January 2020

Application Of Formal Specification Technique To Microgrid Representation

Maksym Tkach

APPLICATION OF FORMAL SPECIFICATION TECHNIQUE TO MICROGRID REPRESENTATION

by

Maksym Vasylevich Tkach
Bachelor of Science, University of North Dakota, 2012

A Thesis

Submitted to the Graduate Faculty

of the

University of North Dakota

in partial fulfillment of the requirements

for the degree of

Master of Science

Grand Forks, North Dakota

December
2020

This thesis, submitted by Maksym Tkach in partial fulfillment of the requirements for the Degree of Master of Science from the University of North Dakota, has been read by the Faculty Advisory Committee under whom the work has been done and is hereby approved.

Emanuel Grant

Hossein Salehfar

Ronald Marsh

This thesis is being submitted by the appointed advisory committee as having met all of the requirements of the School of Graduate Studies at the University of North Dakota and is hereby approved.

Chris Nelson
Dean of the School of Graduate Studies

December 3, 2020

PERMISSION

Title	Application of Formal Specification Technique to Microgrid Representation
Department	Computer Science
Degree	Master of Science

In presenting this thesis in partial fulfillment of the requirements for a graduate degree from the University of North Dakota, I agree that the library of this University shall make it freely available for inspection. I further agree that permission for extensive copying for scholarly purposes may be granted by the professor who supervised my thesis work or, in his absence, by the Chairperson of the department or the dean of the School of Graduate Studies. It is understood that any copying or publication or other use of this thesis or part thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of North Dakota in any scholarly use which may be made of any material in my thesis.

Maksym Tkach
August 23, 2020

ABSTRACT

This thesis uses formal specification techniques to analyze and model a microgrid. A microgrid is a small, local electrical grid, often supplied by a single generator, that can connect to the larger electrical grid, but can also disconnect from it, going into "island mode." Thanks to the growth in renewable energy, microgrids represent a growing segment of the electrical power generation domain. And like any member of the domain they are safety-critical systems, meaning that even a small mistake in their implementation risks damage to life and property.

Formal specification is a way to abrogate the risks of safety critical systems by ensuring that the system under consideration is fully described, modeled, and analyzed prior to implementation, and the description and model are robust and error-free. However, at present there is no established approach to the use of formal specification techniques of microgrid systems. This thesis proposes a specification that can serve as a foundation for future work in the microgrid domain as well as an aid to communication about microgrids. The work uses Unified Modeling Language (UML) graphical notation and an accompanying Object Constraint Language (OCL) formal specification. The model transformation accomplished through the use of Iterative Development techniques is outlined in detail to serve as a guide to future researchers.

I. INTRODUCTION

1.1 Problem Definition:

A microgrid is a localized interconnected group of power consumers (otherwise known as loads) and power producers that can operate in grid-connected or islanded mode. In grid-connected mode a microgrid is connected to a larger electrical grid (sometimes referred to as a host grid) and can exchange power with it as needed. In islanded mode the microgrid is cut off from the larger grid and handles its own power needs [1] [2].

Microgrids are made up of decentralized, modular systems known as Distributed Energy Resources (DERs). DERs are small-scale power generation sources located close to their loads that can service loads individually or have their generated power aggregated to serve the grid as a whole [3]. DERs include renewable energy sources such as photovoltaic cells and wind turbines [2] as well as small non-renewable generators, typically powered by diesel or gas [1]. The definition of DERs also includes localized power storage, which typically takes the form of chemical batteries, but can also take the form of pumped hydro [4], electrical vehicles that double as power storage [5], and Superconducting Magnetic Energy Storage [6].

Microgrids have become increasingly prevalent in the past several years [7]. This is in part due to the increased popularity and effectiveness of renewable energy sources, including rooftop solar panels and wind microturbines, and in part due to the advancements in computing, networking, and communication that allow a distributed grid to be controlled in a flexible and decentralized way [7][8]. There has been a corresponding effort to develop and improve microgrid control schemes. However, there is currently no agreed-upon standard for modeling these schemes, and efforts frequently resort to an ad-hoc approach. To help establish a shared understanding of

microgrid characteristics and facilitate communication between researchers, it's necessary to create a formal specification of a microgrid.

A formal specification is an expression of a system's properties made in a formal language. A formal language is one whose grammar, vocabulary, and syntax follow a set of clear and unambiguous rules. Statements made in a formal language have a single unambiguous meaning, as opposed to statements in a natural language, which may be imprecise or open to misinterpretation [9]. Therefore, a formal specification can be treated as a mathematical entity and have its correctness and self-consistency verified via mathematical methods, including automated methods. A formal specification is not a full implementation but is instead a statement of a system's requirements [9] [10].

The main advantage of a formal model over an informal one is that a formal model is verifiable, meaning that it can be checked for completeness, correctness, and consistency. To be complete, a model has to encompass all relevant concerns about the system. To be correct, a model has to fulfill the requirements it lays out, particularly by making sure that any input given to the system will produce the desired output. To be consistent, a model has to avoid contradicting itself – it should be possible to fulfill all the requirements it lists at the same time, but there should not be multiple ways of doing so [9][10][11].

1.2 Significance of the Work:

A microgrid is a safety-critical system, meaning that any malfunction can result in serious harm to people or property. At the same time microgrids have properties that make ensuring perfect functioning more challenging. Microgrids are heterogeneous, meaning that a designer has to account for different types of structures and components. Microgrids are also changeable, meaning they can have components added or subtracted after the system's initial creation. Taken

together, these properties mean that when a microgrid is designed and built, every measure must be taken to minimize error. This makes a formal model highly desirable, because a formal model can be checked for errors and inconsistencies at design time, preventing costly adjustments or costlier failures later in the process [11].

Formal models can also be used to communicate information about the microgrid with more precision than natural language or ad-hoc models, but without the complexity of a full implementation. Communicating precisely and thoroughly about technical subjects is difficult. As seen in the Methodology section, this issue came up in the course of this project. In the initial exchange of information important details were omitted or left unclear, and the use of formal modeling techniques helped ensure all relevant information was transmitted completely and correctly.

The goal of the project was to create a formal model that can serve as a foundation for future formal microgrid models. Therefore, the model had to have two characteristics. First, it had to rigorously define the basic components of a microgrid in a thorough and error-free manner. Second, it had to be extensible – easy to adapt for specific implementations with different components, structures, and goals. In accomplishing that, the formal specification in this thesis can become an important milestone toward future research.

1.3 Methodology:

My interest in microgrids began during my survey on Multi Agent Systems (outlined in more detail in the Background section). Microgrids came up as one of several systems that benefited from decentralized control and I focused on that subject, researching the Multi Agent approach to microgrid and multi-microgrid control. After observing the different ways microgrids were presented in the papers, I found that none of the models were truly formal and most were made

with a specific implementation in mind. For reasons outlined in the previous section, I decided to pursue the creation of a formal model.

When the background reading phase of my research was finished, I moved onto the practical task of modeling a single microgrid. In accordance with the principles of formal modeling, my first task was requirements elicitation – talking to an engineering team that works with microgrids to find out what aspects of a microgrid were considered important by them. Then I followed a process of iterative development, creating and refining a model with feedback at every step of the way. As is usual during requirements elicitation, the first pass-through did not capture the domain accurately. Attributes deemed important in informal discussion would turn out not to matter as the model became more defined, and on the other hand the process of formalization revealed variables that needed to be included in the model but were initially missed. This difficulty in communicating microgrid-related concepts between professionals with knowledge of the subject is further proof of the importance of a shared model.

1.4 Scope of the Work:

The scope of the model is deliberately limited to a single microgrid. I modeled a relatively small microgrid consisting of an arbitrary number of wind turbines and solar cells connected to an arbitrary number of loads. The components of the microgrid along with all of their potentially important attributes were outlined in a Universal Modeling Language class diagram and defined in a separate document. The desired behavior of the components was described by a set of rules and modeled by a flowchart diagram. The whole of the system was then formally modeled using the Object Constraint Language in the UML-based Specification Environment. Included in the thesis is a description of how each part of the model was constructed and of my application of the iterative development process and how it helped refine the model.

1.5 Expected Results:

My research was expected to define the structure of the microgrid in a formal, unambiguous way and provide the initial set of constraints – mostly to outline scenarios that are either physically impossible or universally dangerous. I also expected to catalogue and describe useful attributes and operation, and to provide an example control scheme. The end goal was a proof of concept and a template for others to build on.

1.6 Thesis Layout:

The rest of the paper is as follows: Section II is a description of the problem domain. Section III is the literature review. Section IV is a description of the methodology used for the research and modeling. Section V describes the results of applying the methodology. Section VI deals with the conclusions and potential future work.

II. BACKGROUND

2.1 Problem Domain:

2.1.1 Microgrids:

As outlined in the Introduction section, a microgrid is a small collection of loads and Distributed Energy Resources that can function in grid-connected or islanded mode. Microgrids are becoming an increasingly important part of the energy generation infrastructure because they allow for local control of distributed energy generation, increasing the grid's overall flexibility [12]. This decentralized approach makes the larger grid more robust by avoiding a single point of failure, since, in case of a breakdown in the host grid, microgrids are able to continue functioning independently and can be called on to help stabilize the host grid [5]. Microgrids can help boost the efficiency of the power generating process by adjusting for local conditions in a way a centrally controlled grid cannot [12]. Microgrids are also attractive from a financial standpoint and are typically installed as a cost-saving measure [13].

Microgrids present special challengers for planners. In a classical grid the flow of power is unidirectional – it's generated at a central location and distributed to consumers. Microgrids, however, feature bi-directional power flow, meaning that depending on the circumstances they may accept power from the host grid or return power to the host grid. That means the structure of a multi-microgrid is necessarily more complex than that of the classical grid [5].

Microgrids are heterogeneous, meaning that they are made up of different components, and the exact components of any two microgrids can differ significantly. Microgrids vary in size, with the smallest being the size of a single household and the largest the size of a university campus. Microgrids vary in their power generation mix, particularly the degree to which they incorporate fossil fuel based power. The earliest microgrids were primarily fossil fuel based, but modern

microgrids are predominantly based on renewable energy [12]. A typical microgrid is owned by a single stakeholder, but there are real world examples of microgrids that have multiple stakeholders with competing interests [14].

Microgrids also vary in terms of their control logic, particularly the degree of centralization within the microgrid. Microgrid control schemes can be broadly broken down into centralized (meaning that the microgrid as a whole is controlled by a single entity), hierarchical (meaning that a single entity exerts overall control but entities "below" it make some independent decisions), and distributed (meaning that there is no central control and individual entities make all the decisions through consensus) [15].

Microgrids are variable – a microgrid can choose to disconnect from the main grid, can have its component parts changed, and is likely to experience fluctuation in its ability to supply power due to small size and a tendency to rely on environment-dependent power sources [5].

Microgrids are autonomous, meaning that they make their own decisions on behalf of their individual owners instead of taking orders from a central authority. The goals microgrids pursue vary greatly, but common priorities include the following: maximization of stability within the microgrid [2]; tracking the physical flow of the electricity within the microgrid [3]; setting up an internal market within the microgrid [14]; performing a simulation of a microgrid's activity at different times of day [16].

While the model described in this thesis is meant can be adapted to different microgrids, it was created with a typical residential microgrid in mind. This means the following assumptions were made: a single stakeholder maintains ownership of all electricity generators and loads within the microgrid, there are no competing interests within the microgrid itself, and all components will serve the microgrid owner's goals cooperatively; the microgrid contains only renewable energy

sources; the microgrid uses chemical batteries for power storage; the microgrid has ready access to the host grid and is unlikely to be forced into islanded mode; the microgrid can be of any size and can have components added and subtracted as necessary.

