7-12-2022

Development of a Configurable DERMS Test System in GridAPPS-D

Sean James Keene
Portland State University

Recommended Citation
Keene, Sean James, "Development of a Configurable DERMS Test System in GridAPPS-D" (2022). *Dissertations and Theses.* Paper 6191.
https://doi.org/10.15760/etd.8052

Development of a Configurable DERMS Test System in GridAPPS-D

by

Sean James Keene

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Robert B. Bass, Chair
John M. Acken
John Lipor

Portland State University
2022

Abstract

Distributed Energy Resources and their increased penetration within the power grid are a suitable method to provide essential reliability services. In order to do so, they must operate in aggregate. Aggregation of a wide variety of systems with varying standards and protocols is a complex task, which could benefit from rules and standards to support interoperability. To this end, the Portland State University Power Lab has developed the Energy Grid of Things project, which includes a Distributed Energy Resource Management System to provide aggregation and demonstrate the usage of an Energy Services Interface: a set of rules governing the interface between resource owners and aggregator businesses.

This thesis presents a Modeling Environment that provides a simulation and test platform reflecting the complex needs of an aggregator. A grid simulator is required that is able to reflect the effects of thousands of Distributed Energy Resources operating in tandem; furthermore, these effects must be communicated to external actors such as the Grid Operator, which provides grid service requests to the aggregator and requires feedback. The Modeling Environment provides all of this functionality by leveraging the GridAPPS-D platform to develop an interoperable class-based system that provides configurable simulations with modular input and output APIs. The Modeling Environment uses abstractions to allow any feasible type of input to be reflected within the model as a Distributed Energy Resource,

and serves as a proof-of-concept for the development of aggregation simulation and testing within GridAPPS-D.

Dedication

To Nicole, Serene, Luke and Ian, and Roger Corman.

Acknowledgements

First off, I'd like to acknowledge my Advisor, Dr. Robert Bass, for the advice, mentorship, support, and for giving me a frame of reference to use someday when I'm in charge of a team. Also, thanks to my thesis committee, Drs. John M. Acken and John Lipor.

Special thanks go to Shiva Poudel and Jamie Kolln from PNNL, without whose support I'd probably never have gotten the system up and running, let alone to its current state.

Along with Dr. Bass, I'd like to thank Dr. Siderius and the ECE department for giving me the opportunity to dust off my old film degree and finally use it for something valuable.

Tylor, I hope we get a chance to work together again some day. Midrar, I could put a thousand things here, but: thanks for the tea. It's always the little things you think back on, right? Mohammed, I don't know if you remember our talks about FRESP way back in the day, but those were one of the first times I felt really confident about my ability to be an engineer, so thanks. Shahad, thanks for the homework tips, and know that, in spite of my attitude, I was happy to help with the lab troubleshooting. To the numerous other GRAs, URAs, and capstone teams I've worked with, good luck with all your endeavors.

To my mom and step-dad, thanks for making sure I made it this far.

To Nicole, Serene, Luke, and Ian, you're the reason I'm here. Thanks.

Table of Contents

List of Tables

List of Figures

Acronyms

ADMS Advanced Distribution Management System

BIS Battery-Inverter Systems

CDTA Central Distributed Trust Aggregator

CIM Common Information Model

DCM Distributed Control Model

DER Distributed Energy Resource

DERMS DER Management System

DTM Distributed Trust Model

DTMC Distributed Trust Model Client

EGoT Energy Grid of Things

ESI Energy Service Interface

GO Grid Operator

GSP Grid Service Provider

MC Model Controller

ME Modeling Environment

PNNL Pacific Northwest National Laboratory

SPC Service Provisioning Customer

1 Introduction

1.1 Problem Statement

The importance of openness and interoperability have been identified for the integration of Advanced Distribution Management Systems (ADMSs), including Distributed Energy Resource (DER) management systems. As new DER Management System (DERMS) are developed with these initiatives in mind, the need arises for a flexible, configurable and modular test system that provides interactions between the DERMS, a grid simulation, and a simulated Grid Operator (GO) in an open and interoperable way.

1.2 Motivation

The Modeling Environment (ME) is a subproject of the Energy Grid of Things (EGoT) project developed by the Portland State University's Power Engineering Group. The EGoT project has two primary goals:

1. Development of an Energy Service Interface (ESI)

2. Development of a Distributed Trust Model (DTM)

The ME is critical for developing and testing these goals. Bidirectional communication between a DER and its aggregator requires a "smart" DER: one that is equipped for external

communications and subject to protocol requirements such as those laid out in the ESI. Distributed Control Models (DCMs) are needed to provide an intermediary between the DER and aggregator: the DCM communicates with DERs by utilizing the IEEE 2030.5 protocol, and translates information to the format required by the aggregator. These DCM-DER combinations represent a prosumer entity called the Service Provisioning Customer (SPC).

The DER aggregator is represented by a Grid Service Provider (GSP). The GSP is a business entity that is contracted with both the GO and SPCs. The purpose of the GSP is to provide grid services to the GO by sending messages to DERs in aggregate. The GSP contracts with SPCs to register their DERs, and communicates with DCMs for registration and dispatch purposes. These interfaces are governed by the rules set forth in the ESI. The system that performs these aggregation and dispatch functions is called the EGoT DERMS.

The DTM consists of the Distributed Trust Model Client (DTMC), located with the DCM, and the Central Distributed Trust Aggregator (CDTA), located with the EGoT DERMS. These work in tandem to provide the GSP with information as to the trustworthiness of individual SPCs.

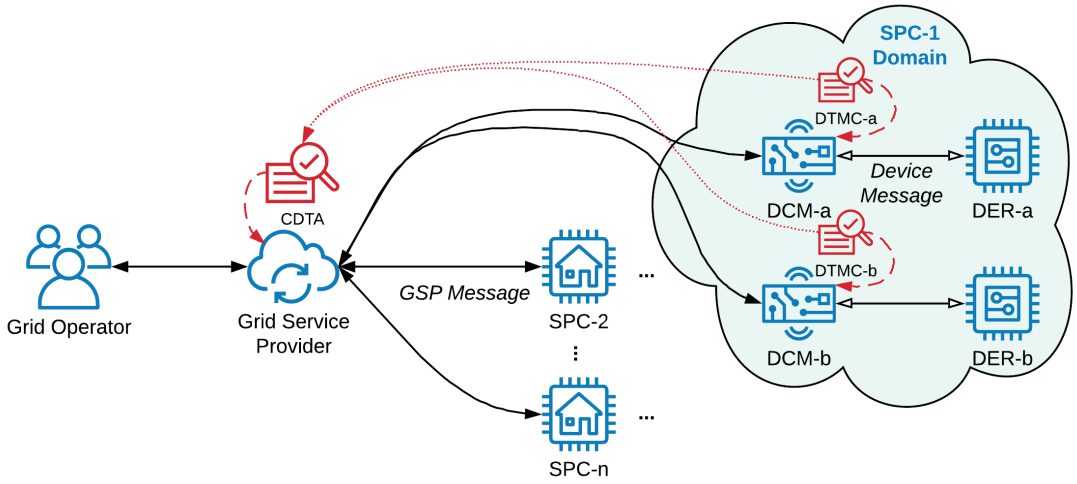

Figure 1.1: The GSP and its interactions for aggregation purposes. The GSP interacts with a GO and multiple SPCs. Each SPC may contain one or more DER with attached DCMs. The DCMs facilitate communication between the DER and GSP. The DTM (represented by CDTA and DTMC) establish trust for security purposes.

It is impossible to procure and install the thousands of physical DERs required for grid service dispatch in a laboratory setting, so DERs and their operating data must be simulated. As the DERs are simulated, a grid simulation system is also required as a testbed. To fully represent the effects of aggregated dispatch, this grid simulation must be able to process data from a variety of DERs. To serve as a DERMS testbed, it must also provide grid measurements to a GO that can determine whether a grid service ought to be dispatched at a given time. This GO can communicate these grid service dispatches to the DERMS, allowing the testbed to provide a full testing loop: the DERMS receives dispatch requests, which it fulfills by dispatching simulated DERs, which in turn will be reflected in the simulation, providing operating data on all aspects of the dispatch cycle.

1.3 Objectives of Work

This thesis presents the Modeling Environment: a configurable test system for DERMS that leverages the Pacific Northwest National Laboratory (PNNL) GridAPPS-D platform to provide an interactive grid simulation containing generic, controllable DERs. The ME's class-based Model Controller (MC) script interacts with the simulation by providing DER inputs and receiving grid state outputs. The input classes allow operating input data from any feasible type of DER to be converted to power signals and directed to the DER representations within the model. Integrating a new type of DER input–be it a new format, new protocol, or new DER entirely–requires only development of a single, standardized input class with no further major modifications to the system. The output classes provide logging as well as a simulated GO, which can be configured to send grid service request messages to a DERMS; this GO-DERMS API is designed to be rewritten to respect the protocol needs of the DERMS under test. The primary purpose of the system development was to provide a test platform for the EGoT DERMS. It also serves as a proof-of-concept for DERMS test applications using GridAPPS-D. Furthermore, the class-based structure and modularity of the system allow it to be used as a testbed for multiple DERMS with minimal extra development, which is a novel use of the GridAPPS-D platform.

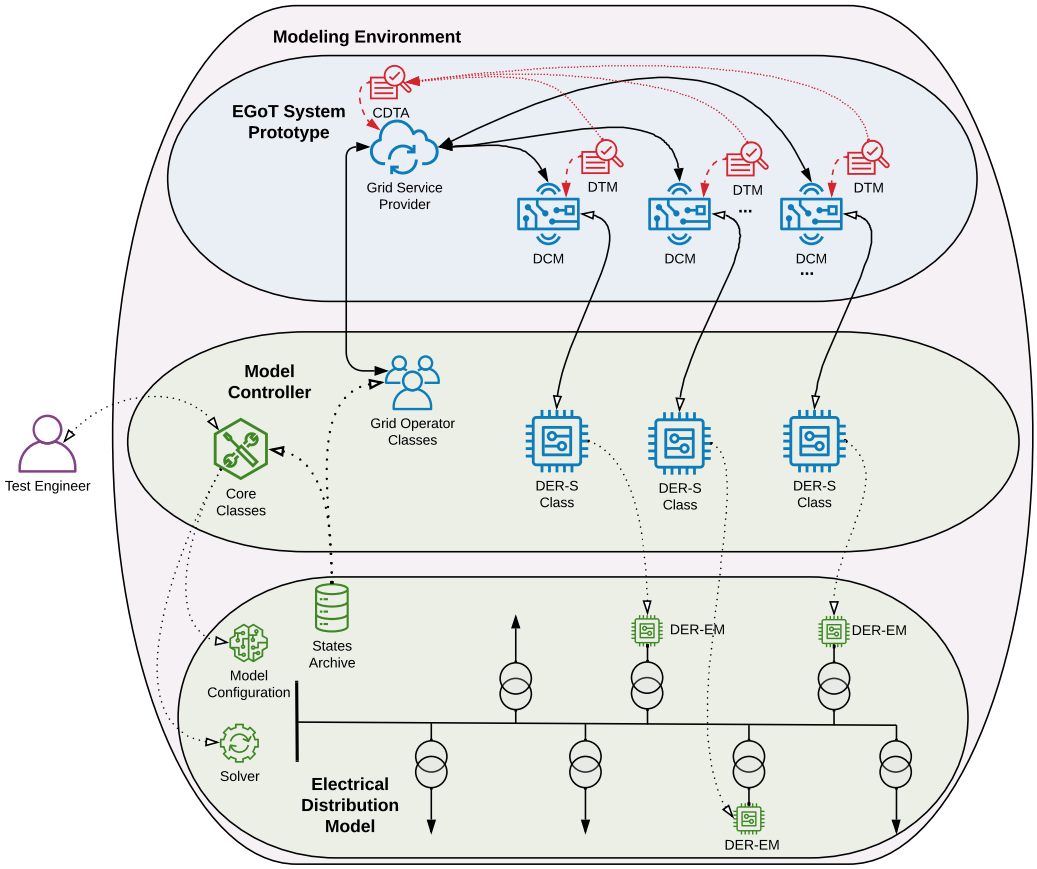

Figure 1.2: The proposed Modeling Environment. This is a high level overview of the proposed EGoT Modeling Environment and its interactions with a Grid Service Provider. The Test Engineer configures and runs a simulation using the Model Controller script. The MC core classes provide configuration and communication with the EDM. The GO classes interact with the GSP, which dispatches DER via DCMs. The DCMs interact with DER-S classes that translate DER input data into a format usable by the EDM. The DER-S classes provide this updated data to DER representations in the model (DER-EMs).

2 Background

2.1 DERs and Aggregation

According to the GMLC, “distributed energy resources include operationally responsive distributed generation, storage, and load [1].” A broader definition used by the Portland State University Power Engineering Group follows:“DERs are customer-owned generation, storage, and load assets that are grid-enabled. These resources are located behind a customer meter and are not traditionally directly managed by utilities” [2]. They have been identified as a desirable method to provide grid flexibility in the form of grid services. These services include frequency control, voltage control, and energy scheduling [3].

Individual DERs are small: a modeled smart water heater is rated at 4500 W. Grid services, however, are on the scale of hundreds of MW or higher. In order to provide sizable services with DERs, they must be operated in aggregate. A study by Kondoh, Lu and Hammerstrom demonstrated that when water heater operational considerations are taken into account, 33,000 electric water heaters would be required to provide a modest 2 MW of frequency regulation over a 24 hour period [4] .

The term “aggregation” can refer to one of two things. First, aggregation may refer to a system or application that is used to enact control on a large number of DERs. Second, a business entity that provides the service of DER aggregation can be called an “aggregator.”

An aggregator develops contracts with DER owners to provide some compensation in return for services provided by the DER; then, when grid services are deployed by utilities, the aggregator "dispatches" the DER resources in the locations and quantities necessary to provide the service [5].

