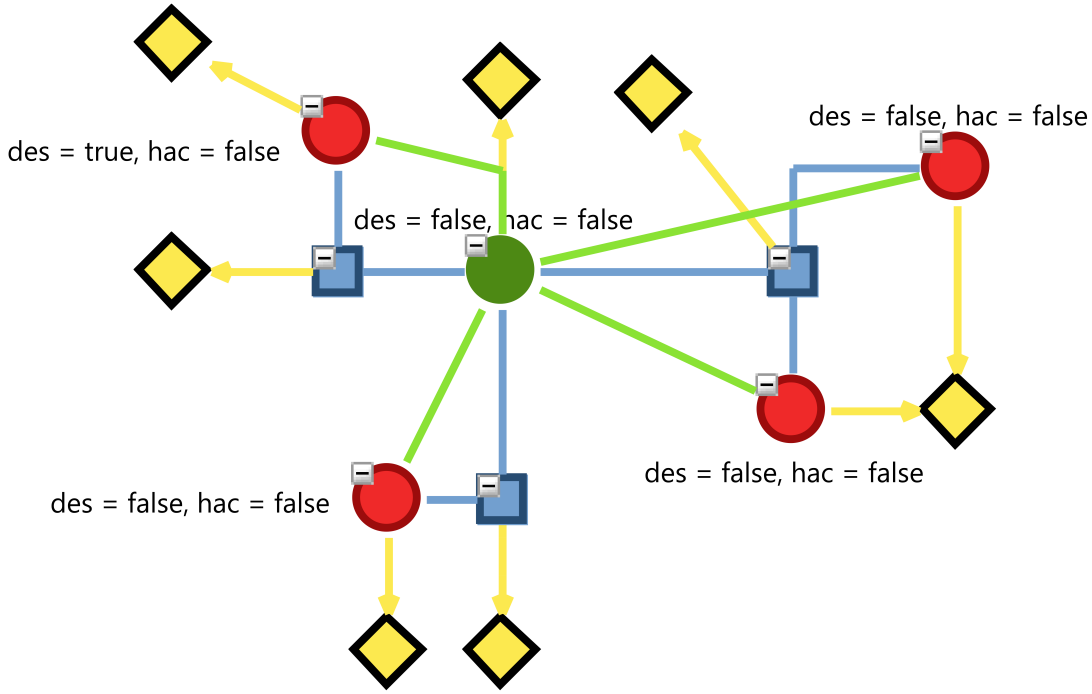

Abbildung 2.3: SGM-Modell mit Input-Daten annotiert

2.2.1.3 SmartgridOutput

Neben einem Modell für die Eingabedaten der Angreiferanalyse wird noch ein Modell für die Ausgabedaten beziehungsweise Ergebnisse benötigt. Dieses ist das *SmartgridOutput*-Modell, in dem jedem Netzwerkknoten ein *EntityState* zugeordnet werden kann. Dabei kann ein Netzwerkknoten entweder ein- oder ausgeschaltet sein und im ersten Fall kann noch gespeichert werden, ob der Knoten kompromittiert ist. Neben diesen für die Angreiferanalyse wichtigen Daten kann das zugrunde liegende *SmartgridTopo*-Modell noch in *Cluster* aufgeteilt werden. Dies übernimmt die ebenfalls im Projekt entwickelte *Impact-Analyse* [6]. Um diese Funktionalität weiter nutzen zu können und die Verwendung in den anderen Projekten des Smart Grid Resilience Frameworks zu ermöglichen, wurde für diese Arbeit kein erweitertes Modell für die Ausgabe erstellt. Stattdessen wurde das bestehende *SmartgridOutput*-Modell zur Speicherung der erzeugten Daten genutzt.

Ein Beispiel für die Darstellung des Modells ist die mit entsprechenden Beschriftungen versehene Abbildung 2.4. Der Übersichtlichkeit halber ist die Eigenschaft *isHacked* mit *hac* abgekürzt. Weiterhin enthalten die Beschriftungen den Zustand der Netzwerkknoten, im aktiven Zustand entweder *Online* oder *NoUplink*, im ausgeschalteten Fall entweder *NoPower* oder *Defect*. Ein defekter oder nicht mit Strom versorgter Knoten kann dabei, wie in der Abbildung am oberen linken Intelligenten Stromzähler zu sehen, nicht kompromittiert sein.

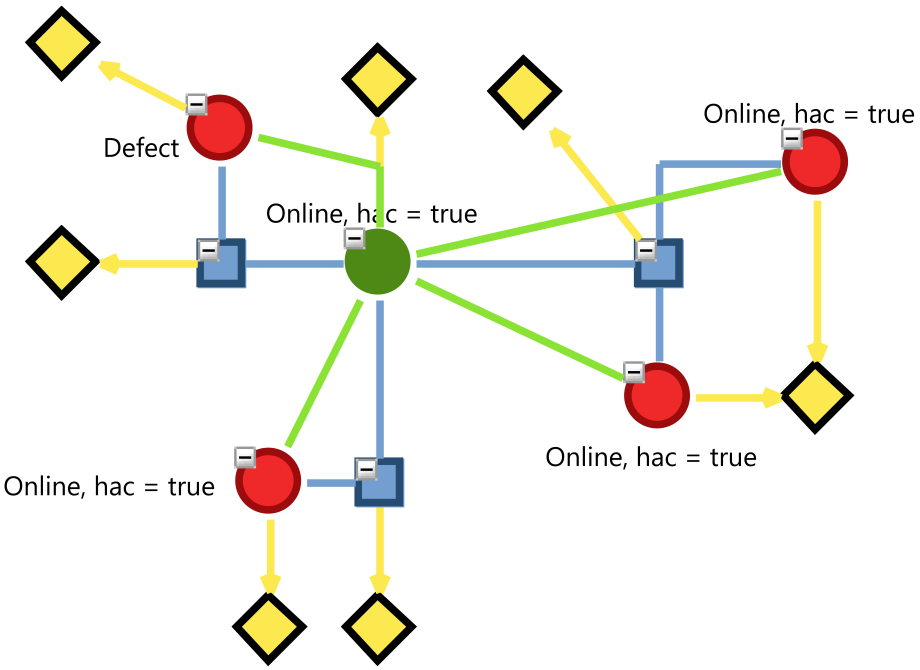

Abbildung 2.4: SGM-Modell mit Output-Daten annotiert

2.2.1.4 Bestehende Analyse

Diese Arbeit baut unter anderem auf der Bachelorarbeit von Eric Klappert [10] und bestehender Vorarbeiten im Smart Grid Resilience Framework auf. Die Vorarbeiten sind in [6] dokumentiert. Sie enthalten unter anderem eine Angreiferimplementierung für das SGM. Der Fokus der Bachelorarbeit von Eric Klappert [10] ist die Unterteilung eines Intelligenten Stromnetzes in kleinere Einheiten, sogenannte *Microgrids*, die sich besser verwalten und schützen lassen. Um dies zu überprüfen, wird das in der Dokumentation beschriebene Angreifermodell unter anderem genutzt.

Das bestehende Angreifermodell enthält verschiedene Arten von Angreifern, die mittels unterschiedlicher Strategien ein Intelligentes Stromnetz kompromittieren können. Ein Angreifer geht dabei entweder lokal oder viral vor. Ein lokaler Angreifer kann pro Zeitschritt nur einen benachbarten Knoten kompromittieren, unabhängig davon, wie viele Knoten er bereits kompromittiert hat. Der virale Angreifer hingegen kann von jedem bereits kompromittierten Knoten aus einen benachbarten Knoten kompromittieren. Neben diesem Unterschied in der Vorgehensweise stehen noch drei Strategien für die Ermittlung des nächsten zu kompromittierenden Knoten zur Verfügung. Diese sind *Fully-Meshed*, *BFS* und *DFS*. Die *Fully-Meshed* Strategie wählt einen zufälligen Knoten als nächstes Ziel aus, die *BFS* Strategie wählt einen Knoten entsprechend einem Breitensuche-Algorithmus aus und die *DFS* Strategie entsprechend einem Tiefensuche-Algorithmus.

2.2.2 Palladio Komponentenmodell

Zur Modellierung der Netzwerktopologie wird in dieser Arbeit das Palladio Komponentenmodell genutzt, das näher in diesen beiden Veröffentlichungen von Reussner et al. [14] und [15] beschrieben ist. Palladio ermöglicht die Beschreibung und Analyse von Software-Architekturen, ohne sie zuerst implementieren zu müssen. Die Beschreibung der Systeme erfolgt dabei im von Palladio definierten Metamodell (PCM), das in dieser Arbeit zur leicht erweiterbaren Beschreibung der Intelligenten Stromnetze genutzt wird. Dies ermöglicht die Betrachtung der Stromnetze aus der Perspektive der Netzwerktopologie heraus und erlaubt es später, die Auswirkungen von Änderungen an der Netzwerktopologie auf die Anfälligkeit gegenüber Angriffen abzubilden. Neben der Möglichkeit, Netzwerktopologien zu beschreiben enthält das Palladio-Projekt auch Analysen für *Quality-of-Service*-Eigenschaften (QoS-Eigenschaften), beispielsweise Ausfallsicherheit oder Leistung. Diese Analysen sind für diese Arbeit allerdings nicht relevant, da hier lediglich der Aspekt der Sicherheit betrachtet wird.

Ein PCM-Modell besteht aus fünf Teilmodellen, von denen vier in dieser Arbeit verwendet werden. Diese sind das *Repository Model*, das im nächsten Kapitel näher beschrieben wird und die Sicht eines Komponenten-Entwicklers widerspiegelt. Auf dieses Modell setzt das *System Model* auf, das den Entwurf der Software enthält. Danach erfolgt mit dem *Resource Environment* die Beschreibung der tatsächlich zur Verfügung stehenden Hardware-Ressourcen und schließlich im *Allocation Model* die Zuordnung der entworfenen Systeme auf die Hardware-Ressourcen. Alle in diesen Modellen definierten Komponenten haben jeweils einen eindeutigen Identifikator. Das fünfte Modell ist das *Usage Model*, in dem Nutzungsszenarien von Domänenexperten erstellt werden.

2.2.2.1 Repository Model

Das *Repository Model* eines PCM-Modells, das von einem Komponentenentwickler spezifiziert wird, enthält Komponenten mit angebotenen oder benötigten Funktionen. Diese Komponenten können aus anderen Komponenten zusammengesetzt werden. Für diese Arbeit relevant sind die enthaltenen Komponenten, die einen unterschiedlichen Funktionsumfang besitzen. Ein *BasicComponent* ist eine einfache nicht zusammengesetzte Komponente, *CompositeComponent* und *SubSystem* sind aus beliebig vielen Komponenten zusammengesetzt. Dabei können alle drei Komponententypen Teil eines *CompositeComponent* oder *SubSystems* sein.

2.2.2.2 System Model

Der Softwarearchitekt entwirft aus den im *Repository Model* enthaltenen Komponenten ein System. Dazu werden vorhandene Komponenten einem *AssemblyContext* zugeordnet, der im System eingesetzt werden kann. Diese *AssemblyContexts* können untereinander oder mit der Systemgrenze verbunden werden. Die so entworfenen Beziehungen werden in einem *System Model* gespeichert.

2.2.2.3 Resource Environment

Das *ResourceEnvironment* beschreibt eine Hardware-Topologie, welche die zur Verfügung stehende Hardware enthält. Dieses Modell enthält *ResourceContainer*, die aus ihnen zur Verfügung stehenden Ressourcen bestehen, beispielsweise Prozessoren oder Festplatten. Um die *ResourceContainer* miteinander zu verbinden, existieren in diesem Modell Verbindungen vom Typ *LinkingResource*. Damit lässt sich eine vollständige Hardware-Topologie entwerfen und speichern.

2.2.2.4 Allocation Model

Da die Modelle zur Software-Konfiguration (*System Model*) und zur Hardware-Topologie (*ResourceEnvironment*) unabhängig voneinander sind, muss noch eine Zuordnung beziehungsweise Allokation zwischen beiden erfolgen. Diese Allokation erfolgt über das *Allocation Model*, in dem einem *AssemblyContext* des *System Model* ein *ResourceContainer* aus dem *ResourceEnvironment* zugeordnet wird. Diese Zuordnung entspricht der Ausführung des *AssemblyContexts* auf dem *ResourceContainer*.

3 Verwandte Arbeiten

Diese Arbeit orientiert sich an verschiedenen bestehenden Arbeiten, die hier näher erläutert werden. Zuerst erfolgt ein Überblick über bereits existierende Angreiferanalysen im Kontext des PCM. Weiterhin wird begründet, warum die Verwendung der darin vorgestellten Ansätze nicht möglich ist. Danach erfolgt eine Zusammenstellung von Veröffentlichungen, die sich mit der Topologie von Intelligenten Stromnetzen beschäftigen. In Kapitel 3.2 werden außerdem Komponenten eines Intelligenten Stromnetzes herausgearbeitet, die in das transformierte PCM-Modell integriert werden. Die Vorgehensweise dazu ist im Kapitel 4 erläutert.