2.1.2 Multi-Agent Systems:

As outlined above, microgrids are distributed, modular, and complex systems. These characteristics make centralized control inefficient and prone to failure [15]. To handle microgrids and multi-microgrids, a control system must be able to deal with a distributed system that gives individual entities local control while coordinating collective action in a way that avoids a single point of failure and provides needed flexibility. One way of handling these challenges is a Multi-Agent System (MAS) [2].

A Multi-Agent System is a collection of interacting, autonomous entities which work in dynamic and uncertain environments to accomplish some goal. The agents are defined by three characteristics:

Autonomy, meaning that each agent has the ability to act independently, free from external intervention. Agents have their own goals, their own rules, and their own awareness of the environment independent of central control [2].

Situatedness/Locality, meaning that each agent is able to receive local data from its environment, but does *not* have a global view of the system. It can also refer to an agent's ability to modify its immediate environment [2].

Flexibility, which means each agent has the ability to react to its environment in a timely manner, take initiative to achieve its goals, and interact with other agents and humans [2].

The basic principle of MAS is that decisions are made at the lowest possible level. This approach prevents the computational complexity of a system from growing exponentially, speeds

up system's response time, and prevents communication bloat [11]. The use of MAS makes systems more robust because it means there isn't a single component whose failure can cause the whole system to fail. In modular systems MAS makes inserting components into the system or subtracting them from the system simpler [11]. In systems composed of agents that have differing priorities or that are competing for the same resources, MAS provides coordination, allowing the goals of the overall system to be achieved [2]. These characteristics have led MAS to see widespread use in microgrid control systems.

While the use of agents in the structure of a microgrid can take many forms, this thesis makes several assumptions. First, I chose to use the two types of agents described by Dimeas, et al [17]: cognitive and reactive agents. Unlike reactive agents, cognitive agents are agents capable of advanced communication that possess memory. I made the assumption that each cognitive agent possesses an internal database whereas a reactive agent does not and that cognitive agents are capable of making sophisticated decisions and holding negotiations with each other while reactive agents are capable of performing specific tasks in an uncertain and dynamic environment.

Second, I assumed agents follow the principle of hardware encapsulation – which is to say that agents overlap individual pieces of hardware to the greatest degree possible. This reflects the aforementioned principle of locality, ensuring that most agents don't need to know the overall state of the system and enjoy low latency. It is assumed that each piece of equipment is overseen by a cognitive agent.

Third, I assumed a two-tier hierarchical architecture with a central Microgrid Control Agent that oversees all communication between different parts of the microgrid, receives inputs from all of them, and is ultimately responsible for load balancing (making sure that electrical power is

available to the users on request). This appears to be the most common implementation within microgrid-related papers, and covers multiple use-cases.

2.1.3 Formal Modeling:

A formal model is a precise description of the system being designed, including components, relationship between components, and behavior. Formal modeling creates mathematically rigorous descriptions of the system that can be validated with replicable results [9].

One function of formal modeling is to increase *precision*. By removing ambiguity and allowing automated validation, formal models remove errors and other unintended behaviors from the process. This increases the safety and robustness of the system being designed. By identifying problems at design time, formal modeling prevents them from appearing at later stages, when they're more costly to resolve [18].

The other function of formal modeling is to increase *discipline*. This means that the process of introducing formality forces a reevaluation of the system. It tells the designer whether the requirements have been met. If they have not, the model may need to be adjusted – or, if the result produced is reasonable, the requirements may need to be adjusted instead. Formal modeling serves as a way to reason about the system and to guide further development of the system [18].

Formal specification can be used to define functional and non-functional requirements. Functional requirements deal directly with the system's behavior; in most basic terms, functional requirements specify what a system should do. Functional requirements define the rules the system must use to transform inputs into outputs, and therefore specify the system's command logic.

Non-functional requirements deal with the users' expectations of a system and define acceptable and unacceptable states for the system to be in. In basic terms, non-functional

requirements specify how a system should behave. One way of expressing non-functional requirements is to make use of quality attributes, which are a set of desirable properties of a system used to indicate which of these properties are most important to the system being considered. The following was the result of describing the non-functional properties of a microgrid in terms of quality attributes:

Similar to other power generation and distribution systems, microgrids are required to prioritize Safety and Security, because of the high risk posed by a malfunctioning electrical component. The microgrid's task of load balancing requires high Availability, to make sure consumers have access to power when they need it. A microgrid must be Scalable, because microgrids frequently have components added to or subtracted from them. It must be Resilient, able to easily deal with hardware and software faults, to prevent interruptions in the service [2] [19] [20].

In this thesis, formal specification was created by supplementing UML diagrams with an OCL specification, as outlined below.

2.2 Solution Domain:

2.2.1 Unified Modeling Language (UML):

Unified Modeling Language is a graphical modeling language developed to standardize the visual depiction of a software intensive system. Developed in mid 1990s to combat standards proliferation, it was officially adopted by the Object Management Group and has since become the industry standard for graphical modeling notation. UML is used to visualize systems, specify requirements, provide a blueprint for construction, and document decisions throughout the development process [21].

UML is semantically rich, using a small set of notational elements to model a broad range of systems in a variety of problem domains. This also allows UML to provide complementary, interlocking models of a system, allowing different aspects to be highlighted based on a developer's current needs. UML also has the advantage of being comprehensible, allowing its diagrams to be understood without prior training, which makes it an important tool for communicating with clients [21].

UML is an Object Oriented programming language and works best on systems made up of self-contained components. This approach has the advantage of making systems more modular, allowing them to be changed or expanded and of allowing for encapsulation, which makes each component more secure by protecting its data from unnecessary contact [21].

UML can be used to create several types of diagrams, depending on what aspect of the system needs to be highlighted. In this research a class diagram shown in Appendix 1 was used to visualize the structure of the microgrid. Classes in a class diagram, also known as objects, represent the major entities within the system. The class diagram also shows the relationships between these objects. At the next step, following the requirements elicitation process, each object was assigned a set of attributes – properties important to its functioning. Finally, during the work with the Object Constraint Language outlined below, each object had its associated operations listed, outlining the actions it can perform. Notably, while operations are listed within the class diagram, they are not fully described within UML. That description is done in Object Constraint Language, as seen in the next section. Figure 1 shows an example object with all associated attributes and operations.

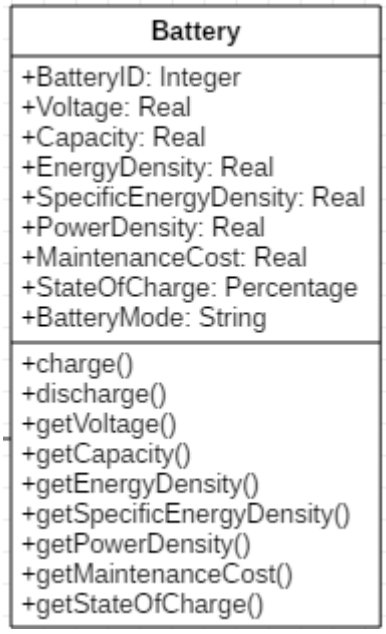


Figure 1: UML Object with associated attributes and operations

In addition, a flowchart diagram was used to help visualize the microgrid's behavior, demonstrating the microgrid's control flow. Figure 2 shows a sample flowchart diagram.

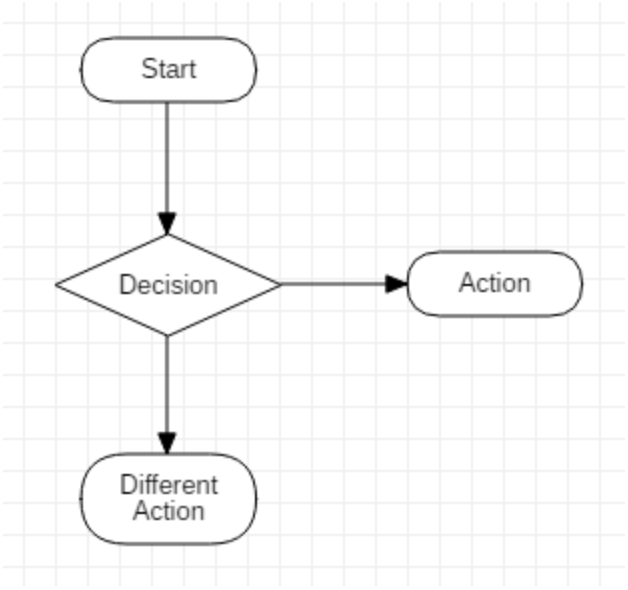


Figure 2: Sample Flowchart

2.2.2 Object Constraint Language (OCL):

Object Constraint Language (OCL) is a formal declarative language originally created to supplement UML. Whereas a UML diagram can be used to show the basic structure of some

system (in this thesis a microgrid), OCL can help define that system in a rigorous way. An OCL file takes the form of a collection of formal expressions which are easy to understand and less complex than a true programming language, but lack the ambiguity of natural language and can be checked for correctness and lack of contradictions [22].

OCL expressions are used to supplement UML in two significant ways. First, OCL expressions are used to create constraints. Constraints are restrictions on attributes that must always hold true. They can be used to limit the value of an attribute to a range that's safe and useful and outline relationships between objects within the system. Constraints are used to model the Non-Functional Requirements, as explained in Section 2.1.3.

```
context Generator
inv:
Cost > 0
inv:
GeneratorOutput >= 0
inv:
Voltage >= 118 and Voltage <= 122
GeneratorKind = 'Solar' or GeneratorKind = 'Wind'
```

Figure 3: OCL Constraints

The OCL fragment seen in Figure 3 shows the constraints on the Generator class. The constraints establish that neither the cost of operating the generator nor its output can be negative (ruling out physically impossible scenarios), that the voltage associated with a generator cannot be lower than 118 volts or higher than 122 volts (ruling out unsafe scenarios), and that all generators must be either solar panels or wind turbines (ruling out scenarios that don't fit client preferences).

Second, OCL expressions are used to define operations and therefore specify the expected behavior of the system. Through the use of OCL a designer can specify the ways objects within the system gain information about themselves, their environment, and each other, the way the system makes decisions, and the expected effect of those decisions. Operations are used to model Functional Requirements, as explained in Section 2.1.3.

```
context Microgrid::changeOperationalMode()
post changePost:
if OperationalMode = 'Islanded'
then OperationalMode = 'GridConnected'
else OperationalMode = 'Islanded'
endif
```

Figure 4: OCL Operation

The OCL fragment in Figure 4 defines the operation that changes the microgrid's operational mode, either connecting to the main grid to enter Grid Connected mode or disconnecting to enter Islanded mode. Because it's a specification and not a full implementation, the model doesn't go into detail on how the connection/disconnection is to be achieved, only establishing that it should exist.

In this thesis OCL was used to ensure the safety of the microgrid by specifying the allowable range of key attributes, including frequency and voltage. OCL was also used to define the system's behavior, including the query operations through which attribute values are checked by appropriate objects, and the non-query operations through which values and states are changed.

2.2.3 UML-based Specification Environment (USE)

The USE tool is used for model transformation and validation of UML/OCL models. The core functionality of USE is to be an interpreter for UML/OCL, allowing developers to analyze model structure and behavior, check for errors, model scenarios, and make quick changes to the model [23].

USE notation supplements OCL by translating UML into textual form. As an example, the fragment of USE notation seen in Figure 5, defines the Solar Panel object present in the UML diagram, establishes that it is a subclass of the Generator class, and lists its attributes, along with an associated data type for each attribute. In this form UML diagrams can be checked for completeness, correctness, and consistency, eliminating ambiguity and forcing all objects and

relationships to conform to proper forms. Combined with traditional OCL notation, this allows for automatic error checking and scope checking of constraints and invariants [23].

```
class SolarPanel < Generator
attributes
        Tilt: Real
        InverterEfficiency: Real
        SolarIrradiance: Real
```

Figure 5: USE object definition

USE specifications can also be used to directly generate UML diagrams, as seen in Figure 6. This aids in the process of iterative development, as described in section 2.2.4, by allowing the developer to easily switch between the OCL/USE specification the UML model in order to make corrections and updates [23].

