Formal Specification framework for Hierarchical Microgrid community

Shravan Kumar Akula
School of Electrical Engineering and Computer Sciences
University of North Dakota
shravankumar.akula@und.edu

Hossein Salehfar
School of Electrical Engineering and Computer Sciences
University of North Dakota
h.salehfar@und.edu

Emanuel Grant
School of Electrical Engineering and Computer Sciences
University of North Dakota
emanuel.grant@und.edu

Abstract: Electrical microgrids are deemed to be the future of modern power systems. Microgrids are sophisticated, decentralized, and self-sufficient small-scale power systems consisting of various resources ranging from wind turbines, solar photovoltaics, electric vehicles, smart energy storage, and complex communication infrastructure. However, renewable energy sources such as solar photovoltaics and wind-based generators are highly intermittent, and if not appropriately planned, they can compromise the stability of the grid. Formal methods can define and analyze the functionality and behavior of any system and show if the system design is correct before the actual system is implemented. Although formal methods have been around for many years, it is surprising that little to none are utilized in the design of safety-critical electrical power systems. Currently, in modeling microgrids, few to no attempts of formalization are being used to improve the design reliability and reduce system operating costs and time. This work demonstrates how complex systems such as microgrids can be modeled elegantly using a formal specification method. In this work, the Z state-based formal specification language (Z-Method) is used to model and verify microgrid designs. In this work, 3-interconnected microgrid systems with a high penetration level of solar and wind-based renewable energy sources with plug-in hybrid electric vehicles (PHEV) as battery energy storage systems (BESS) are modeled using the Z-method. To the best of authors, the knowledge presented formal method is one of the first reported attempts in modeling microgrid communities using Software Engineering formalism.

Index Terms: Formal Specification, Microgrid, Battery Energy Storage Systems, Software Engineering, Plug-in Hybrid Electric Vehicle, Modeling, Renewable Energy, Safety-Critical systems, Microgrid Community

I. Introduction

The U.S. Department of Energy defines a Microgrid (MG) as ‘a group of interconnected loads and distributed energy resources within clearly defined electrical boundaries that act as a single controllable entity with respect to the grid’ [1]. The term MG exemplifies the usage of complex control systems, bidirectional communication, advanced power electronics, and more installed with the aging conventional power grid to convert into an advanced proactive and mercurial system. The focus of the MG is to integrate renewable energy sources into the system. With the dramatic decrease in the cost of disposition, the share of renewables and energy storage will continue to grow in the future, improving the reliability, flexibility of the system and will enhance the economics of the system. With advanced communication systems, MG’s can communicate and coordinate with each other, thereby increasing the overall reliability of the system and decreasing the total operational cost. Intermittent nature of the renewables can disrupt the conventional methods used for operating the utility grid, forcing the grid operators to adjust the real-time operating procedures. With minimal energy storage capacity available to maintain the stability of the grid, operators are forced to increase the reserve capacity to maintain the critical balance between demand and supply and to avoid the blackouts, which in turn will increase the operating cost. An effective way to decrease the effect of intermittency is to implement the theory from the law of large numbers. This probability theorem states that output from random events becomes closer to the expected value as a greater number of trials are performed. Similarly, the output from renewable energy becomes more predictable as the number of generators increases and they can be modeled effectively. To accommodate a large number of renewable energy generators MG’s are a perfect solution as it eliminates the necessity for redesigning the distribution system and is scalable while leaving a small footprint on the system in case of an outage.

The energy management system is the heart and soul of the MG as it monitors, controls, and optimizes the performance of the MG to improve the resiliency and efficiency. The authors of this work are working to develop a self-healing MG energy management system by combining complex optimization techniques and multi agent systems and initial results for a simple 3-MG system in [2] demonstrates the effectiveness of the proposed approach. Each component in the MG is dynamic in nature and the behavior of the MG is governed by the dynamics of its components. MG community can be compared to a Complex Adaptive System (CAS) because it can exhibit global change as a result of local action [3]. It is clear at this point that MG is a complex system with so much complication at hand there is a necessity to model each component of the system. With the help of different types of formalism, modeling the system in a sophisticated way is possible which will help to understand the system better. Formalism domain when applied MG's will decrease the complexity of the system thus making it easy to implement the system. Although formal specification methods are around for a while it is surprising to see that little to no went into implementing the formalism for the microgrid domain which is a safety-critical system.

A formal specification is a mathematical based technique in which a statement is expressed in formally defined syntax and vocabulary to define the properties of the system. Formal specification is not an actual implementation of the system, but it is used to develop an implementation of the system. It provides the mathematical analysis for creating a system and will help to detect flaws in the system before actual implementation thus improving the reliability of the system. Objectives of the formal methods are verification, validation and documentation and it has broad range of effects making the system error prone and mathematically precise. While Formal specification has many advantages, critics claim that it will increase the cost of the system, requires highly trained mathematicians, lack of user-friendly tools and limitations for implementation on large scale software's [4].

Formalism has been applied to many domains in the past some key examples include developing a formal specification for railway signaling systems with the help of the configuration data (Geographical data) in [5]. A case study for developing a system that monitors the operation of UAV's in US airspace with the help of formally defined requirements was proposed in [6] proves that formal specification techniques (FST) are suitable for health & status monitoring of safety critical systems. A model-based software engineering methodology that incorporates FST was applied for aircraft software systems in [7]. Formal agent-based framework for AIDS proposed in [8] uses a formal specification model in combination with agent-based simulation for developing a CAS model. Formal method for earthquake disaster mitigation and management system using Vienna Development Method-Specification language is proposed in [9]. Using UML language supported by specification methods a methodological framework was developed for mechatronic models for industrial control systems in [10]. Usage of a formal specification for the smart grid has been mentioned in the literature [11]. Formalized method-based state machine software in smart microgrid control systems is implemented in [12]. A formal specification framework for smart grid components using CAS is presented in [13].