Danach werden noch zwei Veröffentlichungen erörtert, die sich mit der Analyse von Angriffen auf das Stromnetz befassen. Zunächst wird der Angriff auf das Stromnetz der Ukraine erläutert. Danach erfolgt die Betrachtung einer Veröffentlichung zur Angreifbarkeit eines Stromnetzes über die angeschlossenen Verbraucher. Beide Ansätze werden dargelegt und die Teile erläutert, die in dieser Arbeit verwendet werden können.

3.1 Angreifermodellierung im Palladio Komponentenmodell

Die Betrachtung der Sicherheit eines entworfenen Architekturmodells wurde bereits in anderen Veröffentlichungen erläutert. Die Arbeit von Hilbrich et al. [9] beschäftigt sich dabei mit zwei Ansätzen zur Modellierung von Sicherheit in einem PCM-Modell.

Der erste Ansatz aus dieser Veröffentlichung ist einer anderen Veröffentlichung von Busch et al. [3] entnommen. Diese stellt eine Erweiterung des Palladio Komponentenmodells vor, um die Zeit bis zum nächsten Sicherheitsvorfall zu bestimmen. Dabei werden die zu schützenden Daten und die Zugriffsmöglichkeiten auf diese Daten modelliert. Zusätzlich werden die Fähigkeiten des Angreifers und das zur Verfügung stehende Wissen über das System sowie die Qualität des Systems festgelegt. Aus diesen Daten schätzt das Modell die Zeit bis zum nächsten Sicherheitsvorfall, also einem Zugriff des Angreifers auf die zu schützenden Daten. Der Ansatz kann dabei verschiedene Architekturen vergleichen, ist allerdings schwierig vollständig umsetzbar, da beispielsweise die Fähigkeiten eines Angreifers schlecht einschätzbar sind.

Der beschriebene Ansatz arbeitet mit den unterschiedlichen Bestandteilen einer Softwarearchitektur im *System Model* und leitet aus den Eigenschaften eines Angreifers die Zeit bis zum nächsten Sicherheitsvorfall ab. Da in dieser Bachelorarbeit der Fokus auf der Untersuchung der Auswirkungen der Netzwerktopologie auf Angriffe liegt, welche im *Resource Environment* gespeichert sind, ist der gerade beschriebene Ansatz nicht verwendbar. Für die Untersuchung einzelner Komponenten der Topologie und ihrer Bestandteile, beispielsweise ein Kontrollzentrum mit Servern und anderen Komponenten, deren Verhalten im *System Model* modelliert wurde, ist der Ansatz aber interessant. Da dies jedoch

nicht die Zielsetzung dieser Arbeit war, wurde der Ansatz nicht verwendet.

Der zweite Ansatz, der in der Veröffentlichung [9] erarbeitet wurde, benötigt keine Änderungen am Palladio Komponentenmodell. Er arbeitet stattdessen mit Sicherheitsebenen, mit welchen die Daten und Komponenten des Architekturmodells annotiert werden. Über diese Ebenen kann ermittelt werden, auf welchen Komponenten die mit den entsprechenden Daten arbeitenden Prozesse ausgeführt werden müssen. Die Sicherheit des Systems wird dabei durch das Erfüllen der Sicherheitsebenen ermittelt. Das Ziel dieses Ansatzes liegt darin, für eine gegebene Architektur eine Abwägung zwischen verschiedenen Faktoren wie beispielsweise Schnelligkeit, Kosten und Sicherheit zu finden, welche geforderte Sicherheitsstandards einhält.

Dieser in der Veröffentlichung erarbeitete Ansatz arbeitet ebenfalls mit der Sicherheit einer verteilten Ausführung von Software und Daten. Da in dieser Arbeit allerdings keine Software modelliert wurde, lassen sich mit diesem Ansatz für das erstellte Modell keine Aussagen über seine Sicherheit treffen. Für eine Analyse von einzelnen Komponenten eignet er sich jedoch genauso wie der erste auch.

Die zweite Veröffentlichung von Taspolatoglu und Heinrich [17] beschreibt einen Angreifer als einen Fall schädlicher Nutzung des untersuchten Systems. Auch hierbei werden die Modellelemente mit Sicherheitseigenschaften annotiert und der Angreifer erhält eine Anzahl an Angriffstypen, welche er ausführen kann. Beispielsweise kann eine Komponente *Firewall* mit der sicherheitsrelevanten Beschreibung *Statefull Firewall* annotiert werden. Damit hat die Firewall die Einschränkung, dass sie nur eine bestimmte Anzahl simultaner Eingaben prüfen kann. Damit ist sie nicht in der Lage, einen *Distributed Denial of Service* (DDOS) Angriff abzuwehren, einen eingeschränkten *Denial of service* (DOS) Angriff jedoch schon. Mit solchen aus den annotierten Eigenschaften der Komponenten ermittelten Angriffsvektoren lässt sich eine Sicherheitseinschätzung für die einzelnen Komponenten oder auch das Gesamtsystem treffen.

Wie die beiden anderen vorgestellten Ansätze beschäftigt sich auch der aus der zweiten Veröffentlichung mit der Analyse einer Softwarearchitektur und nicht einer Netzwerktopologie, sodass die verwendete Vorgehensweise für diese Bachelorarbeit nicht direkt anwendbar ist. Für eine genauere Analyse der Angreifbarkeit der einzelnen Bestandteile der Topologie, welche eine mögliche Erweiterung des in dieser Arbeit vorgestellten Ansatzes ist, ist die Vorgehensweise allerdings gut geeignet.

3.2 Aufbau eines Intelligenten Stromnetzes

Ein Intelligentes Stromnetz besteht aus diversen Teilen, die nicht alle im stark abstrahierten SGM enthalten sind, aber dennoch für die Analyse eines Angriffs interessant sind. Daher wird ein aus einem SGM-Modell transformiertes PCM-Modell in einer Fallstudie in Kapitel 4.3.5 um einige der in den nachfolgend vorgestellten Veröffentlichungen erläuterten Konzepte erweitert.

Die Veröffentlichung von Hernandez et al. [8] befasst sich dabei mit dem Entwurf eines neuen Multiagentensystemmodells für Intelligente Stromnetze. Dieser Artikel von

Zaballos et al. [19] beschäftigt sich mit der Beschreibung eines mehrlagigen Kommunikationsnetzwerks für ein Intelligentes Stromnetz, wobei der Fokus auf einer Beteiligung aller Geräte an der Kommunikation liegt. Die Veröffentlichung [20] von Zhabelova und Vyatkin behandelt den Datenfluss in einem Intelligenten Stromnetz, vor allem in einem Multiagentensystem. Da diese Veröffentlichungen alle nicht mit dem PCM arbeiten und auch keine Angriffsmodellierung enthalten, müssen ihre Ergebnisse hier adaptiert werden.

Die erste Veröffentlichung [8] befasst sich mit dem Entwurf eines neuen Multiagentensystemmodells für Intelligente Stromnetze. Der Fokus der Arbeit liegt dabei auf einer sogenannten *Virtual Power Plant*, einem kleinen Intelligenten Stromnetz, welches aus einigen dezentralen Erzeugern, beispielsweise Solaranlagen, sowie Verbrauchern besteht. Dieses wird mithilfe mehrerer Kontrollinstanzen dynamisch über eine in der Veröffentlichung erarbeitete Vorhersage mittels neuronaler Netze gesteuert.

Die für die Verwaltung einer *Virtual Power Plant* definierte Architektur enthält für einen Kontrollagenten verschiedene Aufgaben beziehungsweise Komponenten, die sich in benötigter Hardware widerspiegeln. Ein Kontrollagent muss unter anderem eine autonome Fehlerbehebung beinhalten, außerdem eine Kontrolle über die produzierte und verbrauchte Energie und entsprechende Komponenten für die Vorhersage. Um diese Aufgaben erfüllen zu können wird ein Agent in dieser Veröffentlichung um eine oder mehrere Datenbanken, Server und Überwachung erweitert. Die Überwachung wird für diese Arbeit vereinfacht als Windows Desktop PC umgesetzt.

Der Artikel [19] beschäftigt sich mit dem Entwurf eines Kommunikationsnetzes für Intelligente Stromnetze. Dieses Netz ist aus mehreren Schichten aufgebaut, welche unterschiedliche Gerätearten enthalten und mit unterschiedlichen Protokollen kommunizieren. Eine für diese Arbeit interessante Ebene ist die Kommunikation des Sensornetzwerkes beziehungsweise der Intelligenten Stromzähler. Die Veröffentlichung nutzt dabei die Intelligenten Stromzähler oder Netzwerkknoten selbst, um die Sensordaten zu übertragen. Dazu haben Intelligente Stromzähler und Netzwerkknoten Schnittstellen für den Anschluss von beliebigen Sensoren. Ein Beispiel für die Nutzung eines solchen Sensors wäre ein Temperatursensor in einem Haus, mit welchem die Steuerung der Heizung in Abhängigkeit vom aktuellen Strompreis automatisiert wird.

Neben dieser Erweiterung definieren die Autoren noch eine Schicht Vermittlungshardware, welche den Zugriff auf die anfallenden Daten kapselt und beispielsweise Sicherheitsfunktionen übernimmt. Da eine solche Schicht im Smart Grid Resilience Framework jedoch nicht vorgesehen ist und zu einer direkten Änderung der Topologie führen würde, wird sie für diese Arbeit nicht verwendet. Falls das Metamodell des SGM, welches im Kapitel 2.2.1.1 erläutert wurde, allerdings in Zukunft verändert werden sollte, bietet sich die Einführung einer solchen Schicht an.

Die letzte Veröffentlichung [20] geht vor allem auf die Kommunikation in einem Intelligenten Stromnetz ein, die sich an schnell wechselnde und unter Umständen sehr unterschiedliche Teilnehmer anpassen muss. Um diesen Bedingungen gerecht zu werden, stellen die Autoren eine neue Architektur für ein Intelligentes Stromnetz vor, welches in

verschiedene Ebenen gegliedert ist. Diese Ebenen führen unterschiedliche Aufgaben aus und sind auf verschiedene Knoten des Netzwerks verteilt.

Der in der Veröffentlichung vorgestellte Ansatz erlaubt unter anderem eine beschränkte Selbstheilung des Intelligenten Stromnetzes bei einem Ausfall. Er lässt sich allerdings nicht auf die in dieser Arbeit angefertigte Architektur übertragen, da dafür Änderungen an der Topologie notwendig sind. Diese werden benötigt, da eine Typisierung der Kontrollzentren erfolgen müsste. Diese Annahme kann allerdings in den anderen Projekten, welche eine solche Typisierung ebenfalls durchführen müssten, nicht umgesetzt werden.

3.3 Angriffe auf die Kontrollstrukturen des Stromnetzes

Eine Analyse eines tatsächlichen Angriffs auf die Kontrollstrukturen eines Stromnetzes ist der Angriff auf das Stromnetz der Ukraine, der im Bericht von Booz et al. [18] erläutert wird. Die Vorbereitungen auf den Angriff dauerten fast ein Jahr und begannen mit dem Sammeln von Informationen über das Stromnetz und dem Beschaffen der für den Angriff genutzten Software (*BlackEnergy RAT*). Über Phishing wurden daraufhin Zugriffsmöglichkeiten auf erste Rechner in den Unternehmensnetzwerken der Energieversorger geschaffen. Daraufhin wurde die Software über manipulierte Microsoft Office Dokumente, welche von Angestellten der Energieversorger geöffnet wurden, installiert.

Nach dem Einrichten von *Command-and-Control*-Servern wurden Erweiterungen der Schadsoftware installiert und Zugangsdaten zum Steuerungssystem der Produktionsanlagen gesammelt. Mit den so gesammelten Informationen konnten angepasste Treiber für die gefundenen Netzwerkcontroller geschrieben werden, die beim Stattfinden des Angriffs die Kommunikation mit den physischen Schaltern in Umspannwerken verhinderten. Des Weiteren wurde die unterbrechungsfreie Stromversorgung des Kontrollzentrums manipuliert, um im richtigen Moment neu zu starten und so die Stromversorgung vollständig zu verhindern.

Die Erkenntnisse aus der Analyse dieses Angriffs lassen sich nicht direkt übernehmen, da der Fokus dieser Untersuchung auf dem momentanen Stromnetz mit einer zentralen Steuerung liegt. Der Ansatz im Smart Grid Resilience Framework ist allerdings die dezentrale Steuerung von weitestgehend autonom agierenden Teilnetzen.