2.2 DER Aggregation Methods

Some aggregation is handled manually, while some systems are automated. For example, companies such as Enernoc and Opower provide incentive information to their customers allowing them to make on-site decisions and manually adjust their equipment. The Ohmconnect service focuses on residential customers, but communicates alerts to the customers rather than taking on-site control. Cpower Energy Management supports automated Demand Response in which equipment is controlled via signals, whereas Tendril is a company that provides a platform allowing utilities to aggregate and control devices while providing data to both utilities and customers [6].

Automatic methods of DER control can be further divided into two methods: Direct Load Control and Service-Oriented Load Control. Direct Load Control is a simple scheme that allows the utility to control or disconnect a DER at any time for any duration; however, this has the downside of causing undue customer discomfort or dissatisfaction and decreases likelihood of new or continued enrollment. Service-Oriented Load Control is a Service Oriented Architecture in which utilities provide a list of services for DER controllers to interface with. The DER owner may then override the DER's participation in the service,

restoring full control to the DER owner [7].

The software solution that automates DER control is called a DER Management System, or DERMS. DERMS can plan and manage control of DERs in aggregate. However, DERMS are still an emergent technology, so the term "DERMS" may refer to decentralized software solutions, centralized software for system operators, systems for supply/demand balancing, ancillary service provision, peak congestion and voltage violation protection, stability, sales to wholesale or retail markets, local aggregation, and more. DER aggregator systems can thus be considered a type or DERMS, or a component or subsystem of a DERMS; but, not all DERMS are DER aggregators. A distinction may be made between "DER aggregators" and "utility DERMS" as different levels of a hierarchy, with aggregators handling communication with behind-the-meter DERs and utility DERMS using DER aggregators as well as other resources to provide the user with operating options [8, 9]. An Advanced Distribution Management System (ADMS) can be used by a utility or Distribution System Operator (DSO) to integrate DER aggregators into the distribution system [10]. The next generation of ADMS and EMS platforms are expected to take high DER deployment into account; this would integrate DERMS into these existing systems and could make a standalone DERMS redundant [11].

2.3 The Energy Grid of Things

A broader term describing the use of DERs to provide reliability and efficiency to the grid is the "Energy Grid of Things", or EGoT. An EGoT is both part of the energy sector and

communications sector, as it describes the methods by which signals are communicated to DERs along with their effects on the power grid. One example of an EGoT DERMS is being developed by the Portland State University Power Engineering Group. In this EGoT DERMS, Service Provisioning Customers (SPCs) own DERs and contract with a Grid Service Provider (GSP) for aggregation purposes, handled by the EGoT DERMS. The GSP communicates with a Grid Operator and helps to ensure the GO's operational objectives are met by managing the DER assets owned by SPCs. This management is achieved via communication with Distributed Control Models (DCMs) that serve as an intermediary between the DER and the GSP and translate messages into protocols useful to each recipient.

The PSU EGoT DERMS aims to demonstrate two major advancements: the integration of a Distributed Trust Model, and application of an Energy Service Interface. The DTM is a vector-based trust management system that provides a layer of security for the GSP-DCM interface [12]. The ESI between the GSP and SPC is a "set of rules that defines a bi-directional, service-oriented, logical interface that supports the secure communication of information between entities inside and entities outside of a customer boundary to facilitate various energy interactions between electrical loads, storage, and generation within customer facilities and external entities [13, 1]." The ESI rules define an "open and logical communications interface" that provides standard definitions for the boundaries, functions, and responsibilities of the GSPs and SPCs [14].

In order to demonstrate the effectiveness of the DTM and the usefulness of the ESI, PSU developed the EGoT DERMS. However, a test DERMS in the lab will not have the

grid awareness of an ADMS and its sensors, nor will it be possible to procure and deploy physical DERs in sufficient volume to test the DERMS against the local electrical grid. As such, processes must be designed to emulate a large number of DERs, and a grid simulation system must be designed that has the capability to test the EGoT DERMS including the DER emulators, using a robust grid model. Ideally, the grid states from this simulation should be used to determine grid service requests that are communicated to the GSP to test its ability to respond to grid anomalies and situations in which grid services would be warranted, as well as providing feedback data to the DERMS.

2.4 GridAPPS-D

GridAPPS-D is an open source, standards-based platform for smart grid application development. It is an attempt to move advanced distribution system development away from closed, protected architectures toward a more collaborative industry environment. This will allow development of a wider array of applications that will function in a broader set of environments, accelerating development towards the smart grid and freeing up resources to focus on novel value rather than integration [15]. GridAPPS-D is a novel platform, and it is beginning to find some use. The following section will review applications of GridAPPS-D found in literature as of May 2022.

2.4.1 Review of existing GridAPPS-D applications

The GridAPPS-D model database contains files in the Common Information Model (CIM) format, and the system can output GridLAB-D files. Since OpenDSS is able to export CIM

files and import a variety of file types, GridAPPS-D has found some use as a tool to convert to the GridLAB-D file type. This has been used for SXST files from CYMDIST [16] and Synergee models in the Microsoft Access database (.mdb) format [17].

Other researchers used GridAPPS-D to implement a novel distribution-level marginal price (DLMP) scheme derived from an optimal power flow framework. Measurements are taken and setpoints delivered to GridAPPS-D in this implementation [18]. An optimal energy dispatch strategy for distributed photovoltaic systems was also developed and implemented with GridAPPS-D acting as the ADMS [19]. Another optimal power flow application using Volt/VAr control was developed to demonstrate GridAPPS-D capabilities [20]. Another Volt/VAr control tool, the Coordinative Real-time Sub-Transmission Volt-VAr Control Tool (or CReST-VCT) has been developed, and a high level overview of how it could be implemented in GridAPPS-D was written [21]. A fault location, isolation, and restoration application was developed and implemented in GridAPPS-D as a"proof-of-concept for developing advanced applications in an ADMS environment using the GridAPPS-D platform" [22, 23].

The National Renewable Energy Laboratory is developing an ADMS test bed using industry standard protocols. This can be used to integrate and test GridAPPS-D applications, though few details are given in the literature as to how this is accomplished [24, 25] .

An application service was developed to integrate GridAPPS-D and SurvalentONE. SurvalentONE is a "SCADA, OMS, and DMS solution that allows effective operation, monitoring, analysis, restoration, and optimization of network operations." A major component of

this application was the development of an interface for the DNP3 protocol. This application bears some resemblance to the ME: a custom API analogous to the GO-GSP API allows the GridAPPS-D simulation to drive an external system by functioning as the distribution system of concern. This system then provides control messages back to the GridAPPS-D simulation. However, it differs in both architecture and purpose: SurvalentONE is not a DERMS, the application does not support DER aggregation, and the purpose of development was to test the DNP3 API rather than to develop a general purpose test bed that could utilize a variety of APIs [26].

3 Design Methodology

The following section provides a conceptual overview of the Modeling Environment, including the system architecture, GridAPPS-D functionality, and actor-based class design presented as a high level summary. The implementation of these concepts are presented in the Results section.

3.1 Modeling Environment Architecture

The ME is an electrical grid simulation platform designed to support testing of a DERMS and provide a means for analyzing effects of DER dispatch on the electrical grid. Figure 3.2 shows the components of the ME and its relationships with other parts of the EGoT project. I have selected GridAPPS-D as the Electrical Distribution Model platform. The EDM provides a means for modeling distribution systems and includes a grid states solver. A Model Controller configures, initiates, and coordinates simulations while providing systems that interact with the EDM input and output capabilities. The MC provides communications and processing between the EDM, inputs, and outputs. DER-Ss are classes within the Model Controller that retrieve DER input data from external sources and provide the data to their associated Electrical Model DER representations (DER-EMs); these input data may come from a model, an historical data archive, or communications with a hardware interface such

as a DCM. Outputs from the EDM are real-time grid state data. These are available to a simulated GO, which generates output logs and provides service request postings–and, potentially, feedback data–to the DERMS.

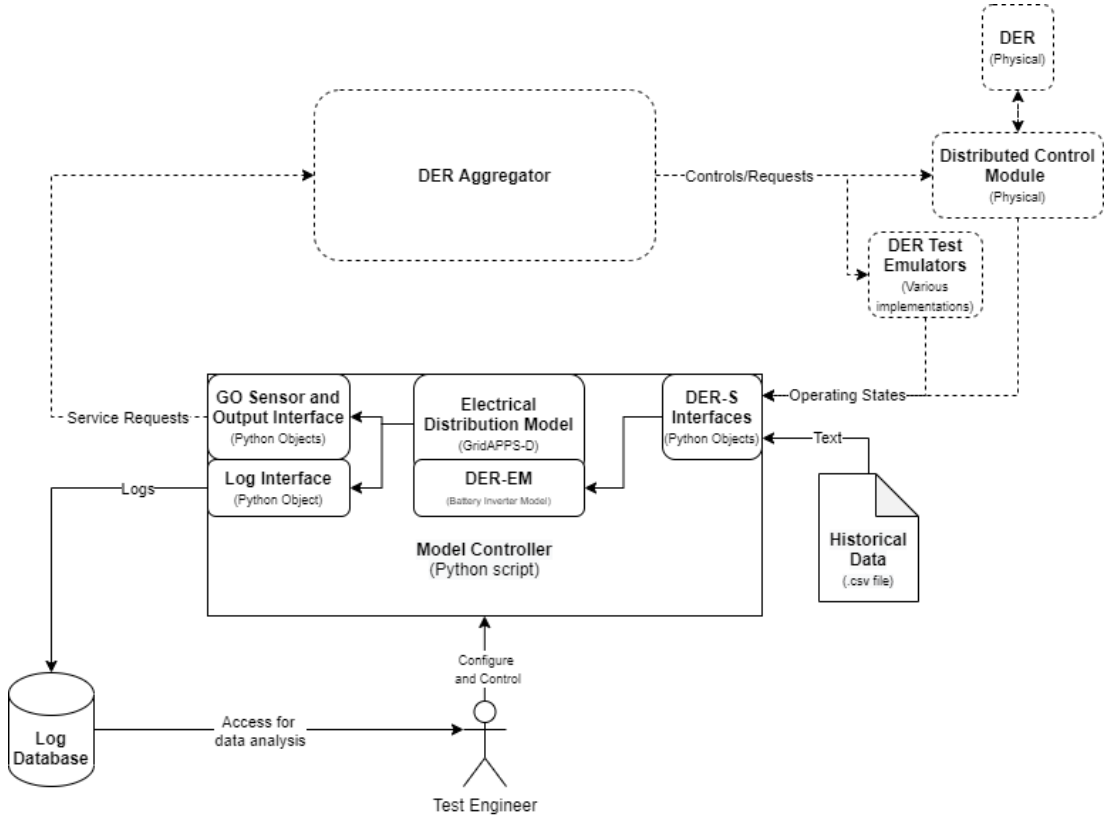

Figure 3.1: A simplified diagram of the Modeling Environment and its interactions. Objects with solid lines fall within the scope of the ME product. The ME will interact with other project objects, shown in dashed lines.

3.1.1 GridAPPS-D

The ME is built using the GridAPPS-D platform. GridAPPS-D is an open-source application development platform designed by the Pacific Northwest National Laboratory to provide an architecture by which applications can easily communicate with grid simulation programs.

The network measurement data in GridAPPS-D originates from GridLAB-D, which is a three-phase unbalanced distribution simulator [15].

The GridAPPS-D installation, and by extension the EDM, is located within a Docker container. Docker containers provide a lightweight alternative to virtual machines, allowing portability, reusability, and versioning. This allows GridAPPS-D to function regardless of platform, provides automatic updates to GridAPPS-D and its database, and allows the Test Engineer to easily install GridAPPS-D regardless of operating system, ensure it is using the newest version, or roll back to a previous version if necessary.

The MC is contained in a script external to the GridAPPS-D simulation; however, the MC and any other potential application can be packaged within the GridAPPS-D container for deployment purposes. GridAPPS-D provides a Python API library that facilitates communication between the script and GridAPPS-D via discrete "topics," which are communications channels designed for a specific purpose. For example, simulation input messages or database queries are sent to their own topics.

GridAPPS-D applications interface with a Blazegraph database, which is also included with GridAPPS-D within the Docker container. This database includes grid models in the Common Information Model (CIM) format. CIM contains alphanumeric identification codes for all components and measurements within this model. These codes are called mRIDs. mRIDs are used by the MC to direct inputs to the proper DERs or identify measurement points, for example.

During a typical GridAPPS-D simulation, the Test Engineer (TE) selects a model from

the database and sets a configuration including simulation start time, duration, and other options pertaining to the simulation. Upon execution, GridAPPS-D converts the CIM model to a GridLAB-D model and runs the simulation in real time while awaiting messages on the platform topics. GridAPPS-D also provides the ability to send regular timekeeping and measurement messages to the MC via callback objects.

3.1.2 Grid Models and the Electrical Distribution Model

The Blazegraph database contains several built-in grid models, such as IEEE test feeders, PNNL taxonomy feeders, and OpenDSS/EPRI circuits, all in the CIM XML format. These models can be run within the simulation directly. However, these models do not by default contain the controllable DER representations required to test input to the simulation. For testing purposes, DER-EMs must be added to the models. These DER-EMs should be generalizable to a variety of DER types to ensure the ME can simulate a wide variety of DERs, including newly developed or modeled resources.

The Grid Modernization Laboratory Consortium has detailed a standard method by which DERs can be modeled by using "battery equivalent models." This standard model with a battery interface has the ability to model a wide variety of DERs in a modular way with the specific purpose of enabling grid services [27]. CIM-based Battery Inverter System (BIS) models can serve as DER-EMs within the grid model. They can be easily controlled via the MC DER input functions, and can be operated to model a wide variety of DER loads.

BIS models can be added directly to an existing model in the database at required locations without going through the process of generating, converting, and ingesting a new

model. This is done using a program called CIMHub (link in Appendix.) Alternatively, new grid models can be created by the test engineer. They must be in the CIM format to be ingested into the database; other formats such as GridLab-D files need to be converted to CIM before ingestion. The BIS models may be added during model creation or using the script, as above.