The rest of the paper is organized as follows. FSM using Z-method in section II, section III discusses application of Z-method schema for a MG community. Finally, conclusions and future work are presented in section IV.

II. Formal Specification

A. Formal Specification Method:

Formal specification techniques (FST) have been in existence for decades it employs mathematical models for writing specifications, development, and verification of software and hardware systems. Using mathematical analysis for an engineering system will increase the robustness and reliability of the system. As mentioned previously, it reveals flaws and errors in the system before actual implementation, and if errors are found, modifications can be done, thereby saving valuable money and time. There are two fundamental approaches for implementing the formal specifications they are:

a. Algebraic Approach: In this approach, a system is described in terms of operations and their relations.
b. Model-Based Approach: In this approach, a model is built using mathematical constraints, and the system operations are defined by modifying the system state and looking for flaws in the system.

In this work, the authors have implemented the model-based approach.

For implementing any formal design, there are three steps involved:

a. Formal Specification: During this phase, the engineer will define the system using a modeling language (Z-Method in this case). This step converts a word problem into mathematical notation. Modeling a system using formal specifications is tough as modeling languages are fixed grammars.
b. Verification: Verification is the toughest of all the phases because even the simplest systems need to develop dozens of theorems, and they must be proven to emphasize the provability and correctness of the system.
c. Implementation: Once the system is defined and verified, then the specification is converted into code using different approaches.

While formal methods are around for a long time, Engineers don’t use them as the formal methods are extremely descriptive. But for large complex systems, some form of formalism is necessary.

B. Z-Specification Language

The Z-specification language is one of the popular and widely accepted formal language and is typically used for software and hardware systems. It is based on Zermelo-Fraenkel set theory and typed first-order predicate logic. By describing the states and the ways they can be changed, a system is modeled using the Z-method. Z-method is just a notation and not a method; it is an abstract formal specification. While Z is not a programming language, it is a model-based notation and uses a combination of boxes, Greek letters, and pictographic symbols.

To model a system in Z, it must be represented by its state, which is an assortment of state variables and its values. To model the complex physical systems such as MG community Z is a good fit as the models can be characterized by abstract data type (ADT). Mathematical objects are used to model the components of the proposed system as well as to model the data structures. A most distinguishable feature of Z, when compared to other methods is it uses schema which is a macro-like abbreviation and it uses smaller schemas to build bigger ones.

Z specification typically has multiple numbers of state schemas and operation schemas. Information about syntax and semantics are described in [14]. The flow chart for Z-Specification implementation is as follows.

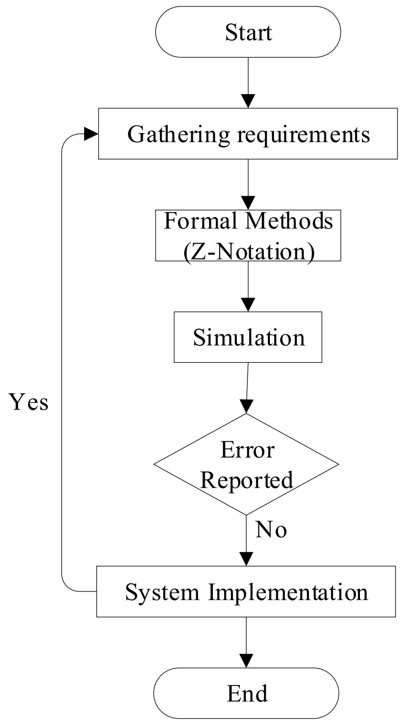

Fig 1. Flowchart of Z-Specification Language

The key difference between standard implementation and Z-schemas used in this work is that schema describes a type, but its description will not introduce any state to the system under implementation.

The Z formalization in the form of a vertical-form schema is as follows:

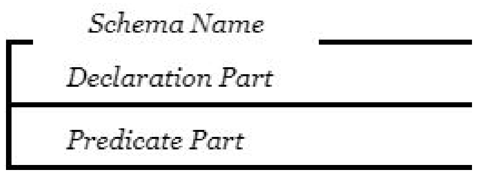

Figure 2. Z-Specification Schema

The above figure shows the generic way of the axiomatic definition of the Z schema. The schema introduces a universal constant in the declaration part, satisfying the predicate part. The set-in part *schema name* is a formal parameter and is the scope of the body of the definition. After the models are transformed into Z notation, they are analyzed by the Z word tools to test the syntax and semantics of the schema. Then the system is checked for errors if any faults are found in the model they can be rectified before actual implementation saving valuable time and money. It is impossible to find all the errors and faults in the system using formalism or any other method. However, it is still essential to test the systems with some manner of testing, especially if the system is a complex system like a microgrid community. The relevant conventions, declarations, sets, relations, functions and schema definitions are discussed in [15].

II. Formal Specification framework for a Microgrid Community

In this section, the formal specification framework is applied to the microgrid community and its components. The MG states considered in this work are islanded, Grid-tied, and Transition mode (Transition between both mode0-s). Components considered in this work are Solar Panel, Wind turbines, Battery Energy storage system, Plug-in Electric vehicle, and Load. A simple MG community with all the components looks as in fig 3. To maintain the voltage/frequency stability in the MG energy storage system within acceptable ranges and to improve the power quality and to increase the dynamic response of the system towards disturbances, BESS is installed. Research states that EV's sit idle in parking lots about 70%of the time, and the big battery packs with high energy and power densities can act as energy storage systems which will eliminate the high capital investment for conventional Li-ion storage systems.