Außerdem dauerte der Angriff inklusive seiner Vorbereitungen mit fast einem Jahr sehr lange und hatte für die Angreifer keine unmittelbaren Vorteile, was für einen staatlich finanzierten Angriff spricht. Dieser hatte laut den Autoren den Zweck, Unruhe zu stiften und das Vertrauen in die Energieversorger zu verringern. Ein solcher Angriff ist wegen seiner Komplexität nicht vollständig durch die Analyse in einem Modell abbildbar.

3.4 Angriffe auf das Stromnetz über Verbraucher

Die Veröffentlichung von Soltan et al. [16] beschreibt die Analyse eines Angriffs mittels kompromittierter Geräte des *Internet der Dinge*, welche die Stabilität der Stromversorgung stören beziehungsweise diese sogar vollständig zum Erliegen bringen. Die Idee hinter dem

Angriff ist, eine große Anzahl von Geräten des *Internet der Dinge* in geographischer Nähe unter die Kontrolle eines Angreifers zu bringen. Diese Geräte sollten einen möglichst hohen Stromverbrauch haben und sich über das Internet an- und ausschalten lassen. Verfügt ein Angreifer über Zugriff auf eine ausreichende Anzahl an Geräten dieser Art in geographischer Nähe, kann er den aktuellen Stromverbrauch einer Region deutlich erhöhen oder verringern, indem er die kompromittierten Geräte gleichzeitig an- oder ausschaltet.

Durch die große Veränderung des Verbrauchs im Netz schwankt die Frequenz des Wechselstroms, welcher zur Übertragung genutzt wird. Wenn diese Änderung einen Schwellwert übersteigt, können die Generatoren in Kraftwerken ausfallen, was zu einem großflächigen Stromausfall führt. Der Schwellwert hängt dabei von der Zusammensetzung des angegriffenen Stromnetzes ab. Der Angriff selbst hat dabei nichts mit dem Stromnetz zu tun, sondern arbeitet lediglich mit den angeschlossenen Verbrauchern.

Diese Art des Angriffs ist nicht in dieser Arbeit verwendet worden. Das Smart Grid Resilience Framework enthält zwar eine Simulation zur Berechnung der Stromverteilung für die einzelnen Intelligenten Stromzähler, die an diese angeschlossenen Stromnetzknoten sind allerdings nicht weiter modelliert. Das in dieser Arbeit vorgestellte erweiterte Modell kann zwar um solche Informationen ergänzt werden, diese können im SGM allerdings nur, als Verbrauchswert der Intelligenten Stromzähler abstrahiert, dargestellt werden. Die Abstraktion verhindert zwar, Wissen über die angeschlossenen Geräte zu nutzen, der zugrunde liegende Ansatz eines Angriffs über die Änderung des Stromverbrauchs lässt sich allerdings bereits mit der bestehenden Implementierung umsetzen. Da dafür die Simulation der Energieverteilung benötigt wird, die zum Zeitpunkt der Erstellung der Bachelorarbeit nicht verfügbar war, wurde diese Art des Angriffs jedoch nicht umgesetzt. Sollen allerdings weitere Informationen über die Geräte hinter einem solchen Intelligenten Stromzähler in einem Modell speicherbar sein, bietet sich der in dieser Arbeit vorgestellte Ansatz an.

4 Konzeption

In diesem Kapitel wird die Konzeption der im Rahmen dieser Bachelorarbeit erstellten Werkzeuge beschrieben. Dazu erfolgt zunächst eine genauere Unterteilung der erstellten Werkzeuge in Unterprojekte, die in den folgenden Kapiteln erläutert werden. Der Einsatz dieser wird zur Veranschaulichung anhand einer Fallstudie vorgestellt.

Zunächst werden die Transformation von SGM-Modellen in PCM-Modelle und die dabei getroffenen Entwurfsentscheidungen vorgestellt. Danach erfolgt eine Beschreibung des erstellten Angreifers, der auf der transformierten Netzwerktopologie arbeitet. Um die mit der Transformation erstellten PCM-Modelle auch beliebig verändern und wieder im ursprünglichen Smart Grid Resilience Framework nutzen zu können, wird danach das Konzept der erstellten Rücktransformation von einem PCM-Modell auf ein SGM-Modell erläutert.

4.1 Aufbau

Um die Auswirkungen von Angriffen auf Netzwerktopologien von Intelligenten Stromnetzen zu untersuchen, müssen diese zunächst auf geeignete Art und Weise modelliert werden. Um diese Modellierung möglichst einfach erweitern zu können erfolgt sie nicht im SGM, weil es im SGM-Metamodell keine Möglichkeit zur Erweiterung der definierten Komponenten gibt. Stattdessen erfolgt eine Transformation in das PCM. Im PCM lassen sich auf Grund des Aufbaus des PCM direkt neue Komponenten hinzufügen. Diese Transformation ist im Kapitel 4.3 zu finden. Dabei werden die einzelnen Schritte der Transformation nach dem entsprechenden Modell, welches sie erstellen, gegliedert.

Danach wird der im Rahmen dieser Bachelorarbeit erstellte Angreifer im Kapitel 4.4 vorgestellt, welcher das zweite Werkzeuge darstellt. Im Gegensatz zum bestehenden Angreifer aus dem Smart Grid Resilience Framework, der näher in Kapitel 2.2.1.4 erläutert ist, arbeitet der neue Angreifer auf der Netzwerktopologie der erstellten PCM-Modellen. Um die Ergebnisse der Angreiferanalyse direkt in den anderen Projekten nutzbar zu machen, arbeitet er allerdings auch mit den vorgestellten Modellen für die Eingabe- und Ausgabedaten (siehe Kapitel 2.2.1.2 und 2.2.1.3). Der Funktionsumfang orientiert sich dabei an der bestehenden Angreiferanalyse .

Das dritte und letzte Werkzeug ist ebenfalls eine Transformation, diesmal allerdings von einem PCM-Modell zurück in ein SGM-Modell. Die Transformation ist näher im Kapitel 4.5 erläutert. Diese Transformation wird benötigt, um Topologieänderungen, welche in einem PCM-Modell vorgenommen wurden, in ein SGM-Modell zu übertragen ohne dieses auf die gleiche Weise manuell anpassen zu müssen. Die damit erstellten SGM-Modell sind wieder in den anderen Projekten des Smart Grid Resilience Frameworks nutzbar.

4.2 Fallstudie

Zur besseren Erläuterung der Konzepte erfolgt die Beschreibung der Transformation anhand der in Abbildung 4.1 dargestellten Beispieltopologie eines Intelligenten Stromnetzes. Eine Erklärung der einzelnen Komponenten ist in Kapitel 3.2 und eine Legende in Abbildung 2.2 zu finden.

Die Topologie ist minimal, um die erstellten Modelle noch lesbar in der Arbeit darstellen zu können, da die generierten Netzwerktopologien schnell groß werden. Sie besteht aus einem Kontrollzentrum (grüner Kreis), einem Intelligenten Stromzähler (roter Kreis), einem Netzwerkknoten (blaues Quadrat) und zwei Stromnetzknoten (gelbe Rauten). Die blauen Verbindungen sind physische, also beispielsweise Kabel und die grüne Verbindung ist eine logische. Die gelben Pfeile zeigen die Zugehörigkeit eines Stromnetzknotens zur verbundenen Komponente.

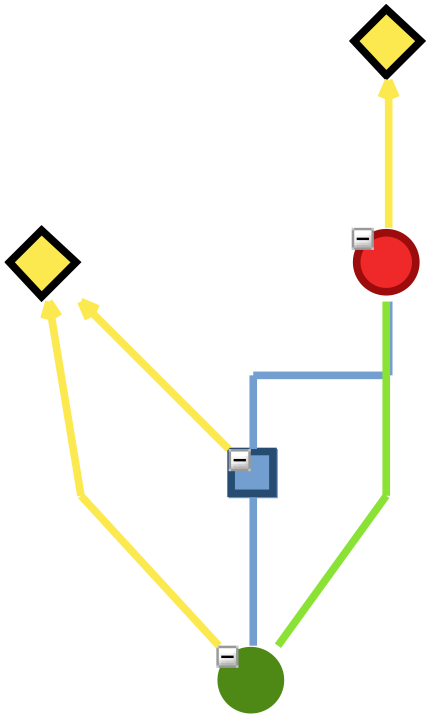

Abbildung 4.1: SGM-Modell der Beispieltopologie

4.3 Transformation von SGM-Modellen in PCM-Modelle

Die Transformation eines SGM-Modells in ein PCM-Modell erfordert das Erstellen von vier Modellen des PCM. Zunächst wird ein *Repository Model* mit den benötigten Komponenten erstellt. Diese werden danach in einem *System Model* zur Beschreibung der Komponenten in der Topologie genutzt. Da das *System Model* aber lediglich Komponenten enthält, können

hier noch keine Informationen über die Kommunikationsverbindungen der Komponenten gespeichert werden.

Diese werden dann in einem *Resource Environment* zusätzlich zu den Hardware-Ressourcen als *LinkingResource* gespeichert. Abschließend werden das *System Model* und das *Resource Environment* über ein *Allocation Model* miteinander verknüpft, um die Software, welche im *System Model* beschrieben ist, der Hardware aus dem *Resource Environment* zuzuordnen. Dabei werden alle Modelle in einer Datei gespeichert, wodurch die Identifikatoren zum Teil angepasst werden müssen, da aus einem Modellelement im SGM mehrere Modellelemente im PCM generiert werden müssen.

4.3.1 Erstellen des Repository Model

Das *Repository Model* enthält für jede Komponente der *SmartgridTopo* eine Komponente, um diese im *System Model* nutzen zu können. Bei der Wahl der Art der Komponenten standen die drei *RepositoryComponents* des PCM zur Auswahl, also *BasicComponent*, *CompositeComponent* und *SubSystem*.

Zur Modellierung wurde das *SubSystem* gewählt, da es sich leicht um beliebig viele Komponenten erweitern lässt. Das so generierte *Repository Model* ist in Abbildung 4.2 zu sehen. Da dieser Schritt als Teil der allgemeinen Transformation ausgeführt wird, enthalten die *SubSystems* an dieser Stelle noch keine weiteren Komponenten. Name und Identifikator eines *RepositoryComponent* werden fest vorgegeben.

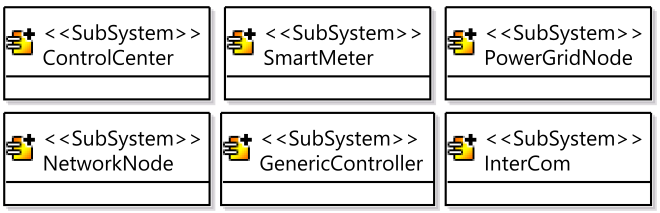

Abbildung 4.2: Repository Model der Beispieltopologie

4.3.2 Transformation des System Model

Das *System Model* wird aus den Komponenten im SGM transformiert, also für jede Komponente im SGM existiert eine Komponente im erstellten *System Model*. Dabei wird der Identifikator aus dem SGM-Modell übernommen und der Name wie im PCM üblich über den Namen des zugehörigen *RepositoryComponent* ermittelt. Außerdem wird das erstellte *Repository Model* verlinkt. Zwischen den einzelnen Komponenten gibt es auf der Ebene des *System Model* keine Beziehungen, da sich diese im Zuge eines Angriffs ändern würden. So würde beispielsweise ein ausgeschalteter Intelligenter Stromzähler keine Daten mehr an das Kontrollzentrum liefern. Da die in dieser Arbeit gewählte Herangehensweise allerdings die SGM- und PCM-Modelle unverändert lässt, um die erzielten Ergebnisse direkt in die anderen Projekte übernehmen zu können, wurde eine Modellierung im *System Model* nicht umgesetzt.

Das aus der Transformation resultierende *System Model* ist in Abbildung 4.3 zu sehen. Entsprechend dem Ursprungsmodell aus Abbildung 4.1 enthält es ein Kontrollzentrum, einen Netzwerkknoten, einen Intelligenten Stromzähler und zwei Stromnetzknoten.