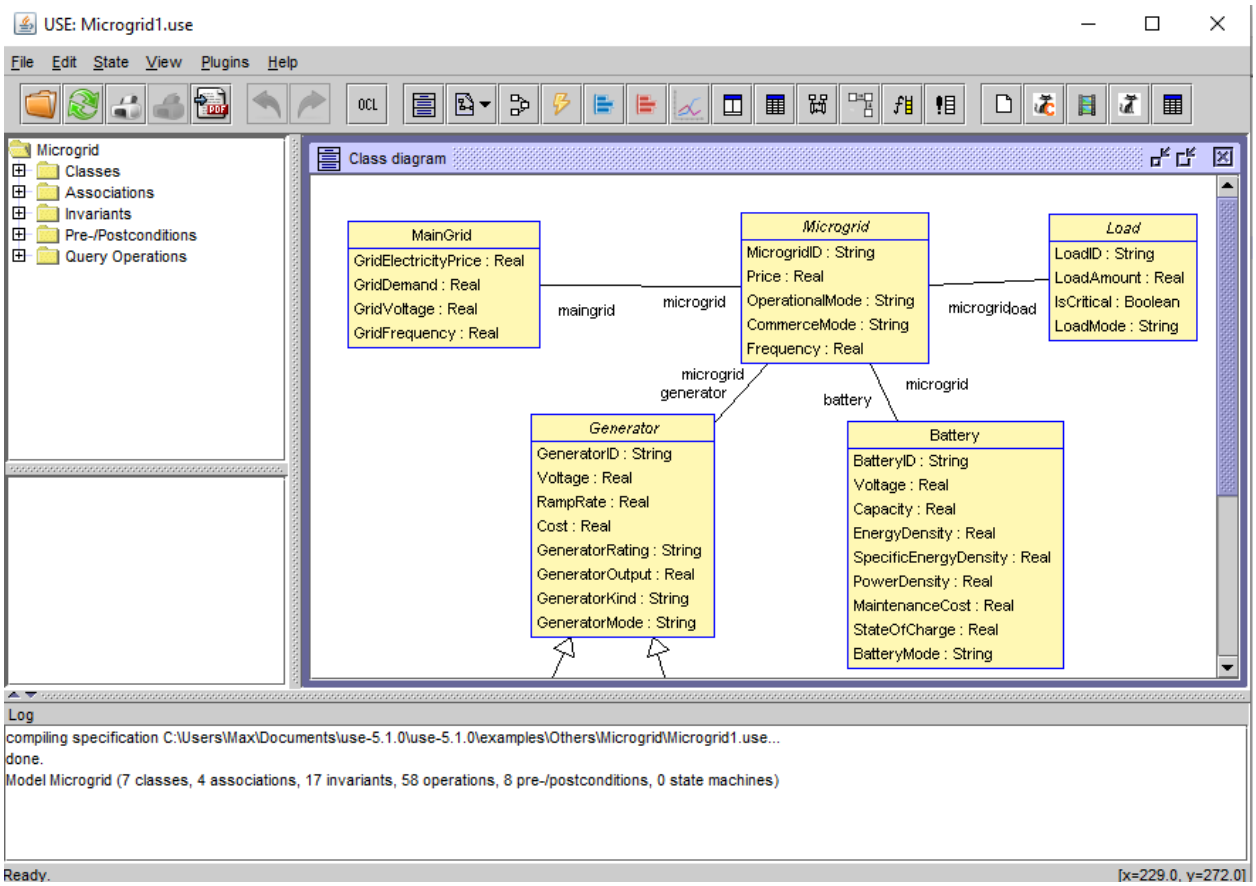


Figure 6: USE in action

In my research USE was an invaluable error-checking tool. It allowed me to ensure the correctness of the OCL syntax and the model behavior, and by extension to modify the model to be able to carry out the needed operations.

2.2.4 Iterative Development:

Iterative Development is a design philosophy that treats software development as a cyclical process. Instead of completing as much of each development phase as possible before moving on to the next, the developer is encouraged to complete a partial, prototypical model in the early phases. That model is then updated with information gained in subsequent phases and models created in those phases are then modified based on that model [24]. This philosophy was followed in this research. The process of developing the UML diagram and its OCL specification was fully bi-directional. While a UML diagram was developed first, the creation of the OCL specification supplemented by USE revealed a number of flaws that caused the UML diagram to be changed. In turn, when the UML diagram was presented to the engineering team with accompanying explanations, new insights were gathered, requiring further changes to be made to both UML and OCL. This process repeated itself several times, and this iterative development resulted in the latest model – still not exhaustive, but a more complete description of a microgrid than the initial attempt.

As an example we look the electrical load – a portion of an electrical system that consumes electrical power (See Section 5.1 for more information and Appendix A for definitions of terms). During the initial requirements elicitation, it was decided that the load would be an attribute of the `Microgrid` class, representing the total demand for electricity in the microgrid. This was its role in the initial UML diagram. However, during OCL modeling, the question of what, if any, actions should be associated with the load came up. A new round of requirements elicitation showed that

it was desirable that some loads should be able to temporarily shut down to conserve power. At this point the `Load` class was created, with each instance of the class representing a separate electricity-using object. To indicate which loads should have the option to shut down and which should keep going no matter what, the concept of critical and non-critical loads was introduced, requiring changes to the UML diagram and the OCL specification. Initially, `Critical Load` and `Non-Critical Load` were introduced as subclasses of `Load`, to make them easier to view in the UML diagram. But during the specification of the operations associated with shutting down non-critical loads, it was established that control flow worked better if the criticality of the load was an attribute instead, so another change was made, again requiring changes in UML and OCL. These changes showcase the Iterative Development process.

III. Literature Review

3.1) Active Power Management in Multiple Microgrids Using a Multi-Agent System with JADE [1]: This paper deals with the problem of maintaining and verifying an active power balance within the microgrid, which is the process of ensuring grid stability by balancing power generation against load demand. The authors model and simulate a multi-microgrid consisting of three microgrids connected to a main grid. Each microgrid is composed of photovoltaic cells, a diesel generator, a battery for power storage, and some loads. Each microgrid also has six inputs to keep track of: Photovoltaic Power, Battery Power, Diesel Generator Power, Load Active Power, State of Charge of the Battery, and a Static Switch between Grid-connected and Isolated modes.

At the core of the control algorithm are the two priority queues. The power supply priority: photovoltaic system, battery system, diesel generator, the grid. And the power delivery priority: local load, battery charging, grid transaction. These priorities are chosen to maximize stability, financial, and environmental advantages.

This paper features a specific microgrid model that considers only the attributes that directly impact its control algorithm. It's more narrowly focused than my own work, but useful as a way to verify the results of my requirements elicitation. The priority queues closely match my own implementation, as outlined in Section 5. Likewise, the paper's implementation of a Multi-Agent System uses the same hardware encapsulation assumptions present in my model, with agents matching specific pieces of hardware. While this work is less formal and less general than my model, the aforementioned commonalities serve to confirm the assumptions I made when setting up the basic structure and the functions.

My work deals with creating an extensible formal model that can be used to formalize models like the one presented in this paper. This paper's model relies on a set of specific parameters and the control algorithm that causes states to change as specific breakpoints. Its formal description could be created from my model by narrowing down the list of used parameters and inserting the breakpoints as invariants. In effect, I'm exploring a more general case than the one presented in this paper.

3.2) An Autonomous Agent for Reliable Operation of Power Market and Systems Including Microgrids [14]: This paper discusses the microgrid as a market in which Generator Agents and Load Agents place bids which are then matched by a Control Agent. This is used in the paper to minimize the price of electricity production, especially if there is more generator capacity than demand from loads. It's also potentially useful for achieving other goals, like minimizing environmental impact or strain on the equipment. It has potential uses if the microgrid has multiple stakeholders who have reason to compete.

In the paper's model a Microgrid Control Agent interacts with Generator Agents and Load Agents at fifteen minute intervals, collecting projected demand from each load and available generation and initial price from each generator. The Microgrid Control Agent creates a priority list based on these inputs and pairs up generators and loads. If total generating capacity isn't enough to meet demand, the Microgrid Control Agent interacts with an outside Grid Agent to buy power. If there's excess, it interacts to sell power.

Currently my model is focused more on the physical flow of power. While money and the sale of electricity are part of it, they are not the primary concern. This paper demonstrates a path through which my model could be altered to focus on the electricity marketplace, perhaps by a future adopter doing follow-up work.

The basic control algorithm described in this paper appears effective at ensuring that demand is met and the costs are minimized whenever possible. It is, however, light on details. Price is always discussed in natural language, and the mechanisms of interaction between agents aren't explicitly outlined. This isn't necessarily a negative in a paper of this length, but any follow-up work on the subject may benefit from formalism, as it would enable the authors to communicate with more precision.

This paper demonstrates another potential application for my model. While it's not suitable to track complex market interactions in its current form, it could be adapted to do so, in which case it would be

3.3) Decentralised coordinated control of microgrid based on multi-agent system [2]: This paper proposes and implements a control scheme based on *coordinated switching*, a strategy for increasing the security and stability of a system by ensuring that state changes take place in an optimized and non-disruptive way.

The control scheme is implemented for a specific type of microgrid – one that includes a wind turbine, a photovoltaic cell, a fuel cell, and a battery. This setup requires the individual microgrid to ramp the operation of its various components up and down depending on environmental conditions, performance, and load, all in a way that doesn't damage the system and optimizes outcomes for the grid's owner.

The switching is governed by "security indexes" which describe the safe operating limits of the system, including voltage, power balance, component capability limit, and mode switching duration limits. When the security indexes are violated, the system has to immediately switch to a more optimum mode of operation to prevent a possible fault. It's desirable to prevent too many switches, since rapidly switching between modes of operation can damage components.

This paper uses a different type of formalism than the one used in my research. It uses Coloured Petri Nets – a kind of graph that can rigorously define a sequence of actions and conditions under which these actions must be taken. It's a better choice for this paper's subject than my approach would be because one of OCL's weaknesses is the lack of native ability to handle timing. Declaring that two or more events must happen simultaneously is essentially impossible in unmodified OCL, but possible through the use of Coloured Petri Nets.

3.4) Multi-Agent based Microgrid Coordinated Control [25]: This paper describes an unusual microgrid: one consisting of a gas turbine, solar and wind power sources, and a hydrogen fuel cell. The hydrogen fuel cell is meant to remain idle as much as possible and serves as a backup power source. The gas turbine handles most of the normal demand due to its stability. Because solar and wind are particularly volatile, their energy is used mostly to run an elecrolyzer which creates hydrogen for the fuel cell. In this way the combination of the renewable energy sources and the fuel cell serves as a balancing mechanism to ensure the stability of the power supply.

The paper goes on to explain the nuances of its approach to the control scheme. It begins by outlining the importance of controlling frequency. The authors choose to maintain frequency stability by ensuring that the demand and supply are balanced, which is done using a set of constraints. The model uses a Multi Agent System that mostly follows the principle of hardware encapsulation, except that it also includes a Database Agent, used to carry out coordinated dispatch. The control strategy is based on constant exchange of parameter values between all agents.

Though this paper doesn't use formal declarative statements, its notation closely approaches formalism, especially because its most important function – load balancing, and by extension fault

prevention – is defined by invariants. It's useful as an example of an invariant-based control strategy, and a way to handle frequency and its stability.

This paper is narrower in scope than my own model, but it's a good example of how an application of formalism can help to better define a model and of how a formal model could be adapted to describe a specific type of microgrid.