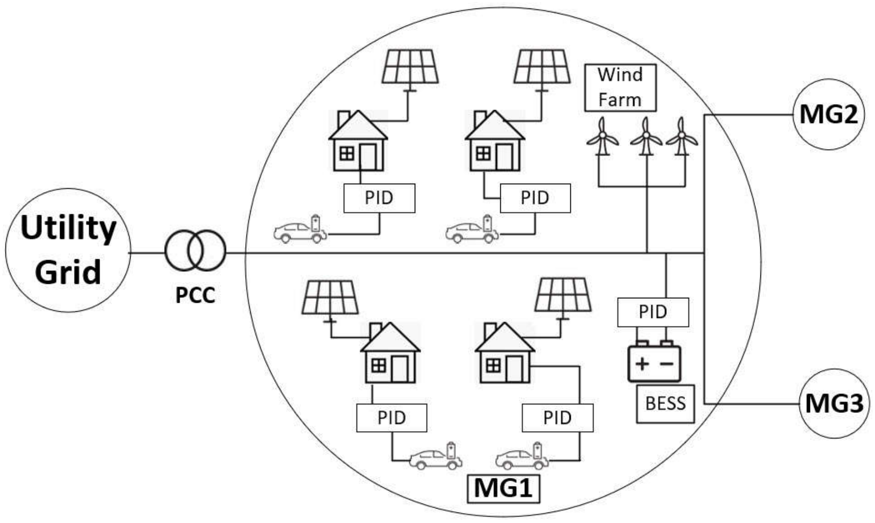

Figure 3. 3-MG community with RES, PHEV, and BESS

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

A. Microgrid community mode of operation

A microgrid can operate in two modes grid-tied where it is connected to the utility grid, Islanded mode, where MG is operating as independent entity, in this work authors, are considering a third mode, i.e., transition mode (transition between grid-tied and island mode). Although this mode lasts for a concise period (in order of milliseconds to microseconds), it is still vital as it will introduce several transients into the system. Each mode and component have different states and events that cause state transitions.

In Grid-Tied or Grid-connected mode, MG is connected to the utility grid at the point of common coupling (PCC) and exchanges power with it depending on generation, demand, and electricity price. When the load in MG is greater than a generation from distributed generators (DG) it buys power from the utility grid or neighboring MG's. During excess generation, it injects the power back into the utility grid or sells the power to neighboring MG's. Grid-connected MG's earn revenue by selling the ancillary services to the utility grid. Some of the well-known services are demand response, voltage and frequency regulation, and black start capability.

Following a grid failure or schedule MG changes its operational mode to islanded mode where the DG's, BESS, and loads operate as an independent entity. When the MG enters an islanded mode, the energy management system (EMS) is responsible for voltage control and power-sharing between critical and non-critical loads and balancing within the MG. In the islanded mode of operation, there is no voltage and frequency support from the utility grid, and MG, which is a power electronic-based grid, lacks the inertia, and there must be other units to support the voltage and frequency fluctuations.

To model the MG mode of operation using Z-Specification, a free type *"MICROGRID"* is used to represent different states of the MG that are Grid-Tied, Islanded, and Transition.

[MICROGRID] = {GridTied, Islanded, Transition}

After the free type, a schema named *MG* is defined that contains "MGState" variable of type "MICROGRID". The value of the MGState can be GridTied, Islanded or Transition.

```
_MG___________________________________________
| MGState: MICROGRID
|______________________________________________
```

After defining the MG mode of operation and MG schema next step is to define the operational schemas. By presenting the initialization schema named *InitMG* and schema, *MG* is declared as a variable. In the predicate section of the schema upon initialization state of the MG is equal to *GridTied*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

InitMG

MGCondition

MGState = *GridTied*

An operational schema *FaultCondition* is defined to show the stressed conditions in the MG, state of the MG changes to *Islanded,* and change of the state is shown by *ΔMG.* In the predicate section, state changes from grid-tied to island mode.

FaultCondition

ΔMG

MGState = *GridTied*
MGState′ = *Islanded*

After the fault clears, MG goes back into grid-connected mode to model that an operational schema named *NormalCondition* is declared. In the predicate section, state of the microgrid changes from Islanded mode to grid-tied mode.

NormalCondition

ΔMG

MGState = *Islanded*
MGState′ = *GridTied*

For the efficient operation of the microgrid, a control strategy is required for smooth microgrid's state transition from islanded to grid-tied and vice versa. To model that an operational schema named *TransitionMode* is defined. In the predicate, the initial state is either grid-tied or islanded mode, and the final state is Islanded or grid-tied depending on the initial mode of operation.

TransitionMode

ΔMG

MGState: *GridTied* ∪ *Islanded*
MGState′: *Islanded*

Similarly,

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

TransitionMode

ΔMG

$MGState: GridTied \cup Islanded$

$MGState': GridTied$

The schemas for the MG and state changes are as shown in figure 4.

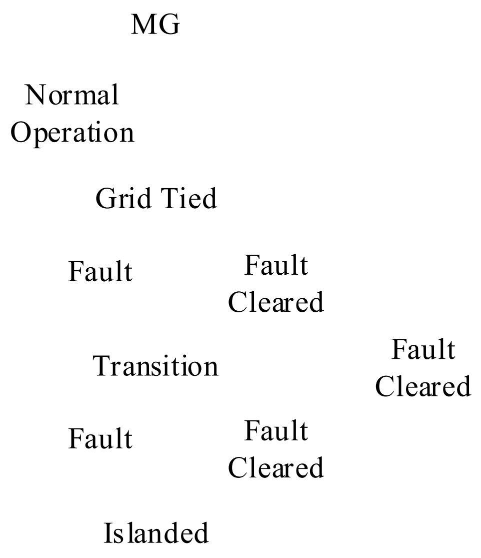

Figure 4. MG state changes

B. Power Transfer

For the MG to be reliable and profitable energy exchange with the neighboring MG's and utility grid is considered in this work. Different states and events involved in the energy transfer between MG's and with utility grid are shown in Figure 5. Whenever MG needs to buy power, it communicates with neighboring MG's and utility grid to check for the electricity prices, and it starts buying power from the lowest price provider. When there is a nominal price or no demand, there is no power exchange. In this work, power transfer has only been viewed from a price perspective, power transfer by demand response (DR) together will be included in future work.