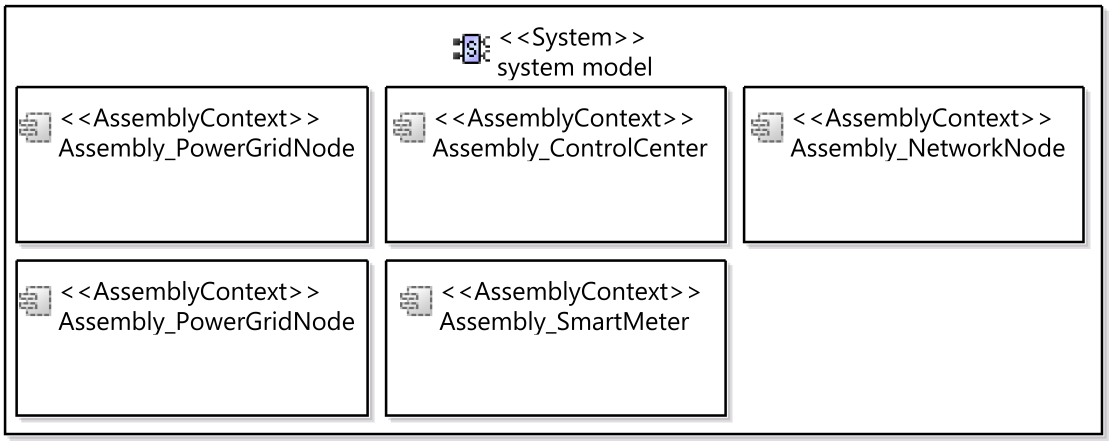

Abbildung 4.3: System Model der Beispieltopologie

4.3.3 Transformation des Resource Environment

Der umfangreichste Teil der Transformation ist die Erstellung des *Resource Environment*, weil hier neben den Komponenten selbst auch die Verbindungen zwischen diesen transformiert werden müssen. Da jede Komponente der Topologie auf eigener Hardware ausgeführt wird, wird im *Resource Environment* zu Beginn für jede Komponente aus dem SGM-Modell ein *ResourceContainer* erstellt.

Der Name der *ResourceContainer* ist dabei der Name der transformierten Komponente und der Identifikator der *ResourceContainer* ist der Identifikator der transformierten Komponente, allerdings mit zwei angehängten Zeichen. Der Grund für diese Vorgehensweise ist, dass Identifikatoren innerhalb eines Modells eindeutig sein müssen. Da alle Modelle zusammenhängen und das SGM nur einen Identifikator pro Komponente enthält, während mit der Transformation drei PCM-Komponenten pro SGM-Komponente erstellt werden, müssen die Identifikatoren hier angepasst werden.

Neben den Komponenten müssen auch deren Verbindungen transformiert werden. Dafür werden die drei verfügbaren Verbindungsarten, physische, logische und Stromverbindungen, jeweils in eine *LinkingResource* transformiert. Diese enthält die an der Verbindung beteiligten *ResourceContainer*. Die vom PCM erwarteten Parameter zur Leistung der Komponenten werden mit Standardwerten belegt.

Das resultierende *Resource Environment* ist in Abbildung 4.4 zu sehen. Die Komponenten sind entsprechend als *ResourceContainer* und die Verbindungen als *LinkingResource* umgesetzt. Der Typ der Komponenten und Verbindungen kann dabei dem Namen entnommen werden.

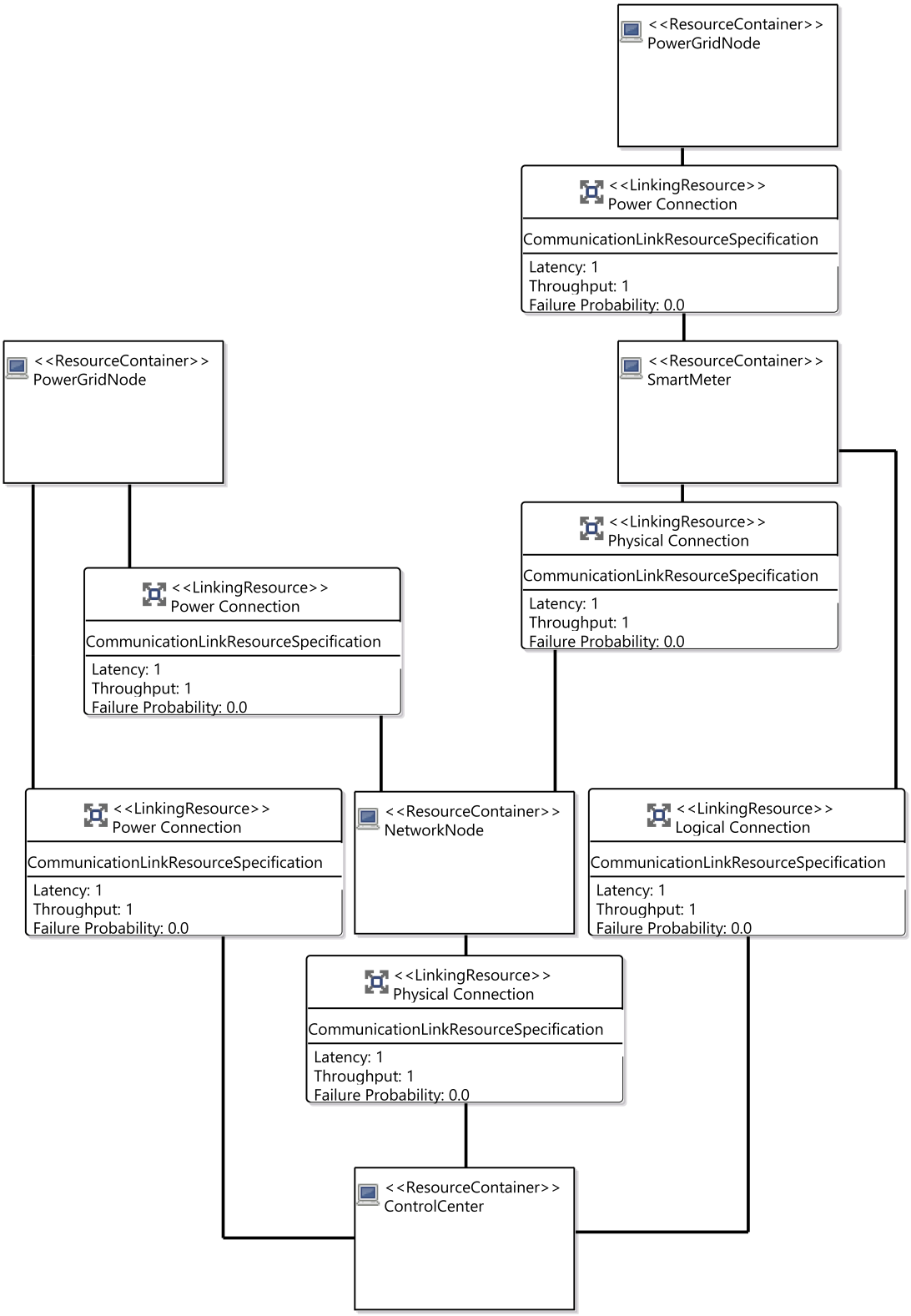

Abbildung 4.4: Resource Environment der Beispieltopologie

4.3.4 Transformation des Allocation Model

Das letzte erstellte Modell ist das *Allocation Model*, welches das *Resource Environment* mit dem *System Model* verknüpft. Dazu müssen der aus dem SGM-Modellelement generierte *AssemblyContext* und der ebenfalls daraus generierte *ResourceContainer* gefunden und einem *AllocationContext* zugeordnet werden. Die Zuordnung erfolgt dabei immer direkt eins zu eins, da für jede Komponente des SGM im *System Model* auch eine Komponente im *Resource Environment* erzeugt wurde.

Auch für den *AllocationContext* wird ein eindeutiger Identifikator benötigt, daher wurde hier der Identifikator des SGM-Modellelements mit einem angehängten festen Zeichen gewählt. Der Name ist wie üblich aus dem String *Allocation_* und dem Namen des *AssemblyContext* zusammengesetzt. Das resultierende *Allocation Model* ist in Abbildung 4.5 zu sehen.

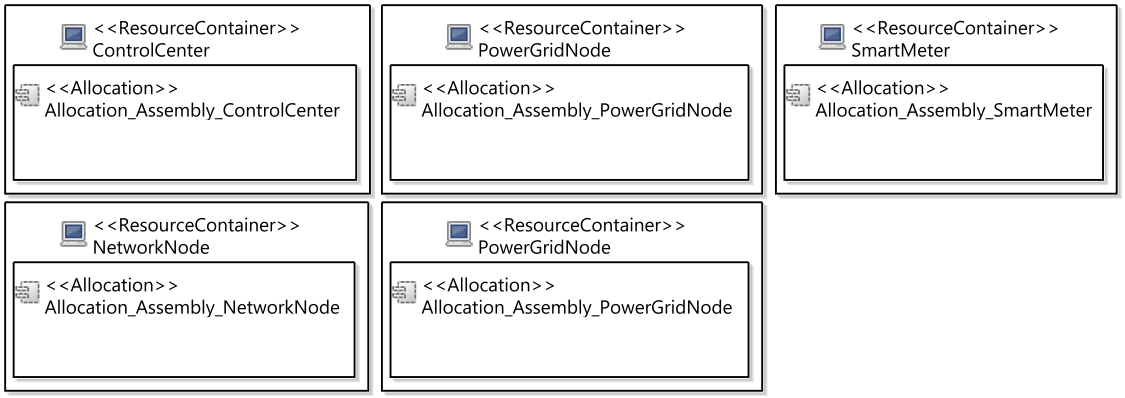

Abbildung 4.5: Allocation Model der Beispieltopologie

4.3.5 Manuelle Erweiterung um Komponenten

Die so erstellte Netzwerktopologie wird nun um die im Kapitel 3.2 erläuterten Komponenten erweitert. Da diese Erweiterungen vom speziellen Aufbau des untersuchten Netzes abhängen, ist hier keine allgemeine Vorgehensweise möglich, sondern es erfolgt eine manuelle Erweiterung der generierten Modelle. Diese Erweiterung der Modelle wird nicht in der Analyse verwendet, da eine Evaluierung mit der bereits bestehenden Angreiferanalyse nicht möglich ist. Eine Evaluation unabhängig vom bestehenden Ansatz ist aufgrund der beschränkten Zeit ebenfalls nicht möglich gewesen. Dieses Kapitel ist eine Demonstration eines Teils der sich durch die neue Darstellung ergebenden Möglichkeiten.

Dazu wird zunächst das *Repository Model* um die beschriebenen Komponenten erweitert. Dies ist in Abbildung 4.6 zu sehen. Die neu hinzugefügten Komponenten sind in der unteren Reihe zu sehen und sind eine Datenbank, ein Server und ein Windows PC sowie ein generischer Sensor. Da für diese Komponenten keine detailliertere Modellierung vorgenommen wird, werden sie als *BasicComponent* dem *Repository Model* hinzugefügt.

Falls konkrete Daten über die zu analysierende Topologie vorliegen, kann natürlich eine genauere Beschreibung der hinzugefügten Komponenten vorgenommen werden.

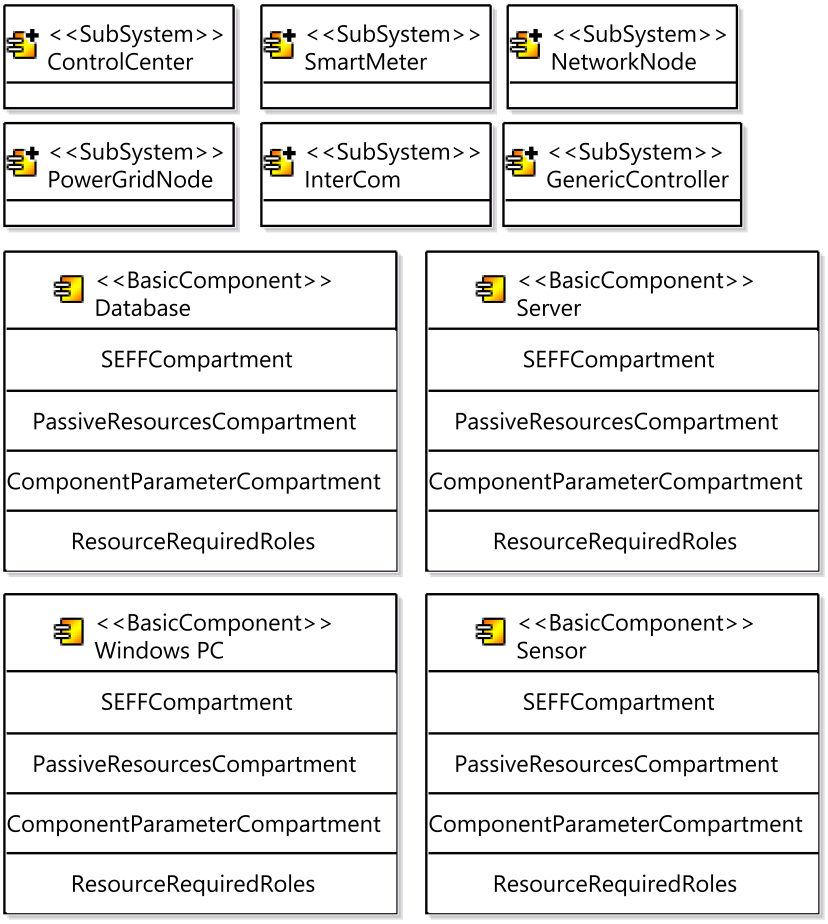

Abbildung 4.6: Erweitertes Repository Model der Beispieltopologie

Mit einem neu hinzugefügten *RepositoryComponent* kann die Beschreibung der Komponenten präzisiert werden, indem das *SubSystem* um die entsprechenden *RepositoryComponents* erweitert wird. Beispielhaft ist das in Abbildung 4.7 für ein Kontrollzentrum zu sehen. Dieses wurde, wie in Kapitel 3.2 erläutert, um zwei Datenbanken, zwei Server und einen Windows PC erweitert. Für diese Arbeit wird dabei keine Modellierung der internen Verbindungen vorgenommen, da die Angriffsmöglichkeiten aus den Komponenten abgeleitet werden und eine vollständige innere Kommunikation angenommen wird.