3.5) Research on multi-agent decision-making model of wind-solar complementary power generation system [8]: This paper proposes a decision-making model for a microgrid containing only solar and wind energy sources based on a Multi Agent System. The paper begins by proposing a model consisting of nine modules and two categories of agents. The first four modules are: data collecting and processing, control, info management, and data resource management. Notably, these do *not* follow the hardware encapsulation principle, but instead the *task encapsulation* principle – meaning that agents that perform similar jobs are grouped together instead. This group of modules is responsible for gathering and remembering data, analyzing it, and making decisions. The other five modules are: wind energy power generation, solar energy power generation, inverter, storage battery, and load. The paper classifies them as "field-level agent modules" and they're responsible for executing the decisions made by the higher-level modules.

Next, the paper goes outlines the parameters of its model. Unlike other models described above, which tend to keep their parameters simple, this paper pursues an exhaustive definition of its components. Four running modes are presented, each one designed to best take advantage of environmental conditions. Finally, the paper outlines its decision making process through a series of state charts.

This paper caught my attention because just like my current model it focuses on wind and solar power. The task oriented module system is too distinct from the object oriented analysis I employ to be directly useful at current time. However, the exhaustive list of parameters gave me the opportunity to compare and contrast the parameters I obtained through requirements elicitation and thereafter decide whether each parameter deserved to be part of the model and whether I should adopt one or more of the parameters found within this paper. The decision outline will prove useful in future work as an example of a more complex control mechanism that better takes advantage of the attributes I've added to the model.

While the paper raises interesting ideas, it's light on detail. It lists parameters but does not take time to explain what role each one plays, nor even how they're quantified. The Data Resource module includes both a knowledge base and a database, but neither of those is mentioned again. The process through which the decision making agent interacts with the executive agent is also unexplained.

For the above reasons, I believe that this paper could benefit from formalism so the concepts it brings up can be better described and communicated. My model could potentially be used as a template to create a formal version of this paper's model, though this would require some follow-up work.

IV. METHODOLOGY

4.1 Problem Domain Analysis

Problem Domain Analysis is the process of defining the problem to be solved and what will need to be done to solve it – the requirements and the functionality of the system to be modeled [26]. The first step of the analysis is establishing basic familiarity with the field by studying the literature. I read the papers outlined in the Literature Review section as well as the additional ones cited in the bibliography to gain understanding of microgrid structure, the basic approaches to describing a microgrid, and the relevant terminology.

Once the basic research was completed, I moved on to the requirements elicitation. Requirements elicitation is the process of talking to the stakeholders of a system to gather information and understand what aspects of the system they find most important. For this project, the stakeholders were an engineering team of Dr. Hossein Salehfar and his graduate student. Over the course of several sessions we established a shared understanding of what the engineering team needed from a model. These discussions formed the base for later work. The first sessions were used to establish a shared understanding of the basic components of a microgrid and to decide which of their attributes belonged in the problem domain. Once this was established, a Vocabulary document was created to hold a definition for each object and attribute. The shared vocabulary allowed us to avoid misunderstandings and allowed for a greater understanding of each proposed attribute.

Some basic decisions were made at this time. First, the model would be object-oriented. It was believed that this would bring the model in line with modern software development practices. The model would also be suitable for describing implementation of Multi Agent Systems, as described above. This was of great interest to both myself and the engineering team for the purposes of

future research and fit with the developing paradigm of microgrid control. Second, the model would be focused on the logical layer rather than the physical one, tracking the flow of information and commands but not the physical exchange of power. Third, the only power storage considered would be chemical batteries, and the only generators considered would be solar cells and wind turbines. While some microgrid models include a non-renewable generator as a backup power source, one was consciously eschewed in this case.

4.2 Class Diagram Development

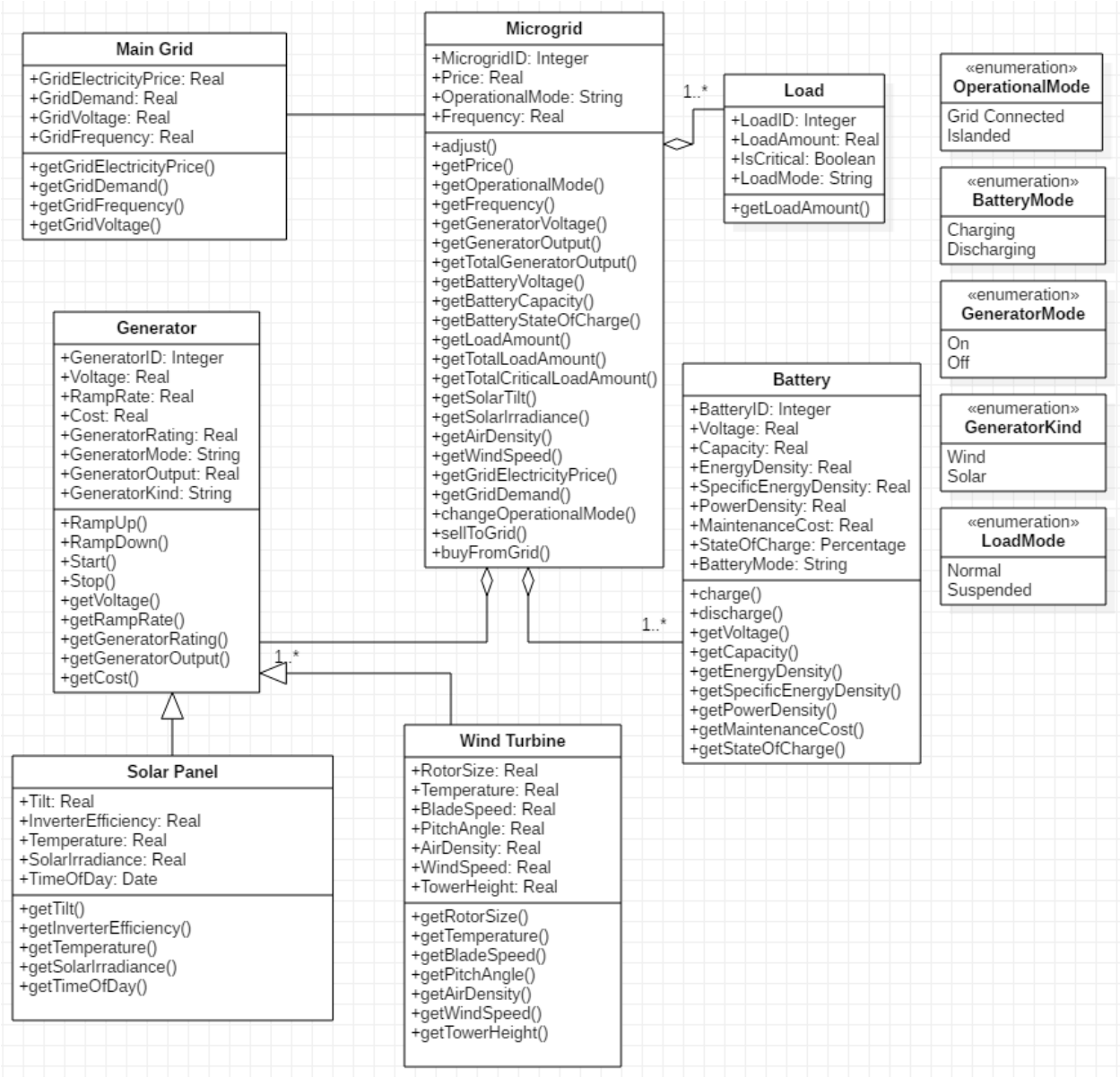


Figure 7: Final Class Diagram

Figure 7 shows the final class diagram of this project. A class diagram is a graphical representation of the model created using the Unified Modeling Language, as outlined in Section 2.2.1. Its purpose is to communicate information about the system in a way that's visual and therefore easier to understand than pure text. In particular, it helps to visualize the relationship between different parts of the system.

In addition, the process of developing a class diagram can serve as a way to improve and correct the model. To illustrate, Figure 8 shows the initial state of the class diagram for this project:

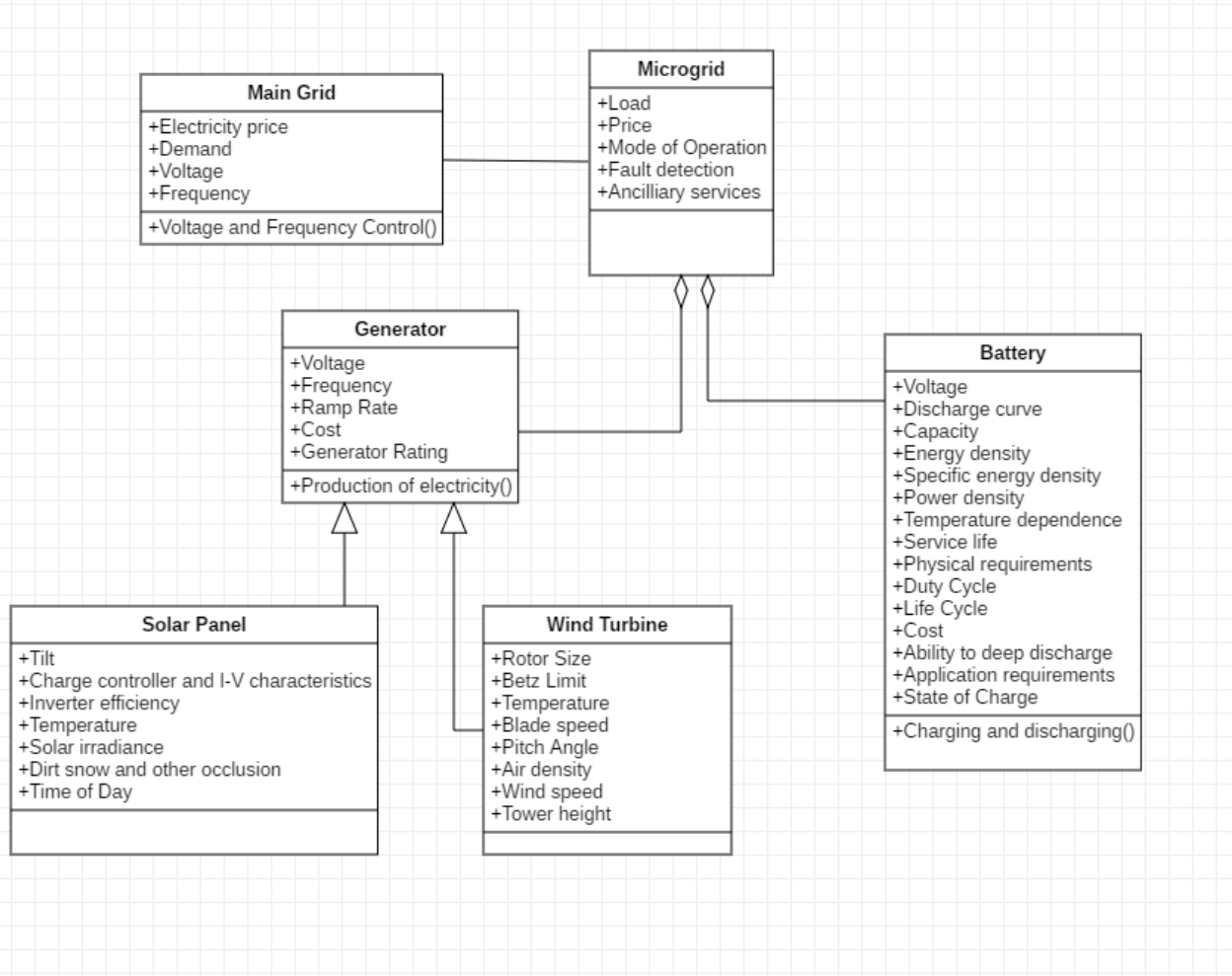


Figure 8: Early Class Diagram

At this stage of the project I had established an initial set of classes and their respective attributes. In my diagram, a *class* represents an entity with agency. Therefore, a generator is assigned a class, but an electrical bus isn't. As seen in the initial class diagram, `Load` was not considered a class at this point because initial requirements elicitation indicated that no decisions would be made in regard to the Load itself.