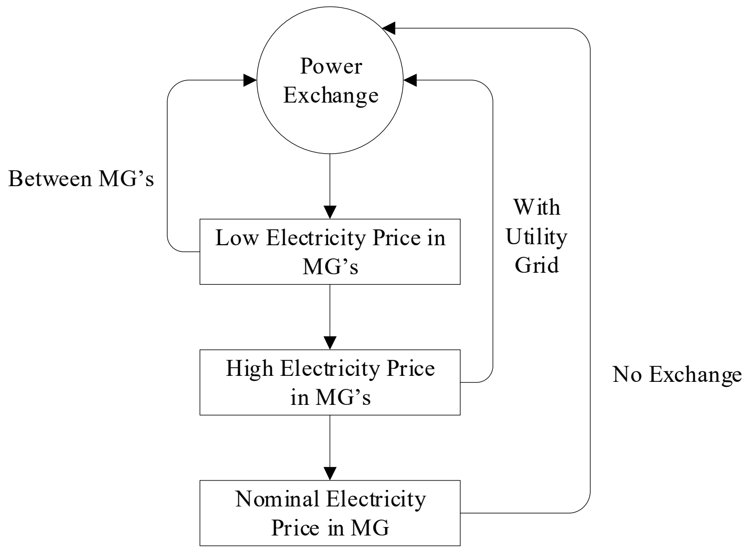
Power Exchange
Between MG's
Low Electricity Price in MG's
High Electricity Price in MG's
Nominal Electricity Price in MG
With Utility Grid
No Exchange

Figure 5. Power Exchange Schema

A free type *EnergyTransfer* is used to define the different states of the energy transfer.

[ET] == {BetweenMG's, WithUtilityGrid, NoTransfer}

After defining the ETState, schema is initialized with variable *InitET*. Initial state in the schema is *NoTransfer*.

```
_EnergyTransfer_________________________________
| ETState: ET
|________________________________________________
```

After defining the ETState schema is initialized with variable *InitET*. Initial state in the proposed schema is *NoTransfer*.

```
_InitET_________________________________________
| EnergyTransfer
|_____________________________________________
| ETState:Notransfer
|________________________________________________
```

If the neighboring MG's offer a better price, then energy is transferred with in the MG community. so, the initial state here is *NoTransfer* and the final state is *BetweenMG's* in the predicate and the transition is shown by defining *LowElectricityPriceinMG's* schema.

```
_LowElectricityPriceinMG's______________________
| ΔEnergyTransfer
|_____________________________________________
| ETState : NoTransfer
| ETState' : BetweenMG's
|
|________________________________________________
```

If the electricity price is higher in MG community than the utility grid, then the power exchange is done between MG and utility grid. To model the transition schema *HighElectricityPriceinMG's* is declared and in the predicate initial state is *BetweenMG's* and the final state is *WithUtilityGrid*

```
_HighElectricityPriceinMG's_______________________
| ΔEnergyTransfer
|__________________________________________
| ETState : BetweenMG's
| ETState' : WithUtilityGrid
|_______________________________________________
```

If the price of electricity is nominal, then there is no power transfer, final state in this schema is *NoTransfer* with initial state as either *BetweenMG's or WithUtilityGrid* in the predicate section and the transition is represented by defining the schema *NominalElectricityPrice.*

```
_NominalElectricityPrice__________________________
| ΔEnergyTransfer
|__________________________________________
| ETState : BetweenMG's ∪ WithUtilityGrid
| ETState': NoTransfer
|_______________________________________________
```

C. Battery Energy Storage System

Due to the intermittent nature of RES, voltage, and frequency fluctuations occur in the power grid, and BESS is a perfect solution to answer that problem. Although PHEV's are considered as BESS in this work, the problem is that they are mobile and not plugged in all the time, so a BESS is installed at the point of common coupling. 100 MW/ 129 MWh Hornsdale Power Reserve in Australia is the world's largest Li-ion battery; it helped in stabilizing the grid, avoid outages and reduces system costs, and surged fast response systems across the continent and saved $40 million [15]. That proves the effectiveness of the BESS in the system. The schema for BESS is like that of PHEV's.

BESS can provide ancillary services like frequency regulation, voltage support, black start, load shifting, and reserve provision. So, a schema is provided for ancillary services using BESS. Battery energy storage systems (BESS) are rechargeable batteries that store energy from renewable energy sources in the microgrid when there is an excess generation for utilization later when the demand is high or high electricity price. To smooth out intermittency from the renewable sources and to assist the grid by providing services like peak shaving, black start, voltage control, and frequency regulation energy storage system is a perfect choice. Energy storage is of many types Flywheel, hydroelectric dams, superconducting magnetic energy, thermal storage, compressed air, and grid oriented large-scale battery technologies (Li-ion batteries).

For modeling the battery energy storage system state of charge (SoC) is used, it is defined as the percentage of the level of the charge left in the battery relative to its capacity. It is a measure of the rated capacity of the battery rather than the current capacity of the battery. As the battery nears the end of its lifetime actual capacity of the battery deteriorates and can only charge up to 80% of its rated capacity when fully charged so, SoC is a good measure for estimation when taking battery aging and depth of discharge into consideration. If the SoC is high, BESS can discharge to provide intended services, and when the SoC is low, it switches to idle mode, and when there is excess generation BESS gets charged and when it's fully charged it goes back into idle mode.

To present the formal specification for a BESS, a free type is defined by the name *BESSState* that comprises the different states of the BESS.

[BESSState] == {Charging, Discharging, Idle}

To define BESS schema, a variable is declared by name *ESSState* of type *BESSState* that has any single value from the above-defined state set.

_*BESS*____

BESSState: *ESS*

By introducing the initialization schema named *InitBESS,* this schema calls the *BESS* as a variable and in the predicate to show the stage of the storage *Idle* is used at the initial time.

_*InitBESS*____

BESS

BESSState = Idle

When an energy storage event occurs, BESS goes into the *Charging* state. This process is shown by the *LowSoC* schema. Change of the state is shown by delta (Δ) sign, and in predicate current state is either *Idle* or *Discharging,* which is changed to *Charging*.