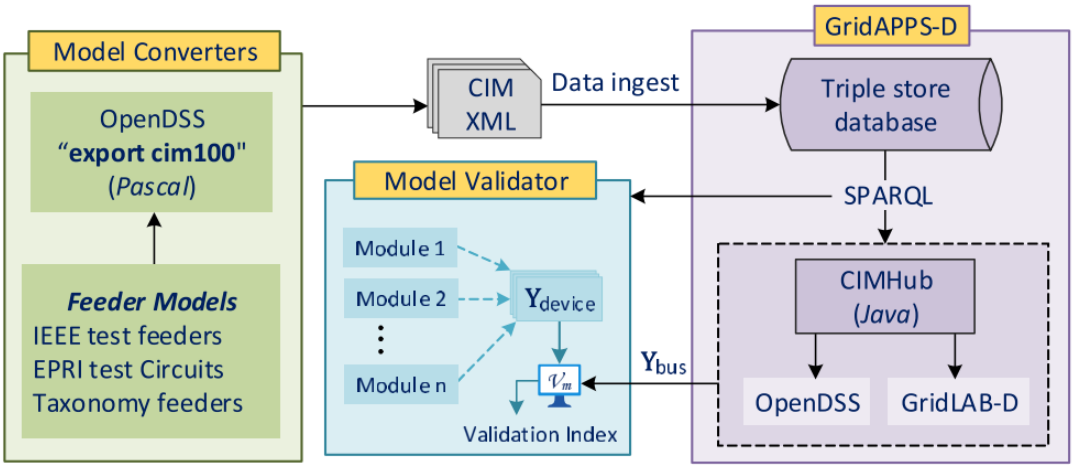

Figure 3.2: A diagram of the model ingestion process. Grid models generated in a variety of file formats can be converted to CIM XML by OpenDSS, which can be added to the GridAPPS-D database which contains several IEEE Test Feeders by default. Modifications to the models can be modified within the triple store database by using the CIMHub and Powergrid-Models scripts, such as the addition of Battery Inverter System models to be used as DER-EMs. Then, during the simulation run process, the model is taken from the database and converted to the GLM format for use by the simulation (as well as providing an OpenDSS file for validation purposes, which are unused by the ME [28].)

3.1.3 Model Controller Script

The MC is an object-oriented Python script that controls the simulation and provides interfaces for inputs and outputs, Figure 3.3. The class-based structure of the script allows the MC to be divided into several "actors" dedicated to individual tasks. For example, a class handles formatting and provides input messages to the EDM, while another class provides an interface between the MC and the DERMS.

GridAPPS-D assists in this implementation by providing "callback methods" that can be included in user-written classes, which then become "callback classes." These callback methods are automatically called by GridAPPS-D at specific times. For example: the EDMTimekeeper callback method is called every time a log message is received from GridAPPS-D. These log messages include a wide variety of data about the simulation operation, status, and errors. These also include incrementation messages tied to the internal simulator time keeping function. The EDMTimekeeper class parses these log messages and incrementation is detected. The EDMTimekeeper then calls a method to perform once-per-timestep functions for the MC. The EDMMeasurementProcessor functions similarly, receiving messages containing grid state measurements as they are delivered by the GridAPPS-D simulation. This occurs once every three seconds. Most MC functions are timed by the EDMTimekeeper "simulation timestep" of one second, except for measurement processing, which is timed by the "measurement timestep" of three seconds.

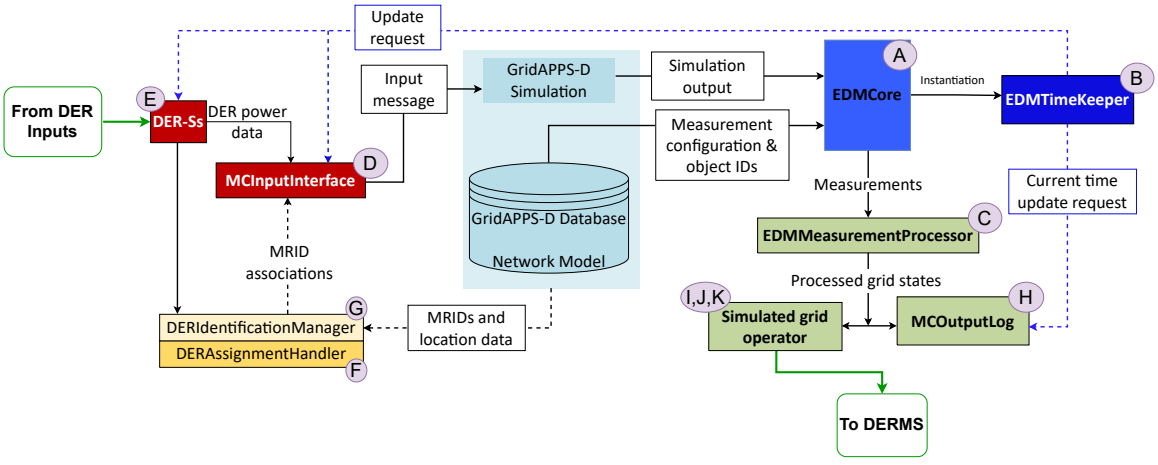

Figure 3.3: An overview of the Class-based architecture of the Model Controller.

.

3.1.3.1 Electrical Distribution Model Classes

The following are the callback classes that make up the core of the EDM-MC communication pipeline, as depicted in Figure 3.3.

- **EDMCore** - Manages configuration and startup of the script as well as class instantiation. See Fig. 3.3(A).
- **EDMTimekeeper** - Parses GridAPPS-D's operating logs to synchronize the MC's simulation time with GridAPPS-D's internal timekeeping. Also, sends regular method calls to all objects that need to be updated once per second (such as the MCInputInterface). See Fig. 3.3(B).
- **EDMMeasurementProcessor** - Once every measurement timestep (three seconds) this object receives updated grid states from the EDM, which it then organizes, translates headers from mRIDs to a human readable format, appends additional information queried from the model database, and sends to the MCOutputLog and GOSensor objects. See Fig. 3.3(C).

The **MCInputInterface** (Fig. 3.3(D)) and the **MCConfiguration** (not depicted) classes interact with the EDM; however, they do not have callback functions and are not callback classes. The MCInputInterface class translates DER operating data from the DER inputs (via DER-S classes) into properly formatted messages sent to the EDM input topic. These inputs are delivered once per timestep on prompting by the EDMTimekeeper. The MCConfiguration attributes should be customized by the user as they contain file directories,

paths, the list of active DER-Ss for the simulation, and any other global configuration data necessary for the script to properly utilize inputs and generate outputs.

3.1.3.2 Distributed Energy Resource Classes

The DERs in the electrical model (DER-EMs) are generically represented as battery- inverter system (BIS) models. BIS models provide sourcing, sinking, and ramp rate controls, which can be adjusted to represent a wide variety of DERs. The control inputs to DER-EMs come from simulated DER input classes (or **DER-Ss**), Fig. 3.3(E). Each DER-S may retrieve inputs corresponding to one or multiple DERs from external simulations, hardware, or data. These DER-Ss perform any processing tasks required to convert these inputs into electrical operating data; specifically, power levels for each DER input along with their unique identifier.

DER-Ss are assigned to DER-EMs at the proper topological location at the start of the simulation. Each DER-EM within the model has a unique control mRID number, which provides a location for inputs to be delivered via the GridAPPS-D messaging function. Each DER-EM location within the grid model can also be queried. However, component mRIDs are randomly generated by CIMHub as they are added to the model. Emulators or hardware that provide inputs to the DER-S will not have foreknowledge of these mRIDs. However, each DER-S will have its own unique identifiers for each DER input. These will be provided by the inputs to the DER-S or generated within the respective DER-S class, as necessary. Each DER-S will also require location data for each of its representative DERs. These locations need to be provided to each DER-S from the DER inputs; they could be the

bus the DER should be assigned to, or a member of a more complex topological grouping. Topological processing is handled in **GOTopologyProcessor**, if necessary.

Using these identifiers and locational data, DER-Ss can be assigned to DER-EMs. The **DERAssignmentHandler** class (Fig. 3.3(F)) automates the task of assigning DER inputs to DER-EMs at the proper location in the model. Then, during the simulation start-up process, the DERAssignmentHandler determines how many DER-Ss are being used by the system and queries their identification and locational data. Using these data, it assigns each DER-S input's unique identifier to a DER-EM mRID existing at the proper node in the grid model.

The data associating DER-S identifiers, mRIDs, and locational data are stored within the **DERIdentificationManager** class, Fig. 3.3(G). The inputs to the DER-EMs are handled by the **MCInputInterface**. The MCInputInterface retrieves the electrical state output data from each DER-S and its respective identifier at each timestep, converts the electrical data to a message format usable by the EDM, queries the DERIdentificationManager to replace the DER-S identifier with the respective target mRID for the DER-EM, then sends the messages. These messages are received by the EDM, which updates the BIS models to reflect the new electrical states communicated by the DER-S. These changed grid states are then reflected in the measurements read by the **EDMMeasurementProcessor**.

3.1.3.3 Output Classes

The **EDMMeasurementProcessor** retrieves grid state measurements once every three seconds. After reciept, it appends the locational association data retrieved from DERIdentifi-

cationManager along with other identifying data queried from the model database. This processed measurement data is sent to two objects at each timestep:

- **MCOutputLog** - Writes the measurements to a log once per timestep for later analysis. Reference Fig. 3.3(H)
- **GOSensor** - Represents the Grid Operator's sensing system and decision-making process. A fully implemented GO could filter the measurements as necessary, compare them to user-defined thresholds, and make determinations as to whether a new grid service should be requested from the DERMS. A simpler implementation may just read grid service requests from a file and initiate them at times designated by the test engineer. Reference Fig. 3.3(I).
  - **GOPostedService** - Grid service requests are packaged into objects of the GOPostedService class. Each of these objects contains attributes holding the service name, group id, type, interval, power, ramp rates, and price. These attributes provide all the information necessary to generate service request messages for a particular GOPostedService object; these objects are maintained in a list in GOSensor and accessed by the GOOutputInterface. Fig. 3.3(J).

Finally, **GOOutputInterface** provides an interface between the MC and DERMS by polling the GOSensor grid service list, packaging grid service requests and feedback data into messages in the proper protocol, and sending them. This interface would be modified

for the required needs and protocols if a system other than the EGoT DERMS were used. See Fig. 3.3(K).

4 Implementation

4.1 Implementation

The following sections describe the implementation of the EDM, MC, DER-Ss, and the Grid Operator representation contained within the MC.

4.1.1 Electrical Distribution Model Implementation

The EDM provides a simulator for a node-based grid model (Fig 4.1) consisting of generation, loads, and distribution; and, provides methods to model and control transformers, breakers and switches, reactive power compensation, etc. The EDM was developed using GridAPPS-D. GridAPPS-D provides the simulation system, a database of grid models, and a communications bus, which allows inputs and outputs from the system to an external program. As such, GridAPPS-D provides the EDM for the ME system. A *gridappsd* python library allows development of a Model Controller script, which interacts with the EDM. The GridAPPS-D system is contained within a Docker container.

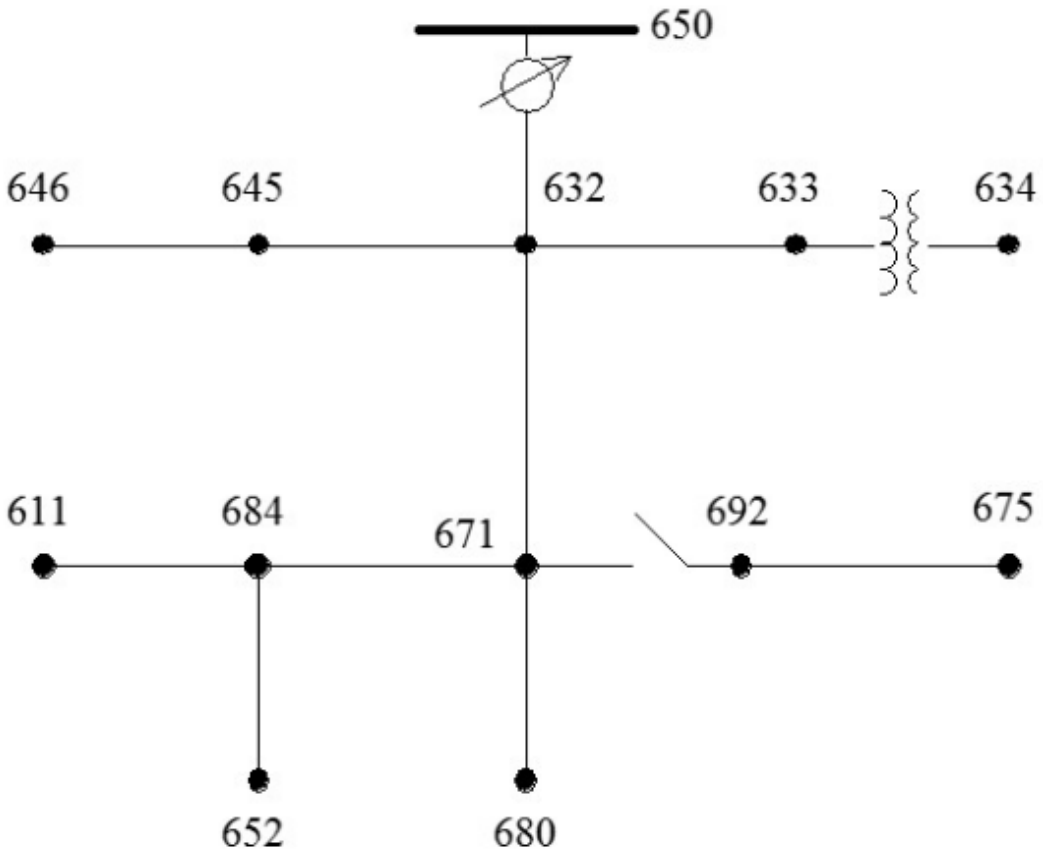

Figure 4.1: The IEEE 13-node test feeder. This is an example of a node-based model [29].

