GridAPPS-D Documentation

Release 1.0

The GridAPPS-D Team

Jan 09, 2018

Contents

TODO Briefly introduce goss-gridapps-d including the simplest path to install and requirements Link to other sections for detailed installation and quickstart.

CHAPTER 1

Overview

Through a series of industry centric meetings and workshops, the U.S. Department of Energy Office of Electricity Delivery and Energy Reliability (DOE-OE) gathered input from utilities throughout the United States on their experiences in implementing, or planning to implement, ADMS. The results of these meetings are documented in a February 2015 report titled Voices of Experience: Insights into Advanced Distribution Management Systems.

The report documents the potential benefits to utilities in implementing ADMS applications, and underscores the need for more affordability, a timely path for deploying ADMS, and the development and deployment of ADMS applications. The high cost and amount of time required for ADMS deployment and application development was highlighted.

In response to these needs, DOE-OE has established an ADMS program with this project specifically tasked with developing an open-source, standards based ADMS application development platform - GridAPPS-D.

1.1 Conceptual Design Summary

A conceptual design for GridAPPS-D was created at the beginning of the project. The conceptual design is summarized below. The full design document may be downloaded from this link - GridAPPS-D Conceptual Design

This document provides a high level, conceptual view of the platform and provides related background and contextual information. This document is intended to both educate readers about the technical work of the project and to serve as a point of reference for the project team. The document will be updated as the project progresses.

1.2 Architecture

A conceptual architecture for the system has five key functional elements as shown in Figure 1:

1. **Tools** help developers enhance the functionality of their applications. Examples might include off-line power flow, optimization tool boxes, state estimators, statistical processing, etc.
2. **I/O** allows convenient access to the power system model and data through standards-based queries and messages. Conversely, applications can send control signals to the simulator using standard message schemas.

3. **Development utilities** include loggers, debuggers, access control, test managers, user interface toolkits, and other application support functions.
4. **Data bus** is based on industry standards like IEC 61968 and 61970 (i.e. the Common Information Model), plus more to be identified.
5. **Distribution simulator** represents the power system operating in real time. Initially, this will be GridLAB-D, but future versions may include EPRI's OpenDSS, ns-3 for communications, and other federated co-simulators.

Figure 1 also shows the relationships between GridAPPS-D, the ADMS application developer and commercial tools. Two different classes of data flow are shown:

1. Control and configuration data are shown with dashed lines; this allows the application developer to manage the platform.
2. Data flowing as a part of an application are shown with solid lines.

For more detailed information about the architecture and design, see *UML from the Functional Specification*

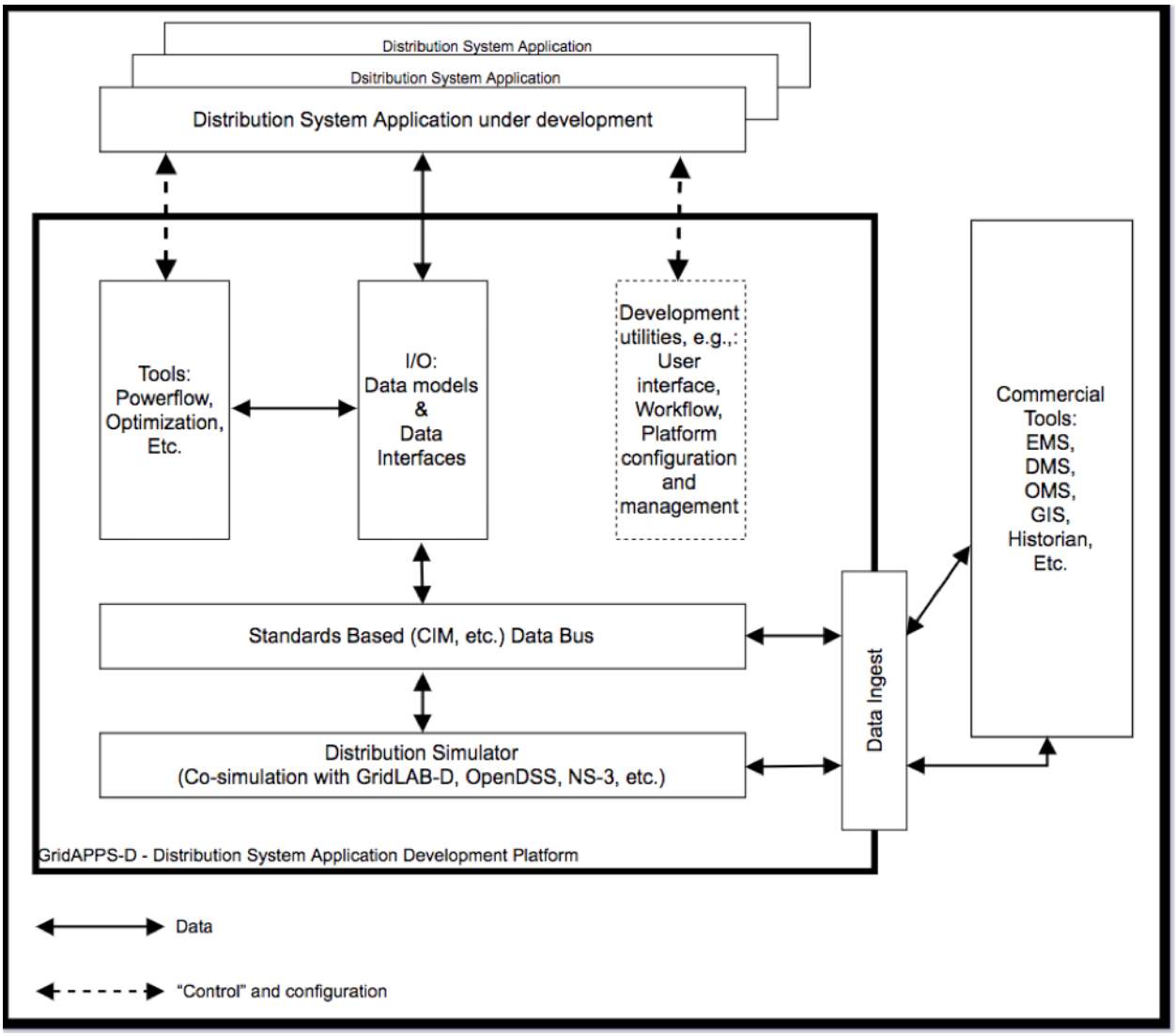


Figure 1: GridAPPS-D provides a method for developers (top) to run their new applications on a real-time simulator with extensive modeling and tool support (heavy box). GridAPPS-D is built around standard data models like the CIM (center). It readily interfaces to existing software products (right), which may also 1) use components of GridAPPS-D

and 2) supplement or replace the built-in distribution simulator (bottom), facilitating the deployment of new ADMS applications to existing software products.

1.3 Definition of Terms

Process Manager - Process Manager keeps track of all the processes running on the platform. These processes may include simulators, requests, applications and other managers. It is also the starting point for a request received by the platform.

Configuration Manager - It receives simulation configuration request from Process Manager and parses it to build the necessary configuration files.

Data Manager - The data manager accesses the database to build the model files used by the simulator.

Simulation Manager - The simulation manager launches the simulator and other required applications such as the FNCS bridge, FNCS, and the VoltVar application. It is in charge of managing the timing of the simulation and reporting output from the simulation out to the simulation status topic.

FNCS-GOSS Bridge - Serves as a bridge between FNCS and Simulation Manager.

FNCS - FNCS is a network co-simulator used to communicate between simulator and FNCS-GOSS bridge

Platform - Refers to GridAPPS-D platform.

RC1 - Release Cycle 1.

Simulation - A real world distribution system currently done by GridLAB-D

Simulator - In current release GridLAB-D serves as the simulator.

VoltVar Application -

Vizualization - A web-based visualization application is developed in RC1 to view power system model with real time values from simulation result.

GOSS - Grid Optics Software System is a middleware architecture designed as a prototype future data analytics and integration platform

GridLAB-D - GridLAB-D is a distribution level powerflow simulator. It acts as the real world distribution system in GridAPPS-D.

Power System Model - IEEE 8500 model is used in RC1.

Model - See Power System Model

CIM - Common Information Model is a standard for representing electrical network and exchange information.

1.4 References

1.5 Version History

Version Name: Release Cycle 1 (RC1)

Release Date: May 2017

Version description: This is the first version for internal release of GridAPPS-D platform. This is not ready for public use yet.

Functional requirements covered in this release:

- 102/202 Command Interface
- 301 Real-time Simulation Data
- 310 Hosted Application, but short-cutting the registration process (partial)
- 401 Distribution Co-Simulator (partial)
- 402 Process Manager (partial)
- 404 Data Manager (partial)
- 405 Simulation Manager (partial)
- 406 Power System Model Manager (partial)
- 413 Platform Manager (encapsulating 401 and 403-406)

1.6 Contact Us

GridAPPS-D team can be reached at gridappsd@pnnl.gov

CHAPTER 2

Installing GridAPPS-D

This document is work in progress.

2.1 Bootstrap

It is recommended to start with a linux platform such as Ubuntu and a 'gridappsd' user created. The bootstrap scripts should be run as root.

```
apt install -y git   (you may need to run apt update first)
git clone https://github.com/GRIDAPPSD/Bootstrap.git
cd Bootstrap
chmod a+x *.sh
./bootstrap.sh
```

2.2 Pre-Requisite

If the bootstrap doesn't work, or you wish to install manually you will need the following prerequisites.

apt upgrade -y (as root user)

GridAPPS-D Dependencies - Use apt install for the following dependencies

apt install -y vim git mysql-server automake default-jdk g++ gcc python python-pip libtool apache2 gradle nodejs-legacy npm curl

- vim
- Git
- Mysql-server (I set the root pw as gridappsd1234)
- Automake
- Default-jdk

- G++
- Gcc
- Python (v 2.x)
- Python-pip
- Libtool
- Apache2
- Gradle
- nodejs-legacy
- npm
- curl

Then apply the following pip installs

pip install –upgrade pip

pip install stomp.py pip install pyyaml

- pip install –upgrade pip
- pip install stomp.py
- pip install pyyaml

As well as the following npm packages

- npm install -g express
- npm install -g ejs
- npm install -g typescript
- npm install -g typings
- npm install -g webpack

  The following structure should be set up to enable the run scripts to execute correctly.

  - Griddapps-project
  - builds/
  - sources/

You will also need to install FNCS and GridLAB-D

FNCS

FNCS

Overview FNCS is the co-simulation engine used by GridAPP-D's simulation manager class to facilitating real-time synchonization and message passing between the GridLAB-D simulation and the GOSS message bus.

Source Code FNCS is maintained by PNNL. The repository is located at https://github.com/FNCS/fncs. GridAPPS-D is using the latest release of FNCS which is v2.3.2.

FNCS Documentation The documentation for FNCS is located at https://github.com/FNCS/fncs/wiki.

Building and Installing the Source ### Linux #### Prerequisites FNCS requires both the ZeroMQ and CZMQ libraries. For the purposes of the tutorial FNCS and it's prerequisites will be installed a custom location refered to by $FNCS_INSTALL. All source code is downloaded to the $HOME directory.