_*LowSoC*____

Δ*BESS*

BESSState = Idle ∪ *Discharging*
BESSState′ = Charging

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

When the BESS charges and reaches High SoC it changes it’s state from *Charging* it enters *Idle* or *Discharging* depending on the grid conditions. To model that a schema named *HighSoC* is declared and in the predicate section state change is declared.

HighSoC

$\Delta BESS$

$BESSState = Charging$

$BESSState' = Idle \cup Discharging$

After clearing the fault in the grid or after fully charging BESS enters idle state to model that *IdleState* schema is declared and in the predicate section the state changes from either *charging* to *discharging* to *Idle*

IdleState

$\Delta BESS$

$BESSState = Charging \cup Discharging$

$BESSState' = Idle$

As mentioned previously, BESS can also provide ancillary services, so to model the ancillary services using Z follow schema has been written. Frequency must be maintained within limits, and when violated, it can impose hefty fines on the MG owner. The authors of this work have proposed a proportional integral derivative (PID) controller with the design based on neural networks to control the frequency deviations in the MG community. To obtain the proper PID parameters in [17], they have used a multilayer feedforward neural network with random numbers as inputs, and the results prove the effectiveness of the method. To model the frequency regulation, a transition *FrequencyRegulation* is defined. Then a change in the state was declared with a delta sign, and in the predicate section, the initial state will be *Idle* or *Charging,* which then changes to *Discharging.*

FrequencyRegulation

$\Delta BESS$

$BESSState: Idle\ U\ Charging$

$BESSState': Discharging$

When the voltage event occurs, BESS goes into discharging state to show this transition, a schema called *VoltageSupport* is called. In the predicate section, the state of the BESS changes from either idle or charging to Discharging.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

VoltageSupport

Δ*BESS*

BESSState:*Idle U Charging*

BESSState':*Discharging*

Black start is the process of restoring power to an MG or a part of MG without relying on the utility grid. To model that a transition *BlackStart* schema is declared, with the initial state as *Idle* or *Charging* in the predicate section, and the final state is *Discharging.*

BlackStart

Δ*BESS*

BESSState:*Idle U Charging*

BESSState':*Discharging*

Reserve provision allows conventional thermal generators to go into offline mode so that renewables can supply the demand during the peak generation hours of solar and wind. To model reserve provision using formal specifications, a transition schema named *ReserveProvision* is introduced, and in the predicate, the initial state is *Idle,* and the final state is *Discharging.*

ReserveProvision

Δ*BESS*

BESSState:*Idle U Charging*

BESSState':*Discharging*

D. Plug-in Hybrid Electric Vehicle

PHEV's are penetrating the market every year, and if not planned properly, they can have adverse effects on the power grid. According to research, electric vehicles spend only 10% of the time on the roads. They are essentially big battery packs sitting idle for 90% of the time. If planned properly, they can act as a battery energy storage system and help in mitigating the voltage and frequency fluctuations in the MG and earning income for the owners.

There are three states, namely "Charging, Discharging and Idle" for the PHEV. First, we start by defining free type *PHEV* that comprises the different states of the EV.

[EV] == {Charging, Discharging, Idle}

For initializing schema, a variable *PHEVState* is declared that has a single value *PHEVSTATE*.

PHEV

PHEVState:*EV*

Defining the operational schema by introducing the initialization schema called *InitPHEV*. By calling the *PHEV* as a variable and *Idle* as predicate at the initial time.

InitPHEV

PHEV

PHEVState: *Idle*

When price of electricity is low, and BESS is at medium or low SoC then it starts charging to model this condition a *LowElectricityPrice* schema is declared and changing in PHEV is shown by delta sign. In predicate the current state is *Idle*, and the final state is *Charging*.

LowElectricityPrice

Δ*PHEV*

PHEVState: *Idle*

PHEVState': *Charging*

If the generation is low or demand is high utilities following spot pricing will raise the electricity price. So, when the price of electricity is high BESS starts discharging bringing down the electricity price. A schema named *HighElectricityPrice* is declared and in the predicate section state of the BESS changes from *Charging* to *Discharging*

HighElectricityPrice

Δ*PHEV*

PHEVState: *Charging*

PHEVState': *Discharging*

During a fault condition MG enters the islanded mode, to supply the critical loads BESS will start discharging to model this condition. This process is defined by declaring a schema named *FaultCondition* and in the predicate section state of the PHEV changes from either *Charging* or *Idle* to *Discharging.*

FaultCondition

Δ*PHEV*

PHEVState: *Charging U Idle*

PHEVState': *Discharging*

After the fault is cleared BESS is in low SoC condition and should start charging to model this a schema named *AfterFault* is declared and in the predicate section state of the PHEV changes from *Discharging* to *Charging*

AfterFault

$\Delta PHEV$

PHEVState: *Discharging*
PHEVState': *Charging*

Like BESS, PHEV follows the charging and discharging conditions based on the SoC. This part of the schema looks like that of the PHEV. When SoC is low, an event named *LowSoC* occurs and in the predicate state changes from *Idle* to *Charging.*

LowSoC

$\Delta PHEV$

PHEVState: *Idle*
PHEVState': *Charging*

After charging PHEV reaches high SoC and stops charging. A schema named *HighSoC* is declared with *Charging* as initial state and *Idle* as final state in the predicate

HighSoC

$\Delta PHEV$

PHEVState: *Charging*
PHEVState': *Idle*

To enter idle state after a fault is cleared or charging schema *IdleState* is declared. In the predicate section initial state is either *Charging* or *Discharging* and final state is *Idle*

IdleState

$\Delta PHEV$

PHEVState: $Charging \cup Discharging$
PHEVState' : *Idle*

PHEV with distinct states and events is shown in figure 6.