4.1.1.1 Model Database

The GridAPPS-D database contains models in the CIM format. When a simulation is started, GridAPPS-D automatically converts the selected CIM model into a GLM file as part of the startup process [28]. There are five steps to the process of adding models to the Blazegraph database:

1. The TE generates a model in a suitable format.

2. The TE converts the model format to CIM using OpenDSS (or a comparable software package.)

3. The TE uses CIMHub to ingest the CIM model to the database.

4. Using CIMHub, the TE adds measurement points to the model in the database.

5. The TE uses CIMHub to validate the model, as necessary.

CIMHub also provides a set of utility functions that allow the TE to add objects such as houses and DERs to a model in the database, along with the requisite measurement and control points. DER-EMs in the form of BIS models are added to particular nodes in the model and assigned unique mRIDs, allowing them to be associated with DER-Ss in the DER assignment process within the MC.

4.1.1.2 DER-EM Implementation

DERs may function as either load, source, or both. Emulation of thousands of DERs over time is a complex and computationally-intensive task; furthermore, hard-coding the simulator with these emulation processes would unduly couple those specific DER types to the simulator, complicating the process of simulating new types of DERs or modifying the profiles of existing ones with updated models. For this reason, DER-EMs do not contain time-function DER modeling, and instead are generic DERs that respond to control inputs. This allows their modeling to be offloaded to external scripts or to be drawn directly from hardware or sensors. The DER-EMs then update their parameters to reflect changes in these external entities.

DER-EMs are implemented in the EDM using BIS models. BIS models exist in the CIM standard and can be controlled in the simulation using input messaging topics within

GridAPPS-D. The MC generates DER-EM control messages based on input data provided by DER-Ss. Each DER-S receives DER input data from an external file, emulator, or system; these input data may represent one or many DERs. Each DER-S is assigned to an appropriate number of DER-EMs automatically during the simulation startup process. Location data provided by DER inputs is matched to location data within the model, allowing DERs to be assigned to the proper topological location. Commonly, this location data will be the bus within the model the DER should be assigned to, but more complex topological processing and assignment can be implemented within the GOTopologyProcessor class (Fig 4.10).

4.1.1.3 EDM-MC Communications

Communications to or from the EDM are handled by the GridAPPS-D message bus. The *gridappsd* Python library simplifies communications by packaging requests in intuitive functions. For example, the *DifferenceBuilder* object contains an "add difference" method that accepts an mRID and power values as arguments. This function automatically packages the request and sends it to the proper GridAPPS-D API. Another way the library automates communications is by providing “callback functions” or “callback methods” within classes. These functions are called automatically by the simulation; for instance, when the simulation has updated measurements of grid states, or when a simulation timestep has elapsed. Though the methods are called by the system, the contents of these callback methods are user-defined. This allows for automation of the MC: processing measurements into a format useful for logs can be performed automatically as the measurements come in, or on-timestep functions can be handled by a callback method that is called once per timestep, for example.

4.1.2 Model Controller Implementation

The MC is the programmed implementation of the ME and is responsible for configuration, execution, and I/O features of the simulation. It is a class-based python script, with each class encapsulating some functions of the system into "actors." Organizing the system into a series of class-based actors makes the system more easily extensible: inputs from new DER emulators and controllers, different logging schemes, varied Grid Operator functions, and external communication schemes can be added, removed, or reconfigured within their respective classes without requiring substantial refactoring outside of the class.

The classes are roughly organized into three groups: core classes, input classes, and output classes.

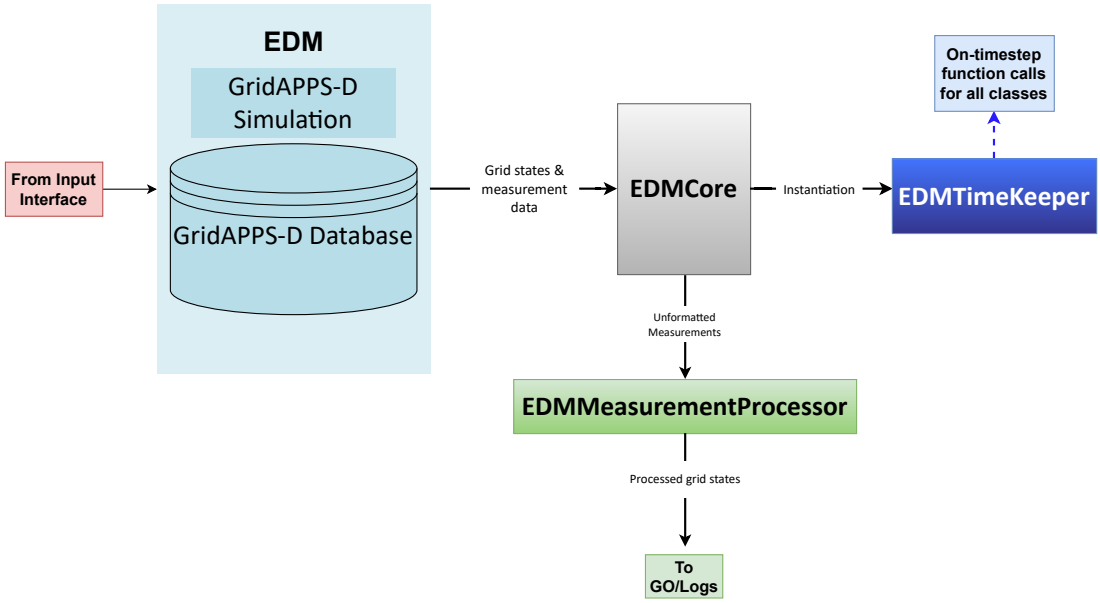

Figure 4.2: Core classes and their interactions

- Core classes govern the simulation configuration and startup, and also include classes with “callback methods,” which are methods that are automatically called by the

simulation on startup, once per second, or once per measurement update (roughly three seconds). These callback classes are EDMCore, EDMTimekeeper, and EDM-MeasurementProcessor. MCConfiguration is not a callback class, but rather is a core class that allows the user to configure system settings within its attributes.

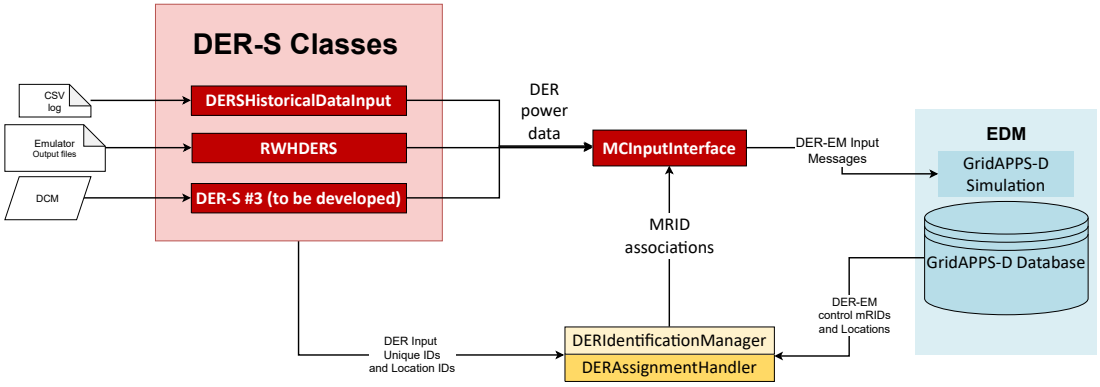

Figure 4.3: Input classes and their interactions

- Input Classes provide all functions necessary to read DER inputs from external sources and translate them into updated DER-EM states. DER-EMs can be added to the grid model and assigned measurement and control mRIDs prior to the simulation. DER-S classes are added and enabled as required by the needs of the test. The input classes handle communication between the DER-S and MC, assign the unique identifiers of each DER input to the proper DER-EM mRIDs, keep track of the DER-S to DER-EM association data for inputs and logging, and convert processed state data from the DER-Ss into input messages for the EDM. These classes are the DER-S interfaces (DERSHistoricalDataInput, RWHDERS, and any other DER-S developed in future), DERAssignmentHandler, DERIdentificationManager, and MCInputInterface.

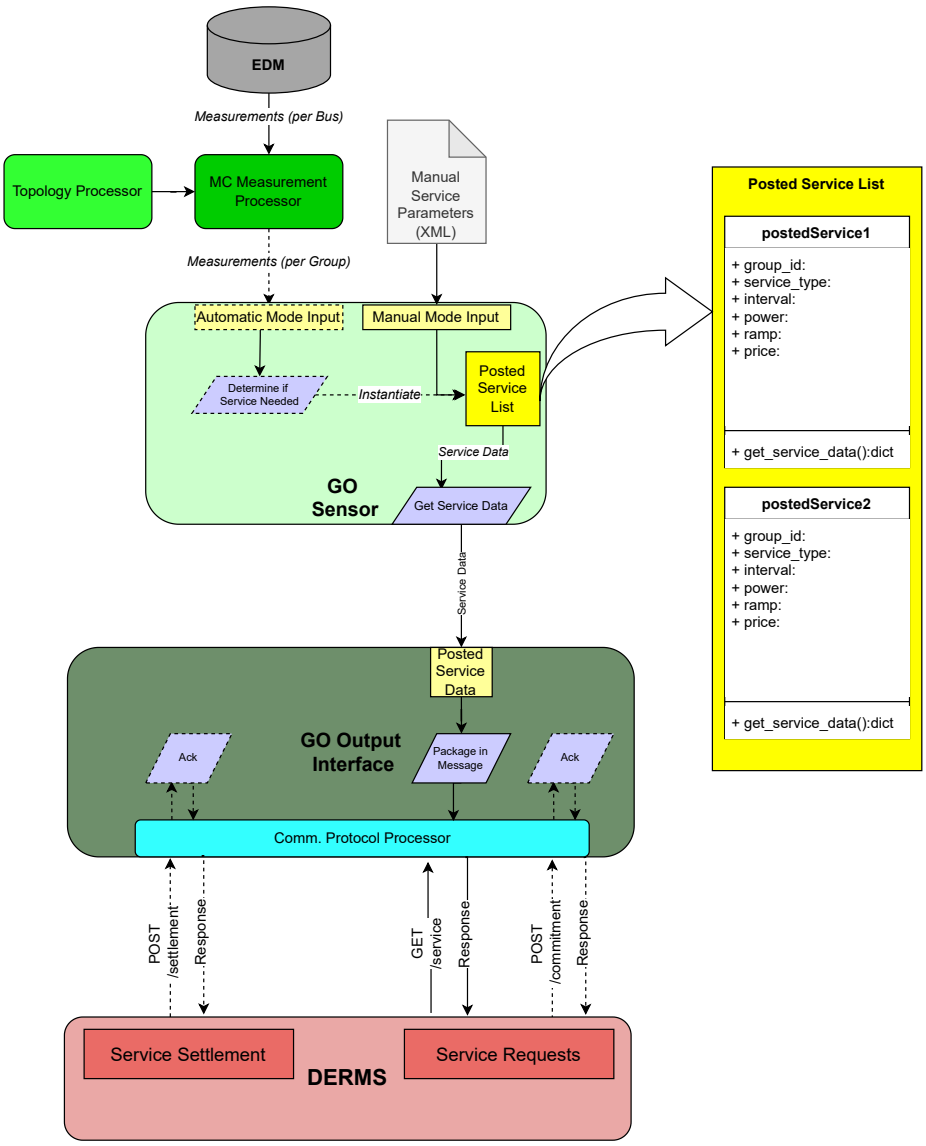

Figure 4.4: The output classes and their interactions. Classes shown are planned (dashed lines) or implemented (solid lines).

- Output classes take the formatted grid state data from EDMMeasurementProcessor and send it to logging or simulated Grid Operator actors. The logger writes the data to a file for later analysis. The Grid Operator is a simulated actor that functions in one of two modes. The first mode, “automatic mode”, responds to grid state changes by requesting grid services from the DERMS when user-programmed thresholds are

met. The second mode, “manual mode,” does not retrieve grid measurement updates; instead, it generates grid service requests based on an XML input file containing service request times and parameters. Manual mode is currently implemented to support testing of the EGoT DERMS; automatic mode is a potential extension of the system to support future research.

Grid service requests are objects of the GOPostedService class, each of which contains all necessary information for one grid service request along with methods to access this data. These requests are provided to the DERMS via an interface class: GOOutputInterface. The GOOutputInterface polls the list of posted service objects once per timestep and accesses the data in any new entries to package them into the proper message format.

Finally, the GOTopologyProcessor allows for topological identification. For instance, if the DERMS is configured to register DERs as members of a group of buses rather than a single bus, the topology XML file can be updated to reflect each bus membership in a group, allowing proper assignment, association, and feedback to the DERMS. As with the GOSensor automatic mode of operation, this functionality is not yet required by the EGoT system and as such has not been implemented. Plans for its implementation have been developed and will be presented in following sections.

The following class diagrams outline each class in terms of attributes and methods. Note that this is not a full outline of the system but rather an overview. A full class outline is available in the GitHub repository (see Appendix.)

4.1.2.1 Model Controller Core Classes

The core classes include EDMCore, EDMTimeKeeper, EDMMeasurementProcessor, and MCConfiguration. Class diagrams for each are shown in Figure 4.5.