The next step in developing the class diagram was establishing the *relationships* between the classes. On a class diagram, the type of relationship is indicated by the connection between them. In my model (As seen in Figure 7 and the appendixes), the connection between the `Main Grid` and `Microgrid` is a simple association, which denotes a relationship between equals. The connection between `Load`, `Generator`, or `Battery` and the `Microgrid` is an aggregation, meaning that the `Microgrid` is in effect a collection of the other classes. The relationship between `Solar Panel` or `Wind Turbine` and the `Generator` is *inheritance,* which shows that they're subclasses of `Generator` and not classes in their own right.

At one point during the development a reflexive relationship was added to the `Microgrid`. This would have been used to denote a direct connection between microgrids that didn't use the main grid as an intermediary. This was a project considered by the engineering team, but due to lack of concrete data and time pressures this relationship was cut from the model.

4.3 Formal Specification Development

At this stage of the project I began the development of a formal specification document to complement the class diagram. A class diagram is useful for describing the structure of a system, but it cannot express all relevant information about a model. Formal specification can add details about how the system functions that increase the model's security, reliability, and usability, and open it up to validation. In addition, formal specification allows for clearer communication with the model's stakeholders and helps further the model's development. As outlined in Sections 2.2.2 and 2.2.3, I used Object Constraint Language (OCL) in conjunction with the UML-based Specification Environment (USE) to create the formal specification. OCL was chosen due to its object-oriented nature, synergy with UML, and the fact that its statements are easy to understand

even for those with no background. USE was chosen for its ability to serve as an interpreter, give textual form to UML objects, and do automatic error checking.

The first step in creating the specification was translating the existing classes and attributes from the class diagram to USE notation. As seen in Figure 9, the structures and relationships that make up a UML diagram have direct equivalents in USE.

```
abstract class Load
attributes
      LoadID: String
      LoadAmount: Real
      IsCritical: Boolean
      LoadMode: String
operations
      getLoadAmount() : Real = LoadAmount
end
```

Figure 9: USE Class

The creation of the textual notation was important because it forced the model to become complete and consistent. UML allows for placeholder attributes with no data types and relationships without defined roles or multiplicity. USE ensures that all of these are present before it validates the model, meaning that by the time a model has been validated by USE, it is free of errors and can be safely expanded.

The next step is the creation of constraints, using OCL. A constraint is an invariant statement that must hold true at every system initialization – in other words, a condition or restriction built into the model. Some constraints are based on logical or physical limitations. For example, “The capacity of a battery cannot be less than zero.” Other constraints are created for safety reasons. For example, “the frequency of the microgrid must be between 59.98 and 60.02 Hertz.” Others may be created for security or usability reasons, such as “the ID assigned to a generator must be unique.” These constraints can be seen in Figure 10.

```
    context Microgrid
    -- All IDs must be unique
    inv:
    self.generator->forAll(a, b | a.GeneratorID <> b.GeneratorID)
```

```
inv:
self.battery->forAll(a, b | a.BatteryID <> b.BatteryID)
inv:
self.load->forAll(a, b | a.LoadID <> b.LoadID)
-- Frequency Must be within a safe range
inv:
  Frequency > 59.98 and Frequency < 60.02
inv:
Price > 0
inv:
OperationalMode = 'Islanded' or OperationalMode = 'GridConnected'
```

Figure 10: Constraint Examples

An operation is an action that a class can take. Operations are present in the class diagram, but only as dummy values. What they actually do must be explained elsewhere, and an OCL specification provides a way to do it.

Operations break down into two kinds. The first are *query* operations. These are operations that, when given appropriate parameters, return information about the current state of the system to the user. They're used to model the way the system gathers information about itself and its surroundings and the way different parts of the system communicate. Examples of query operation are seen in Figure 11.

```
getGeneratorVoltage(): Bag(Real) = self.generator.Voltage
getGeneratorOutput(): Bag(Real) = self.generator.GeneratorOutput
getTotalGeneratorOutput(): Real = self.generator.GeneratorOutput->sum()
```

Figure 11: Query Operation Examples

In my model an object is generally allowed to query itself and the Microgrid may query other objects, but other objects are unable to directly query each other. This represents a hierarchical model in which all communication and commands flow through a single agent but subordinate agents are allowed to deal with themselves and their environment.

The other type of operation makes changes to the state of the system. Since the specification isn't meant to be a full implementation, it's not desirable to model every possible action. Thus, for example, the way the system updates each individual value isn't modeled. Instead the model shows major changes in the system's functioning, specifically the way it decides whether to sell

or buy power, charge or discharge batteries, and ramp power generation up or down. This is discussed in more detail in Section 5.2.

4.4 Iterative Development

As explained in Section 2.2.4., my research followed the Iterative Development design philosophy. I began with a provisional model based on the results of requirements elicitation. At that time, it wasn't a true class diagram, only a placeholder identifying the basic parts of the microgrid and the properties considered important by the engineering team. The development of this placeholder model into the initial class diagram and the concurrent development of the Vocabulary document was the first major iteration.

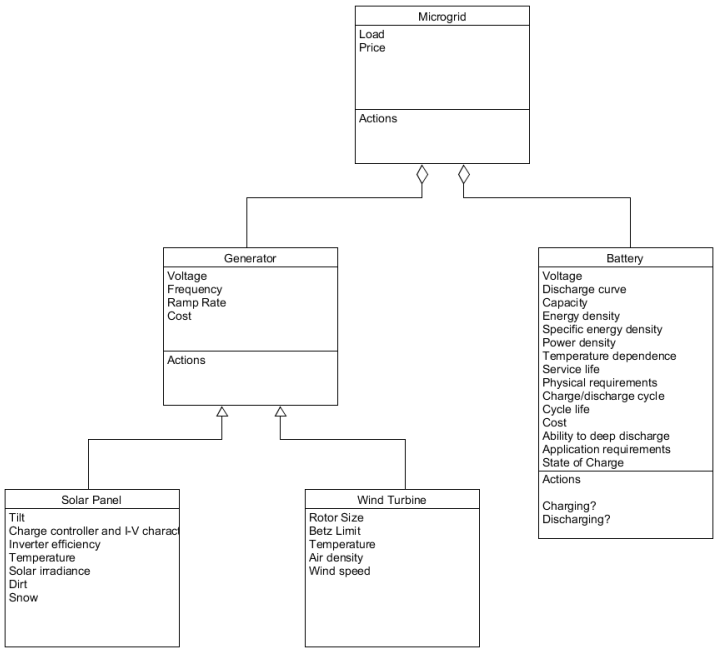


Figure 12: Placeholder Model

The placeholder model seen in Figure 12 was constructed from answers to the question "what matters in a microgrid"; at each subsequent meeting a new question was discussed, using the information gained in previous meetings as a base. The next question to be answered was "how can these attributes be quantified?" For most attributes this meant assigning a data type on the

class diagram and a metric unit in the Vocabulary. Other, ill-defined attributes were changed or replaced when it became apparent that they could not be quantified. For example, the `Battery` class had proposed attributes like `Life Cycle`, `Temperature Dependence`, and `Ability to Deep Discharge` which turned out to be too difficult to express using units and were therefore cut.

The next question addressed was "what should the microgrid do?" At this point I was looking for basic functionality and not a full set of operations. It was necessary to determine what concrete actions each object within the microgrid could take; how the system would decide when to take these actions was not yet a concern. At the same time the question of how the system's command and communication logic should be handled. It was decided that it should be handled primarily by the Microgrid class and other classes should not be able to communicate directly with each other.

In preparation for the construction of the formal specification, I created an informal list of desirable limits. In discussion it was decided that wherever there was room for doubt, limits would be left open-ended. For example, the `Turbine Height` attribute doesn't have a maximum even though there are realistic limits to the height of a given turbine. While a *specific* microgrid may limit the height of a turbine – it can even specify that all turbines should be the same size, with only a very small tolerance for variance – it was decided that it wasn't desirable that this extensible model should make that limitation.

At this point the class diagram was properly formatted and was closer to its final form, but it still contained errors. The `Load` was not a class at this point. Attributes necessary to denote system state didn't exist. The `Microgrid` class had a reflexive relationship with itself, denoting a proposition that microgrids should be able to directly interact with each other in a way different from their interactions with the `Main Grid`. Every object had its own `Voltage` and `Frequency`

rating. These aspects of the model, among others, would change with the creation of the formal specification, which represented the next major iteration.

As shown in Section 2.3, the first step of creating an OCL specification was translating the current state of the Class Diagram into OCL. During this process the ID attributes were added to each object when formalizing the relations between them showed that it was impossible to address an individual object in then-current state.

Then next step was formalizing the informal list of constraints. During this process the `Betz Limit` was removed from the list of `Wind Turbine` attributes. It was inserted into the class diagram during the discussion because it was considered important to the functioning of a wind turbine, but the process of creating constraints showed that it wasn't free to vary. Likewise, it was clarified that while each `Battery` and `Generator` would have to maintain their own voltage, only the Microgrid as a whole had to limit electrical frequency.

The creation of query operations established the communication logic of the microgrid. It also required further changes in the model, including the addition of the `GeneratorKind` attribute to the `Generator` class to enable the `Microgrid` to query its subclasses. When creating the class diagram I believed that setting `Wind Turbine` and `Solar Panel` as subclasses of `Generator` would be sufficient, but experimentation with OCL showed that wasn't true. While the class diagram already held placeholder operations for query operations, the process of formally creating them in OCL showed that the `Microgrid` class should have different operations for querying individual objects vs. collecting the totals of certain values (such as total available power output of all generators).

The construction of non-query actions, discussed in Section 5.2, required further changes to the model. As discussed previously, the `Load` had previously been an attribute of the `Microgrid` class,

serving as a stand-in for total demand. The discussion of decisions a microgrid may have to make required splitting it off into its own class because the decision was made that in certain situations a load may have to be temporarily suspended, which meant that each load had to be treated as a separate object with agency instead of just a value. The `Load` was initially split into `Critical` and `Non-Critical` subclasses, but further refinement of the model showed that a Boolean variable was a better way to distinguish between critical and non-critical loads, at least in the context of a specification. This prompted the creation of further query actions. Several enumerated attributes were added to model different system states, enabling the results of actions to be shown.

Each revision of the earlier parts of the model prompted alterations to the newer parts. When attributes were added to the Class Diagram, for example, they had to be inserted into the Vocabulary document (along with definitions) and into the OCL diagram, which could reveal previously unquestioned assumptions or errors. Though this mechanism the Iterative Development cycle enabled the creation of a model that was more complete and correct than could have been created with a more linear process.

V. Results and Analysis

5.1 Structure

The structure of the Microgrid is outlined in detail in Appendix A (The UML Class Diagram), Appendix B (The Vocabulary) and Appendix C (The OCL specification). Presented here is a more thorough explanation of each component part and the history of its iterated development.

Main Grid: a large-scale traditional electrical grid, typically connected to one or more conventional power stations. It's not the focus of the model and is therefore represented only in terms of its relationship with the microgrid, which consists primarily of selling and buying power.

In the model the `Main Grid` is assumed to be able to supply a functionally limitless amount of energy and to demand a specific amount of energy, with a set selling and buying price visible to the microgrid, with the microgrid able to make buying and selling decisions in real time. This is not universally true. In some grids power consumers and power producers may participate in a marketplace, bidding for specific amounts of power, leading to an uncertain price. In other grids the microgrid may have to make commitments over fixed periods of time, only enabling it to make decisions at fixed intervals. Since these mutually exclusive cases would be impossible to model in a single specification, the model present in the microgrid was chosen for the sake of simplicity.

Because the specification in this thesis is focused specifically on the microgrid and not the multi-microgrid, the `Main Grid` is deliberately kept as general as possible. As can be seen in section 4.4 it was not even included in the original placeholder model, though the decision to add it came early on.