‘‘‘bash # download and install ZeroMQ :~$ wget http://download.zeromq.org/zeromq-3.2.4.tar.gz # if you do not have wget, use # curl -O http://download.zeromq.org/zeromq-3.2.4.tar.gz

unpack zeromq, change to its directory :~$ tar -xzf zeromq-3.2.4.tar.gz :~$ cd zeromq-3.2.4

configure, make, and make install :~/zeromq-3.2.4$ ./configure –prefix=$FNCS_INSTALL :~/zeromq-3.2.4$ make :~/zeromq-3.2.4$ make install

download and install CZMQ :~/zeromq-3.2.4$ cd $HOME

:~$ wget http://download.zeromq.org/czmq-3.0.0-rc1.tar.gz # if you do not have wget, use # curl -O http://download.zeromq.org/czmq-3.0.0-rc1.tar.gz

unpack czmq, change to its directory :~$ tar -xzf czmq-3.0.0-rc1.tar.gz :~$ cd czmq-3.0.0

configure, make, and make install :~/czmq-3.0.0$ ./configure –prefix=$FNCS_INSTALL –with-libzmq=$FNCS_INSTALL :~/czmq-3.0.0$ make :~/czmq-3.0.0$ make install ‘‘‘

Building FNCS In this tutorial FNCS source code will be downloaded using git to the $HOME directory. The code will be installed at t/FNCShe loacation $FNCS_INSTALL. ‘‘‘bash # download FNCS :~$ git clone https://github.com/FNCS/fncs.git

change to FNCS directory :~$ cd fncs

configure, make, and make install :~/fncs$ ./configure –prefix=$FNCS_INSTALL –with-zmq=$FNCS_INSTALL :~/fncs$ make :~/fncs$ make install ‘‘‘

Environment Setup In order for GridAPPS-D to be able to run FNCS and for GridLAB-D to be built with FNCS The following environment variables need to be setup: * $PATH must contain $FNCS_INSTALL/bin * $LD_LIBRARY_PATH must contain $FNCS_INSTALL/lib

GridLAB-D

GridLAB-D

Overview

GridLAB-D is a steady-state Distribution System simulation tool. It solves full three phase unbalanced network power flows and provides highly detailed enduse load models. It is part of GridAPPS-D's Simulation Engine. It serves for providing the real world distribution system environment for third party GridAPPS-D applications to monitor and control in real time.

Source Code GridLAB-D is maintained by Pacific Northwest National Laboratories in GitHub. The repository is located at https://github.com/gridlab-d/gridlab-d. GridAPPS-D uses the 4.0 release which is in release candidate currently and located on branch release/RC4.0.

GridLAB-D Documentation GridLAB-D's Documentation is located at http://gridlab-d.shoutwiki.com/wiki/Main_Page

Building and Installing the Source ### Linux #### Prerequisites The following packages are needed in order to build GridLAB-D. `bash :~$ sudo apt-get install \ gcc \ g++ \ automake \ libtool \ git `

For GridAPPS-D GridLAB-D will need to be compiled with the FNCS shared Library so FNCS will need to be installed. For instructions on building and installing FNCS, please go [here](). For the purposes of this document the location of where you installed FNCS will be known as $FNCS_INSTALL.

Building GridLAB-D For the purposes of this instruction set, the location to where you download the repository will be known as $GLD_INSTALL. `bash #download the release/RC4.0 branch repository :$GLD_INSTALL$ git clone https://github.com/gridlab-d/gridlab-d.git -b release/RC4.0 --single-branch #build and install xerces located in the third_party folder of the repository :$GLD_INSTALL$ cd gridlab-d/third_party :$GLD_INSTALL/gridlab-d/third_party$ tar -xzf

```
xerces-c-3.1.1.tar.gz :$GLD_INSTALL/gridlab-d/third_party$ cd xerces-c-3.1.1
:$GLD_INSTALL/gridlab-d/third_party/xerces-c-3.1.1$ ./configure :$GLD_INSTALL/
gridlab-d/third_party/xerces-c-3.1.1$ sudo make :$GLD_INSTALL/gridlab-d/
third_party/xerces-c-3.1.1$ sudo make install #build and install GridLAB-D
with FNCS :$GLD_INSTALL/gridlab-d/third_party/xerces-c-3.1.1$ cd ../../
:$GLD_INSTALL/gridlab-d$ autoreconf -if :$GLD_INSTALL/gridlab-d$ ./configure
--prefix=$GLD_INSTALL/install --with-fncs=$FNCS_INSTALL --enable-silent-rules
'CFLAGS=-g -O0 -w' 'CXXFLAGS=-g -O0 -w' 'LDFLAGS=-g -O0 -w' #before performing
make. Make sure the envirionment variable $LD_LIBRARY_PATH contains the
path $FNCS_INSTALL/lib if it doesn't then it will need to be added to
$LD_LIBRARY_PATH :$GLD_INSTALL/gridlab-d$ make :$GLD_INSTALL/gridlab-d$
install `
```

Environment Setup In order for GridAPPS-D to be able to run GridLAB-D The following environment variables need to be setup: * $PATH must contain $GLD_INSTALL/install/bin and $FNCS_INSTALL/bin * $GLPATH must contain $GLD_INSTALL/install/lib/gridlabd and $GLD_INSTALL/install/share/gridlabd * $CXXFLAGS must contain $GLD_INSTALL/install/share/gridlabd * $LD_LIBRARY_PATH must contain $FNCS_INSTALL/lib

2.3 Download

You will need to clone the GOSS-GridAPPS-D and viz repositories and build each

- **GOSS-GridAPPS-D**
  - git clone https://github.com/GRIDAPPSD/GOSS-GridAPPS-D.git
  - cd GOSS-GridAPPS-D
  - ./build-goss-test.sh
  - mkdir -p $GRIDAPPSD_INSTALL/builds/log
- **Vizualization**
  - git clone https://github.com/GRIDAPPSD/viz.git
  - cd viz
  - npm install
  - webpack
- **Blazegraph**
  - wget https://downloads.sourceforge.net/project/bigdata/bigdata/2.1.1/blazegraph.jar -O $GRIDAPPSD_INSTALL/builds/lib/blazegraph.jar

2.4 Activate Environment

You will need to populate the mysql database with the ieee8500 model

wget https://github.com/GRIDAPPSD/Bootstrap/raw/master/gridappsd_mysql_dump.sql

mysql -u root -p < gridappsd_mysql_dump.sql

To populate Blazegraph with the ieee8500 model

- Download https://github.com/GRIDAPPSD/Powergrid-Models/blob/master/CIM/ieee8500.xml

- java -Dbigdata.propertyFile=$GRIDAPPSD_INSTALL/builds/lib/conf/rwstore.properties -jar $GRIDAPPSD_INSTALL/builds/lib/blazegraph.jar >> $GRIDAPPSD_INSTALL/builds/log/blazegraph.log 2>&1 &
- Go to http://localhost:9999
- Click on the Update tab
- Choose the ieee8500 model file and change the format to RDF/XML
- Click Update

2.5 Start Platform

To start the platform, open two terminal windows:

- **To start GridAPPS-D, this should also start the web visualization**
  - cd gridappsd_project/sources/GOSS-GridAPPS-D
  - ./run-goss-test.sh
  - In a browser go to http://localhost:8082/ieee8500

2.6 Testing

Testing information will be added later

CHAPTER 3

Using GridAPPS-D

3.1 Overview

3.1.1 RC1 Demonstration

In order to run the RC1 demonstration you will need to have access to a Linux machine configured according to the instructions in the Installing GridAPPS-D section. In order to run the demonstration you will need both ssh and web access to the machine.

In this procedure, we connect to a Linux virtual machine (VM) in the PNNL Energy Infrastructure Operations Center (EIOC) using a virtual private network (VPN) connection. The ssh host for GridAPPS-D in the EIOC is at 172.20.128.20, and you would need user credentials for both that VM and the VPN connection.

The procedure could vary if logging into a local build of GridAPPS-D, i.e. not hosted in PNNL's EIOC. If logging into your own build, you will need to replace the IP in the following steps with that of your system. You can use the ssh command from a Terminal or any SSH client such as *ssh -X -Y username@your.host.ip.address* and then supply your local password when prompted. If you don't have direct web access, you may also need to take extra steps in setting up a secure SSH tunnel for your browser to work in step 3 below; please ask your local IT administrator for those details.

Once connected to the VM hosting GridAPPS-D, there are three basic steps involved in starting the RC1 demo:

1. **Start GridAPPS-D; this prepares the platform to configure a power system model and generate results.**
    (a) Open the first terminal to 172.20.128.20.
    (b) Switch to the gridappsd user by typing *sudo su - gridappsd*
    (c) Type *cd $HOME/gridappds_project/sources/GOSS-GridAPPS-D*
    (d) Type *./run-goss-test.sh* You may not see any output and it doesn't exit until you press Ctrl-C.
2. **Start the node server for the viz application, which allows you to start the simulation and see its results.**
    (a) Open a second terminal to 172.20.128.20.
    (b) Switch to the gridappsd user by typing *sudo su - gridappsd*

(c) Type *cd $HOME/gridappsd_project/sources/viz*

(d) Start the node server by typing *node server.js* You may not see any output and it doesn't exit until you press Ctrl-C.

3. **Start the viz demo. In some cases this may require a browser using an SSH tunnel.**

   (a) In a browser go to http://172.20.128.20:8082/ieee8500

   (b) Click on the IEEE 8500 link in the top left of the webpage (see Figure 1).

   (c) Click the play button in the top right of the webpage. It will take 5-10 seconds before you see the graphs being generated.

The demonstration runs a continuous loop of load variations with a Volt-Var Optimization (VVO) application *[CIT5]* controlling capacitor banks on the IEEE 8500-node test system *[CIT2]*. Most of Figure 1 is devoted to a map layout view of the test circuit, with updated labels for capacitor banks and voltage regulators. On the right-hand side, strip chart plots of the phase ABC voltages at capacitors and regulators, phase ABC substation power levels, and phase ABC regulator taps are continually updated. Capacitor bank labels on the circuit map view change between OPEN and CLOSED to show the bank status as load varies and the VVO application issues control commands. While GridAPPS-D runs the demo, GridLAB-D *[CIT8]* simulates power system operation and exchanges information with the VVO application using GOSS *[CIT6]* and FNCS *[CIT7]*.

For an orderly shutdown of the demonstration:

1. Close the web browser (i.e. step 3 above).
2. Enter Ctrl-C and then *Exit* in the node server's terminal window / tab (i.e. step 2 above).
3. Enter Ctrl-C and then *Exit* in the GOSS terminal window (i.e. step 1 above).
4. Log off the VM and the VPN.

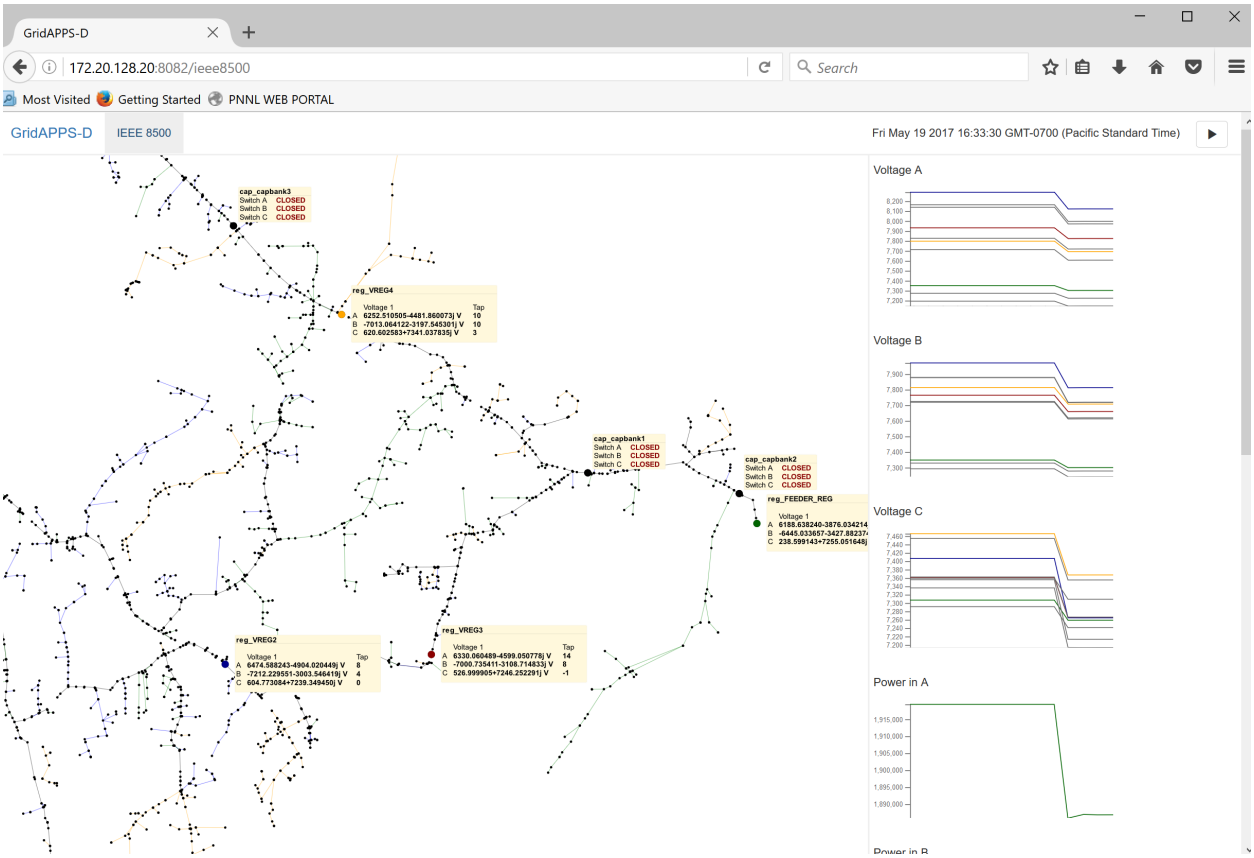


Figure 1: GridAPPS-D Release Cycle 1 Demo of the Volt-Var Optimization Running on the IEEE 8500-node test system.