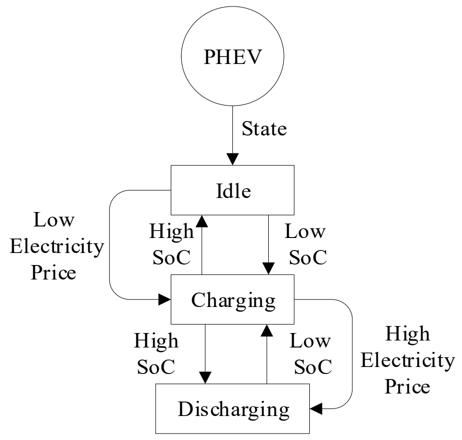

Figure 6. PHEV states and events

E. Solar Cell

Renewables considered in this work are the Solar Panels and Wind Turbine. In [13], a simple framework is presented for solar panel and wind generation based on the type of day, i.e., Sunny, Cloudy, and Night, and the states are NoGeneration, Partial Generation, and Full Generation. But a solar panel generation depends on many other factors like the type of solar cell, temperature, orientation, irradiance, Dirt, and Snow. These factors are included in this work to make the system realistic.

A free type *PVCell* is defined for presenting different states, which comprise of states *FullGeneration*, *PartialGeneration,* and *LowGeneration*.

[PVCell] == {LowGeneration, NominalGeneration, HighGeneration}

Then a PV system schema is initialized. Schema takes a variable *PVState* of type *PVCell* is introduced, which takes the value mentioned above.

```
_SolarPanel______________________________________
| PVState: PVCell
|_________________________________________________
```

The next step is declaring operational schema by initializing *InitPV* schema. Initially, a schema *SolarPanel* is introduced in the declaration part, and in the predicate, *LowGeneration* is considered as an initial state.

```
_InitSolar_______________________________________
| SolarPanel
|____________________________________________
| PVState: LowGeneration
|_________________________________________________
```

Temperature affects the generation of solar panels, as excessive heat can reduce the output of a PV system. As the temperature of the solar panel increases, output current increases exponentially while the voltage decreases linearly. Most of the solar cells are tested at a temperature of 25°C. For modeling the solar panel characteristics based on the temperature in this work, Panasonic HIT 330W N-Type 96 is considered, and its product specifications are documented in [18]. For raise in every 1°C, solar panel efficiency is decreased by 0.0258% and vice versa.

For a temperature equal to 25°C, the solar panel is operating normally to model that a schema called *TempEqualTo25C* is declared, and in the predicate, the state change is indicated by Δ*SolarPanel* in with initial state *LowGeneration* while the final state is *PartialGeneration*.

_*TempEqualTo25C*________________________
Δ*SolarPanel*
PVState= *LowGeneration*
PVState'= *NominalGeneration*

When the temperatures are below 25°C, solar panels will generate more electricity. By declaring schema *TempBelow25C* and in the predicate section with initial state *NominalGeneration* and final state *FullGeneration* schema is as follows.

_*TempBelow25C*________________________
Δ*SolarPanel*
PVState= *NominalGeneration*
PVState'= *HighGeneration*

Whenever temperatures are above 25°C, solar panels efficiency decreases, which means less power is generated to model that schema named *TempAbove25C* is introduced with initial state *HighGeneration* and final state *LowGeneration* in the predicate.

_*TempAbove25C*________________________
Δ*SolarPanel*
PVState= *HighGeneration*
PVState'= *LowGeneration*

Dust and Snow can affect the generation of solar cell studies that are performed to estimate the losses taking these factors into consideration. So, it is important to include them in the model. The type of material used in the manufacturing process of solar panels can determine the efficiency of the panel. Traditionally they are made out of silicon as they are more efficient and long-lasting. There are currently three types of silicon-based panels used they are Monocrystalline (24.2% Efficiency), Polycrystalline (19.3% Efficiency), and Amorphous silicon cell (10% Efficiency). There is also the fourth kind of solar cell named Hybrid Silicon Cell with an efficiency of 33% per unit area but is ignored in this work as it is still work in progress. When modeled using Z-Method schema is as follows.

The polycrystalline solar cell has a moderate efficiency to model that a schema named *Polycrystalline* is introduced and in predicate initial state is *LowGeneration,* and the final state is *NominalGeneration*.

PolyCrystalline

ΔSolarPanel

PVState=LowGeneration
PVState′= NominalGeneration

Monocrystalline cells are the most efficient among all the cells, so they generate more power to model them. A schema *Monocrystalline* is declared with *LowGeneration* as initial state and *HighGeneration* as the final state in the predicate

Monocrystalline

ΔSolarPanel

PVState= LowGeneration
PVState′= HighGeneration

Amorphous cells are the least efficient, so they generate less electricity per unit area than any other cell. A schema named *Amorphous* is declared with the initial state as *NominalGeneration* and final state *LowGeneration* in the predicate.

Amorphous

ΔSolarPanel

PVState= NominalGeneration
PVState′= LowGeneration

Solar panels generate maximum power when the surface is perpendicular to the sun. The position of the sun is plotted using two angles, namely Azimuth (Angle of the sun as it moves from East to West) and Zenith (Angle of the sun measured between ground and horizon). It is important for solar panels to be mounted at the correct angle for them to point them at the sky directly. Solar panel orientation refers to the azimuth setting; it is ignored in this work because the solar panel is assumed to be stationary on the vertical axis. Solar panel tilt refers to a zenith setting and is key for the efficient operation of a solar cell. Authors of this work are located in Grand Forks, ND and from [19] ideal solar tilt is calculated, and the values are as follows for optimal year-round generation tilt angle is 42°, for best performance in winter tilt angle is 27° and in summer best tilt angle is 57°.

To model the tilt angle, a schema names *TiltAngleAbove42Deg* is introduced, and in the predicate section initial state is *LowGeneration,* and the final state is *NominalGeneration*. This schema is for angles above 42° where solar panels are operating at moderate efficiency.

_*TiltAngleAbove*42*Deg*______________________________

Δ*SolarPanel*

PVState= *LowGeneration*
PVState′ = *NominalGeneration*

For tilt angles equal to 42 degrees, solar panels are operating at maximum efficiency to model that a schema *TitleAngle24Deg* is declared with initial state *NominalGeneration* and final state *HighGeneration* in the predicate.