Microgrid: the `Microgrid` class represents an entity in the overall control of the microgrid. It takes in information from the other agents, makes decisions, and sends out instructions. The

`Microgrid` governs the relations between `Generators`, `Batteries`, and `Loads`, and thereby handles load balancing (making sure power demands are filled) and ancillary services (error checking and keeping voltage, current, and frequency within acceptable tolerances). The `Microgrid` is also responsible for negotiating the buying and selling of power with the `Main Grid` and establishing or severing the connection to the `Main Grid`.

Depending on how the microgrid is structured, the `Microgrid` class may represent a distinct agent that's controlling the other parts of the microgrid hierarchically, or it may represent a mechanism through which other agents reach a consensus.

In the early design on the model consideration was given to creating a separate agent for the purpose of interacting with the main grid. When the decision was made that the `Microgrid` agent would handle all communication within the microgrid, however, it was decided that communication with the `Main Grid` should pass through it too.

The possibility of direct microgrid-to-microgrid interaction independent of interactions with the larger grid was discussed during the requirements elicitation phase. However, this was dropped later in the process because the concept was too novel and inadequately explored and would introduce guesswork into the model.

Generator: an entity responsible for producing power. In this model either a photovoltaic cell or a wind turbine. A single `Generator` agent may be responsible for one physical device, or for several devices that aggregate their decisions.

The `Generator` agent's main task is to control the intensity of power generation, which may need to be ramped up or down depending on the load, the environmental conditions, safety requirements, equipment longevity, or financial considerations. The `Generator Agent` collects information about its hardware and its environment through its sensors, and gains relevant

information about the state of the microgrid by communicating with other agents, enabling it to make informed decisions.

The decision to focus on photovoltaic solar panels and wind turbines as the sole considered power sources was made during the requirements elicitation phase to fit with the needs of the engineering team. Additional power sources may be added to the model in the future.

The `Solar Panel` and `Wind Turbine` subclasses went through alterations that discarded attributes that are important to the functioning of such generators but not to a microgrid's decision making. For example, early in the requirements elicitation phase the decision was made to consider dirt and snow. When formalizing attributes these were fused into occlusion. But during the design of the operations it was decided that how much of the panel was covered by occlusion is not directly relevant to a microgrid's control logic, only the way it affects solar irradiance. Since solar irradiance was already an attribute, occlusion was dropped as an attribute at a fairly late stage.

The `Generator` class and its subclasses contain attributes which aren't used by the algorithm presented in Section 5.2. They were nonetheless kept in the model because they may be used by other algorithms, fitting with the goal of making the model extensible.

Battery: fulfils a role similar to the `Generator Agent`, but in respect to power storage. In my model only chemical batteries are considered, but other means of power storage may be implemented in future work.

The `Battery` agent's primary role is to charge and discharge electrical power. The goals to be considered when making the decision include maximizing financial benefit, maintaining a store of power for emergencies, minimizing pollution, and other considerations dictated by the nature of the microgrid.

In the requirements elicitation phase attributes of a battery such as its life cycle, its physical properties, and its discharge curve were considered important to the functioning of the battery. However, later in the process it was determined that while these may be important to the construction of a microgrid, they did not have a direct effect on the microgrid's decisions.

Load: a section of the microgrid that consumes electrical power. The role of the `Load Agent` is to monitor the demand for power, and to coordinate with other parts of the microgrid to make sure that demand is met.

A critical load is a demand on power that must be filled immediately. A non-critical load is a demand on power that can be postponed. In the context of a residential microgrid critical loads encompass typical household applications such as lights and appliances. Jiang, et al [4] provide a good example of a non-critical load in the form of electrically-pumped water tanks for residential houses. The tanks provide water to the household, so they must be filled at some point, but the pumping may take place in off-peak hours without jeopardizing reliability.

The critical/non-critical divide can work differently depending on what the microgrid is powering. For a hospital microgrid life-support equipment is considered critical, while monitors and phones are not. For a business microgrid the equipment critical to the organization is critical. Thus, printers may be considered critical for a printing company, they may be considered non-critical by a typical office. The ability to distinguish between critical and non-critical loads is a key part of ensuring a microgrid's stability.

As outlined in above sections, the `Load` was originally considered an attribute of the `Microgrid` Class, with only the total demand on the microgrid being considered. When it was decided that a distinction should be made between critical and non-critical loads, `Load` was split off into its own class, with each load counting separately.

5.2 Behavior

As discussed in above sections, the system's operations can be broken down into query and non-query operations. Query operations probe the state of the system without altering it, while non-query operations alter the state of the system. Thus, query operations represent the work of the sensors and the communication within the system, while non-query operations represent concrete actions taken by the system.

As previously outlined (and seen in Appendix B), query operations in this model are laid out in a way that ensures the `Microgrid` agent is in control of all communication between other agents. This creates a single point of failure, losing some of the flexibility and robustness of a fully distributed framework, but it enables stronger coordination between the system's various parts. This set up is suitable for a typical household microgrid.

The `Microgrid` agent also triggers the non-query operations that adjust the microgrid's behavior according to a given microgrid's goals. The algorithm that governs the behavior of the modeled algorithm is presented here as an illustrative example. During requirements elicitation, I was given the following priorities for the behavior of the microgrid:

1. The loads should be serviced whenever possible, including non-critical loads.
2. Charging the batteries always takes priority over selling power.
3. Power should not be purchased to charge batteries.
4. When power is needed for the microgrid, self-generated power should always be used first.

These priorities represent a typical residential microgrid whose stakeholders expect any interruptions in service to be temporary, consider grid stability to be more important than

maximizing profits, and whose non-critical loads aren't set up in a way that allows them to take advantage of off-peak hours.

These priorities were then translated into a set of rules shown in Figure 13.

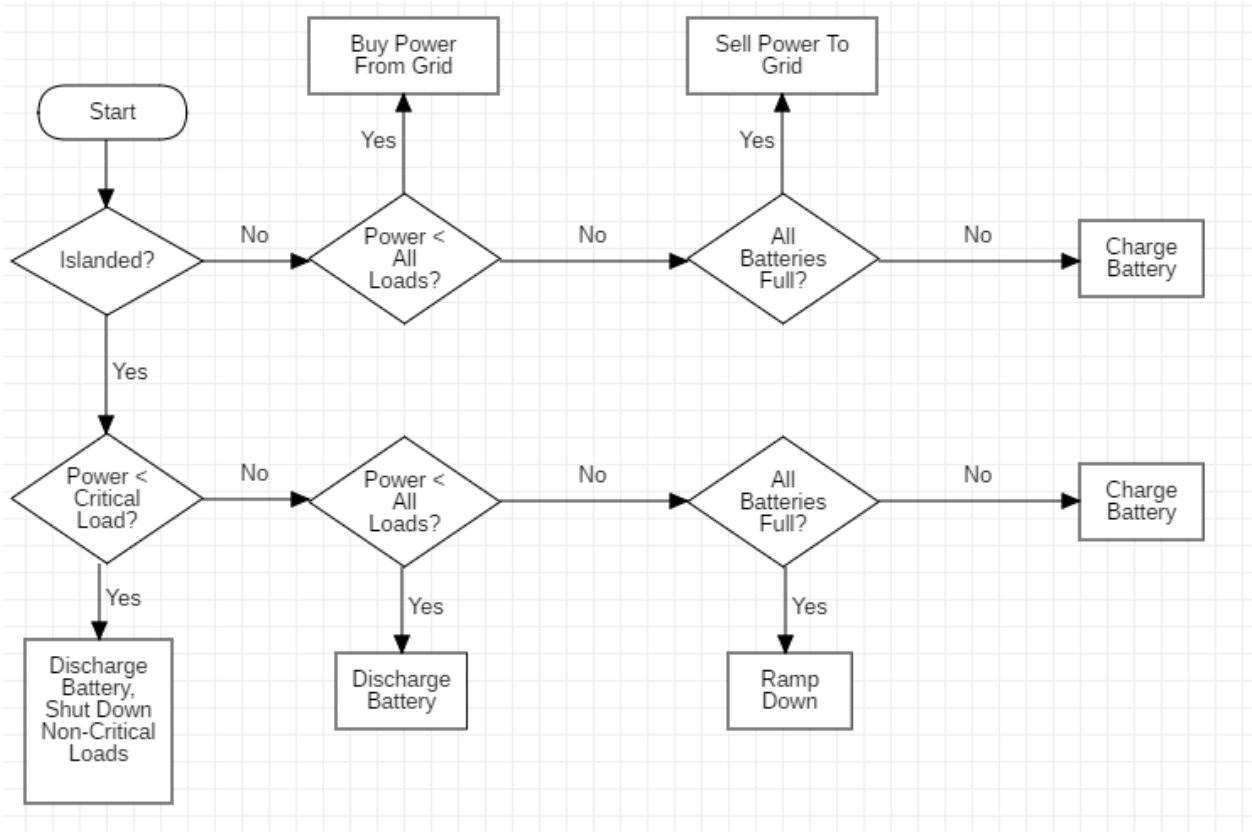


Figure 13: Microgrid Control Flow

This rule set was then implemented in OCL as a set of conditional statements that alter the state of the system, as seen in Figure 14.

```
context Microgrid::adjust()
 post adjustment:
  if OperationalMode = 'Islanded' then
      if (getTotalGeneratorOutput()) < (getTotalCriticalLoadAmount()) then
            battery->forAll(BatteryMode = 'Discharging')
            --and load->forAll(l:Load | l.IsCritical = false).LoadMode = 'Off'
      else
            if (getTotalGeneratorOutput()) < (getTotalLoadAmount()) then
                  battery->forAll(BatteryMode = 'Discharging')
            else
                  if battery->forAll(StateOfCharge = 100) then
                        getTotalGeneratorOutput() = getTotalLoadAmount()
```

```
                        else
                                battery->forAll(BatteryMode = 'Charging')
                        endif
                endif
        endif
       else
        if (getTotalGeneratorOutput()) < (getTotalLoadAmount()) then
                CommerceMode = 'Buying'
        else
                if battery->forAll(StateOfCharge = 100) then
                        CommerceMode = 'Selling'
                else
                        battery->forAll(BatteryMode = 'Charging')
                endif
        endif
endif
```

Figure 14: OCL Operation

The modeled behavior accomplishes the elicited priorities, ensuring that the grid prioritizes making power available to the consumers, only discharging batteries when the grid is in islanded mode, and only selling excess power when all current power needs are met and the batteries are fully charged.

This does not represent the only possible set of goals for a microgrid. For instance, a microgrid that provides power to safety critical equipment could place even more emphasis on having power available in case of emergency, while a microgrid that had ample non-critical loads could prioritize minimizing costs by shutting down non-critical loads or discharging batteries when power is most expensive. Section 6.3 discusses some possible goal sets to be modeled in future work.

The model of system behavior modeling went through the same Iterative Development process as the model of its structure. The initial flow chart was incomplete, and only the development of the textual specification ensured that all relevant scenarios were covered by the model. Behavior modeling also forced changes to structure modeling by adding the required *state* attributes to the `Microgrid`, `Generator`, and `Battery` classes.

The current behavior model is simplified for the sake of comprehensibility. It does not, for example, deal with the possibility of shutting down one `Generator` while keeping the rest going. That possibility was considered during the development, but ultimately eschewed because it would be too complex to implement in vanilla OCL. Notably, the use of state changes in the model is partially due to the fact that it's one of the few available ways to model control flow in OCL. Future work modeling more complex behavior, particularly time-dependent behavior, may need to incorporate additional tools, such as the use of one of several OCL-based imperative languages.

VI. CONCLUSION

6.1 Work accomplished

As outlined in the Methodology section, my work went through an iterative process, beginning with requirements elicitation and continuing with the creation of a class diagram and a formal specification, with each step increasing the accuracy and detail of the model.

6.2 Outcomes achieved

After undergoing the steps described above, I was able to create a formal model of a microgrid consisting of a UML class diagram, an OCL specification verified by the application of USE, and an associated vocabulary document. The model describes a residential microgrid made up of photovoltaic solar panels, wind turbines, and Critical and Non-Critical Loads, controlled by a Multi Agent System, with rules governing its behavior based on a priority queue.