3.2 Run Configuration

A publish and subscribe mechanism is utilized for clients and applications to communicate with the GridAPPS-D platform. The next sections describe the topics and expected message formats for starting a simulation, receiving data from a simulation, and interacting with an ongoing simulation. In order to start a simulation, you must send a simulation request to the process manager. The Process manager listens to topic **goss/gridappsd/process/request/simulation** and returns a simluationId. The simulation request should look like the example below. It should incluse a power system config section, which specifies which model to run. A simulation config, which includes parameters for the simulation, such as the simulator or power flow solver method. Within the power system config is the power system output, this specifies which objects and properties should be returned by the simulation, these should match the objects in the chosen model.

Once started, the ongoing process status messages will be sent on **goss/gridappsd/simulation/status/<Simulation_ID>**

{

power_system_config: the CIM model to be used in the simulation

```
"power_system_config": {
        "GeographicalRegion_name": "ieee8500nodecktassets_Region",
        "SubGeographicalRegion_name": "ieee8500nodecktassets_SubRegion",
        "Line_name": "ieee8500"
},
```

simulation_config: the paramaters used by the simulation

```
"simulation_config": {
        "start_time": "2009-07-21 00:00:00",
        "duration": "120",
        "simulator": "GridLAB-D",
        "timestep_frequency": "1000",
        "timestep_increment": "1000",
        "simulation_name": "ieee8500",
        "power_flow_solver_method": "NR",
```

simulation_output: the objects and fields to be returned by the simulation

```
"simulation_output": {
        "output_objects": [{
                "name": "rcon_FEEDER_REG",
                "properties": ["connect_type",
                "Control",
                "control_level",
                "PT_phase",
                "band_center",
                "band_width",
                "dwell_time",
                "raise_taps",
                "lower_taps",
                "regulation"]
        },
        .....]
},
```

model creation config: the paramaters used to generate the input file for the simulation

```
        "model_creation_config": {
                "load_scaling_factor": "1",
                "schedule_name": "ieeezipload",
                "z_fraction": "0",
                "i_fraction": "1",
                "p_fraction": "0"
        }
},
```

application config: inputs to any other applications that should run as part of the simluation, in this case the voltvar application

```
"application_config": {
        "applications": [{
                "name": "vvo",
                "config_string": "{\"static_inputs\": {\"ieee8500\" : {\"control_
↪method\": \"ACTIVE\", \"capacitor_delay\": 60, \"regulator_delay\": 60, \"desired_
↪pf\": 0.99, \"d_max\": 0.9, \"d_min\": 0.1,\"substation_link\": \"xf_hvmv_sub\",\
↪"regulator_list\": [\"reg_FEEDER_REG\", \"reg_VREG2\", \"reg_VREG3\", \"reg_VREG4\
↪"],\"regulator_configuration_list\": [\"rcon_FEEDER_REG\", \"rcon_VREG2\", \"rcon_
↪VREG3\", \"rcon_VREG4\"],\"capacitor_list\": [\"cap_capbank0a\",\"cap_capbank0b\", \
↪"cap_capbank0c\", \"cap_capbank1a\", \"cap_capbank1b\", \"cap_capbank1c\", \"cap_
↪capbank2a\", \"cap_capbank2b\", \"cap_capbank2c\", \"cap_capbank3\"], \"voltage_
↪measurements\": [\"nd_l2955047,1\", \"nd_l3160107,1\", \"nd_l2673313,2\", \"nd_
↪l2876814,2\", \"nd_m1047574,3\", \"nd_l3254238,4\"],       \"maximum_voltages\":␣
↪7500, \"minimum_voltages\": 6500,\"max_vdrop\": 5200,\"high_load_deadband\": 100,\
↪"desired_voltages\": 7000,   \"low_load_deadband\": 100,\"pf_phase\": \"ABC\"}}}"
        }]
}
```

}

3.3 Input/Output Topics

The FNCS Bridge input and output topics are the main driver behind controlling the simulation and subscribing to the latest data from the simulation. FNCS Bridge listens for input on topic **goss/gridappsd/fncs/input** and publishes responses on topic **goss/gridappsd/fncs/output**

Applications that wish to interact with the simulation can do so by subscribing to to the output topic and publishing commands to the input topic.

Each message should contain a command field, this specifies the operation that is either sent to teh input topic or responded to on the output topic. The available values for the command field are *isInitialized*, *nextTimeStep*, *update*, and *stop*. These are each described in more depth below.

- **isInitialized** *goss/gridappsd/fncs/input* - Checks to see if the simulator is initialized, meaning that it has established a connection to both GOSS and the simulator. This command takes no other parameters.

  ```
  {"command": "isInitialized"}
  ```

  goss/gridappsd/fncs/output - Returns simulator initializtion status (true/false) and any initialization messages.

  ```
  {"command": "isInitialized", "response":<true/false>, "output":"Any messages␣
  ↪from simulator regarding initialization"}
  ```

- **nextTimeStep** *goss/gridappsd/fncs/input* - Increments the simulator to the specified timestep, in a typical real-time simulation it will be incremented once per second. The only parameter is the current time in seconds (after the start of the simulation), this command will initiate the next time step in the simulator.

```
{"command": "nextTimeStep", "currentTime":<seconds from start of simulation>}
```

goss/gridappsd/fncs/output - Returns the current state of the objects and properties in the simulator, which objects and properties are specified by the simluation output. The is the output that applications will wish to subscribe to. The visualization application subscribes to this output to display the latest capacitor and regulator state. The volt-var optimization application subscribes to this output when managing voltage levels within the simulation


```
{"command": "nextTimeStep", "output": "{\"ieee8500\":{\"cap_capbank0a\":{\
↪"capacitor_A\":400000.0,\"control\":\"MANUAL\",\"control_level\":\"BANK\",\
↪"dwell_time\":100.0,\"phases\":\"AN\",\"phases_connected\":\"NA\",\"pt_
↪phase\":\"A\",\"switchA\":\"CLOSED\"},\"cap_capbank0b\":{\"capacitor_B\
↪":400000.0,\"control\":\"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\
↪":101.0,\"phases\":\"BN\",\"phases_connected\":\"NB\",\"pt_phase\":\"B\",\
↪"switchB\":\"CLOSED\"},\"cap_capbank0c\":{\"capacitor_C\":400000.0,\
↪"control\":\"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\":102.0,\
↪"phases\":\"CN\",\"phases_connected\":\"NC\",\"pt_phase\":\"C\",\"switchC\
↪":\"CLOSED\"},\"cap_capbank1a\":{\"capacitor_A\":300000.0,\"control\":\
↪"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\":100.0,\"phases\":\"AN\",
↪\"phases_connected\":\"NA\",\"pt_phase\":\"A\",\"switchA\":\"CLOSED\"},\
↪"cap_capbank1b\":{\"capacitor_B\":300000.0,\"control\":\"MANUAL\",\"control_
↪level\":\"BANK\",\"dwell_time\":101.0,\"phases\":\"BN\",\"phases_connected\
↪":\"NB\",\"pt_phase\":\"B\",\"switchB\":\"CLOSED\"},\"cap_capbank1c\":{\
↪"capacitor_C\":300000.0,\"control\":\"MANUAL\",\"control_level\":\"BANK\",\
↪"dwell_time\":102.0,\"phases\":\"CN\",\"phases_connected\":\"NC\",\"pt_
↪phase\":\"C\",\"switchC\":\"CLOSED\"},\"cap_capbank2a\":{\"capacitor_A\
↪":300000.0,\"control\":\"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\
↪":100.0,\"phases\":\"AN\",\"phases_connected\":\"NA\",\"pt_phase\":\"A\",\
↪"switchA\":\"CLOSED\"},\"cap_capbank2b\":{\"capacitor_B\":300000.0,\
↪"control\":\"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\":101.0,\
↪"phases\":\"BN\",\"phases_connected\":\"NB\",\"pt_phase\":\"B\",\"switchB\
↪":\"CLOSED\"},\"cap_capbank2c\":{\"capacitor_C\":300000.0,\"control\":\
↪"MANUAL\",\"control_level\":\"BANK\",\"dwell_time\":102.0,\"phases\":\"CN\",
↪\"phases_connected\":\"NC\",\"pt_phase\":\"C\",\"switchC\":\"CLOSED\"},\
↪"cap_capbank3\":{\"capacitor_A\":300000.0,\"capacitor_B\":300000.0,\
↪"capacitor_C\":300000.0,\"control\":\"MANUAL\",\"control_level\":\
↪"INDIVIDUAL\",\"dwell_time\":0.0,\"phases\":\"ABCN\",\"phases_connected\":\
↪"NCBA\",\"pt_phase\":\"\",\"switchA\":\"CLOSED\",\"switchB\":\"CLOSED\",\
↪"switchC\":\"CLOSED\"},\"nd_190-7361\":{\"voltage_A\":\"6410.387411-4584.
↪456974j V\",\"voltage_B\":\"-7198.592139-3270.308372j V\",\"voltage_C\":\
↪"642.547265+7539.531175j V\"},\"nd_190-8581\":{\"voltage_A\":\"6485.244722-
↪4692.686497j V\",\"voltage_B\":\"-7183.641237-3170.693324j V\",\"voltage_C\
↪":\"544.875720+7443.341013j V\"},\"nd_190-8593\":{\"voltage_A\":\"6723.
↪279162-5056.725836j V\",\"voltage_B\":\"-7494.205738-3101.034602j V\",\
↪"voltage_C\":\"630.475857+7534.534977j V\"},\"nd__hvmv_sub_lsb\":{\"voltage_
↪A\":\"6261.474438-3926.148203j V\",\"voltage_B\":\"-6529.409296-3466.
↪545236j V\",\"voltage_C\":\"247.131622+7348.295282j V\"},\"nd_l2673313\":{\
↪"voltage_A\":\"6569.522312-5003.052614j V\",\"voltage_B\":\"-7431.486583-
↪3004.840139j V\",\"voltage_C\":\"644.553331+7464.115915j V\"},\"nd_l2876814\
↪":{\"voltage_A\":\"6593.064915-5014.031801j V\",\"voltage_B\":\"-7430.
↪572726-3003.995538j V\",\"voltage_C\":\"643.473396+7483.558765j V\"},\"nd_
↪l2955047\":{\"voltage_A\":\"5850.305846-4217.166594j V\",\"voltage_B\":\"-
↪6729.652722-2987.617376j V\",\"voltage_C\":\"535.302083+7395.127354j V\"},\
↪"nd_l3160107\":{\"voltage_A\":\"5954.507575-4227.423005j V\",\"voltage_B\":\
↪"-6662.357613-3055.346879j V\",\"voltage_C\":\"600.213657+7317.832960j V\"},
↪\"nd_l3254238\":{\"voltage_A\":\"6271.490549-4631.254028j V\",\"voltage_B\
↪":\"-7169.987847-3099.952683j V\",\"voltage_C\":\"751.609655+7519.062260j V\
↪"},\"nd_m1047574\":{\"voltage_A\":\"6306.632406-4741.568924j V\",\"voltage_
↪B\":\"-7214.626338-2987.055914j V\",\"voltage_C\":\"622.058711+7442.125124j␣
↪V\"},\"rcon_FEEDER_REG\":{\"Control\":\"MANUAL\",\"PT_phase\":\"CBA\",\
↪"band_center\":126.5,\"band_width\":2.0,\"connect_type\":\"WYE_WYE\",\
```