_*TiltAngle*42*Deg*______________________________

Δ*SolarPanel*

PVState=*NominalGeneration*
PVState′= *HighGeneration*

To model the solar panel with a tilt angle less than 42 degrees, a schema *TiltAngleBelow42Deg* is initialized, and in predicate initial state is *HighGeneration*, and the final state is *LowGeneration*. In this work, all the tilt angles are taken for Grand Forks Location, and they vary from place to place.

_*TiltAngelBelow*42*Deg*______________________________

Δ*SolarPanel*

PVState= *HighGeneration*
PVState′ = *LowGeneration*

Solar irradiance is the power generated at a particular location, and it varies depending on the season. It is measured in kWh/m^{2,} and for Grand Forks location for a Flat lying solar panel average solar insolation values calculated from [19] are as follows: Jan-1.36, Feb-2.30, Mar-3.38, Apr-4.68, May-5.66, Jun-5.99, Jul-6.05, Aug-5.22, Sep-3.76, Oct-2.49, Nov-1.58, and Dec-1.17.

During Jan-Mar generation is Nominal to model that a schema named *JanToMar* is declared and in predicate initial state is *LowGeneration* and final state is *NominalGeneration* in the predicate

JanToMar

$\Delta SolarPanel$

$PVState = LowGeneration$

$PVState' = NominalGeneration$

During the summer solar panels generate maximum power. For Apr-Aug transition is implemented by decalring schema named *AprToAug*, with initial state *NominalGeneration* and final state *HighGeneration* in the predicate.

AprToAug

$\Delta SolarPanel$

$PVState = NominalGeneration$

$PVState' = HighGeneration$

During the spring and winter solar panels have a low generation to include them as a part of the model and to demonstrate the transition a schema named *SepToDec* is declared, and in the predicate section initial state is *HighGeneration,* and the final state is *LowGeneration*

SepToDec

$\Delta SolarPanel$

$PVState = NominalGeneration \cup HighGeneration$

$PVState' = LowGeneration$

Accumulation of dust, snow, and water can distract the light from reaching the surface of the solar panels and can affect the generation of the solar panel, in extreme cases, generation was reduced by up to 85%. A slightly dirt-covered solar panel can't generate electricity at full capacity, and the transition is shown by declaring with a schema *SlightlyDirty* with initial state *LowGeneration* and *NominalGeneration* as the final state in the predicate, and the schema is as follows

SlightlyDirty

$\Delta SolarPanel$

$PVState = LowGeneration$

$PVState' = NominalGeneration$

A clean solar panel generates power at full capacity, and to model the transition, a schema named *Clean* and in predicate initial state is *NominalGeneration,* and the final state is *HighGeneration*.

```
_Clean________________________________________
| ΔSolarPanel
|_____________________________________
| PVState= NominalGeneration
| PVstate' = HighGeneration
|______________________________________________
```

If a solar panel is covered completely in the dirt it produces little to no electricity, which is highly undesirable. To represent the and to model it, a schema named *CompletelyCoveredWithDust* is declared while in the predicate section initial state is *NominalGeneration,* while the final state is *LowGeneration*.

```
_CompletelyCoveredWithDirt____________________
| ΔSolarPanel
|_____________________________________
| PVState= NominalGeneration
| PVState' = LowGeneration
|______________________________________________
```

Solar panel states, and the events that trigger the state changes are shown in figure 7.

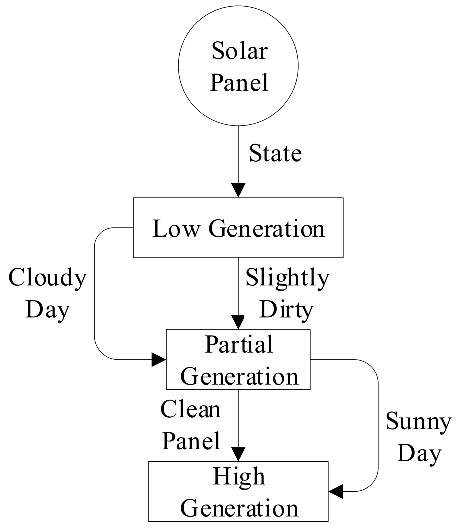

Figure 7. Solar panel cell states and events

F. Wind Turbines

A simple framework for wind turbines based on wind speed is presented in [13]. In this work taking temperature into consideration, a formal framework is presented. When temperature drops from +20°C to

-20°C turbine output will increase by 16%. The power generated by a wind turbine in terms of density of air and area swept out by turbines is given by.

$$P = \frac{1}{2} * Density\ of\ Air * Area\ Swept\ by\ Turbines * (Windspeed)^3$$

The air is denser when it is cold and vice versa. As air density is dependent on the temperature, it is important to model the temperature using formalism. To present the formal specification for the wind turbine, a free type *WINDGEN* is defined, and the set comprises different states of wind turbines, namely *LowOutput*, *NominalOutput,* and *HighOutput*.

[WindGen] == {LowOutput, NominalOutput, HighOutput}

A wind turbine schema is developed by declaring *WindGenerator* and a schema consisting of variable *WTState* of type *WINDGEN*. The value of this variable is presented in the above set.

```
_WindGenerator______________________________________
| WTState: WindGen
|___________________________________________________
```

Schema is initialized by presenting *InitWT* and calling the *WindGenerator,* and in the predicate initial state is *LowOutput*

```
_InitWind___________________________________________
| WindGenerator
|______________________________________________
| WTState: LowOutput
|___________________________________________________
```

At room temperatures, wind turbines produce nominal output. Transition is represented by declaring *RoomTemperature* as the schema, and the change of state is indicated by *ΔWindGenerator* in the predicate initial state is *LowOutput,* and the final state is *NominalOutput*.

```
_RoomTemperature____________________________________
| ΔWindGenerator
|______________________________________________
| WTState= LowOutput
| WTState′= NominalOutput
|___________________________________________________
```

Low temperatures improve the performance of the wind turbine and can ramp up the power production to capture the transition a schema named *LowTemperature* is defined, and in the predicate, the initial state is *NominalOutput* while the final state is *HighOutput*.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 30 June 2022 doi:10.20944/preprints202206.0431.v1

_*LowTemperature*_______________________________________

Δ*WindGenerator*

WTState= NominalOutput
WTState′= HighOutput

To model the high temperatures schema named *HighTemperature* is declared to show the transition, and in the predicate, the initial state is either *NominalOutput* and the final state is *LowOutput*. This schema presents the output for a wind turbine at high temperatures at which their efficiency is low.