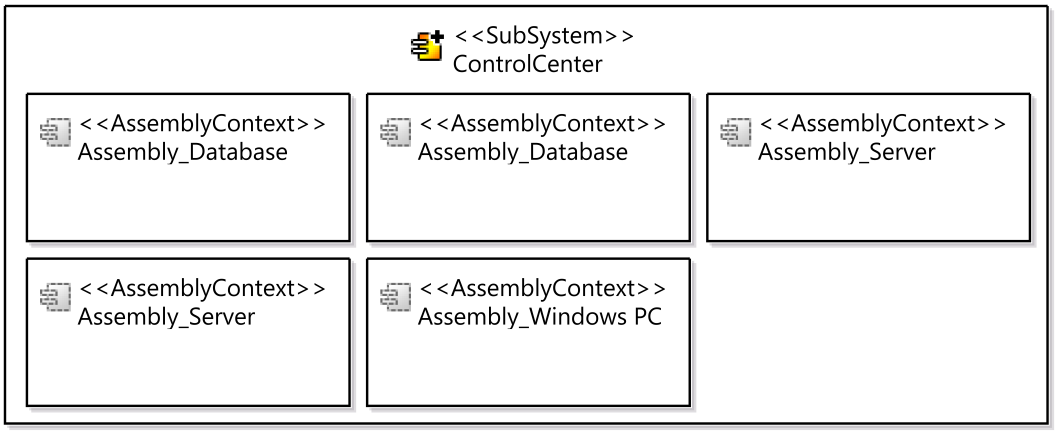

Abbildung 4.7: Erweitertes SubSystem der Beispieltopologie

Die vorgenommenen Änderungen wirken sich auch auf das *Allocation Model* aus, welches nun die *AllocationContexts* mit erweiterten *AssemblyContexts* enthält. Da die Erweiterung keine explizite Verbindungsmodellierung enthält, wird das *Resource Environment* nicht verändert und die Elemente eines *SubSystem* laufen alle auf einem Hardwareknoten. Im konkreten *Allocation Model*, das in Abbildung 4.8 zu sehen ist, sind auch die Sensoren in Netzwerkknoten und Intelligenten Stromzählern zu finden.

Abbildung 4.8: Erweitertes Allocation Model der Beispieltopologie

4.4 Angreiferanalyse auf der erstellten Netzwerktopologie

Mit der Netzwerktopologie des PCM-Modells lässt sich eine Angreiferanalyse implementieren. Da für diese die Eingabe- und Ausgabeformate des Smart Grid Resilience Framework genutzt werden, welche den Status der Komponenten über eine Referenz auf das SGM-Modellelement speichern, wird auch das SGM-Modell benötigt. Die Modelle werden genutzt, um eine direkte Verwendbarkeit der Ergebnisse in den anderen Projekten des Smart Grid Resilience Frameworks zu ermöglichen.

In einem ersten Schritt müssen daher alle Modelle geladen werden. Um die Modelle einfacher nutzen zu können, wird für jedes eine Adapter-Klasse angelegt. Die so geladenen Daten werden in einem Objekt zusammengeführt, welches die Topologie mit den vier Modellen darstellt. Die Informationen über die Knoten, also ihr Aufbau und die Verbindungen untereinander, werden dabei aus dem generierten PCM-Modell extrahiert. Der Status eines Knotens wird über das *SmartgridInput*-Modell eingelesen und der Status nach erfolgter Angreiferanalyse wird über das *SmartgridOutput*-Modell gespeichert. Für Knoten, die während der Angreiferanalyse nicht betrachtet wurden, beispielsweise weil sie keine logische Verbindung zum Startknoten des Angriffs haben, wird abschließend der Status aus der Eingabe für die Ausgabe übernommen.

Um sinnvolle Ergebnisse zu erzielen, müssen die übergebenen Modelle die gleiche Topologie beschreiben. Wenn das generierte PCM-Modell mit Topologieänderungen versehen wird, passen die anderen Modelle nicht mehr und die Daten können nicht korrekt verknüpft werden. Um Änderungen an der Topologie dennoch zu ermöglichen, kann eine Transformation von einem PCM-Modell wieder zurück in ein SGM-Modell vorgenommen werden, für das ein neues *SmartgridInput*-Modell generiert werden kann. Wie ein solches *SmartgridInput*-Modell aus einem SGM-Modell generiert werden kann, ist in [6] beschrieben.

Der implementierte Angreifer erhält die in den vorherigen Absätzen beschriebene Topologie. Diese besteht aus den unterschiedlichen geladenen Modellen. Außerdem erhält er einen Startknoten, von dem der Angriff ausgeht, und weitere Parameter, die hier kurz erläutert werden. Die Geschwindigkeit des Angreifers gibt an, wie viele Knoten der Angreifer auf einmal kompromittieren kann. Wie oft er diesen Schritt ausführen kann, gibt die Anzahl der Runden an. Weiterhin wird übergeben, ob der Angreifer logische Verbindungen berücksichtigen soll. Soll er dies, kann er nur Knoten kompromittieren, die sowohl physisch als auch logisch mit bereits kompromittierten Knoten verbunden sind. Logische Verbindungen können allerdings auch ignoriert werden, in diesem Fall kann der Angreifer alle Knoten kompromittieren, welche physisch mit bereits kompromittierten verbunden sind.

Außerdem erhält der Angreifer noch die zu nutzenden Vorgehensweise für die Ermittlung des nächsten zu kompromittierenden Knotens. Wie in dem Kapitel 2.2.1.4 zur bestehenden Analyse erläutert, gibt es hier drei Möglichkeiten. Die Vorgehensweise *BFS* befüllt die Liste der zu kompromittierenden Knoten nach einem Breitensuche-Algorithmus, *DFS* nach einem Tiefensuche-Algorithmus und *Fully-Meshed* befüllt die List in einer zufälligen Reihenfolge. Die Vorgehensweise *Fully-Meshed* geht davon aus, dass Knoten direkt kompromittiert werden können, unabhängig davon, ob schon ein benachbarter Knoten kompromittiert wurde oder nicht.

Für zukünftige Arbeiten, welche eine Erweiterung wie die im letzten Kapitel erläuterte vornehmen wollen, muss neben der so übergebenen Definition der Fähigkeiten des Angreifers noch festgelegt werden, inwieweit die hinzugefügten Elemente, wie beispielsweise ein Windows PC, die Möglichkeiten, eine Komponente zu kompromittieren, erhöhen. Neben Elementen, die das Risiko erhöhen, sind auch solche denkbar, die das Risiko verringern, beispielsweise eine Firewall in einem Netzwerkknoten, welche nur eine unidirektionale

Kommunikation zulässt. Um dort weiter zu kommen, muss der Angreifer also detailliertes Wissen über den Netzwerkknoten erlangen und ihn separat angreifen.

4.5 Transformation von PCM-Modellen in SGM-Modelle

Um die generierten Modelle auch verändern zu können, ohne die Änderungen auch im entsprechenden SGM-Modell vornehmen zu müssen, wurde eine Transformation von einem PCM-Modell auf ein SGM-Modell erstellt. Da im SGM die genauere Darstellung der Komponenten nicht beschrieben werden kann, gehen in diesem Schritt einige Informationen verloren, allerdings bleibt die generierte Netzwerktopologie erhalten.

Zunächst werden die Komponenten des *SmartgridTopo*-Modells aus den *AssemblyContexts* des *System Models* transformiert. Dabei wird der Identifikator übernommen. Neben den Komponenten müssen noch die Verbindungen transformiert werden, die im *Resource Environment* als *LinkingResource* gespeichert sind. Die Unterscheidung der Art der Verbindung erfolgt dabei anhand des Namens. Mit diesen beiden Transformationsschritten ist die Transformation abgeschlossen. Da das *Repository Model* nicht verwendet wird, können daraus auch keine erweiterten Informationen ins SGM-Modell übernommen werden. Diese wären ohnehin nicht im SGM speicherbar. Das Ausführen der ersten Transformation auf ein SGM-Modell und das Ausführen dieser Transformation auf das generierte Modell führen dann zu einem zum Ursprungsmodell semantisch gleichen SGM-Modell.

5 Implementierung

Die Implementierung der Projekte wurde in den Sprachen Java und *QVT Operational* (QVTo) vorgenommen. Als Entwicklungsumgebung kamen die *Eclipse Modelling Tools* zum Einsatz. Die Transformationen und die Angreiferanalyse wurden auf dem Smart Grid Resilience Framework aufbauend implementiert.

Das erste Unterkapitel beginnt mit einer kurzen Erläuterung der für die Entwicklung genutzten Werkzeuge. Danach erfolgt eine Beschreibung der erstellten Tests, welche in der Evaluation verwendet werden. Das nächste Kapitel gibt einen Überblick über die verwendete Transformationssprache QVTo. Daran anschließend erfolgt eine Erläuterung der tatsächlichen Umsetzung der Transformationen. Abschließend wird noch die Implementierung der Angreiferanalyse erläutert.

5.1 Ingenieursmäßige Softwareentwicklung

Neben den *Eclipse Modelling Tools* werden für die Benutzung noch die Modellprojekte der verwendeten Metamodelle benötigt, also die Modelle des Smart Grid Resilience Frameworks[1]. Diese sind das *SmartgridTopo*-Metamodell, das *SmartgridInput*-Metamodell und das *SmartgridOutput*-Metamodell. Außerdem wird das *Palladio Komponentenmodell* benötigt. Für die Transformationen ist das *QVT Operational SDK* erforderlich.

Zur Versionskontrolle wurde ein privates *Github Repository* verwendet. Die Ausarbeitungen und der Code, der im Rahmen dieser Bachelorarbeit erstellt wurde, ist außerdem in diesem SVN[2] zu finden. Um die Ergebnisse der durchgeführten Analysen zu überprüfen wurden Unit-Tests implementiert. Diese sind im Ordner *Tests* zu finden und sind aufgeteilt in einen Test, welcher die Genauigkeit der Transformationen überprüft und manuelle Nachkontrolle erfordert und einen automatischen Test, der die Genauigkeit der Angreiferanalyse überprüft, sowie einem Test zur Evaluation der Skalierbarkeit der erstellten Werkzeuge.

5.2 Transformationssprache QVTo

Die in dieser Bachelorarbeit erstellten *Modell-zu-Modell*-Transformationen überführen die SGM- und PCM-Modelle ineinander. Bei den verwendeten Transformationen wird ein Quellmodell in ein anderes Zielmodell transformiert, daher handelt es sich um eine *Outplace*-Transformation.

[1] https://svnserver.informatik.kit.edu/i43/svn/code/SmartGrid/
[2] https://svnserver.informatik.kit.edu/i43/svn/stud/ThomasWeber/Code/Tests

Für diese Transformationen wurde die Transformationssprache QVTo gewählt, da sie eine verbreitete Sprache für Modelltransformationen ist. Die Informationen dieses Kapitels stammen aus diesem Buch [12] über QVTo. Eine Transformation besteht aus einer Signatur und einem Ausführungsteil. Weiterhin hat sie einen Namen. Die Signatur deklariert die Ein- und Ausgabemodelle der Transformation. Der Ausführungsteil enthält im Regelfall eine *main*-Funktion, welche den Einstiegspunkt der Transformation darstellt. Neben der *main*-Funktion stehen noch sogenannte *mapping*- und *helper*-Funktionen zur Verfügung.