- **update** *goss/gridappsd/fncs/input* - Sends an update command which can change the capacitor and regulator status within the simulator, this is used by the volt-var optimization application. Parameters include a message field, which contains the simulation name and the desired values for the objects to be updated.

```
{"command": "update", "message": {"ieee8500": {"reg_FEEDER_REG": {"tap_C": -3,
↪ "tap_B": -2, "tap_A": -1}, "reg_VREG4": {"tap_C": 1, "tap_B": 8, "tap_A":␣
↪8}, "reg_VREG2": {"tap_C": -1, "tap_B": 2, "tap_A": 6}, "reg_VREG3": {"tap_C
↪": -3, "tap_B": 6, "tap_A": 12}}}}
```

- **stop** *goss/gridappsd/fncs/input* - Stops the simulator and shuts down the bridge. No additional parameters are required

```
{"command": "stop"}
```

3.4 Starting Simulation Using API

GridAPPS-D communicates over a publish subscribe architecture implemented in ActiveMQ. A number of communication protocols are supported, including Openwire, STOMP, and websockets. Many programming languages support communication over these protocols, below are three examples.

Java

The request simulation can be called using the GOSS Client API. https://github.com/GridOPTICS/GOSS The Client API is used to send a run configuration to the GOSS simulation request topic, once the simulation has started it listens to the FNCS output topic for the simulation data.

```
import org.apache.http.auth.Credentials;
import org.apache.http.auth.UsernamePasswordCredentials;
import pnnl.goss.core.Client;
import pnnl.goss.core.Client.PROTOCOL;
import pnnl.goss.core.ClientFactory;
import pnnl.goss.core.GossResponseEvent;
import pnnl.goss.core.Request.RESPONSE_FORMAT;
import pnnl.goss.core.client.ClientServiceFactory;
import pnnl.goss.gridappsd.dto.PowerSystemConfig;
import pnnl.goss.gridappsd.dto.RequestSimulation;
import pnnl.goss.gridappsd.dto.SimulationConfig;
import pnnl.goss.gridappsd.utils.GridAppsDConstants;



ClientFactory clientFactory = new ClientServiceFactory();

Client client;

//Step1: Create GOSS Client
Credentials credentials = new UsernamePasswordCredentials(
                                  username, pw);
client = clientFactory.create(PROTOCOL.STOMP, credentials);


//Create Request Simulation object, you could also just pass in a json string with␣
↪the configuration
PowerSystemConfig powerSystemConfig = new PowerSystemConfig();
```

```
powerSystemConfig.GeographicalRegion_name = "ieee8500_Region";
powerSystemConfig.SubGeographicalRegion_name = "ieee8500_SubRegion";
powerSystemConfig.Line_name = "ieee8500";

SimulationConfig simulationConfig = new SimulationConfig();
simulationConfig.duration = 60;
simulationConfig.power_flow_solver_method = "";
simulationConfig.simulation_id = ""; //.setSimulation_name("");
simulationConfig.simulator = ""; //.setSimulator("");

SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
simulationConfig.start_time = sdf.format(new Date()); //.setStart_time("");

RequestSimulation requestSimulation = new RequestSimulation(powerSystemConfig,␣
↪simulationConfig);

Gson  gson = new Gson();
String request = gson.toJson(requestSimulation);
//Step3: Send configuration to the request simulation topic
String simulationId = client.getResponse(request, GridAppsDConstants.topic_
↪requestSimulation, RESPONSE_FORMAT.JSON)

//Subscribe to bridge output
client.subscribe("goss/gridappsd/fncs/output", new GossResponseEvent() {
    public void onMessage(Serializable response) {
      System.out.println("simulation output is: "+response);
    }
});
```

Websockets/Javascript

In order to call the simulation API from javascript you will need to install stomp.js In order to start the simulation through the websocket API you will need to send the configuration to the gridappsd simulation topic in the format descibed on *the Simulation Request page #simulation-request_*

```
<script src='js/jquery-2.1.4.min.js'></script>
<script src="js/stomp.js" type="text/javascript"></script>
configString = "...........  See developer resources"
simulationTopic = "/queue/goss/gridappsd/process/request/simulation";
gossHost = "gridappsdhost";
//Create client
var client = Stomp.client( "ws://"+gossHost+":61614");
client.heartbeat.incoming=0;
client.heartbeat.outgoing=0;

var connect_error_callback = function(error) {
   $("#debug").append("Error "+error + "\n");
};
var outputCallback = function(message){
   $("#debug").append("Output "+message.body + "\n");
}
//Make connection with server
client.connect( "username", "pw", connect_callback, connect_error_callback);

var request = JSON.stringify(JSON.parse(configField));
client.send(simulationTopic, {"reply-to" :"/temp-queue/response-queue"}, request);
      client.subscribe("/temp-queue/response-queue", function(message) {
          var simulationId = JSON.parse(message.body);
```

```
          $("#debug").append("Received Simulation ID: " +simulationId + "\n");
          client.subscribe("/topic/goss/gridappsd/simulation/status/"+simulationId,␣
↪statusCallback);
      });
client.subscribe("/topic/goss/gridappsd/fncs/output", outputCallback);
```

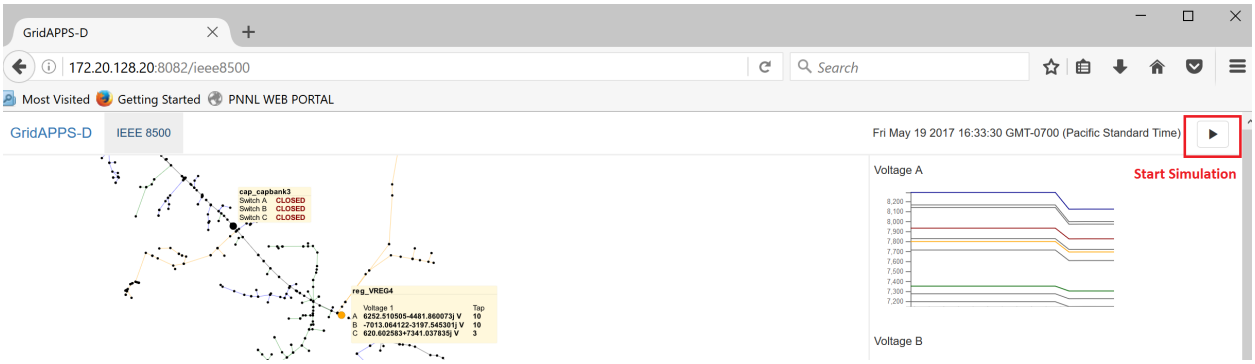
Python The python API requires that you install the stomp.py package, you can do this using pip with the command *pip install stomp.py* For additional documentation see https://github.com/jasonrbriggs/stomp.py/wiki/Simple-Example You will need to create a stomp connection, listen to the output topic, and then send a message to start the simulation.

```
import json
import sys
import stomp
import time

goss_output_topic = '/queue/goss/gridappsd/fncs/output'
goss_simulation_status_topic = '/topic/goss/gridappsd/simulation/status/'
gossConnection= None
isInitialized = None
simulationId = None

class GOSSStatusListener(object):
  def on_message(self, headers, msg):
    message = ''
    print('status ',msg)
  def on_error(self, headers, msg):
    print('simulation status error      ',msg)
class GOSSSimulationStartListener(object):
  def on_message(self, headers, msg):
    message = ''
    print('simulation start ', msg)
    _registerWithGOSS('system','manager', msg,gossServer='localhost',stompPort='61613
↪')
  def on_error(self, headers, msg):
    print('simulation start error     ',msg)

def _registerWithGOSS(username,password,simulationId,gossServer='localhost',
                      stompPort='61613'):
    '''Register with the GOSS server broker and return.

    Function arguments:
        gossServer -- Type: string. Description: The ip location
        for the GOSS server. It must not be an empty string.
            Default: 'localhost'.
        stompPort -- Type: string. Description: The port for Stomp
        protocol for the GOSS server. It must not be an empty string.
            Default: '61613'.
        username -- Type: string. Description: User name for GOSS connection.
        password -- Type: string. Description: Password for GOSS connection.

    Function returns:
        None.
    Function exceptions:
        RuntimeError()
    '''
    if (gossServer == None or gossServer == ''
            or type(gossServer) != str):
        raise ValueError(
```

```
            'gossServer must be a nonempty string.\n'
            + 'gossServer = {0}'.format(gossServer))
    if (stompPort == None or stompPort == ''
            or type(stompPort) != str):
        raise ValueError(
            'stompPort must be a nonempty string.\n'
            + 'stompPort = {0}'.format(stompPort))
    gossConnection = stomp.Connection12([(gossServer, stompPort)])
    gossConnection.start()
    gossConnection.connect(username,password)
    gossConnection.set_listener('GOSSStatusListener', GOSSStatusListener())
    gossConnection.subscribe(goss_output_topic,1)

def _startSimulation(username,password,gossServer='localhost',stompPort='61613'):
    simulationCfg = '{"power_system_config":{"GeographicalRegion_name":
↪"ieee8500nodecktassets_Region","SubGeographicalRegion_name":"ieee8500nodecktassets_
↪SubRegion","Line_name":"ieee8500"}, "simulation_config":{"start_time":"03/07/2017␣
↪00:00:00","duration":"60","simulator":"GridLAB-D","simulation_name":"my test␣
↪simulation","power_flow_solver_method":"FBS"}}'
    if (gossServer == None or gossServer == ''
            or type(gossServer) != str):
        raise ValueError(
            'gossServer must be a nonempty string.\n'
            + 'gossServer = {0}'.format(gossServer))
    if (stompPort == None or stompPort == ''
            or type(stompPort) != str):
        raise ValueError(
            'stompPort must be a nonempty string.\n'
            + 'stompPort = {0}'.format(stompPort))
    gossConnection = stomp.Connection12([(gossServer, stompPort)])
    gossConnection.start()
    gossConnection.connect(username,password, wait=True)
    gossConnection.set_listener('GOSSSimulationStartListener',
↪GOSSSimulationStartListener())
    gossConnection.subscribe(destination='/queue/reply',id=2)
    gossConnection.send(body=simulationCfg, destination=goss_simulation_topic,␣
↪headers={'reply-to': '/queue/reply'})
    time.sleep(3)
    print('sent simulation request')


if __name__ == "__main__":
    #TODO: send simulationId, fncsBrokerLocation, gossLocation,
    #stompPort, username and password as command line arguments

    _startSimulation('username','pw',gossServer='127.0.0.1',stompPort='61613')
```

3.5 Starting Simulation Using Viz Application

In the web-based visualization, click the Play button at the top right to start the simulation using the default run configuration. Behind the scenes this uses the websockets/javascript API described in a previous section.