The end result is an extensible model that can be used as foundation and template for future work with formal modeling of microgrids. This model satisfies my initial goals, though the next section indicates further work that could improve or extend the model.

6.3 Future Work

As noted above, my model is made to be extensible and can therefore be used to pursue multiple avenues of further research.

A potential first step would be to expand the model to cover more use cases. Non-renewable power sources would be added to the Generator class, and additional power storage options would be added to the battery class. The Microgrid class would be expanded with more explicit error checking. Direct interaction between different microgrids would be added to the model.

To demonstrate the usefulness of formal modeling to the planning stages of an engineering project, I would like to create a sample microgrid bounded by more specific constraints. As

outlined in the Methodology section, the currently existing constraints deal mostly with conditions that are physically impossible or universally unsafe. The sample microgrid would operate under more specific restrictions; it would have a maximum number of batteries, a maximum allowable height for wind turbines, a minimum amount of total provided power and stored power based on the average load, and other realistic restrictions to be included as part of a potential engineering project.

I would also like to create several alternative rule sets for control flow to better showcase the versatility of the model. The rule set presented in this thesis emphasizes stability and keeping the batteries full when possible. Alternative goals to be explored in the future include: maximizing financial advantage by selling power during peak hours and buying it during off-peak hours; ensuring that power storage is full at a specific time (such as the beginning of the night in predominantly solar-powered microgrid); balancing non-critical loads and power storage; maximizing environmental benefits; minimizing wear and tear on microgrid components. All of these rule sets can be modeled using the model in its current state.

APPENDICES

Appendix A

Vocabulary

	Description	Notes And Constraints	Data Type	Units
Classes:				
Microgrid	A small network of electricity users with a local source of supply			
Load	Consumes electrical power			
Battery	Any power storage system	We are exclusively focused on chemical batteries		
Generator	A device that converts motive power (mechanical energy) into *electrical* power for use in an external circuit			
SolarPanel	A panel designed to absorb the sun's rays as a source of energy			
WindTurbine	A turbine having a wheel rotated by the wind to generate electricity			
Attributes:				
	MainGrid			
GridElectricityPrice	Amount of money paid per watt of electricity bought from the grid		Real	
GridDemand	Amount of electricity the grid is willing to buy	Due to difference in sizes may be irrelevant for smaller microgrids	Real	
GridFrequency	Nominal frequency of the oscillations of alternating current (AC) in an electric power grid		Real	

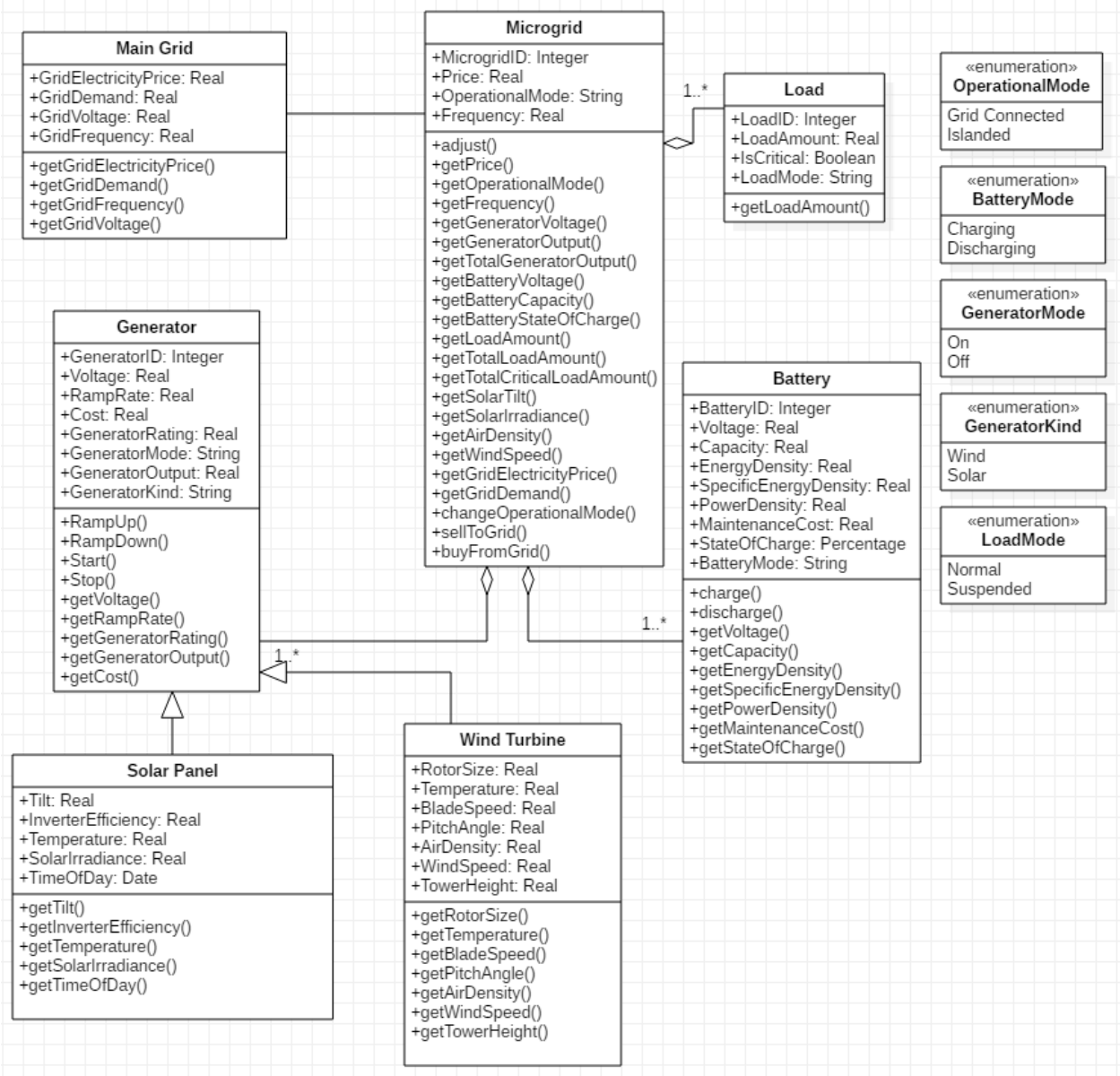
GridVoltage	An electromotive force or potential difference expressed in volts		Real	
	Microgrid			
MicrogridID	A unique ID	Must be unique	String	
Price	Amount of money paid per watt of electricity sold to the grid	Price for buying and Selling is different usually utility power companies will charge more when buying and will pay less when buying electricity	Real	Dollar
OperationalMode	Grid Connected or Islanded	Transition is not being considered by the current model	String	
Frequency	Nominal frequency of the oscillations of alternating current (AC) in an electric power grid	Must be between 60.02 and 59.98 HZ	Real	Hertz
	Generator			
GeneratorID	A unique ID	Must be unique	String	
Voltage	An electromotive force or potential difference expressed in volts		Real	Volt
RampRate	The increase or reduction in output per minute		Real	Mw/min
Cost	Cost of generating electricity	In our case cost is sum of principal plus maintenance cost	Real	Dollars
GeneratorRating	The product of the voltage per phase, the current per phase, and the number of phases		Real	kVA
Output	Current output of the generator measured in Watts		Real	Watt
	Load			

LoadID	A unique ID	Unique	String	
LoadAmount	Total amount of power demanded by the load		Real	kWh
IsCritical	Whether this load is a critical load		Real	kWh
LoadMode	Whether this load should be filled	On/Off	String	
	Battery			
BatteryID	A unique ID	Unique	Real	
Voltage	An electromotive force or potential difference expressed in volts		Real	Volt
Capacity	A measure (typically in Amp-hr) of the charge stored by the battery		Real	Amp-hr
Energy density	The amount of energy stored per unit volume		Real	
Specific energy density	The amount of energy stored per unit mass		Real	
Power density	The amount of power per unit volume		Real	
MaintenanceCost		Of the battery, not the electricity	Real	Dolalrs
State of Charge	The equivalent of a fuel gauge		Real	Percent
BatteryMode	Whether the battery is receiving/Ready to receive power or putting it out	Charging/Discharging	String	
	Solar Panel			
Tilt	Angle of the solar panel		Real	Degrees
Inverter efficiency	It converts the variable direct current of a photovoltaic solar panel into alternating current		Real	Percent
Time of Day		Related to solar irradiance		
Temperature	Temperature of the air		Real	Celsius

Solar irradiance	How much light is hitting the solar cell	Determined party by occlusion		
	Wind Turbine			
Rotor Size	The size of the rotors		Real	Meters
Temperature	Temperature of the air			
Blade speed	Speed of the blades		Real	m/s
Air density	How dense air is - varies with elevation		Real	
Tower Height	The height of the wind turbine		Real	Meters
Wind speed	Wind speed		Real	

Apppendix B

Class Diagram (created in UML)



Appendix C

Formal Specification

Created in USE, incorporating USE notation for objects and relationships and OCL notation for constraints and operations