_*HighTemperatures*_______________________________________

Δ*SolarPanel*

PVState= Nominal ∪ *HighOutput*
PVState′= LowOutput

Figure 8. demonstrate the states, events that trigger the change if states of a wind turbine.

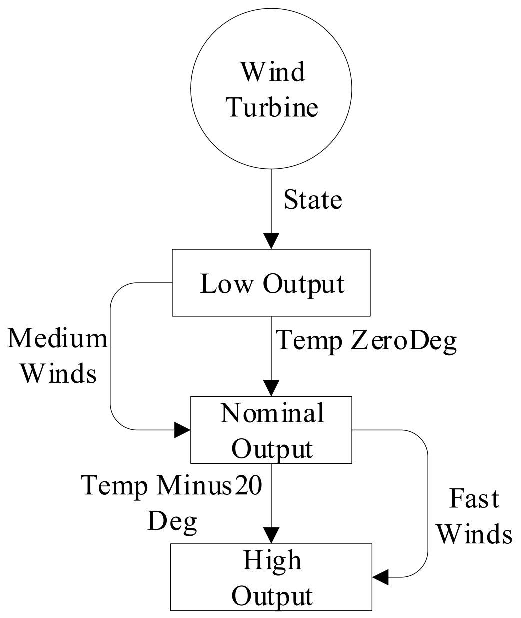

Figure 8. Wind turbine states and events

Conclusion and Future work

In this work formal specification framework for MG components is developed. This work proves that complex power systems like MG community can be modeled using software engineering methods like Formal specifications. this paper only includes part of the MG and can be expanded to include more components. future work includes other key factors that can influence the reliability of the MG and also using other formalism approaches like petrinets and B-method.

References

[1] Dan T. Ton and Merrill A. Smith, The U.S. Department of Energy's Microgrid Initiative, The Electricity Journal, Volume 25, Issue 8, 2012, Pages 84-94, https://doi.org/10.1016/j.tej.2012.09.013.

[2] S. K. Akula and H. Salehfar, "Energy Management System for Interconnected Microgrids using Alternating Direction Method of Multipliers (ADMM)," 2018 North American Power Symposium (NAPS), Fargo, ND, 2018, pp. 1-6. doi: 10.1109/NAPS.2018.8600647.

[3] S. K. Akula and H. Salehfar, "Frequency Control in Microgrid Communities Using Neural Networks," *2019 North American Power Symposium (NAPS)*, Wichita, KS, USA, 2019, pp. 1-6. doi: 10.1109/NAPS46351.2019.9000219

[4] S. M. Amin and B. F. Wollenberg, "Toward a smart grid: power delivery for the 21st century," IEEE power and energy magazine, vol. 3, no. 5, pp. 34–41, 2005.

[5] Hall, A.: Seven myths of formal methods, Software, IEEE , vol.7, no.5, 11--19, (1990).

[6] Lars-Henrik Eriksson, "Use of Thoeries in Applied Formal Methods", http://user.it.uu.se/~lhe/publications/domaintheory.pdf.

[7] Clachar, S. Grant, E.S.: A Case Study in Formalizing UML Software Models of Safety Critical Systems. In Proceedings of the Annual International Conference on Software Engineering. Phuket, Thailand (2010).

[8] A. Siddiqa and M. Niazi, "A novel formal agent-based simulation modeling framework of an aids complex adaptive system," International Journal of Agent Technologies and Systems, vol. 5, no. 3, pp. 33–53, -2013.

[9] N. A. Zafar and H. Afzaal, "Formal model of earthquake disaster mitigation and management system," Complex Adaptive Systems Modeling, vol. 5, no. 1, p. 10, Sep 2017. [Online]. Available: https://doi.org/10.1186/s40294-017-0049-8.

[10] M. Bonfe and C. Fantuzzi, "Design and verification of mechatronic object-oriented models for industrial control systems, " in Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference, vol. 2, sept. 2003, pp. 253 - 260 vol.2.

[11] G. Hackenberg, M. Irlbeck, V. Koutsoumpas, and D. Bytschkow, "Applying formal software engineering techniques to smart grids," in Software Engineering for the Smart Grid (SE4SG), 20.

[12] G. K. Turner, A formalized method for state machine software implementation in smart microgrid control systems. The University of Texas at Arlington, 2014.

[13] W. Akram and M. A. Niazi, "A formal specification framework for smart grid components," Complex Adaptive Systems Modeling, vol. 6, no. 1, p. 5, Sep 2018.

[14] Spivey J (1989) The Z notation: a reference manual. Prentice-Hall, Englewood Cliffs.

[15] d'INVERNO, M. 1998. Agents, Agency and Autonomy: A Formal Computational Model. Ph.D. Dissertation, University of London

[16] https://na.panasonic.com/us/energy-solutions/solar/solar-panels/n330-photovoltaic-module-hitr

[17] S. K. Akula and H. Salehfar, "Frequency Control in Microgrid Communities Using Neural Networks," 2019 North American Power Symposium (NAPS), Wichita, KS, USA, 2019, pp. 1-6.

[18] https://na.panasonic.com/us/energy-solutions/solar/solar-panels/n330-photovoltaic-module-hitr

[19] http://www.solarelectricityhandbook.com/solar-irradiance.html