3.6 Logging messages and process status

All processes should publish their log messages with process status to Process Manager. These processes include applications, simulations, services, and test runs.

3.6.1 Topic:

Log message with process status should be published on the following topic. Process id should be attached to the topic name at the end.

```
goss.gridappsd.process.log.simulation.[simulation_id]
goss.gridappsd.process.log.service.[service_id]
goss.gridappsd.process.log.application.[app_id]
goss.gridappsd.process.log.test.[test_id]
```

3.6.2 Message structure:

```
{
        "process_id": ""
        "timestamp": "",
        "process_status": "[started|stopped|running|error|passed|failed]",
        "log_message": "",
        "log_level": "[info|debug|error]",
        "store_to_db": [true|false]
}
```

3.6.3 Receving multiple logs:

User can either receive individual process's log by subcribing to topics mentioned above or recevie all logs of a type by subcribing to following topics.

```
goss.gridappsd.process.log.simulation.*
goss.gridappsd.process.log.service.*
goss.gridappsd.process.log.application.*
goss.gridappsd.process.log.test.*
```

Similarly, to receive to all logs subscribe to following topic:

```
goss.gridappsd.process.log.>
```

CHAPTER 4

GridAPPS-D Development Resources

This section is useful for developers for understanding or changing platform's internal workings and for those wishing to develop their own applications for GridAPPS-D. For developing application for GridAPPS-D platform see *Using GridAPPS-D* .

4.1 Developing Applications using GridAPPS-D

4.1.1 Supported Application Types

- Python
- Java (Jar)

4.1.2 Registering Application With Platform

Assumptions: GOSS-GridAPPS-D repository (https://github.com/GRIDAPPSD/GOSS-GridAPPS-D.git) is cloned under [ROOT_DIR]

1. Create a [app_name].config file in JSON format with keys and values as described below. where app_name should be unique for the application.

```
{
        "id":"app_name",
        "description":"This is desxription of the app",
        "creator":"orgnization name",
        "inputs":["topic.goss.gridappsd.input1", "topic.goss.gridappsd.input2", ..],
        "outputs":["topic.goss.gridappsd.output1", "topic.goss.gridappsd.output2", ..
↪],
        "options":"space saperated command line input options",
        "execution_path":"absolute/execution/path",
        "type":"PYTHON|JAVA",
        "launch_on_startup":true|false,
```

```
        "prereqs":["other_app","other_service",..],
        "multiple_instances":true|false
}
```

2. Put [app_name].config file in applications folder under cloned repository location
3. Put your application under applications/[app_name] folder under cloned repository location as shown below.

```
applications
        [app_name]
                app
                        Your application goes here
                test
                        Test scripts for your application goes here.
```

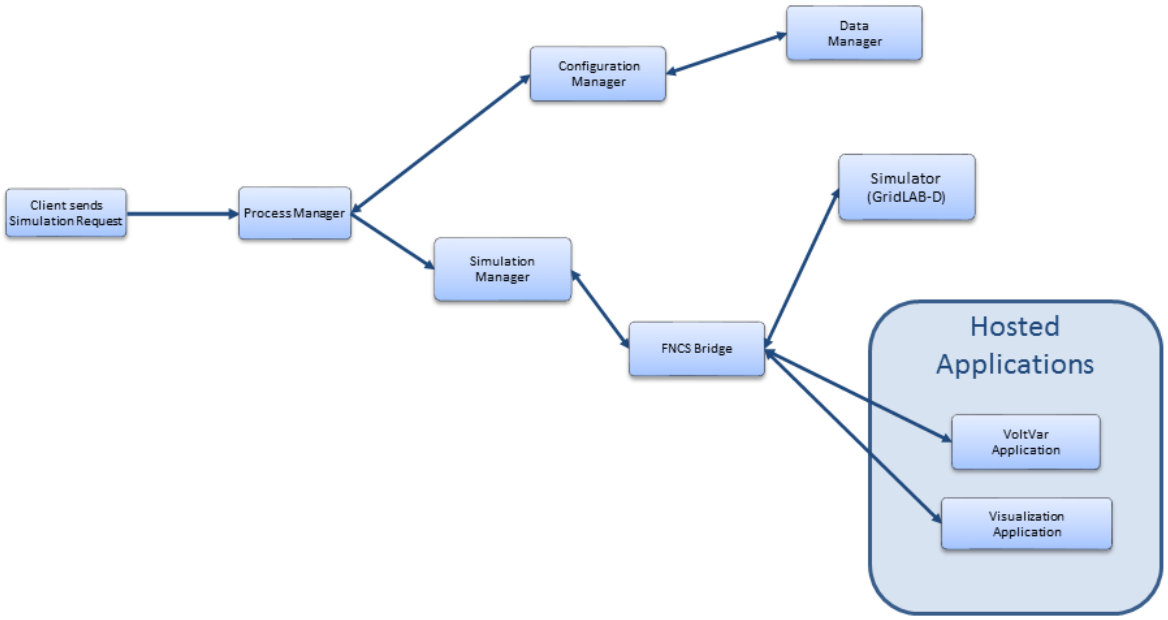
See Using GridAPPSD section for details on starting a simulation from an application and communicating with plaform. It also has an example in Python and Java to start a simulation.

4.2 Eclipse IDE Setup

1. **Download or clone the repository from github**
   (a) Install github desktop https://desktop.github.com/ or sourcetree https://www.atlassian.com/software/sourcetree/overview and Clone the GOSS-GridAPPS-D repository (https://github.com/GRIDAPPSD/GOSS-GridAPPS-D)
   (b) Or download the source (https://github.com/GRIDAPPSD/GOSS-GridAPPS-D/archive/master.zip)
2. Install java 1.8 SDK and set JAVA_HOME variable
3. Install Eclipse http://www.eclipse.org/downloads/packages/release/Mars/1 (Mars 4.5.1 or earlier, 4.5.2 appears to have bugs related to bundle processing) TODO what about neon?
4. Open eclipse with workspace set to GOSS-GridAPPS-D download location, eg. C:UsersusernameDocumentsGOSS-GridAPPS-D
5. Install BNDTools plugin: Help->Install New Software->Work with: http://dl.bintray.com/bndtools/bndtools/3.0.0 and Install Bndtools 3.0.0 or earlier
6. **Import projects into workspace**
   (a) File->Import General->Existing Projects into workspace
   (b) Select root directory, GOSS-GridAPPS-D download location
   (c) Select cnf, pnnl.goss.gridappsd
7. If errors are detected, Right click on the pnnl.goss.gridappsd project and select release, then release all bundles
8. **If you would like to you a local version of GOSS-Core (Optional)**
   (a) Update cnf/ext/repositories.bnd
   (b) Select source view and add the following as the first line
   (c) aQute.bnd.deployer.repository.LocalIndexedRepo;name=GOSS Local Release;local=/GOSS-Core2/cnf/releaserepo;pretty=true,
   (d) verify by switching to bndtools and verify that there are packages under GOSS Local Relase
9. Open pnnl.goss.gridappsd/bnd.bnd, Rebuild project, you should not have errors

10. Open pnnl.goss.gridappsd/run.bnd.bndrun and click Run OSGI

4.3 Execution Workflow



Process Manager - The workflow begins when a simulation request is sent to the request topic monitored by the Process Manager, the process manager gathers the necessary configurations from the Configuration Manager. Then sends the configuration to the simulation manager to run the simulation.

Configuration Manager - The configuration manager parses the request and builds the necessary configuration files. It also uses the data manager to pull the model data from the CIM database.

Data Manager - The data manager accesses the CIM database to build the model files used by the simulator.

Simulation Manager - The simulation manager launches the simulator and other required applications such as the FNCS bridge, FNCS, and the VoltVar application. It is in charge of managing the timing of the simulation and reporting output from the simulation out to the simulation status topic.

FNCS Bridge - Serves as input and output from the simulator to the rest of GridAPPS-D, receives initialization, timestep, update, and finalize requests from the simulation manager and other applications, such as voltvar. It also publishes output from the simulator on a pre-defined topic for the simulation manager and other applications to subscribe to.

Simulator - In this case GridLAB-D serves as the simulator.

Hosted Application - Applications can be developed to use the data generated by the simulation and submit feedback and updates to the simulator. Two examples of this have been developed in RC1, the VoltVar application and a vizualization application

Log Manager - Process Manager recieves a log message. It retrieves the username associated with the message and forwards the message and username to Log Manager. Log Manager writes the message on a file and if store_to_db key is true in log message then log manager calls the data manager to store the log message in the database.

4.4 Messaging

Please see run_config for more details.

4.5 CIM Documentation

This section summarizes the use of a reduced-order CIM[1] to support feeder modeling for the volt-var application in Release Cycle 1 (RC1). The full CIM includes over 1100 tables in SQL, each one corresponding to a UML class, enumeration or datatype. In RC1, we're using approximately 100 such entities, mapped onto 100+ tables in SQL. Later versions of GridAPPS-D will use a triple-store or graph database, both of which appear to be better suited for CIM.

The CIM subset described here is based on the profile adopted for the most recent distribution CIM interoperability test, which was held in 2011 at EDF. For GridAPPS-D, we have updated that profile for compatibility with the most recent CIM base standard.

4.5.1 Class Diagrams for the Profile

Figure 1 through Figure 11 present the UML class diagrams generated from Enterprise Architect[2]. These diagrams provide an essential roadmap for understanding:

1. How to ingest CIM XML from various sources into the database
2. How to generate native GridLAB-D input files from the database

For those unfamiliar with UML class diagrams:

1. Lines with an arrowhead indicate class inheritance. For example, in Figure 1, ACLineSegment inherits from Conductor, ConductingEquipment, Equipment and then PowerSystemResource. ACLineSegment inherits all attributes and associations from its ancestors (e.g. length), in addition to its own attributes and ancestors.
2. Lines with a diamond indicate composition. For example, in Figure 1, ConnectivityNodes make up a TopologicalNode, and then TopologicalNodes make up a TopologicalIsland.
3. Lines without a terminating symbol are associations. For example, in Figure 1, ACLineSegment has (through inheritance) a BaseVoltage, Location and EquipmentContainer.
4. Italicized names at the top of each class indicate the ancestor (aka superclass), in cases where the ancestor does not appear on the diagram. For example, in Figure 1, PowerSystemResource inherits from IdentifiedObject.

[1] See http://cimug.ucaiug.org/default.aspx and the EPRI CIM Primer at: http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000003002006001

[2] Suggest "Corporate Edition" from http://www.sparxsystems.com/ for working with CIM UML. The free CIMTool is still available at http://wiki.cimtool.org/index.h tml, but support is being phased out.

Please see *OSPRREYS_RC1.eap*[3] in the repository[4] on GitHub for the latest updates. The EnterpriseArchitect file includes a description of each class, attribute and association. It can also generate HTML documentation of the CIM, with more detail than provided here.

The diagrammed UML associations have a role and cardinality at each end, source and target. In practice, *only one end of each association* is profiled and implemented in SQL. In some cases, the figure captions indicate which end, but see the CIM profile for specific definitions, as described in the object diagram section.

Nearly every CIM class inherits from IdentifiedObject, from which we use two attributes:

1. mRID is the "master identifier" that must be unique and persistent among all instances. It's often used as the RDF resource identifier, and is often a GUID.
2. Name is a human-readable identifier that need not be unique.