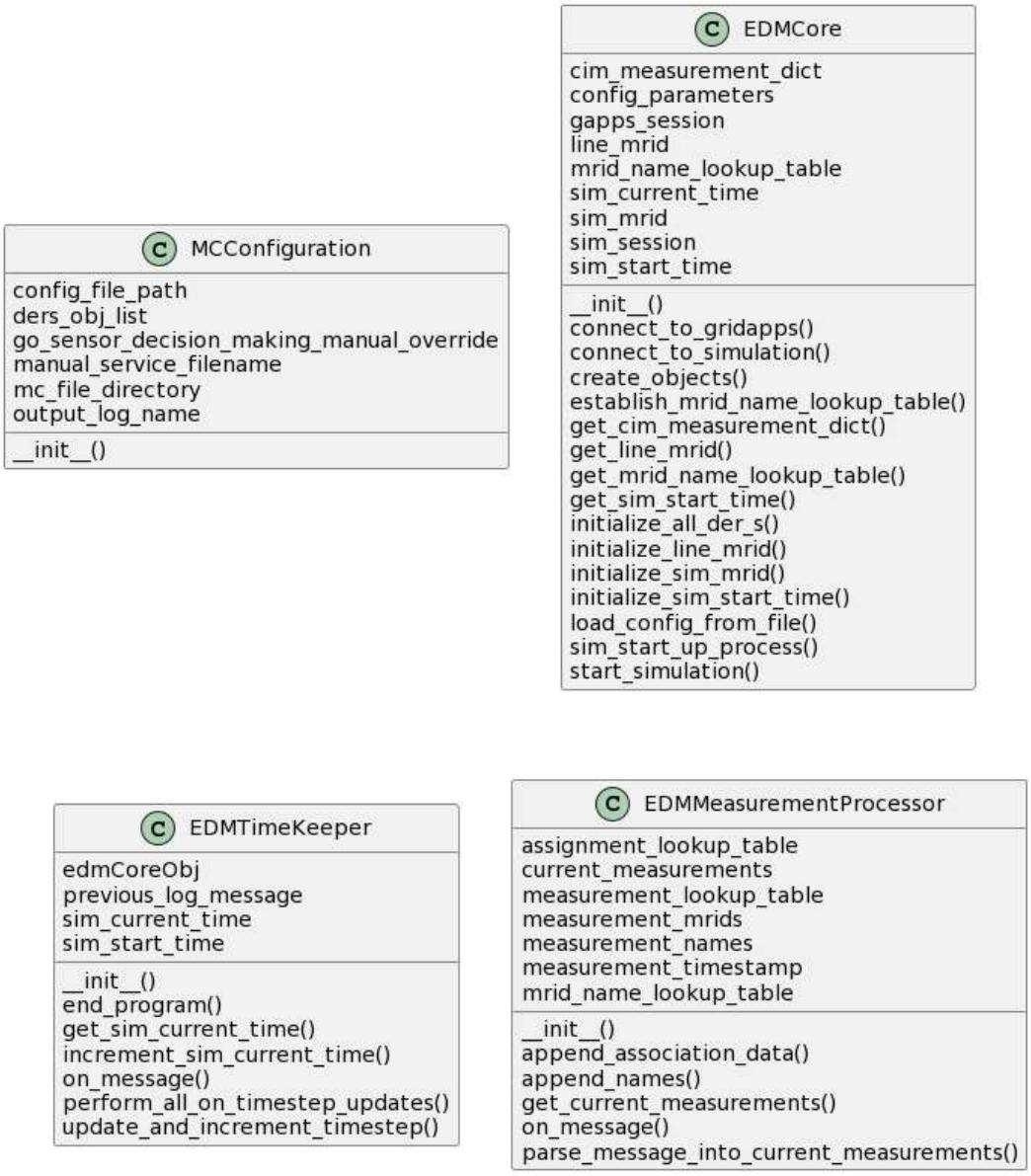

Figure 4.5: UML class diagrams of the callback classes.

The **EDMCore** class provides EDM configuration, GridAPPS-D connection, and sim-

ulation startup functionality. It also instantiates the other callback classes that are used to handle the EDM to MC communications.

The **MCConfiguration** class contains configuration settings for the Model Controller in its attributes, making it convenient for the user to configure paths, file names, and active DER-S classes.

The **EDMTimeKeeper** callback class receives log messages which are parsed to isolate the simulation timestep incrementation messages, which occur once per second. On receipt, the class updates the global simulation time for all classes that require it. This class also instructs actors to call their update methods if they are meant to do so once per timestep (such as the MCInputInterface), rather than once per measurement (such as the MCOutputLog)

The **EDMMeasurementProcessor** callback class operates once every three seconds. GridAPPS-D sends a message to this class containing a timestamped dictionary including measured quantities from the simulation. These measurements are keyed by mRID; they do not include human-readable information such as component names, locational data, etc. This class draws said information from lookup tables generated by DERIdentificationManager during startup, then reformats the message into a single "row" of data for the logs, and sends it to MCOutputLogs. If the GO is updated with an automatic mode of operation, this data will also be provided to the GOSensor class for decision-making and feedback purposes.

4.1.2.2 Model Controller Input Classes

The Input classes include the DERIdentificationManager, DERAssignmentHandler, and MCInputInterface. Also included are DER-S classes which handle inputs to the system.

Two examples of DER-S classes are provided: the Resistive Water Heater DER-S class (RWHDERS), and a general-purpose logged input data processor class, DERSHistorical-DataInput. Class diagrams for each are shown in Figure 3.6.

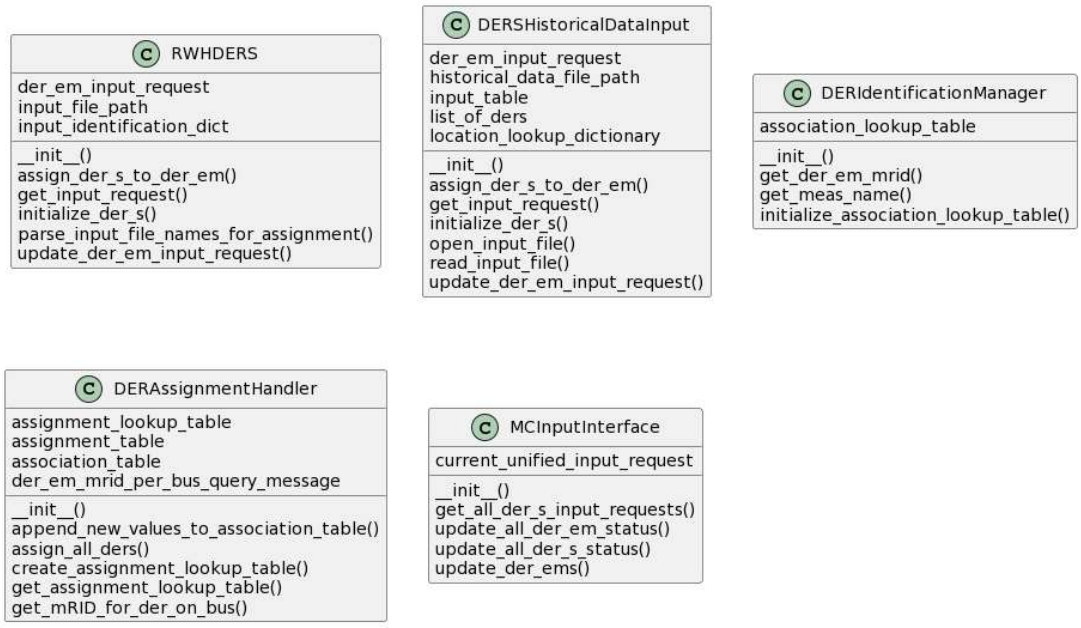

Figure 4.6: UML class diagrams of the input classes.

The Resistive Water Heater DER-S class **RWHDERS** is one example of a DER-S interface within the ME. This class reads emulated water heater values from an input buffer, which in the current configuration is a folder containing CSV files representing each DER that are updated on a running basis by a DER emulator. This DER emulator is a subsystem of the EGoT DERMS developed for test purposes. It simulates the operation of water

heaters over time. Once per timestep, RWHDERS reads the input files, parses their file names for identification, their contents for electrical states data, and provides that data to the MCInputInterface.

The **DERSHistoricalDataInput** class is a DER-S that uses timestamped historical data from a formatted CSV file. The input CSV provides unique identification data for each DER in the header, as well as location information (such as a bus) for each DER. This allows the DER inputs to be assigned to DER-EMs.

Time	DER1_mag	DER1_loc	DER2_mag	DER2_loc	DER3_mag	DER3_loc	DER4_mag	DER4_loc
1570041118	0	632	0	633	1000	634	1000	645
1570041119	0	632	1000	633	0	634	0	645
1570041120	0	632	1000	633	0	634	0	645
1570041121	0	632	1000	633	0	634	0	645
1570041122	0	632	1000	633	1000	634	1000	645
1570041123	0	632	1000	633	1000	634	1000	645
1570041124	0	632	1000	633	1000	634	1000	645

Table 4.1: Example data from a DERSHistoricalDataInput CSV file. Timestamps are in unix time format. Each pair of columns serves a single DER Input; DER1_mag and DER1_loc are the power magnitude (in Watts) and the locational identifier (or bus, in this case). The DER Input's unique identifier is taken from the header, and in this case is "DER1_mag". The unique and locational identifiers are used for assignment.

DERIdentificationManager class provides lookup tables for all association data in the simulation. "Association data" refers to dictionaries that correlate each DER input's unique identifier with its associated DER-EM control mRID. Identification data are used primarily by the MCInputHandler to send input messages to the correct DER-EM, or by the measurement processor to make the measurements more useful to the test engineer or GO.

The **DERAssignmentHandler** class manages the creation of the DER association lookup table during startup. The role of the assignment handler is to query the model within the GridAPPS-D database for DER mRIDs and location data, associate them with

provided unique identifiers for each DER-S, and provide that data to the DERIdentification-Manager in the form of a table. See Figure 4.7

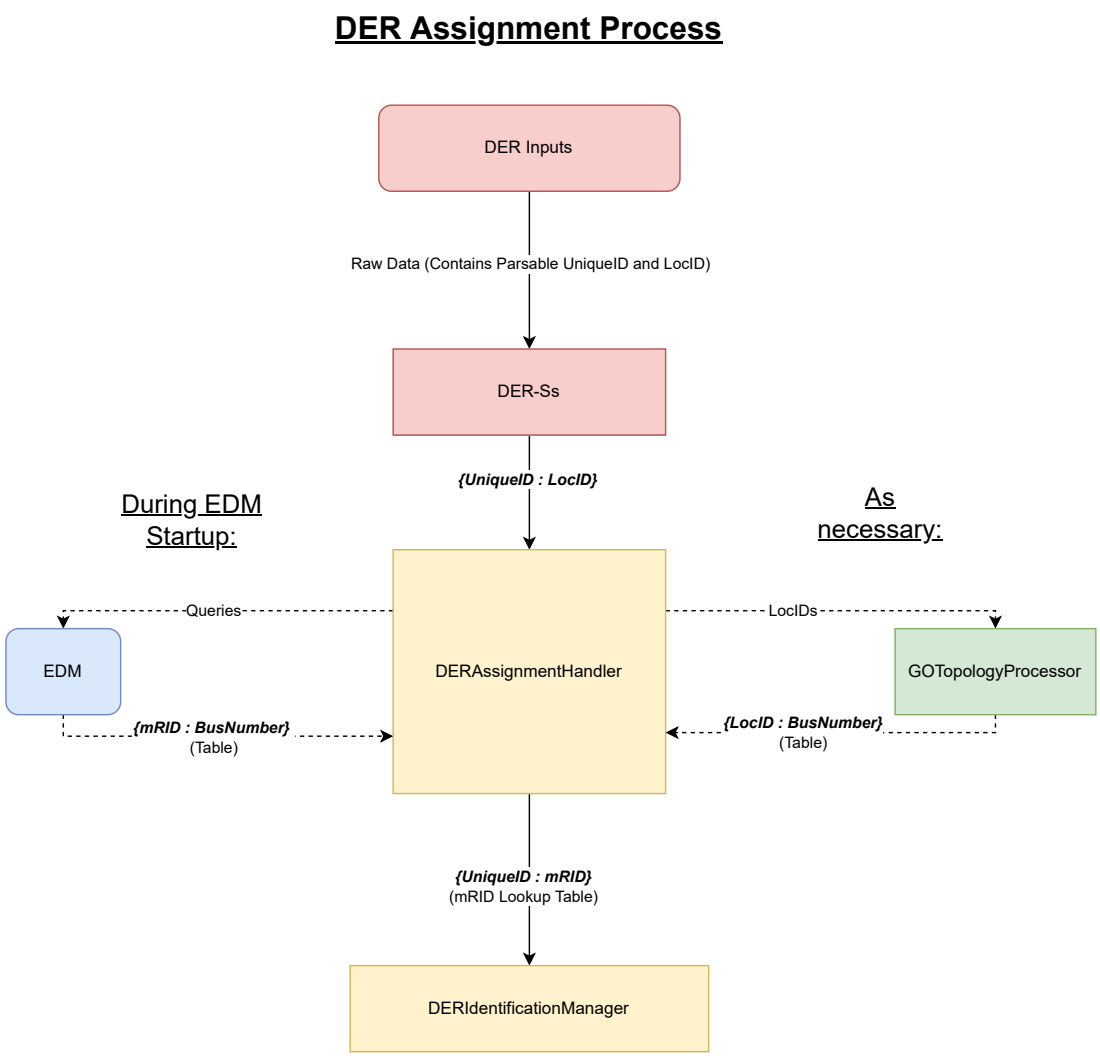

Figure 4.7: The DER assignment process. DER Input unique and locational identifiers are used to assign a DER input to a DER-EM by its control mRID.

The **MCInputInterface** class injects updated DER states into the model. Each timestep, it retrieves updated data from each DER-S, packages it into formatted messages, and delivers the messages to the EDM via the GridAPPS-D *DifferenceBuilder* function.