---

```
model Microgrid

--  datatypes

--  enum OperationalMode { GridConnected, Islanded }
--  enum GeneratorKind { Solar, Wind }
--  enum GeneratorMode {On, Off}
--  enum LoadMode {On, Off}
--  enum BatteryMode {Charging, Discharging}
--  enum CommerceMode {Selling, Buying, Neither}

-- datatype Time
-- operations
--   static Time(hour: Integer, minute: Integer, second: Integer) : Time
--   static now() : Time
--   < (time2 : Time) : Boolean
--   > (time2 : Time) : Boolean
--   = (time2 : Time) : Boolean
--   <> (time2 : Time) : Boolean
--   after(when : Time) : Boolean
--   before(when : Time) : Boolean

-- end

-- classes

abstract class Microgrid
attributes
  MicrogridID: Integer
  Price : Real
  OperationalMode : String -- GridConnected, Islanded
  CommerceMode: String -- Selling, Buying, Neither
  Frequency: Real
operations
  adjust()
  getPrice() : Real = Price
  getOperationalMode() : String = OperationalMode
  getFrequency() : Real = Frequency
```

```
  getGeneratorVoltage(): Bag(Real) = self.generator.Voltage
  getGeneratorOutput(): Bag(Real) = self.generator.GeneratorOutput
  getTotalGeneratorOutput(): Real = self.generator.GeneratorOutput->sum()
  getElectricityPriceMain(): Real = self.maingrid.GridElectricityPrice
  getBatteryVoltage(): Bag(Real) = self.battery.Voltage
  getBatteryCapacity(): Bag(Real) = self.battery.Capacity
  getBatteryStateOfCharge(): Bag(Real) = self.battery.StateOfCharge
  getLoadAmount(): Bag(Real) = self.load.LoadAmount
  getTotalLoadAmount(): Real = self.load.LoadAmount->sum()
  getTotalCriticalLoadAmount(): Real = self.load->select(l:Load | l.IsCritical =
true).LoadAmount->sum()
  getSolarTilt(): Bag(Real) = self.generator->select(g:Generator | g.GeneratorKind =
'Solar').oclAsType(SolarPanel).Tilt
  getSolarIrradiance(): Bag(Real) = self.generator->select(g:Generator | g.GeneratorKind =
'Solar').oclAsType(SolarPanel).SolarIrradiance
  getAirDensity(): Bag(Real) = self.generator->select(g:Generator | g.GeneratorKind =
'Wind').oclAsType(WindTurbine).AirDensity
  getWindSpeed(): Bag(Real) = self.generator->select(g:Generator | g.GeneratorKind =
'Wind').oclAsType(WindTurbine).WindSpeed
  getGridElectricityPrice(): Real = self.maingrid.GridElectricityPrice
  getGridDemand(): Real = self.maingrid.GridDemand
  changeOperationalMode()
  sellToGrid()
  buyFromGrid()

end

abstract class Load
attributes
  LoadID: Integer
  LoadAmount: Real
  IsCritical: Boolean
  LoadMode: String
operations
  getLoadAmount() : Real = LoadAmount
end

abstract class Generator
attributes
  GeneratorID: Integer
  Voltage: Real
  RampRate: Real
  Cost: Real
  GeneratorRating: String
  GeneratorOutput: Real
  GeneratorKind: String
```

```
  GeneratorMode: String
operations
  getVoltage() : Real = Voltage
  getRampRate() : Real = RampRate
  getCost() : Real = Cost
  getGeneratorRating() : String = GeneratorRating
  getGeneratorOutput() : Real = GeneratorOutput
  RampUp()
  RampDown()
  start()
  stop()
end

class SolarPanel < Generator
attributes
        Tilt: Real
        InverterEfficiency: Real
        Temperature: Real
        SolarIrradiance: Real
        TimeOfDay: Real
operations
  getTilt() : Real = Tilt
  getInverterEfficiency() : Real = InverterEfficiency
  getTemperature() : Real = Temperature
  getSolarIrradiance() : Real = SolarIrradiance
  getTimeOfDay() : Real = TimeOfDay
end

class WindTurbine < Generator
attributes
        RotorSize: Real
        BetzLimit: Real
        Temperature: Real
        BladeSpeed: Real
        PitchAngle: Real
        AirDensity: Real
        WindSpeed: Real
        TowerHeight: Real
operations
  getRotorSize() : Real = RotorSize
  getTemperature() : Real = Temperature
  getBladeSpeed() : Real = BladeSpeed
  getPitchAngle() : Real = PitchAngle
  getAirDensity() : Real = AirDensity
  getWindSpeed() : Real = WindSpeed
  getTowerHeight() : Real = TowerHeight
```

```
end

class Battery
attributes
  BatteryID: Integer
  Voltage: Real
  Capacity: Real
  EnergyDensity: Real
  SpecificEnergyDensity: Real
  PowerDensity: Real
  MaintenanceCost: Real
  StateOfCharge: Real
  BatteryMode: String
operations
  getVoltage() : Real = Voltage
  getCapacity() : Real = Capacity
  getEnergyDensity() : Real = EnergyDensity
  getSpecificEnergyDensity() : Real = SpecificEnergyDensity
  getPowerDensity() : Real = PowerDensity
  getMaintenanceCost() : Real = MaintenanceCost
  getStateOfCharge() : Real = StateOfCharge
  charge()
  discharge()
end

class MainGrid
attributes
  GridElectricityPrice: Real
  GridDemand: Real
  GridVoltage: Real
  GridFrequency: Real
operations
  getGridElectricityPrice() : Real = GridElectricityPrice
  getGridDemand() : Real = GridDemand
  getGridVoltage() : Real = GridVoltage
  getGridFrequency() : Real = GridFrequency
end

association GridGenerator between
  Microgrid[1] role microgrid
  Generator[*] role generator
end

association GridBattery between
  Microgrid[1] role microgrid
  Battery[*] role battery
```

```
end

association GridLoad between
  Microgrid[1] role microgrid
  Load[*] role load
end

association MicroMain between
  Microgrid[1] role microgrid
  MainGrid[1] role maingrid
end


-- Constraints

constraints

context Microgrid
-- All IDs must be unique and positive
inv:
self.generator->forAll(a, b | a.GeneratorID <> b.GeneratorID) and self.generator->forAll(a |
a.GeneratorID > 0)
inv:
self.battery->forAll(a, b | a.BatteryID <> b.BatteryID)and self.battery->forAll(a | a.BatteryID >
0)
inv:
self.load->forAll(a, b | a.LoadID <> b.LoadID)and self.load->forAll(a | a.LoadID > 0)
-- Frequency Must be within a safe range
inv:
  Frequency > 59.98 and Frequency < 60.02
inv:
Price > 0
inv:
OperationalMode = 'Islanded' or OperationalMode = 'GridConnected'

context Load
inv:
LoadAmount > 0

context Generator
inv:
Cost > 0
inv:
GeneratorOutput >= 0
inv:
Voltage >= 118 and Voltage <= 122
```

```
inv:
GeneratorKind = 'Solar' or GeneratorKind = 'Wind'

context Battery
inv:
Capacity > 0
inv:
EnergyDensity > 0
inv:
SpecificEnergyDensity > 0
inv:
PowerDensity > 0
inv:
MaintenanceCost > 0
inv:
StateOfCharge >= 0 and StateOfCharge <= 100

context Microgrid::adjust()
  post adjustment:
    if OperationalMode = 'Islanded' then
            if (getTotalGeneratorOutput()) < (getTotalCriticalLoadAmount()) then
                    battery->forAll(BatteryMode = 'Discharging')
                    --and load->forAll(l:Load | l.IsCritical = false).LoadMode = 'Off'
            else
                    if (getTotalGeneratorOutput()) < (getTotalLoadAmount()) then
                            battery->forAll(BatteryMode = 'Discharging')
                    else
                            if battery->forAll(StateOfCharge = 100) then
                                    getTotalGeneratorOutput() = getTotalLoadAmount()
                            else
                                    battery->forAll(BatteryMode = 'Charging')
                            endif
                    endif
            endif
           else
            if (getTotalGeneratorOutput()) < (getTotalLoadAmount()) then
                    CommerceMode = 'Buying'
            else
                    if battery->forAll(StateOfCharge = 100) then
                            CommerceMode = 'Selling'
                    else
                            battery->forAll(BatteryMode = 'Charging')
                    endif
            endif
    endif
```

```
context Microgrid::changeOperationalMode()
  post changePost: if OperationalMode = 'Islanded' then OperationalMode = 'GridConnected' else
OperationalMode = 'Islanded' endif


context Battery::charge()
  pre: self.BatteryMode = 'Discharging'
  post: self.BatteryMode = 'Charging'

context Battery::discharge()
  pre: self.BatteryMode = 'Charging'
  post: self.BatteryMode = 'Discharging'


context Microgrid::sellToGrid()
  post: self.CommerceMode = 'Selling'

context Microgrid::buyFromGrid()
  post: self.CommerceMode = 'Buying'
```

REFERENCES

[1] Frank Ibarra Hernandez, Carlos Alberto Canesin, Ramon Zamora, Anurag K Srivastava, “Active Power Management in Multiple Microgrids Using a Multi-Agent System with JADE,” 11th IEEE/IAS International Conference on Industry Applications, doi: 10.1109/INDUSCON.2014.7059471, December 2014

[2] Chunxia Dou, Mengfei Lv, Tianyu Zhao, Yeping Ji, Heng Li, “Decentralised coordinated control of microgrid based on multi-agent system,” IET Generation, Transmission & Distribution, doi: 10.1049/iet-gtd.2015.0397, November 2014.

[3] M. Reyasudin Basir Khan, Razali Jidin, Jagadeesh Pasupuleti, “Multi-agent based distributed control architecture for microgrid energy management and optimization,” Energy Conversion and Management 112 (2016) 288–307, 25 January 2016

[4] Jiang, R.; Wang, J.; Guan, Y. Robust unit commitment with wind power and pumped storage hydro. IEEE Trans. Power Syst. 2012, 27, 800–810.

[5] Abhilash Kantamneni, Laura E. Brown, Gordon Parker, Wayne W. Weaver, “Survey of multi-agent systems for microgrid control,” Engineering Applications of Artificial Intelligence 45 (2015) 192–203.

[6] A. F. Burke, “Batteries and ultracapacitors for electric, hybrid, and fuelcell vehicles,”Proc. IEEE, vol. 95, no. 4, pp. 806–820, Apr. 2007

[7] F. Blaabjerg, F. Iov, T. Kerekes and R. Teodorescu, "Trends in power electronics and control of renewable energy systems," *Proceedings of 14th International Power Electronics and Motion Control Conference EPE-PEMC 2010*, Ohrid, 2010, pp. K-1-K-19, doi: 10.1109/EPEPEMC.2010.5606696.

[8] Shuyun Jia, Jiang Chang, “Research on multi-agent decision-making model of wind-solar complementary power generation system”, 2009 Second International Conference on Intelligent Computation Technology and Automation, doi: 10.1109/ICICTA.2009.718, October 2009

[9] Axel van Lamsweerde. 2000. “Formal specification: a roadmap,” Proceedings of the Conference on The Future of Software Engineering (*ICSE ’00*). Association for Computing Machinery, New York, NY, USA, 147–159. DOI:https://doi.org/10.1145/336512.336546

[10] Ahmed Taki Eddine Dib, Zaidi Sahnoun, “Formal Specification of Multi-Agent System Architecture,” International Conference on Advanced Aspects of Software Engineering ICAASE, November, 2-4, 2014, Constantine, Algeria.

[11] Mustapha Bourahla, Mohamed Benmohamed , “Formal Specification and Verification of Multi-Agent Systems,” Electronic Notes in Theoretical Computer Science 123 (2005) 5–17, doi: 10.1016/j.entcs.2004.04.042.

[12] D. E. Olivares *et al*., "Trends in Microgrid Control," in *IEEE Transactions on Smart Grid*, vol. 5, no. 4, pp. 1905-1919, July 2014, doi: 10.1109/TSG.2013.2295514.

[13] “Tracking Corporate Solar Adoption in the U.S.,” Solar Energy Industries Association April 2018. seia.org

[14] T. Funabashi, T. Tanabe, T. Nagata, R. Yokoyama, “An Autonomous Agent for Reliable Operation of Power Market and Systems Including Microgrids,” Third International Conference on Electric Utility Deregulation and Restructuring and Power Technologies, doi: 10.1109/DRPT.2008.4523397, April 2008

[15] A. Sujil, Jatin Verma, Rajesh Kumar, “Multi agent system: concepts, platforms and applications in power systems,” Artif Intell Rev (2018) 49:153–182 https://doi.org/10.1007/s10462-016-9520-8, 12 October 10`6

[16] C.M. Colson, M.H. Nehrir, “A Review of Challenges to Real-Time Power Management of Microgrids,” IEEE Power & Energy Society General Meeting, 2009

[17] A. L. Dimeas, S.I. Hatzivasiliadis, N. D. Hatziargyriou, “Control agents for enabling customer-driven Microgrids,” IEEE Power & Energy Society General Meeting, 2009

[18] M. Collins, “Formal methods,” Dependable Embedded Systems, 1998.

[19] Soukaina Boudoudouh, Mohamed Maâroufi, “Multi agent system solution to microgrid implementation,” Sustainable Cities and Society 39 (2018) 252–261

[20] Enrique Kremers, Jose Gonzalez de Durana, Oscar Barambones, “Multi-agent modeling for the simulation of a simple smart microgrid,” Energy Conversion and Management 75 (2013) 643–650.

[21] Booch, Grady & Rumbaugh, James & Jacobson, Ivar. (1999). Unified Modeling Language User Guide, The (2nd Edition) (Addison-Wesley Object Technology Series). J. Database Manag.. 10.

[22] OM Group. Object constraint language (ocl) v2. 4, February 2014.

[23] Gogolla, M., Büttner, F., & Richters, M. (2005). A UML-based specification environment for validating UML and OCL. *Science of Computer Programming*.

[24] X. Ge, R. F. Paige and J. A. McDermid, "An Iterative Approach for Development of Safety-Critical Software and Safety Arguments," *2010 Agile Conference*, Orlando, FL, 2010, pp. 35-43, doi: 10.1109/AGILE.2010.10.

[25] ZHOU Xiaoyan, LIU Tianqi, LIU Xueping, “Multi-Agent based Microgrid Coordinated Control”, 2011 2nd International Conference on Advances in Energy Engineering.

[26] Prieto-Díaz, R. (1990). Domain analysis: An introduction. *ACM SIGSOFT Software Engineering Notes*, *15*(2), 47-54.