[3] OSPRREYS is an older name for GridAPPS-D
[4] https://github.com/GRIDAPPSD/Powergrid-Models/CIM

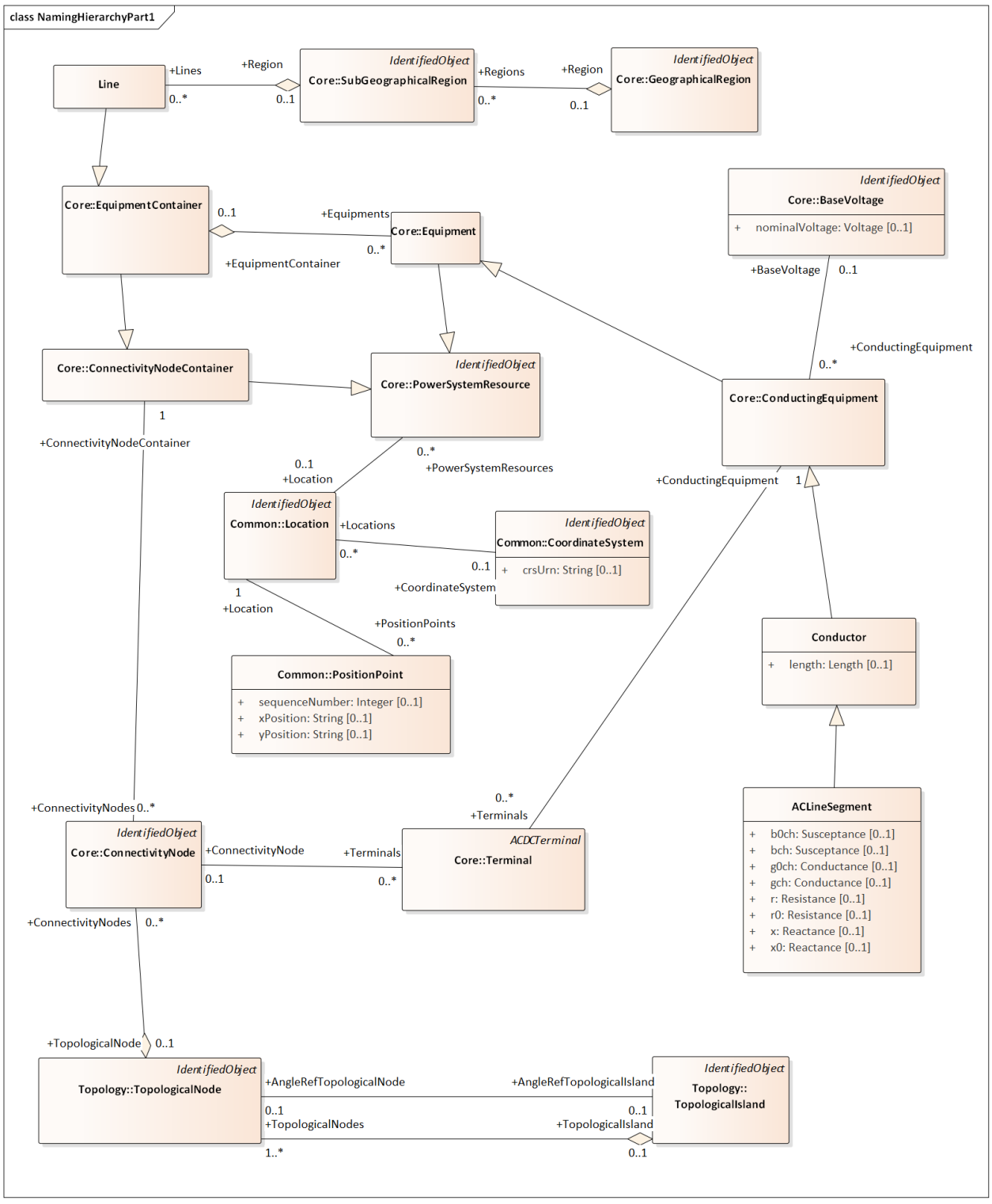


Figure 1: Placement of ACLineSegment into a Line (aka Feeder). In GridAPPS-D, the Line is the EquipmentContainer for all power system components and the ConnectivityNodeContainer for all nodes. It also corresponds to one TopologicalIsland. It's part of a SubGeographicalRegion and GeographicalRegion for proper context with other CIM models. For visualization, ACLineSegment can be drawn from a sequence of PositionPoints associated via Location. The Terminals are free-standing; two of them will "reverse-associate" to the ACLineSegment as ConductingEquipment, and each terminal also has one ConnectivityNode. In RC1, we have a one-to-one association between Connec-

tityNode and TopologicalNode. The AngleRefTopologicalNode association can be used to identify the swing bus for GridLAB-D. Otherwise, we're only using the topology classes to facilitate state variables, as described in Figure 11. The Terminal:phases attribute is not used; instead, phases will be defined in the ConductingEquipment instances. The associated BaseVoltage:nominalVoltage attribute is important for many of the classes that don't have their own rated voltage attributes, for example, EnergyConsumer.

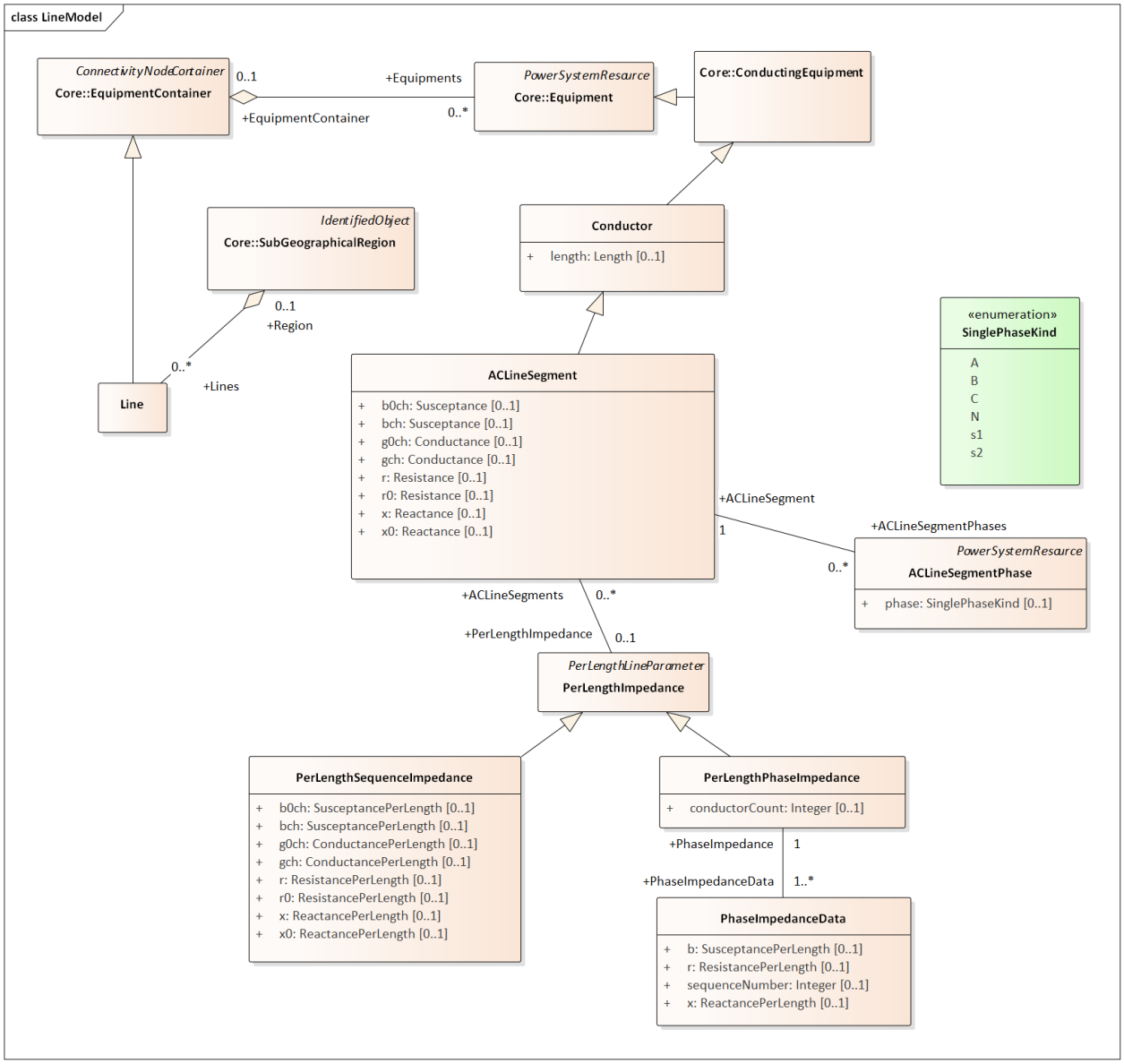


Figure 2: There are four different ways to specify ACLineSegment impedances. In all cases, Conductor:length is required. The first way is to specify the individual ACLineSegment attributes, which are sequence impedances and admittances, leaving PerLengthImpedance null. The second way is to specify the same attributes on an associated PerLengthSequenceImpedance, in which case the ACLineSegment attributes should be null. The third way is to associate a PerLengthPhaseImpedance, leaving the ACLineSegment attributes null. Only conductorCount from 1 to 3 is supported, and there will be 1, 3 or 6 reverse-associated PhaseImpedanceData instances that define the lower triangle of the Z and Y matrices per unit length. The sequenceNumber goes from 1 to N+N*(N-1)/2 in column order. The fourth way to specify impedance is by wire/cable and spacing data, as described with Figure 10. If there are ACLineSegmentPhase instances reverse-associated to the ACLineSegment, then per-phase modeling applies. There are several use cases for ACLineSegmentPhase: 1) single-phase or two-phase primary, 2) low-voltage secondary using phases s1 and s2, 3) associated wire data where the neutral exists, 4) associated wire data where the phase wires are

different. It is the application's responsibility to propagate phasing through terminals to other components, and to identify any miswiring.