Eine *mapping*-Funktion (kurz *mapping*) transformiert ein Modellelement des Eingabemodells in eines des Ausgabemodells. Quell- und Zielmodellelement werden dabei in der Signatur aufgeführt. Weiterhin kann ein *mapping* Parameter benötigen. Ein *mapping* wird auf eine Menge von Modellelementen mittels des Schlüsselwortes *map* angewendet. Dabei kann ein *mapping* nur Modellelemente des Eingabemodells auf Modellelemente der Ausgabemodelle abbilden, allerdings keine neuen Modellelemente erzeugen.

Neben der *mapping*-Funktion gibt es noch eine *helper*-Funktion. Diese dient vor allem dem Berechnen von Aggregaten oder dem Überprüfen von Bedingungen, kann allerdings auch Modellelemente in der Ausgabe erzeugen. Diese Funktionalität ist wichtig für das Erstellen des *Repository Model*, da hierbei kein Quellmodell existiert, sondern lediglich ein Zielmodell erzeugt wird. Eine weitere wichtige Funktion ist *resolveone*, da damit für ein Modellelement des Eingabemodells die im Verlauf der Transformation generierten Modellelemente aus den Ausgabemodellen ermittelt werden können. Dies ist vor allem relevant für das Erstellen des *Allocation Model*, da dabei die erstellten *AssemblyContexts* und *ResourceContainer* ermittelt werden müssen.

5.3 Aufbau der Transformationen

Das Konzept der Transformationen ist bereits in den Kapiteln 4.3 und 4.5 dargelegt worden, daher erfolgt hier lediglich eine Erläuterung mit Bezug auf den Einsatz von QVTo. Die Transformationen sind im SVN-Repository[3] in den Ordnern *SGMtoPCM* und *PCMtoSGM* zu finden und werden hier näher beschrieben.

Die Transformation startet mit der Erstellung des *Repository Model* mittels *helper* Funktionen, da hierbei neue Elemente erstellt werden und nicht nur Elemente aufeinander abgebildet werden. Die Identifikatoren und Namen der erzeugten *RepositoryComponent* sowie des erzeugten *Repository* sind statisch gewählt. Die einzelnen Komponenten werden dabei über einen Konstruktoraufruf erzeugt und dem *Repository Model* hinzugefügt.

Darunter folgt die Erstellung des *System Models*, bei welchem *NetworkEntities* auf *AssemblyContexts* gemappt werden. Wichtig ist dabei der Parameter des *mappings*, da jeder erzeugte *AssemblyContext* darüber seinen zugehörigen *RepositoryComponent* als Attribut erhält. Danach folgt die Erstellung des *Resource Environment*, wobei die einzelnen *NetworkEntities* jeweils in den *toResourceContainer-mappings* auf *ResourceContainer* gemappt werden.

[3] https://svnserver.informatik.kit.edu/i43/svn/stud/ThomasWeber/Code/

Nachdem die Komponenten gemappt sind können nun die Verbindungen transformiert werden. Dies geschieht für alle physischen Verbindungen beziehungsweise *PhysicalConnections* im *createLinkingResourcePhys-mapping*. Danach werden die Stromverbindungen erstellt. Da das SGM-Metamodell dafür allerdings keine Verbindung vorsieht, sondern lediglich einen Stromnetzknoten als Attribut einer Komponente speichert, erfolgt das Mapping etwas abgewandelt im *addLinkToPGN-mapping*. Danach werden alle logischen Verbindungen (*LogicalCommunication*) im *mapping createLinkingResourcelog* auf eine *LinkingResource* gemappt.

Die für die erstellten *LinkingResources* benötigten *CommunicationLinkResourceSpecifications* werden über die *helper* am Ende mit Standardwerten erstellt. Wenn die zur Verfügung stehende Topologie genauere Informationen über die Verbindungseigenschaften enthält, können diese in der Transformation ergänzt werden.

Mit dem generierten *System Model* und dem generierten *Resource Environment* lässt sich das *Allocation Model* füllen. Dazu werden über die *resolveone*-Methode die *AssemblyContexts* und *ResourceContainer* ermittelt, die aus einem SGM-Modellelement erstellt wurden und einander in einem *AllocationContext* zugeordnet.

Die Rücktransformation, deren Konzept in Kapitel 4.5 erläutert wurde, arbeitet mit dem *System Model* und dem *Resource Environment* des gegebenen PCM-Modells, um daraus ein SGM-Modell zu erzeugen. Zunächst werden daher diese beiden Modelle aus dem Eingabemodell gesucht. Da ein PCM-Modell theoretisch mehrere Modelle des gleichen Typs haben kann, was aber im Kontext dieses Projekts nicht vorkommt, wird das erste und gleichzeitig letzte gefundene Modell gewählt. Danach werden die Komponenten des SGM-Modells aus den *AssemblyContexts* des *System Model* transformiert, wobei aufgrund der Typisierung von QVTo für jede Komponente ein eigenes *mapping* notwendig ist.

Danach werden noch die Verbindungen aus dem *Resource Environment* auf ihren entsprechenden Verbindungstyp im SGM gemappt. Dies geschieht in den *mappings toPhysConnection* und *toLogConnection* sowie dem *helper toPowConnection*, da im SGM keine Stromverbindungen existieren. Um die zugehörigen Komponenten für die Verbindungen zu finden werden wieder über *resolveone* die beteiligten Komponenten ermittelt.

5.4 Umsetzung der Angreiferanalyse

Für die transformierte Netzwerktopologie kann nun eine Angreiferanalyse implementiert werden, deren Konzept im Kapitel 4.4 erläutert ist. Zur Übersicht werden hier die Klassendiagramme der drei Pakete des Projekts *AttackerAnalysis* vorgestellt. Das erste Paket ist das Paket *model*, welches die Adapter für die einzelnen Modelle enthält. Ein Klassendiagramm ist in der Abbildung 5.1 zu sehen. Die Adapter erben von der Klasse *Model*, welche die gemeinsame Funktionalität und spezifische Einschubmethoden für die erbenden Klassen enthält. Außerdem ist die Klasse *AdaptedModelExtent* enthalten, die für die Ausführung der Transformationen benötigt wird. Die implementierten Klassen für die genutzten Modelle enthalten noch Zusatzfunktionen, die einen einfacheren Zugriff auf die in ihnen gekapselten Modelle und deren Komponenten ermöglichen.

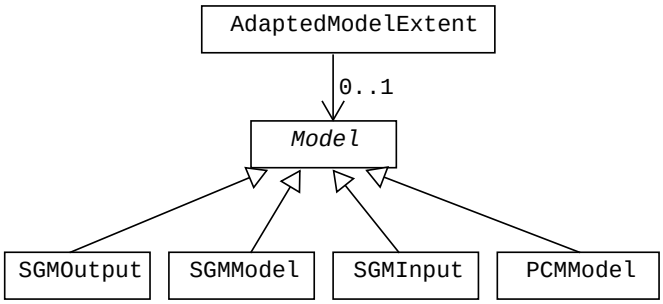

Abbildung 5.1: Klassendiagramm des Pakets model

Das zweite Paket *analysis* enthält Daten zur Verwaltung der geladenen Topologie. Das Klassendiagramm des Pakets ist in Abbildung 5.2 zu sehen. Die Klassen *ExtendedPowerGridNode* und *ExtendenNetworkEntity* kapseln alle Daten über die entsprechende Komponente aus allen geladenen Modellen, also aus dem erstellten PCM-Modell, dem SGM-Modell sowie dem Ein- und dem Ausgabemodell. Über die entsprechenden Methoden kann auf den Status der Komponente zugegriffen beziehungsweise dieser verändert werden. Die vollständige Topologie ist in der Klasse *AnalysisData* gekapselt. Über eine Instanz dieser Klasse kann ein Angreifer eine Topologie angreifen und mit den gegebenen Methoden Informationen darüber erlangen.

Weiterhin enthält das Paket das Enum *HackedType*, welches die im *SmartgridOutput*-Modell definierten Zustände einer Komponente enthält, da diese nicht aus dem Metamodell extrahierbar waren. Außerdem ist die Klasse *UsedUris* zu finden, welche die genutzte Ordnerstruktur, die in diesem Projekt verwendet wurde, kapselt und *URIs* für die unterschiedlichen Modelle gemäß ihres Speicherortes generiert.

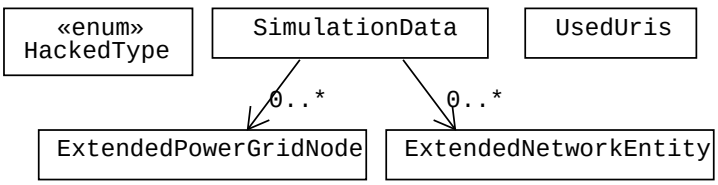

Abbildung 5.2: Klassendiagramm des Pakets analysis

Das letzte Paket *attacker* enthält die Implementierung der tatsächlichen Angreifer. Das Klassendiagram dieses Pakets ist in Abbildung 5.3 zu sehen. Diese erben von der abstrakten Klasse *Attacker*, die gemeinsame Funktionalität und Einschubmethoden enthält. Die Ausführung eines Angriffs erfolgt dabei über die Methode *launchAttack*. Wenn vorher nicht über *setData* ein befülltes *SimulationData*-Objekt übergeben wurde, wird eine Fehlermeldung erzeugt, da der Angreifer keine anzugreifende Topologie hat.

Weiterhin gibt es das Enum *HackingStyle*, welches die in Kapitel 4.4 erläuterten Strategien des Hackers implementiert. Außerdem wurden die ebenfalls in diesem Kapitel erläuterten Parameter im Objekt *SimulationParameters* zusammengefasst.

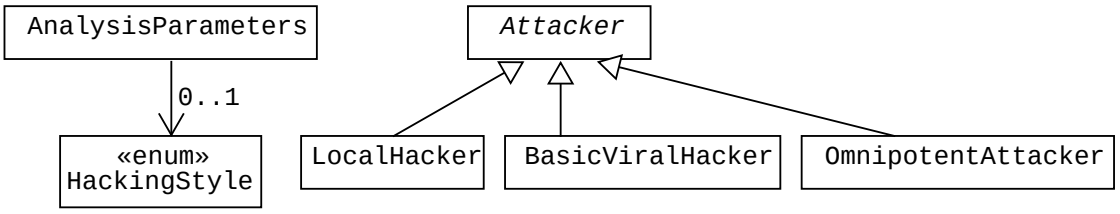

Abbildung 5.3: Klassendiagramm des Pakets attacker

5.5 Implementierung einer Fassade

Um die Verwendung der im Zuge dieser Bachelorarbeit erstellten Werkzeuge zu vereinfachen, wurde eine Fassade erstellt. Diese ist im Ordner *ProjectFacade* im SVN-Repository[4] zu finden und ein Klassendiagramm ist in Abbildung 5.4 zu sehen. Sie enthält vier wichtige Methoden, die hier kurz erläutert werden. Die erste ist die Methode *transformSgm2Pcm*, welche ein Objekt der Klasse *SGMModel* und eines der Klasse *PCMModel* als Parameter erwartet. Das enthaltene SGM-Modell wird mit der in Kapitel 4.3 erläuterten Transformation in ein PCM-Modell umgewandelt, welches im übergebenen *PCMModel* gespeichert wird. Die zweite Methode *transformPcm2Sgm* erwartet die gleichen Parameter in umgekehrter Reihenfolge und transformiert das übergebene PCM-Modell das übergebene SGM-Modell.

Die dritte und vierte Methode sind sich sehr ähnlich, da die dritte Methode *executeTransformAndAttack* lediglich eine andere Parameterliste aufweist und aus dieser die Parameter für die vierte Methode *transformAndAttack* generiert und diese aufruft. Die vierte Methode ruft zunächst die Transformation von einem SGM- zu einem PCM-Modell auf und führt danach eine Angreiferanalyse mit dem übergebenen Angreifer aus.

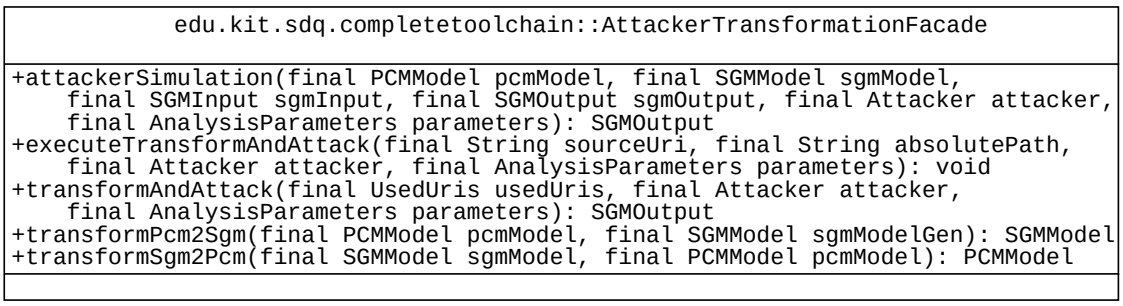

Abbildung 5.4: Fassade für alle Projekte

[4]https://svnserver.informatik.kit.edu/i43/svn/stud/ThomasWeber/Code/ProjectFacade/

6 Evaluation

Das Ziel dieser Bachelorarbeit ist einerseits das Erstellen einer Netzwerktopologie für ein vorhandenes SGM-Modell im Palladio Komponentenmodell und andererseits die Implementierung eines Angreifers auf diesem. Die vorgenommene Implementierung der beiden Transformationen und der Angreiferanalyse, die im Kapitel 5 vorgestellt wurde, wird in den folgenden Kapiteln evaluiert.