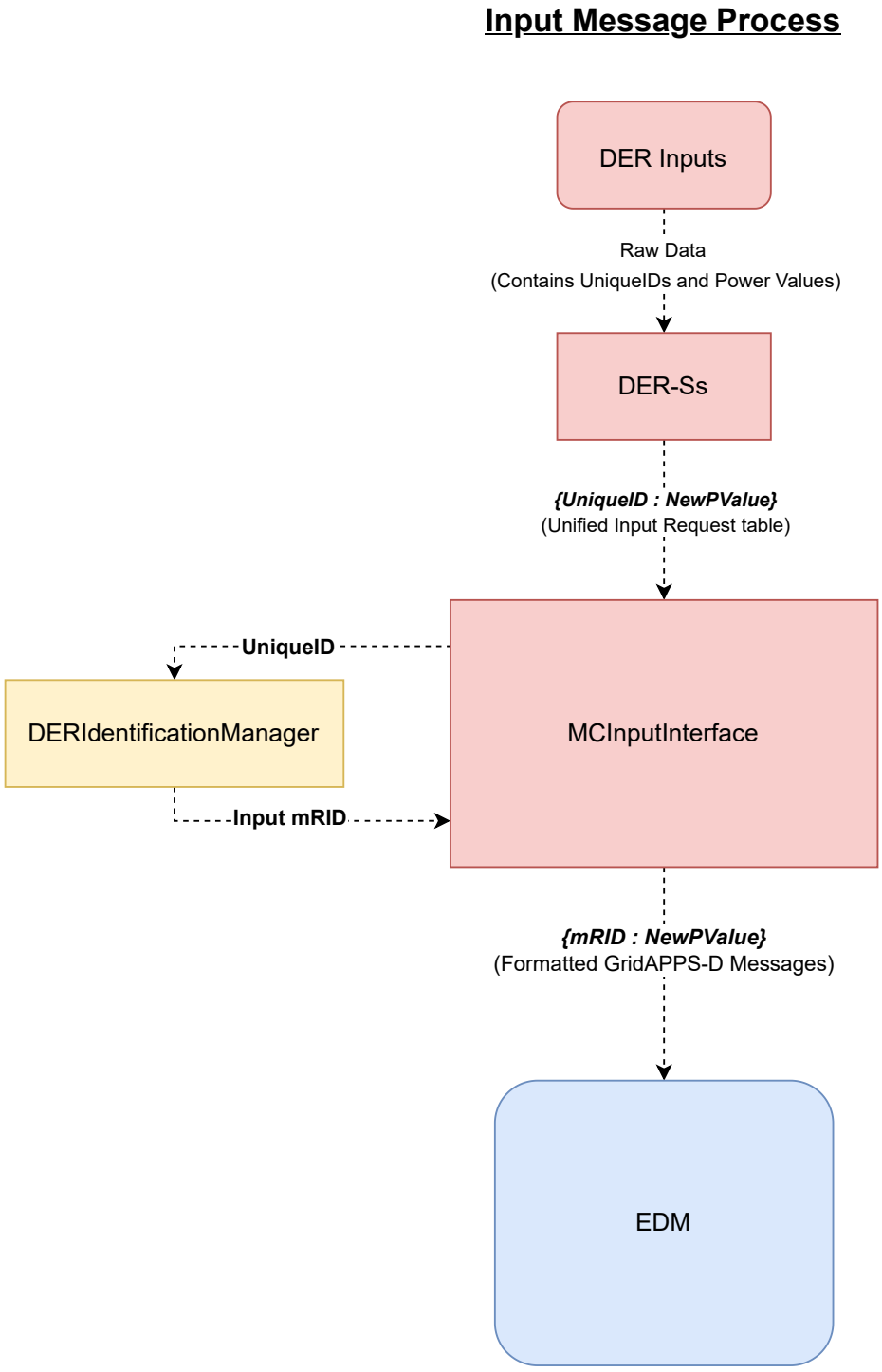

Figure 4.8: Input Message process including DER association lookups. After assignment, the association data is held within the DERIdentificationManager, automating the process of looking up DER-EM control mRIDs.

4.1.2.3 Model Controller Output Classes

The Output classes include GOSensor, GOOutputInterface, MCOutputLog, and GOTopologyProcessor. Also included is GOPostedService, which is unique in that it is the only class that may be instantiated into more than one object. Class diagrams for each are shown in Figure 4.9.

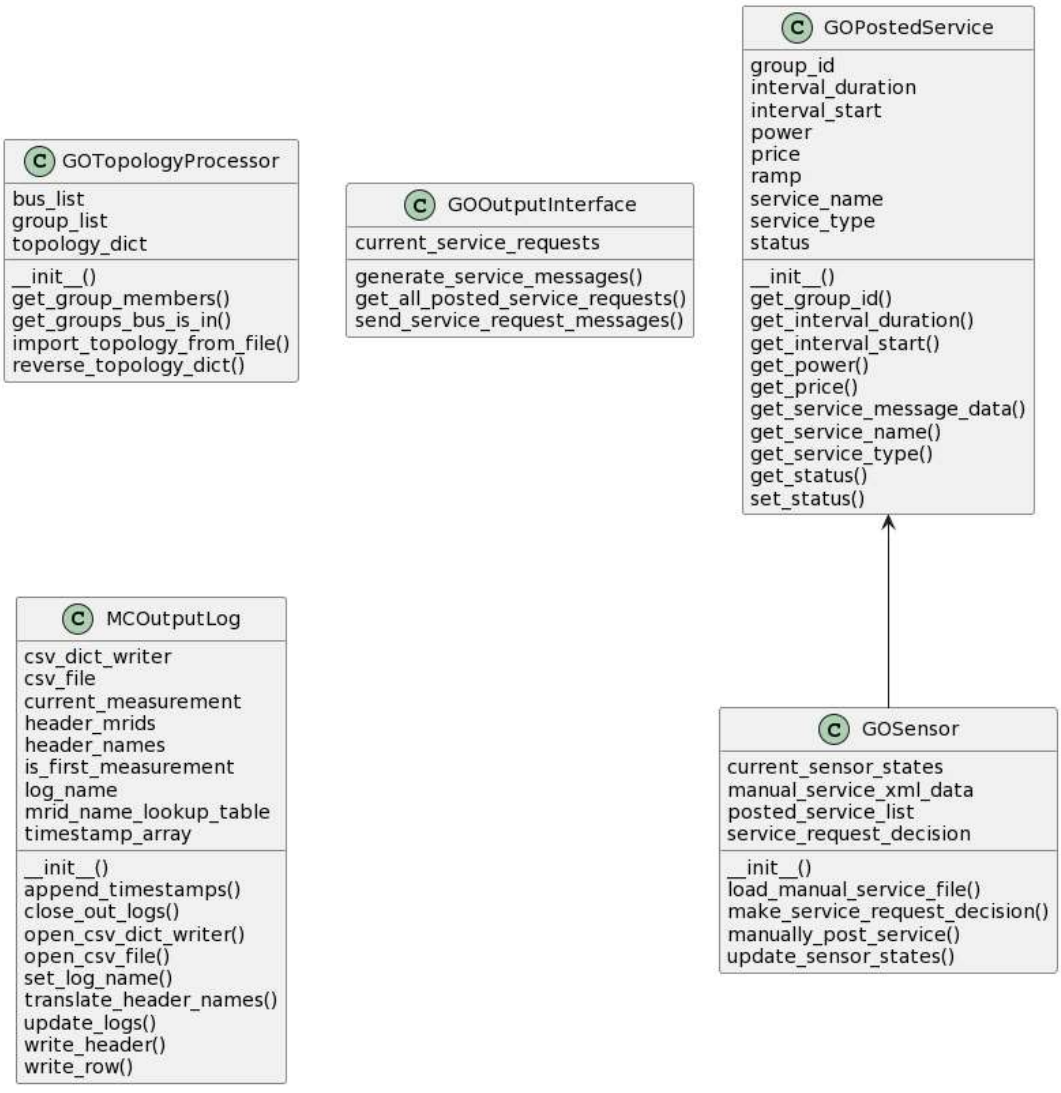

Figure 4.9: UML class diagrams of the Output classes.

The **GOSensor** class represents the operations of the Grid Operator. It is meant to be configurable for the needs of the model or the test. There are two possible operating modes: automatic mode and manual mode. Both implementations have been designed, but due to EGoT testing priorities only manual mode is currently implemented in the ME. In automatic mode, GOSensor reads in grid states from the measurement processor (hence the name "GOSensor"); these are compared to thresholds based on a user-defined algorithm and service requests are automatically generated. This mode is useful for more realistic grid simulation purposes but requires considerable work to develop detection and response algorithms. In manual mode, a list of services, parameters and request times are read from an XML file during ME startup. The GOSensor then generates service requests with the proper parameters at the designated times. This mode is sufficient for most functional testing purposes and is considered the "default" mode of operation. In either case, service requests are generated by the instantiation of an object of the GOPostedService class; these objects contain the service request parameters, are kept in a list by GOSensor and accessed by the GOOutputInterface.

The **GOOutputInterface** class handles service request messaging between the MC and the DERMS. Like the DER-S, it is intended to be designed by the test engineer to serve the needs of its client DERMS. It makes connections as necessary, polls the GOSensor for new service requests each timestep, accesses posted service request parameters, packages them into the proper message format, and sends the messages to the DERMS. The actual decision-making process is in GOSensor.

Timestamp	PowerTransformer_sub3_Power	PowerTransformer_sub3_Power.1
2019-10-02 18:31:58	{'measurement_mrid': '_01e96721-222d-42be-bcc3-8cb6fe23d44c', 'magnitude': 1474276.2673145642, 'angle': -11.711333830924843, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650', 'Phases': 'B', 'MeasType': 'VA'}	{'measurement_mrid': '_122affee-e7dd-4d8f-bd10-2696fd95c950', 'magnitude': 1415691.1267786373, 'angle': -7.507786248792342, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650z', 'Phases': 'A', 'MeasType': 'VA'}
2019-10-02 18:31:59	{'measurement_mrid': '_01e96721-222d-42be-bcc3-8cb6fe23d44c', 'magnitude': 1474276.2673145642, 'angle': -11.711333830924843, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650', 'Phases': 'B', 'MeasType': 'VA'}	{'measurement_mrid': '_122affee-e7dd-4d8f-bd10-2696fd95c950', 'magnitude': 1415691.1267786373, 'angle': -7.507786248792342, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650z', 'Phases': 'A', 'MeasType': 'VA'}
2019-10-02 18:32:00	{'measurement_mrid': '_01e96721-222d-42be-bcc3-8cb6fe23d44c', 'magnitude': 1474276.2673145642, 'angle': -11.711333830924843, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650', 'Phases': 'B', 'MeasType': 'VA'}	{'measurement_mrid': '_122affee-e7dd-4d8f-bd10-2696fd95c950', 'magnitude': 1415691.1267786373, 'angle': -7.507786248792342, 'Measurement name': 'PowerTransformer_sub3_Power', 'Meas Name': 'PowerTransformer_sub3_Power', 'Conducting Equipment Name': 'sub3', 'Bus': '650z', 'Phases': 'A', 'MeasType': 'VA'}

Table 4.2: Subset of data contained within an output log file. Note that each cell contains a dictionary of data which can be used or filtered out during analysis.

The **MCOutputLog** class handles logging. Processed measurements from EDMMeasurementProcessor are retrieved, converted to dictionary format, and written to a CSV file for later data analysis.

GOTopologyProcessor, when implemented, will allow for more complex topological assignment. A topology file will be read into the object on startup. This file must be user-

defined to match the topology expected by the DERMS, and will contain XML dictionaries clustering buses into groups. This information can then be used during the DER assignment process, allowing groups to be entered as locational identifiers instead of buses. These groups are translated to appropriate buses by the topology processor. These buses are fed back to the assignment handler to continue the assignment process. The following is an example of a simple topology input file containing two groups consisting of three buses. In the intended implementation, the topolology processor would be fed a group, and would respons with one of the three assigned buses for DER-EM assignment.

```
<all>
    <group name= "group-1" >
        <bus name="632" />
        <bus name="646" />
        <bus name="645" />
    </group>

    <group name= "group-2" >
        <bus name="632" />
        <bus name="633" />
        <bus name="634" />
    </group>
</all>
```

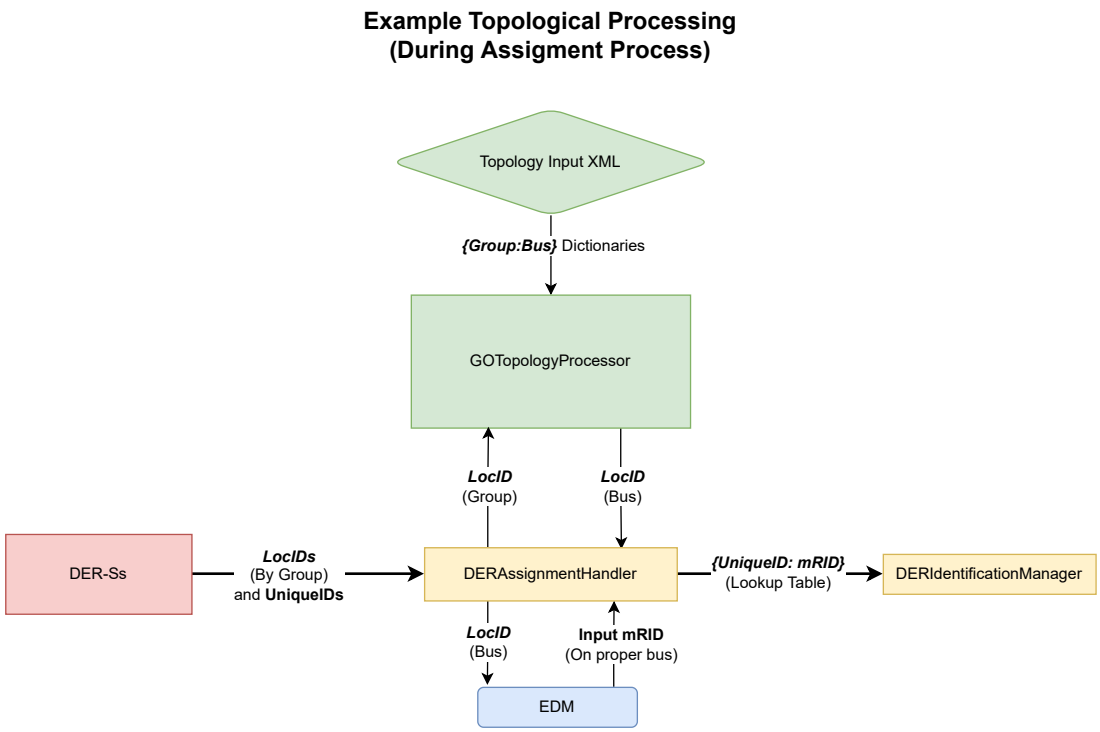

Figure 4.10: Block diagram demonstrating topological processing interactions. During assignment, the GOTopologyProcessor translates non-bus locational identifiers to buses using a predetermined topology.