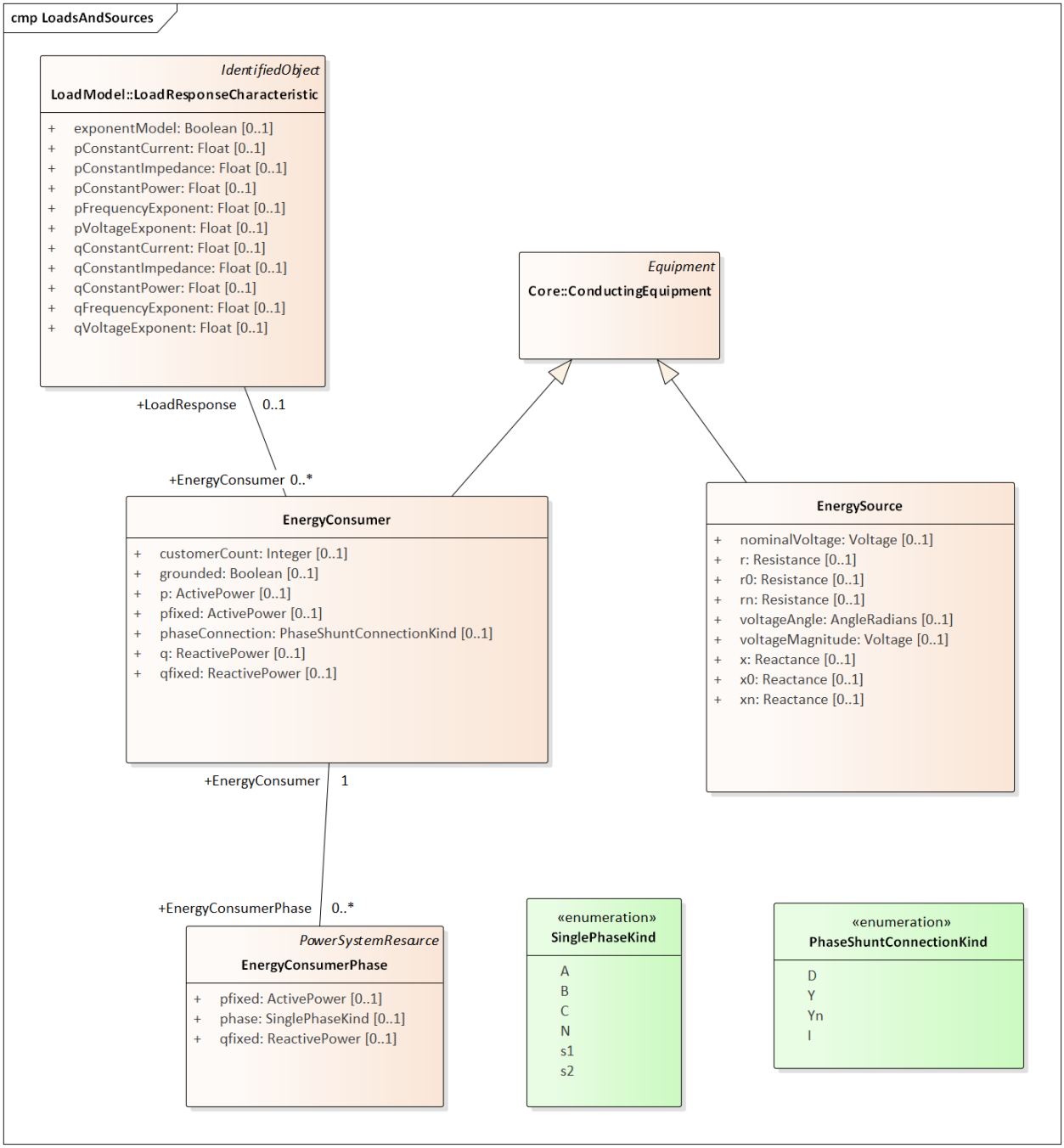


Figure 3: The EnergySource is balanced three-phase, representing a transmission system source (this is probably not the way we'll model distributed generation in future versions). The EnergyConsumer is a ZIP load, possibly unbalanced, with an associated LoadResponse instance defining the ZIP coefficients. For three-phase delta loads, the phaseConnection is D and the three reverse-associated EnergyConsumerPhase instances will have phase=A for the AB load, phase=B for the BC load and phase=C for the AC load. A three-phase wye load may have either Y or Yn for the phaseConnection. Single-phase and two-phase loads, including secondary loads, should have phaseConnection=I (for individual).

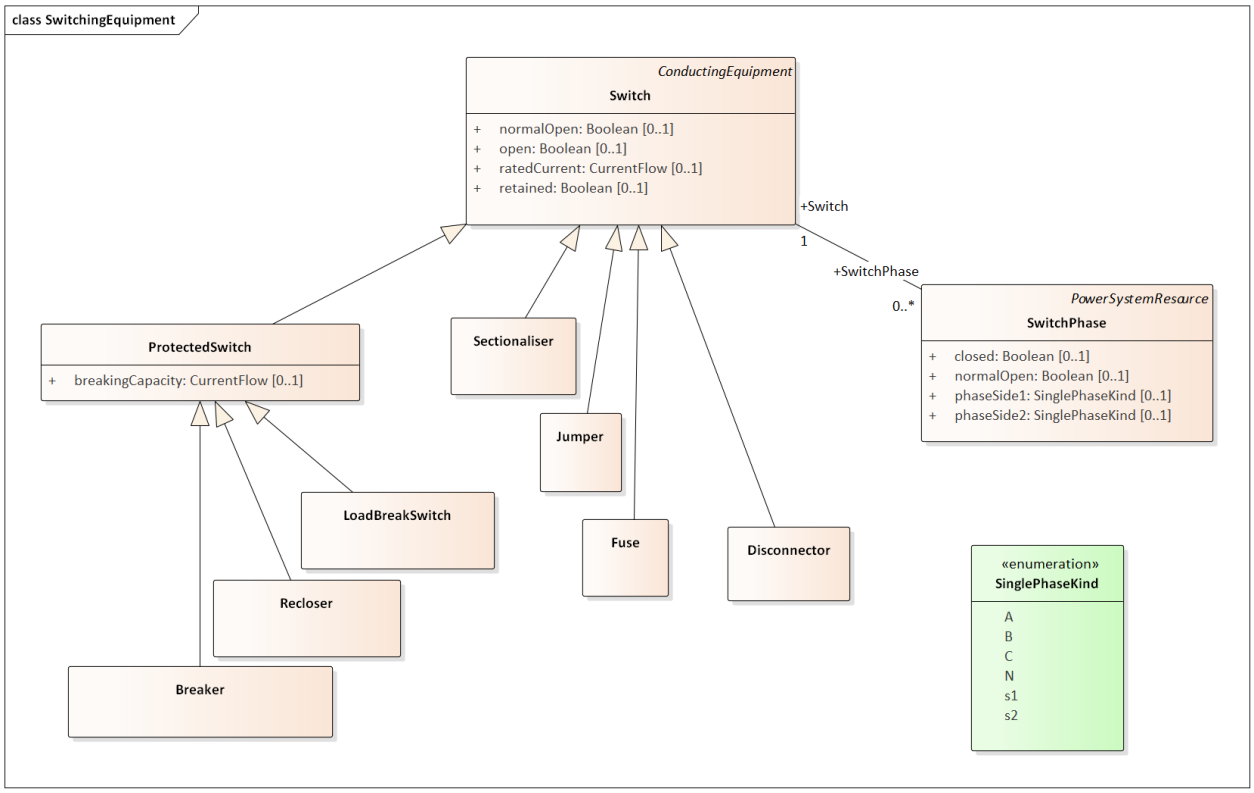


Figure 4: There are seven different kinds of Switch supported in the CIM, and all of them have zero impedance. They would all behave the same in power flow analysis, and all would require many more attributes than are defined in CIM to support protection analysis. The use cases for SwitchPhase include 1) single-phase, two-phase and secondary switches, 2) one or two conductors open in a three-phase switch or 3) transpositions, in which case phaseSide1 and phaseSide2 would be different.

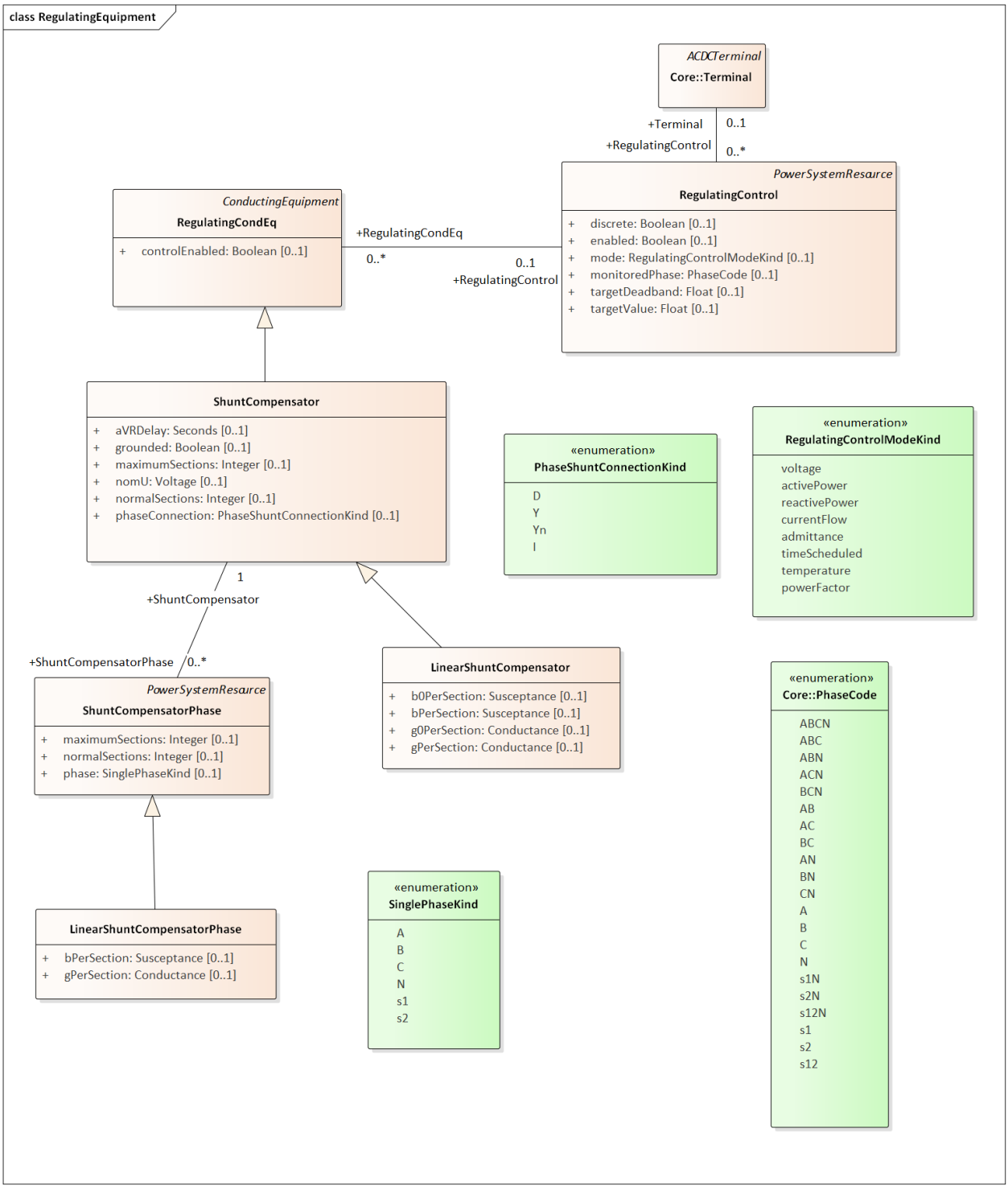


Figure 5: On the left, LinearShuntCompensator and LinearShuntCompensatorPhase define capacitor banks, in a way very similar to EnergyConsumer in Figure 3. The kVAR ratings must be converted to susceptance based on the nominal voltage, nomU. Note that aVRDelay is really a capacitor control parameter, to be used in conjunction with RegulatingControl on the right-hand side. The RegulatingControl associates to the controlled capacitor bank via RegulatingCondEq, and to the monitored location via Terminal. There is no support for a PT or CT ratio, so targetDeadband and targetValue have to be in primary volts, amps, vars, etc.