Die Evaluation erfolgt dabei anhand eines Goal-Question-Metric(GQM)-Plans [4], welcher in der Tabelle 6.1 zu sehen ist. Dabei werden zunächst Ziele formuliert, die mit der Evaluation überprüft werden sollen. Das Erreichen der Ziele wird als Fragenliste beschrieben. Diese Fragen müssen für eine Evaluation des Ziels beantwortet werden. Danach folgt die Festlegung von Metriken, mit denen die Fragen beantwortet werden, was dann klärt, ob die Ziele erreicht werden konnten.

GQM			
Ziel	Z1: Genauigkeit der Transformationen	Z2: Genauigkeit der Angreiferanalyse	Z3: Skalierbarkeit
Frage	Q1: Führt die Anwendung beider Transformationen zu einer zur Starttopologie semantisch gleichen Topologie?	Q2: Führt die Analyse eines Angriffs im PCM-Modell zum gleichen Ergebnis wie im bereits existierenden Modell?	Q3: Ist die Transformation und die Analyse eines Angreifers auch für große Modelle performant?
Metrik	M1: Anzahl semantischer Unterschiede der durch Transformation erzeugten SGM-Modelle und der Ausgangsmodelle	M2: Anzahl scheiternder übernommener Tests	M3: Betrachtung der Laufzeit im Verhältnis zur Größe der betrachteten Modelle.

Tabelle 6.1: GQM-Plan

6.1 Evaluation der Genauigkeit der Transformationen

Das erste Ziel der Evaluation ist es, die Genauigkeit der Transformationen zu evaluieren. Für die beiden Transformationen ergibt sich so die Frage, ob die Transformationen eine

semantisch gleiche Topologie erzeugen. Dies lässt sich am einfachsten überprüfen, indem beide Transformationen nacheinander ausgeführt werden. Wenn das resultierende SGM-Modell semantisch identisch zum Ursprungsmodell ist, haben beide Transformationen die Topologie erhalten. Sollte das Ergebnis der zweiten Transformation eine nicht semantisch gleiche Topologie enthalten, so hat eine der beiden Transformationen diese verändert. Als Metrik ergibt sich also die Anzahl semantischer Unterschiede zwischen dem Ursprungsmodell und dem durch Anwenden beider Transformationen entstandenen Modell.

Die Ermittlung der semantischen Unterschiede findet dabei manuell statt, da die Ausführung der Transformation in QVTo nichtdeterministisch abläuft und somit die Reihenfolge, in der die einzelnen Komponenten und Verbindungen hinzugefügt werden, nicht vorhersehbar ist. Da die Verbindungen allerdings lediglich durch ihre Position im Modell identifiziert werden, ist eine syntaktischer Vergleich nicht möglich. Die Ergebnisse dieser manuellen Evaluation sind in der Tabelle 6.2 zu sehen und entsprechend den Erwartungen. Die genutzten Modelle stammen aus den bestehenden Tests für die Angreiferanalyse .

Die Daten für die Evaluation sind im Ordner *Tests* zu finden, wobei die Ursprungsmodelle im Ordner *resources/smartgridtopo* liegen. Dieser ist im SVN-Repository[1] zu finden. Die für den Test notwendigen Daten können dann entweder über den *TransformationTest* im Paket *transformation* generiert werden, oder die bereits früher generierten Modelle können nochmals überprüft werden. Die manuelle Überprüfung der Modelle ergab, dass sie die gleiche Topologie enthalten, also dass die Transformation genau ist.

Modellname	1Lokal BFS	2Lokal BFS	3Lokal BFS	4Lokal BFS	1Lokal DFS	2Lokal DFS	3Lokal DFS	4Lokal DFS
sem. Untersch.	0	0	0	0	0	0	0	0

Tabelle 6.2: Semantische Unterschiede der getesteten Modelle

6.2 Evaluation der Genauigkeit der Angreiferanalyse

Für die Angreiferanalyse ergibt sich für das Ziel der Genauigkeit hingegen eine andere Frage, nämlich ob identische Ergebnisse im Vergleich zur bestehenden Angreiferanalyse erzielt werden können. Zur Überprüfung bieten sich die für die bestehende Angreiferanalyse erstellten Unit-Tests an.

Da beide Angreiferanalysen mit den gleichen Ein- und Ausgabedateien arbeiten, wäre als alternative Metrik die semantische Gleichheit der Ausgabedateien möglich. Weil die bestehende Angreiferanalyse zum Zeitpunkt der Erstellung dieser Bachelorarbeit in ihrer

[1] https://svnserver.informatik.kit.edu/i43/svn/stud/ThomasWeber/Code/Tests

aktuellen Version nicht lauffähig war, konnte diese Metrik nicht umgesetzt werden.

Dementsprechend kann keine allgemeine Überprüfung vorgenommen werden. Daher erfolgt die Evaluation anhand einer Auswahl der im bestehenden Projekt implementierten Tests. Diese sind, auf das neue System angepasst, im Paket *attackersimulation* zu finden und werden erfolgreich ausgeführt. Die Ergebnisse der Tests sind in Tabelle 6.3 zu finden.

Test	testExample1	testExample2	testExample2 HighSpeed	testExample3	testExample3 HighSpeed
Test erfolgreich	ja	ja	ja	ja	ja

Tabelle 6.3: Ergebnisse der übernommenen Tests

6.3 Evaluation der Skalierbarkeit

Das dritte zu evaluierende Ziel ist das der Skalierbarkeit. Bei diesem Ziel ist die Art der Evaluation für alle drei Projekte gleich, da die Laufzeit der Programme auf die selbe Art gemessen wird. Um die Skalierbarkeit zu evaluieren, wird die Laufzeit der Programme in Abhängigkeit zur Größe der eingegebenen Modelle betrachtet.

Um das zu testen werden in der Klasse *GenerateModels* Modelle mit einer unterschiedlichen Anzahl von Modellelementen erzeugt. Für diese Modelle erfolgen beide Transformationen sowie eine Angreiferanalyse . Dabei werden jeweils die Ausführungszeiten ermittelt. Die Ergebnisse sind in den folgenden Unterkapiteln dargelegt.

6.3.1 Testsystem

Die Laufzeiten zur Evaluation der Skalierbarkeit wurden alle auf dem gleichen Desktop PC erhoben. Dieser hatte zum Testzeitpunkt einen Ryzen 7 2700X (nicht übertaktet) sowie 16GB Arbeitsspeicher installiert. Die Daten und Programme lagen auf einer Samsung SSD 860 EVO. Das Betriebssystem war zum Testzeitpunkt Windows 10 Pro Build 1903 und die verwendete Java-Version war 1.8.

6.3.2 Evaluation der Transformationen

Um die Skalierbarkeit zu evaluieren wurden zufällige SGM-Modelle mit einer Sterntopologie erstellt. Dies geschieht in der Klasse *GenerateModels*. Eine Sterntopologie besteht dabei aus einem Kontrollzentrum in der Mitte, welches von Intelligenten Stromzählern umgeben ist. An diesen hängt jeweils ein Stromnetzknoten und sie sind sowohl physisch als auch logisch mit dem Kontrollzentrum verbunden.

Die so generierten Modelle werden dann mit der Methode *transformSgm2Pcm* aus der Klasse *AttackerTransformationFacade* in ein PCM-Modell transformiert und die Laufzeiten mit *System.currentTimeMillis()* ermittelt. Dies geschieht in der Klasse *ScalabilityTest* in der Methode *testTransformationSGM2PCM*. Das Verhältnis zwischen der Anzahl der Intelligenten Stromzähler und der Laufzeit der Transformation ist in Abbildung 6.1 zu sehen. Der Ausreißer am Anfang ist unabhängig von der Größe des transformierten Modells, was vermutlich durch Ladezeiten der Transformationsumgebung zustande kommt. Für den Ausreißer bei 4800 Intelligenten Stromzählern konnte kein Grund bestimmt werden. Das beobachtete Verhältnis der Laufzeit im Vergleich zur Größe der Eingabe ist quadratisch.

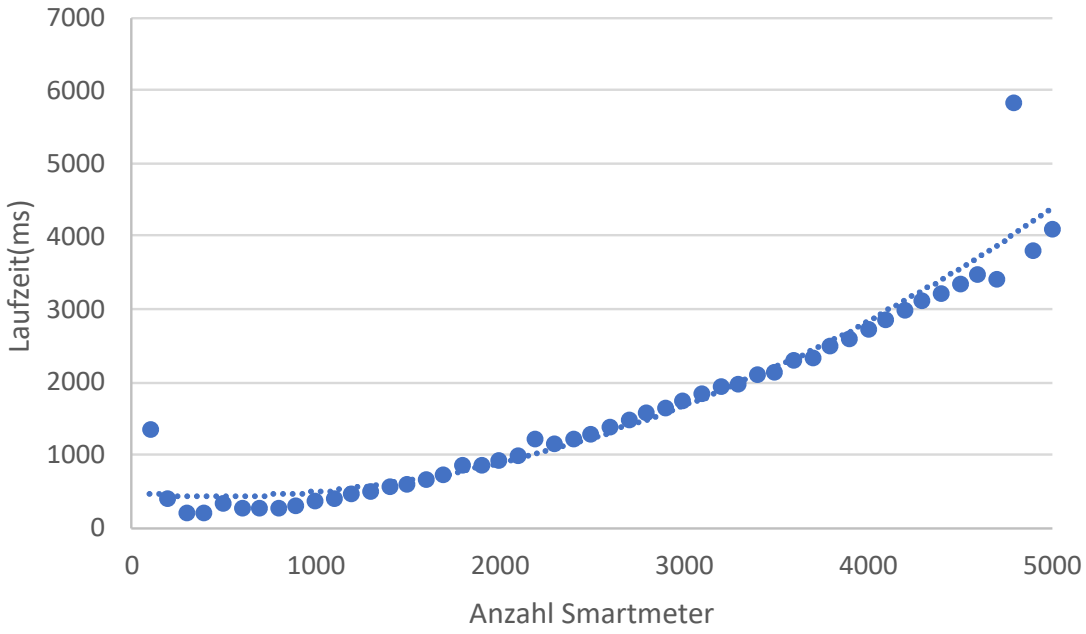

Abbildung 6.1: Laufzeitmessung der Transformation eines SGM-Modells in ein PCM-Modell

Mit den so erstellten PCM-Modellen kann nun die andere Transformation überprüft werden. Wichtig ist hierbei, dass die zweite Transformation nicht ohne die von der ersten erstellten Dateien getestet werden kann. Da der Test der ersten Transformation allerdings mehrere Minuten dauert, wird auf das explizite generieren der PCM-Modelle im zweiten Test verzichtet. Die Existenz der PCM-Modelle ist vom Benutzer der Tests durch das vorherige Ausführen des ersten Tests sicherzustellen.

Um die zweite Transformation zu testen wird die Methode *transformPcm2Sgm* aus der Klasse *AttackerTransformationFacade* genutzt und die Laufzeiten werden wieder mit *System.currentTimeMillis()* ermittelt. Dies geschieht in der Klasse *ScalabilityTest* in der Methode *testTransformationPCM2SGM*. Der zugehörigen Abbildung 6.2 kann das Verhältnis der Laufzeit zur Größe des zugehörigen SGM-Modells entnommen werden. Auf Grund der hohen Laufzeiten von fast einer Minute für das größte Modell, erfolgte dieser Test nur bis zur Größe von 2000 Intelligenten Stromzählern. Das Verhalten ist quadratisch.