GOPostedService is the only class in the MC that can be instantiated more than once. The GOSensor class creates a GOPostedService object each time it determines that a service should be requested. All service request parameters are included in attributes of the GOPostedService object; this includes a name, a group id, the service type, interval, and parameters. These objects are contained within a list in the GOSensor class and queried by the GOOutputInterface once per timestep.

4.1.3 DER-S Implementation

The ME is designed with extensibility in mind, particularly regarding the ability to implement models of a variety of DERs, both existing and yet to be developed. Each DER-S is a

conceptual "black box": it receives data from one or more DER inputs, and provides unique and locational identifiers (for assignment and association) and power sourced or sunk (for DER-EM control). However, the contents of the DER-S are deliberately left to the test engineer to define based on the testing needs and what DER inputs are available. This provides a greater amount of flexibility in how a particular DER-S can be developed and allows for hardware-in-the-loop implementations, inputs from emulators, and historical data to be used. This means that any type of DER can be included in the MC as long as a DER-S is developed for said DER.

Due to this flexibility, it is impossible to provide an exhaustive list of possible implementations, particularly with regards to inputs or internal processing within a DER-S. Some examples are presented below.

4.1.3.1 DERMS to DER-S communication

One example of DERMS to DER-S communication is direct exchange. Typically, information exchange between the DERMS and DER is beyond the scope of the ME since the DER-S is concerned with reading the operating state of a DER, not how that state is achieved via DERMS controls or aggregator dispatch. However, it is conceivable that for test purposes a DERMS could request resources that would be convenient to implement directly within the MC. One example would be a DERMS that controls BISs directly. In this example, communications between the DERMS and DER-S need to be established via development of an API within the DER-S class that handles and parses these communications natively.

The communication protocol is dictated by the DERMS. After the communications are parsed, they are processed into the proper output format within the DER-S class.

A second example of DERMS to DER-S communication involves interfaces with a DER's control module. A DER controller allows the DER to operate normally at most times based on the needs of the owner. At times, however, it allows participation with the DERMS. A Distributed Control Module provides the communication pathway between the DER and DERMS. The DERMS-to-DCM information exchange is managed by the DERMS, and is beyond the scope of the ME.

In this case, the DER-S class interfaces with one or more DCMs via a custom communications API within the DER-S class. The contents of these communications are dictated by the capabilities and protocols of the DCM. The DER-S governs processing these communications into a readable by the MCInputProcessor and DERAssignmentHandler classes. For example, one method of modeling a DER is to observe its power consumption over time. A sophisticated enough DCM with the proper sensors in its DER could provide power consumption readings directly to the DER-S, along with unique identifiers. A more rudimentary DCM may only provide a flag that the DER is "turned on" to the DER-S class, which would require the DER-S to "emulate" power consumption patterns based on the DER name plate information. If not provided by the DCM, identifiers need to be provided to the DER-S by the test engineer using an input configuration file or an algorithm for automatic generation.

Due to the flexibility in DER-S design, no communication protocols or standards are

suggested. These should be selected based on the needs of the particular DERMS, DCM, and DER-S implementation in use.

A third example of DERMS to DER-S communication demonstrates communication between outside software and DER-S classes. Sophisticated external emulators controlled by the DERMS or even functions within the DERMS may provide time-valued DER modeling. The modeled DER states communicate to the DER-S via an appropriate protocol, which the DER-S can read. The DER-S class needs to extract some information from these emulators that can be parsed into electrical data. These "DER inputs" may be electrical information, such as power consumption, that may be fed directly into the simulation. Or, they may be more generic operating data that are converted to electrical information within the DER-S by the inclusion and processing of label plate data or similar parameters.

RWHDERS is one example of this process. One or many resistive water heaters are emulated within the DERMS; the power consumption of each water heater is placed in a constantly updated file containing unique identifiers, location data, and operation data in the form of power. These files are read in real time by the RWHDERS processing method. While RWHDERS provides a power reading directly, another DER-S implementation may not. One conceivable DER-S would function similarly to RWHDERS but instead of receiving power readings, it may only receive a flag indicating whether the water heater is "importing" or "exporting" power. The import/export flags would be translated into power consumption data by the DER-S using provided label plate ratings. The role of the DER-S class is to read the data, perform the operational/electrical power conversion, and pass those data in

the proper form to the MCInputInterface for use in DER-EM updates.

4.1.3.2 DER Modeling using Historical Data

"Historical data" in this case refers to any input to the simulation that is not being generated in real-time. This could be data generated by an emulator, states read from DCM sensors and written to a file, or generated manually by the test engineer based on prior research or deterministically for functional test purposes. Provided the data are properly formatted, all of these data can be processed by single DER-S. The data must be time stamped to be used at the proper time in the simulation, and must include unique and locational identifiers so DERs in the data can be assigned to DER-EMs in the EDM. This is a preferred method for generating inputs to the DER-EMs in many functional test cases due to its simplicity: direct communications do not need to be established between the DERMS/emulators and DER-S during simulation runtime and large amounts of data can be processed without encountering communication obstacles such as bandwidth and latency.

Time	DER1_mag	DER1_loc	PowerElectronicsConnection_BatteryUnit_DER_Association_Test_6321_Battery.3
1570041118	0	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041119	1000	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041120	1000	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041121	0	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041122	0	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041123	1000	632	{ 'magnitude': 1333.333333, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041124	1000	632	{ 'magnitude': 1666.666667, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041125	0	632	{ 'magnitude': 1666.666667, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}
1570041126	0	632	{ 'magnitude': 1666.666667, 'Bus': '632', 'MeasType': 'VA', 'Input Unique ID': 'DER1_mag'}

Table 4.3: An example of the effects of DER inputs on grid state data over time. The inputs are drawn from an input CSV file for the DERHistoricalDataInput DER-S, and the outputs are log outputs that have been edited for clarity. Note that each column represents a single phase; as such, an added 1 kW of load shows as 333 W per phase.

Time	DER2_mag	DER2_loc	PowerElectronicsConnection_BatteryUnit_DER_Association_Test_6331_Battery.3
1570041118	1000	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041119	0	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041120	1000	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041121	0	633	{ 'magnitude': 1333.333333, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041122	1000	633	{ 'magnitude': 1333.333333, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041123	0	633	{ 'magnitude': 1333.333333, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041124	1000	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041125	0	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}
1570041126	1000	633	{ 'magnitude': 1666.666667, 'Bus': '633', 'MeasType': 'VA', 'Input Unique ID': 'DER2_mag'}

Table 4.4: Another example of the effects of DER inputs on grid state data over time. See 4.3. Also note the discrepancy between simulation and measurement timesteps: while inputs are delivered once per second, the measurements only update once every three seconds.

4.1.4 Grid Operator Implementation

As a component of the ME, the GO is a set of classes that simulate the decision-making process and actions of a real Grid Operator with respect to DER dispatch. The GO also provides an interface for communicating grid service requests and feedback data to the DERMS, and also provides a method for topological translation. The GO is designed for two modes of operation: Automatic and Manual mode. Automatic mode is designed to realistically simulate GO operations by retrieving grid states once per timestep, making determinations based on the grid states and built-in detection algorithms, and posting service requests automatically based on grid conditions. In Manual mode, grid service requests are read in from an XML file and posted at the proper times, allowing for simple, direct control for functional testing.

In a fully functional test system such as the EGoT prototype, during a running simulation the GOSensor may post a service request for the DERMS via the GOOutputInterface. Externally to the ME, the DERMS will dispatch DERs per this request. These changes in DERs will be communicated to the DER-Ss and reflected by the DER-EMs within the

EDM. These changes in the DER-EMs affect the grid states, which are gathered by the EDMMeasurementProcessor and (in Automatic mode) sent to the GOSensor. The GOSensor will parse the EDM grid states to verify the dispatch had occurred and incurred the required grid effects. The GO is meant to be a high-level representation of the GO's decision making; as such, the GOSensor decision-making function can be a simple threshold detection script, or more advanced detection capabilities can be designed. The GOOutputInterface communication with the DERMS is meant to be configured by the end user rather than requiring a specific protocol.

4.1.4.1 Grid States Inputs (Proposed)

During the simulation process, the EDMMeasurementProcessor holds a table containing the most recent grid states read from the ME at each time step, including electrical characteristics, the status of each DER-EM, and its association data (mRIDs, locational data, etc.) along with any other relevant grid state data. These tables are sent to the MCOutputLog once per time step to construct the logs. If Automatic Mode is configured and enabled, they will also be sent to the GOSensor, which will parse them.

4.1.4.2 Decision-Making Process (Proposed)

- Manual Decision-Making Mode (Implemented) - In this mode, the TE selects a particular desired request and a specific time to send the request. This is done for one or more requests by placing the service request parameters and times in a properly formatted XML file and configuring the MC to read it during the startup process by

enabling the Manual operation flag in MCConfiguration. As the simulation is running, the GO keeps time with the simulation using the MCTimekeeper `current_time` attribute. At the designated times, the GOSensor generates a GOPostedMessage object containing the parameters of the grid service and places this object in a list. Each timestep, the GOOutputInterface reads this posted service list and, if a new object is found, generates a message in the predetermined GO-DERMS format containing the service parameters. The GO then sends the message to the DERMS. The format and protocol of this message will be dependent on the needs of the DERMS; generally, the message will contain a request for a particular service and any related information such as durations, levels, etc. This mode is suitable for testing the output of the DERMS and its effects on the grid without necessarily requiring an abnormal grid condition to exist.

- Automatic Decision-Making Mode (Proposed) - In this mode, the TE does not request a service at a particular time. Instead, the GOSensor contains an algorithm which, for example, uses measurements from the EDMMeasurementProcessor to compare component values against programmed thresholds once per time step. If these thresholds are exceeded, a GOPostedService object will be generated. The remainder of the process is identical to Manual mode. This is representative of the normal functionality of a DERMS and its interactions with a GO. This can be used to test effectiveness of a DERMS dispatch scheme to address the causes of a grid service request.

4.1.4.3 GO-DERMS Communications

Since the ME is designed to be used with a variety of DERMSs, there is no single protocol for information exchange between a GO and a DERMS. This will necessarily be defined by the user within the GOOutputInterface class. To simplify this process, all results of the decision-making process are simplified into "grid service requests" and "feedback data." As such, the TE will need to develop an API between the GO and the DERMS in order to translate request types and magnitudes into messages that both actors can understand.

In the case of the EGoT DERMS, testing with the ME will be highly coordinated. GO-DERMS feedback data is not necessary for our test plans. As such, only posted grid service requests need to be communicated to the EGoT DERMS. To simplify testing and development, this is achieved by writing an XML file containing posted service requests once per timestep. This XML file is almost identical to the GOSensor Manual mode input XML file, but with the timestamps removed. Because of this requirement, the EGoT DERMS and ME must run on the same system so the DERMS can read these files from the drive and process them in real time.

5 Discussion

5.1 Limitations and Potential Weaknesses

The EGoT project was planned to be developed over three years. The ME presented in this thesis is the result of two years of development. Efforts to integrate the ME and EGoT DERMS are currently incomplete, awaiting further development of the DERMS. The communication schemes between the DERMS and ME as presented in this thesis have not been fully tested and may require further planning and development. As such, while the system has been tested using the historical data input and log output classes, and while as much testing as possible has been conducted with mock inputs to RWHDERS and outputs from the GO, more testing is required to verify certain aspects of the ME-DERMS interactions are sound.

One critical testing phase will involve scaling. Due to the nature of DER aggregation, the ME will eventually need to manage thousands of DERs simultaneously. GridAPPS-D is able to perform simulations on models with thousands of nodes, so it is expected that there will not be any issue with their addition; however, bottlenecks may arise in the simulation's interactions with external sources. Importantly, communications latency between the DER Inputs and DER-S classes will need to be considered, and stress testing should be performed on the EDM input messaging class. Finally, system memory requirements should be kept in

mind. Memory requirements are not currently a limitation: the biggest object in memory is currently the processed system measurement dictionary, which is cleared and replaced every timestep. However, there is the possibility that advanced DER-S classes may be designed to store information to memory, so care should be taken in how this is done so as not to overly burden the system running the ME.

5.2 Further Research and Future Work

5.2.1 DER-EM Clustering

There is likely to be an upper limit to the number of discrete inputs that the EDM will accept per timestep. This upper limit is a function of GridAPPS-D and details are unknown and likely to remain unknown until stress testing is completed. In any case, there is a strong likelihood that future simulations involving massive quantities of DERs will encounter issues.

Bottlenecks between the DER-S and DER-EMs can be relieved by reconsidering the definition of a DER-EM. Currently, each DER-EM represents a single DER: that is, a single water heater, a single battery inverter, a single household, etc. If there are 20 DERs providing inputs to a single location, there will be 20 DER-EMs on the corresponding bus. This is efficient for functional testing: each DER-EM will have measurement points that will be reflected in the logs, so the TE can ensure that individual DERs are being dispatched to the expected levels at the expected times.

The downside to this approach is that each DER-EM needs to be updated individually

each timestep. If there are 20 DER-EMs, and they all change states on a time step, that requires 20 input messages to GridAPPS-D. If individual DER-EM measurements are not explicitly required by the test parameters, it would be much more efficient for each bus to contain one DER-EM representing all DERs on that bus. As long as DER inputs can be retrieved by the DER-S each timestep, it would be trivial to modify the MCInputInterface to sum the power values of all input requests to a given bus each timestep. Then the number of input messages provided to GridAPPS-D on a certain timestep would never be greater than the number of buses containing DERs in the model.

Furthermore, while inputs to the EDM are handled once per simulation timestep (or one per second), measurable outputs from the EDM only occur once per MEASUREMENT timestep; that is, once every three seconds. If necessary, the buses could be split into three equally sized groups; DER-EMs on group A would input on the first timestep, group B the second timestep, and group C the third timestep before new measurements are produces. This would divide the input messages over three seconds, reducing the maximum input messages per timestep to the number of buses divided by three.