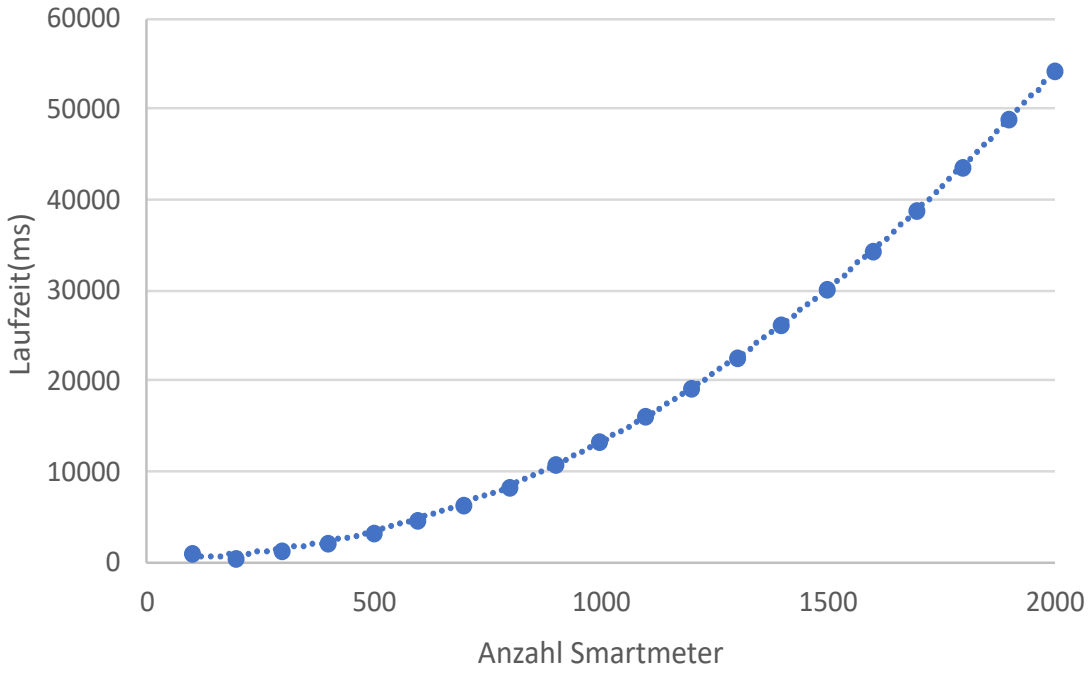

Abbildung 6.2: Laufzeitmessung der Transformation eines PCM-Modells in ein SGM-Modell

6.3.3 Evaluation der Angreiferanalyse

Um auch die Angreiferanalyse hinsichtlich ihrer Skalierbarkeit evaluieren zu können, wurde die Klasse für das Erstellen der SGM-Modelle erweitert. Diese kann für jedes generierte SGM-Modell ein zugehöriges *SmartgridInput*-Modell generieren. Dabei sind *EntityStates* und *PowerStates* für alle Modellelemente enthalten. Diese sind mit den Standardwerten initialisiert, also nicht gehackt und eine Stromversorgung der Stromnetzknoten besteht.

Dieser Test benötigt das SGM- und das *SmartgridInput*-Modell sowie das transformierte PCM-Modell. Wie beim zweiten Test ist aufgrund der langen Dauer der Erstellung der benötigten Modelle eine explizite Generierung im Test nicht vorgesehen. Stattdessen muss die Existenz der benötigten Modelle durch das vorherige Aufrufen der ersten Transformation beziehungsweise vorher der Methoden der *GenerateModels* Klasse sichergestellt werden. Damit der Test funktionieren kann, müssen die generierten Modelle zueinander passen. Wenn also die SGM- und *SmartgridInput*-Modelle neu generiert werden, müssen auch die zugehörigen PCM-Modelle neu generiert werden.

Zur Ermittlung der Laufzeit wird die Methode *attackerSimulation* aus der Klasse *AttackerTransformationFacade* genutzt. Die Laufzeiten werden mit *System.currentTime-Millis()* ermittelt. Dies geschieht in der Klasse *ScalabilityTest* in der Methode *testAttacker*. Der zugehörigen Abbildung 6.3 kann das Verhältnis der Laufzeit zur Größe des zugehörigen SGM-Modells entnommen werden. Der Graph lässt auf ein quadratisches Laufzeitverhalten schließen.

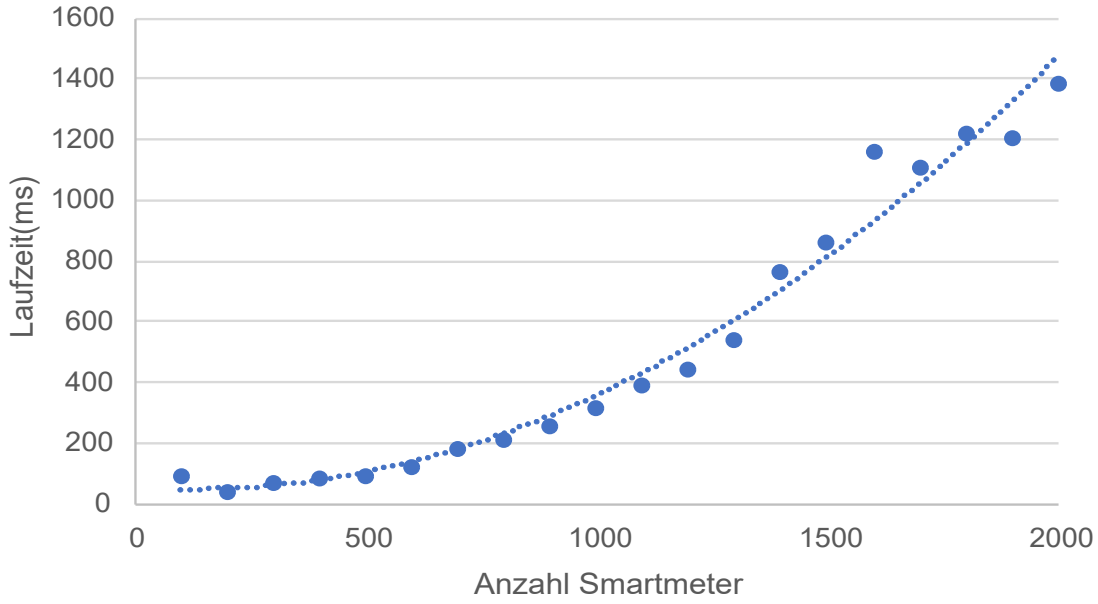

Abbildung 6.3: Laufzeitmessung der Angreiferanalyse

7 Zusammenfassung und Ausblick

Im Rahmen dieser Bachelorarbeit wurde die existierende Modellierung von Intelligenten Stromnetzen im Smart Grid Resilience Framework in Form der *SmartgridTopo* durch eine Transformation in eine Netzwerktopologie im Palladio Komponentenmodell übertragen. Die so generierte Netzwerktopologie lässt sich beliebig um neue Komponenten erweitern, um das Intelligente Stromnetz präziser beschreiben zu können. Außerdem wurden eine Angreiferanalyse auf diesem generierten PCM-Modell sowie eine Rücktransformation implementiert. Durch die Rücktransformation lassen sich Änderungen in der Topologie des generierten PCM-Modells wieder in ein SGM-Modell übertragen. Die dabei getroffenen Entwurfs- und Implementierungsentscheidungen wurden erläutert und die erstellten Werkzeuge hinsichtlich Genauigkeit und Skalierbarkeit evaluiert. Dabei wurden die gesetzten Ziele erreicht.

Der vorgestellte Ansatz zur Modellierung der Topologie eines Intelligenten Stromnetzes mit erweiterbaren Komponenten kann als Grundlage für weiterführende Arbeiten genutzt werden.

Ein Ansatz zur Verbesserung der Transformation wäre das Erarbeiten eines Standards für die Erweiterung der Komponenten, der nur noch in wenigen Fällen geändert werden muss, anstelle der manuellen Erweiterung des transformierten Modells. Dafür muss zunächst ermittelt werden, wie die Komponenten während der Transformation modelliert werden können. Danach kann die Transformation um die gefundenen Elemente erweitert werden.

Neben der Modellierung auf Netzwerktopologieebene kann das Modell noch auf Systemebene modelliert werden. Diese Erweiterung im *System Model* des generierten PCM-Modells erlaubt die Analyse der Auswirkungen von Angriffen auf die Softwarearchitektur. Aus dieser lassen sich Sicherheitseigenschaften der Komponenten bestimmen.

Eine weitere Möglichkeit zur Verbesserung des in dieser Arbeit erarbeiteten Ansatzes ist die Implementierung eines Angreifers auf Komponentenebene, der Ergebnisse für die Angreiferanalyse auf Topologieebene liefern kann. Ein geeigneter Ansatz dazu ist in Kapitel 3.1 mit der Sicherheitsanalyse für ein Palladio Komponentenmodell zu finden.

Literatur

[1] *Average frequency and duration of electric distribution outages vary by states*. https://www.eia.gov/todayinenergy/detail.php?id=35652. Aufgerufen: 26.09.2019.

[2] R. Brown. *Electric power distribution reliability*. CRC press, 2017.

[3] A. Busch, M. Strittmatter und A. Koziolek. "Assessing security to compare architecture alternatives of component-based systems". In: *2015 IEEE International Conference on Software Quality, Reliability and Security*. IEEE. 2015, S. 99–108.

[4] V. Caldiera und H. Rombach. "The goal question metric approach". In: *Encyclopedia of software engineering* (1994), S. 528–532.

[5] *Die neuen Stromzähler kommen*. https://www.verbraucherzentrale.de/wissen/umwelt-haushalt/wohnen/die-neuen-stromzaehler-kommen-13275. Aufgerufen: 21.09.2019.

[6] M. Ebada. *Smart Grid Resilience Framework*. KIT, SDQ.

[7] X. Fang u. a. "Smart Grid — The New and Improved Power Grid: A Survey". In: *IEEE Communications Surveys Tutorials* 14.4 (Fourth 2012), S. 944–980. ISSN: 1553-877X. DOI: 10.1109/SURV.2011.101911.00087.

[8] L. Hernandez u. a. "A multi-agent system architecture for smart grid management and forecasting of energy demand in virtual power plants". In: *IEEE Communications Magazine* 51.1 (Jan. 2013), S. 106–113. ISSN: 0163-6804. DOI: 10.1109/MCOM.2013.6400446.

[9] M. Hilbrich, M. Frank und S. Lehrig. "Security Modeling with Palladio—Different Approaches". In: *Proceedings of the Symposium on Software Performance 2016, 7–9 November 2016, Kiel, Germany*. 2016.

[10] E. Klappert. "Microgrid-Topologien für Smart Grids". KIT, SDQ. Bachelorarbeit.

[11] E. Lakervi und E. Holmes. *Electricity distribution network design*. 21. IET, 1995.

[12] S. Nolte. "Operational Mappings–die Sprache". In: *QVT-Operational Mappings*. Springer, 2010, S. 53–116.

[13] S. Ottenburger, T. Münzberg und M. Strittmatter. "Smart Grid Topologies Paving the Way for an Urban Resilient Continuity Management". Englisch. In: *International Journal of Information Systems for Crisis Response and Management* 9.4 (2017). 32.02.13; LK 01, S. 1–22. ISSN: 1937-9390, 1937-9420. DOI: 10.4018/IJISCRAM.2017100101.

[14] R. Reussner u. a. *Modeling and Simulating Software Architectures – The Palladio Approach*. Cambridge, MA: MIT Press, Okt. 2016. 408 S. ISBN: 9780262034760. URL: http://mitpress.mit.edu/books/modeling-and-simulating-software-architectures.

[15] R. Reussner u. a. "The Palladio Component Model". In: Karlsruhe reports in informatics (2011). URL: http://digbib.ubka.uni-karlsruhe.de/volltexte/1000022503.

[16] S. Soltan, P. Mittal und H. Poor. "BlackIoT: IoT botnet of high wattage devices can disrupt the power grid". In: *27th {USENIX} Security Symposium ({USENIX} Security 18)*. 2018, S. 15–32.

[17] E. Taspolatoglu und R. Heinrich. "Context-Based Architectural Security Analysis". In: *2016 13th Working IEEE/IFIP Conference on Software Architecture (WICSA)*. Apr. 2016, S. 281–282. DOI: 10.1109/WICSA.2016.55.

[18] *When the lights went out*. https://boozallen.com/content/dam/boozallen/documents/2016/09/ukraine-report-when-the-lights-went-out.pdf. Aufgerufen: 14.05.2019.

[19] A. Zaballos, A. Vallejo und J. M. Selga. "Heterogeneous communication architecture for the smart grid". In: *IEEE Network* 25.5 (Sep. 2011), S. 30–37. ISSN: 0890-8044. DOI: 10.1109/MNET.2011.6033033.

[20] G. Zhabelova und V. Vyatkin. "Multiagent Smart Grid Automation Architecture Based on IEC 61850/61499 Intelligent Logical Nodes". In: *IEEE Transactions on Industrial Electronics* 59.5 (Mai 2012), S. 2351–2362. ISSN: 0278-0046. DOI: 10.1109/TIE.2011.2167891.