5.2.2 New DER-S Classes and Improvements

The DER-S classes developed for this phase of the EGoT project were designed to facilitate functional testing and troubleshooting of the EGoT DERMS. Despite some user-unfriendliness in development of the input logs, DERSHistoricalDataInput is certainly flexible enough to be used in most testing situations; RWHDERS, on the other hand, was designed expressly to test the current implementation of the water heater emulators quickly

and easily. The custom protocol is non-standard and will not work with other water heater emulators or systems.

New DER-S classes will need to be written for new DER input concepts. One proposed DER-S would facilitate direct communications between a physical battery inverter system and the ME. This would utilize whatever communications protocol the inverter or DCM is equipped with to allow real-time data to be sent to the DER-S. Another potential DER-S would do the same, but for water heaters rather than Battery-Inverter Systemss (BISs).

5.2.3 Log Usability Optimization

The MCOutputLog class was written to support functional testing of the EGoT DERMS; this functional testing was simple enough that manual log analysis by the TE was sufficient to ensure that resources were dispatched at the proper time, for example. However, future testing and research will demand more complex observations such as trend analysis. These will require tools to be written that generate plots, for instance.

Currently, the logs contain the measurements processed by the EDMMeasurementProcessor in a spreadsheet format. Each row represents a time, and each column a measurement point. However, each cell is a dictionary containing multiple pieces of information: this includes the measurement, measurement type, phase, locational data, etc. While this data is important for TE understanding of the logs, it complicates development of tools that automate processing and analysis of the data.

There are at least two potential avenues to approach this problem. The first is to rewrite the MCOutputLog class to produce simplified logs containing the numeric data only; that is,

each cell contains a single numeric value for power or voltage. These logs could be saved in place of or in parallel with the existing data logs. The second approach is to develop an external tool that processes the existing logs into a more manageable format, potentially as a component of the analysis tools to be written.

5.2.4 Topological Processing

The ME currently assumes that all inputs will provide locational identification in the form of the bus number that the DER should be assigned to. This is suitable for direct functional testing, but future implementations should include more complex topological processing. This is facilitated by the inclusion of the GOTopologicalProcessor class: this class reads an XML file containing the desired topology of the model, with buses organized into groups or branches. This topology is shared between the ME and the DERMS, and the class will include methods allowing for a group-bus lookup table for DER assignment and association purposes as well as feedthrough to the logs and GO.

While a class definition for the GOTopologicalProcessor exists, and an example XML topology file is included with the ME, the methods and attributes providing functionality have not yet been written.

5.2.5 GO Automatic Operation

The GO was conceptualized to have two discrete operating modes: Manual mode and Automatic mode. In Manual mode, grid services are "manually" requested by the TE: a prewritten XML file containing the grid service request times and parameters is loaded into

the GO during the startup process, and grid services are requested based on that schedule. No grid state measurements are required by or provided to the GO. In Automatic mode, however, grid services would be requested in a realistic way by the GO based on grid states and incipient events.

Manual mode is sufficient for functional testing since it tests all the necessary components of the DERMS: service request message generation and communication, DERMS operations and dispatch of resources via the water heater DER-S, and the results of the dispatch as observed in the logs by the TE. Automatic mode functionality was not prioritized in this stage of the project since it is not a method of testing the DERMS, but instead is a method of developing and testing GO algorithms for detection and dispatch.

Of course, once the DERMS is functionally tested, developers will want to provide optimal solutions for its usage, along with data on effectiveness, response times, etc. At that stage, the GOSensor class should be modified to include an Automatic operation mode, including algorithms that read grid states and detect anomalies similarly to a real GO.

Looking even further into the future, the ME and a functional DERMS could be used to develop, test and optimize advanced GO detection and request generation algorithms. For example, a machine learning algorithm could be implemented to optimize the timing, scale or location of grid service requests

6 Conclusion

6.1 Contributions

The Modeling Enviroment successfully demonstrates a GridAPPS-D-based, open source, configurable and modular, DER-aware grid modeling system for DERMS testing. In its current state, the ME provides functional testing abilities for the EGoT DERMS by generating grid service dispatch requests, sending them to the DERMS, retrieving resultant updated DER input data from the associated water heater emulators, updates the grid simulation, and demonstrates grid state changes via output logs. The class-based architecture provides modularity, as demonstrated by the concurrent operation of multiple DER-S types as well as the decoupled API between the simulated GO and DERMS.

This open source modularity is in service to a greater push towards openness and interoperability within the DER-DERMS ecosystem. The DER-S concept will accelerate testing of new types of DERs in whatever form they take, and the per-DER-S input API means that new standards and protocols, open or proprietary, can be easily and independently implemented. The DER-S can adapt to the needs of the input and the GO output API can be reconfigured to the communication protocol needs of the DERMS. This eliminates the need to develop DER dispatch test platforms for individual DERMS.

6.2 Conclusion

As with many open-source projects, the ME was developed with the intent that future collaborators would modify and iterate on it over time to fulfill the individual requirements of the collaborator, as well as any others who might share those requirements. To this end, I developed a modular, class-based system using the GridAPPS-D platform to provide communications to and from a grid simulator to model the effects of mass DER dispatch on a grid model. The grid model DER-EMs were made using generic BIS models, allowing the development of the concept of the DER-S: a "simulated DER" that can take any kind of DER data in any format, translate it in real time to electrical DER operating states, and provide it to a DER-EM at the proper location at the proper time. Along with logs, outputs from the simulation are filtered through a simulated Grid Operator providing a configurable API between the ME and a DERMS.

By using a decoupled class-based architecture and generic, configurable APIs and DER representations, the ME is designed to test any system that dispatches DERs. The ME is configurable to the needs of the DERMS or DER inputs, not the other way around, thus removing barriers to testing and development. Open-sourcing the ME allows anyone to develop their own APIs for different DERs or DERMS and provide them to the public, allowing for the growth of an ecosystem of test development utilizing a variety of models, DERs, DERMS, and even GO detection algorithms. This will remove the time and effort required to develop individual test and simulation systems for DERMS.

Bibliography

[1] Steven E. Widergren, Mark R. Knight, Ronald B. Melton, David Narang, Maurice Martin, Bruce Nordman, Aditya Khandekar, and Keith S. Hardy. Interoperability Strategic Vision. Technical report, Pacific Northwest National Laboratory, Richland, WA (United States), 2018.

[2] Tylor Slay, John Acken, and Robert Bass. Incentivizing Distributed Energy Resource Participation in Grid Services. In *9th IEEE Conference on Technologies for Sustainability*, 2022.

[3] Ricardo Silva, Everton Alves, Ricardo Ferreira, José Villar, and Clara Gouveia. Characterization of TSO and DSO Grid System Services and TSO-DSO Basic Coordination Mechanisms in the Current Decarbonization Context. *Energies*, 14(15):4451, July 2021.

[4] Junji Kondoh, Ning Lu, and Donald J. Hammerstrom. An evaluation of the water heater load potential for providing regulation service. *IEEE Transactions on Power Systems*, 26(3):1309–1316, August 2011.

[5] Manasseh Obi, Tylor Slay, and Robert Bass. Distributed energy resource aggregation using customer-owned equipment: A review of literature and standards. *Energy*

Reports, 6:2358 – 2369, 2020.

[6] A Ransil. Report 2.1: Price signals and demand-side management in the electric distribution and retail system. Technical report, Protocol Labs, September 2018.

[7] Midrar Adham, Manasseh Obi, and Robert Bass. A Field Test of Direct Load Control of Water Heaters and its Implications for Consumers. *IEEE Power and Energy Society General Meeting*, July 2022.

[8] Luka Strezoski and Izabela Stefani. Utility DERMS for Active Management of Emerging Distribution Grids with High Penetration of Renewable DERs. *Electronics*, 10(16):2027, August 2021.

[9] Luka Strezoski, Izabela Stefani, and Branislav Brbaklic. Active Management of Distribution Systems with High Penetration of Distributed Energy Resources. In *18th International Conference on Smart Technologies*, pages 1–5. IEEE, July 2019.

[10] Hee Seung Moon, Young Gyu Jin, Yong Tae Yoon, and Seung Wan Kim. Prequalification Scheme of a Distribution System Operator for Supporting Wholesale Market Participation of a Distributed Energy Resource Aggregator. *IEEE Access*, 9:80434–80450, 2021.

[11] Mani Vadari. Evolving Architectures and Considerations to address Distributed Energy Resources and Non-Wired Alternatives. Technical report, Pacific Northwest National Laboratory, Richland, WA (United States), July 2021.

[12] Narmada Sonali Fernando. The Distributed Trust Model Applied to the Energy Grid of Things. Master's thesis, Portland State University, 2021.

[13] D. Hardin. Customer Energy Services Interface White Paper. *Grid-Interop Forum*, 2011.

[14] Tylor Slay and Robert B. Bass. An Energy Service Interface for Distributed Energy Resources. In *IEEE Conference on Technologies for Sustainability*, pages 1–8, 2021.

[15] Ronald B. Melton, Kevin P. Schneider, Thomas E. McDermott, and Subramanian V. Vadari. GridAPPS-D Conceptual Design v1.0. Technical report, Pacific Northwest National Laboratory, Richland, WA (United States), May 2017.

[16] M. Alam, Abhishek Somani, Ronald Melton, and Thomas McDermott. Transactive Approach for Engaging Distribution Network Assets for Voltage Management in Southern California Edison Distribution Feeders. Technical report, Pacific Northwest National Laboratory, Richland, WA (United States), June 2018.

[17] Bishnu Bhattarai, Sri Nikhil Gourisetti, Priya Thekkumparambath Mana, and Jason Fuller. CleanStart DERMS: Modeling Effort Update. Technical report, Pacific Northwest National Laboratory (PNNL), Richland, WA (United States), June 2018.

[18] Mengmeng Cai, Rui Yang, and Yingchen Zhang. Iteration-Based Linearized Distribution-Level Locational Marginal Price for Three-Phase Unbalanced Distribution Systems. *IEEE Transactions on Smart Grid*, 12(6):4886–4896, November 2021.

[19] Fei Ding, Yingchen Zhang, Jeffery Simpson, Andrey Bernstein, and Subramanian Vadari. Optimal energy dispatch of distributed PVS for the next generation of distribution management systems. *IEEE Open Access Journal of Power and Energy*, 7(1):287–295, 2020.

[20] Rahul R. Jha. *Network-Level Optimization for Volt/VAr Control in Unbalanced Electric Power Distribution Systems*. PhD thesis, Washington State University, December 2020.

[21] Quan Nguyen, Jim Ogle, Xiaoyuan Fan, Xinda Ke, Mallikarjuna R. Vallem, Nader Samaan, and Ning Lu. EMS and DMS Integration of the Coordinative Real-time Sub-Transmission Volt-VAr Control Tool under High DER Penetration. In *2021 IEEE Power & Energy Society General Meeting (PESGM)*, pages 1–5, July 2021.

[22] Shiva Poudel, Poorva Sharma, Anamika Dubey, and Kevin P. Schneider. Advanced FLISR with Intentional Islanding Operations in an ADMS Environment Using GridAPPS-D. *IEEE Access*, 8:113766–113778, 2020.

[23] Shiva Poudel. *Quantifying and Improving Resilience in Power Distribution Systems*. PhD thesis, Washington State University, August 2020.

[24] Annabelle Pratt. ADMS Test Bed Capabilities. In *Workshop on Advanced Distribution Management System Test Bed*, Golden, CO, November 2019.

[25] Annabelle Pratt, Harsha Padullaparti, Young Ngo, and Herschel Arant. Defining a Use Case for the ADMS Test Bed: Fault Location, Isolation, and Service Restoration with

Distributed Energy Resources. In *IEEE Power & Energy Society Innovative Smart Grid Technologies Conference*, pages 1–5, 2021.

[26] Alka Singh, Andrew Fisher, Craig Allwardt, and Ronald B. Melton. A Data Exchange Interface for a Standards Based Data Integration Platform. In *IEEE Power & Energy Society Innovative Smart Grid Technologies Conference*, pages 1–5, February 2020.

[27] Robert Pratt, Teja Kuruganti, Hung Ngo, Ebony Mayhorn, David Winiarski, Hayden Reeve, Chuck Booten, Jeff Maguire, Michael Duoba, Alejandro Fernandez Canosa, Rasel Mahmud, David Rosewater, Sig Gonzales, Rob Hovsapian, Julian Osorio-Ramirez, Rahul Kadavil, Svetomir Stevic, Jin Dong, Jeffrey Munk, Yaosuo Xue, Vishaldeep Sharma, Mahabir Bhandari, Todd Taylor, Yuan Liu, Jingjing Liu, Fei Ding, Tom Edmund, and Vaibhav Donde. Grid Services from DER Device Fleets: Volume 1 - Battery-Equivalent Models of Devices and Fleets. Technical report, Grid Modernization Laboratory Consortium, June 2020.

[28] Shiva Poudel, Gary D. Black, Eric G. Stephan, and Andrew P. Reiman. Admittance Matrix Validation for Power Distribution System Models Using a Networked Equipment Model Framework. *IEEE Access*, 10:9108–9123, 2022.

[29] W.H. Kersting. Radial distribution test feeders. *IEEE Transactions on Power Systems*, 6(3):975–985, 1991.

Appendix A: Useful Links

A.1 The EGoT Project at the PSU Power Lab

A.1.1 GitHub Repositories

A.1.1.1 Modeling Environment

- https://github.com/PortlandStatePowerLab/doe-egot-me
- Includes instructions for downloading, installing, and configuring the EGoT Modeling Environment.

A.1.1.2 EGoT DERMS

- https://github.com/PortlandStatePowerLab/doe-egot-system
- Includes code for DCM, DTM, and GSP.

A.1.1.3 EGoT DTM

- https://github.com/PortlandStatePowerLab/doe-egot-dtm
- DTM server and DCM client.

A.2 GridAPPS-D

A.2.1 Documentation

- GridAPPS-D User Manual
  - https://gridappsd.readthedocs.io/

A.2.2 GitHub Repositories

- GridAPPS-D Group
  - https://github.com/GRIDAPPSD
- Docker-based Runtime for GridAPPS-D
  - https://github.com/GRIDAPPSD/gridappsd-docker
- CIMHub tools
  - https://github.com/GRIDAPPSD/CIMHub