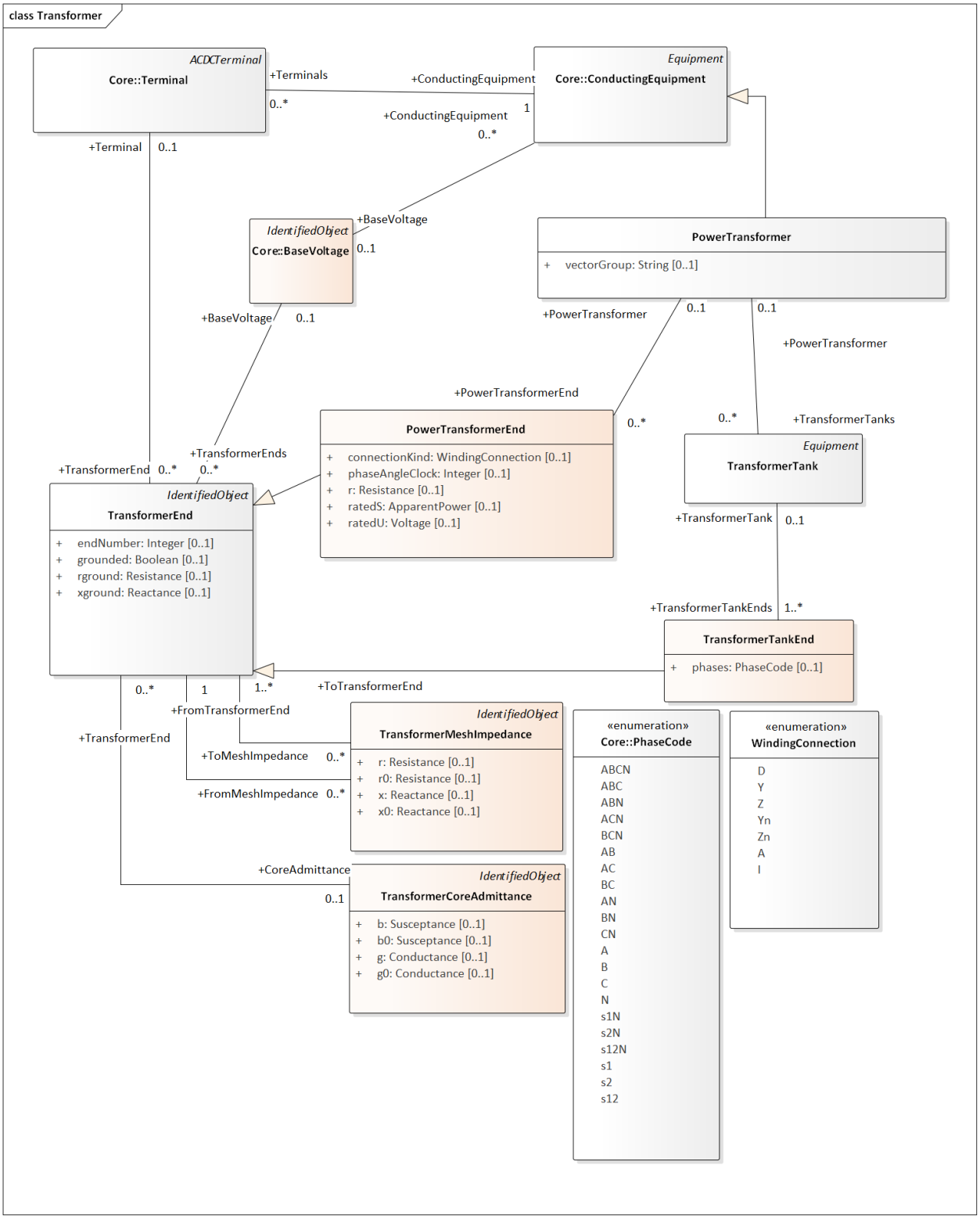


Figure 6: PowerTransformers may be modeled with or without tanks, and in both cases vectorGroup should be specified according to IEC transformer standards (e.g. Dy1 for many substation transformers). The case without tanks is most suitable for balanced three-phase transformers that won't reference catalog data; any other case should use tank-level modeling. In the tankless case, each winding will have a PowerTransformerEnd that associates to both a Terminal and a BaseVoltage, and the parent PowerTransformer. The impedance and admittance parameters are

defined by reverse-associated TransformerMeshImpedance between each pair of windings, and a reverse-associated TransformerCoreAdmittance for one winding. The units for these are ohms and siemens based on the winding voltage, rather than per-unit. WindingConnection is similar to PhaseShuntConnectionKind, adding Z and Zn for zig-zag connections and A for autotranformers. If the transformer is unbalanced in any way, then TransformerTankEnd is used instead of PowerTransformerEnd, and then one or more TransformerTanks may be used in the parent PowerTransformer. Some of the use cases are 1) center-tapped secondary, 2) open-delta and 3) EHV transformer banks. Tank-level modeling is also required is using catalog data, as described with Figure 9.

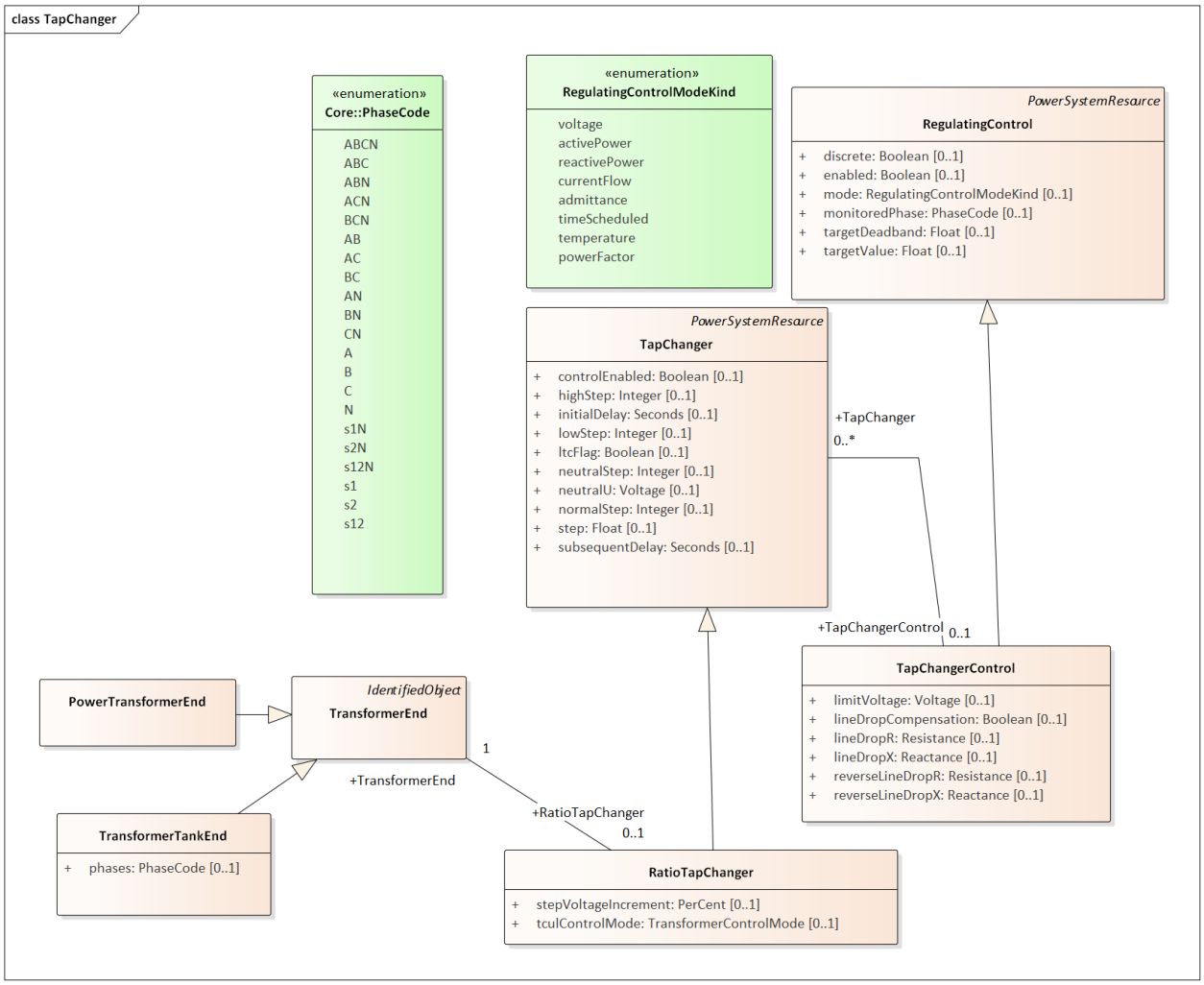


Figure 7: A RatioTapChanger can represent a transformer tap changer on the associated TransformerEnd. The RatioTapChanger has some parameters defined in a direct-associated TapChangerControl, which inherits from RegulatingControl some of the same attributes used in capacitor controls (Figure 5). Therefore, a line voltage regulator in CIM includes a PowerTransformer, a RatioTapChanger, and a TapChangerControl. The CT and PT parameters of a voltage regulator can only be described via the AssetInfo mechanism, described with Figure 8.

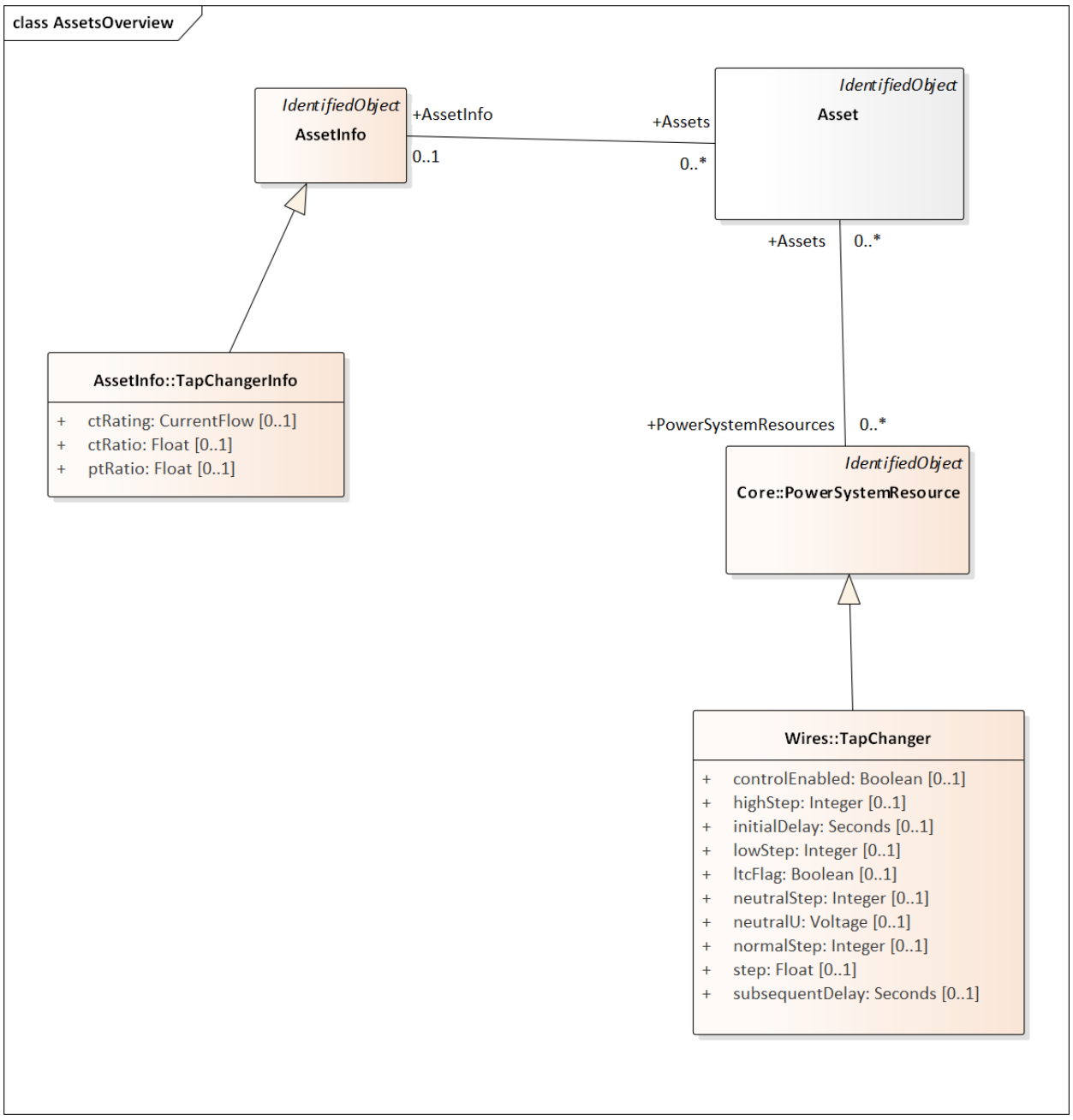


Figure 8: Many distribution software packages use the concept of catalog data, aka library data, especially for lines and transformers. We use the Asset and AssetInfo packages to implement this in CIM. Here, the TapChangerInfo class includes the CT rating, CT ratio and PT ratio parameters needed for line drop compensator settings in voltage regulators. Catalog data is a one-to-many, and sometimes a many-to-many, relationship. For these lookups, we create an Asset instance that has one association to AssetInfo, and one-to-many associations to PowerSystemResources. In this case, many TapChangers can share the same TapChangerInfo data, which saves space and provides consistency.

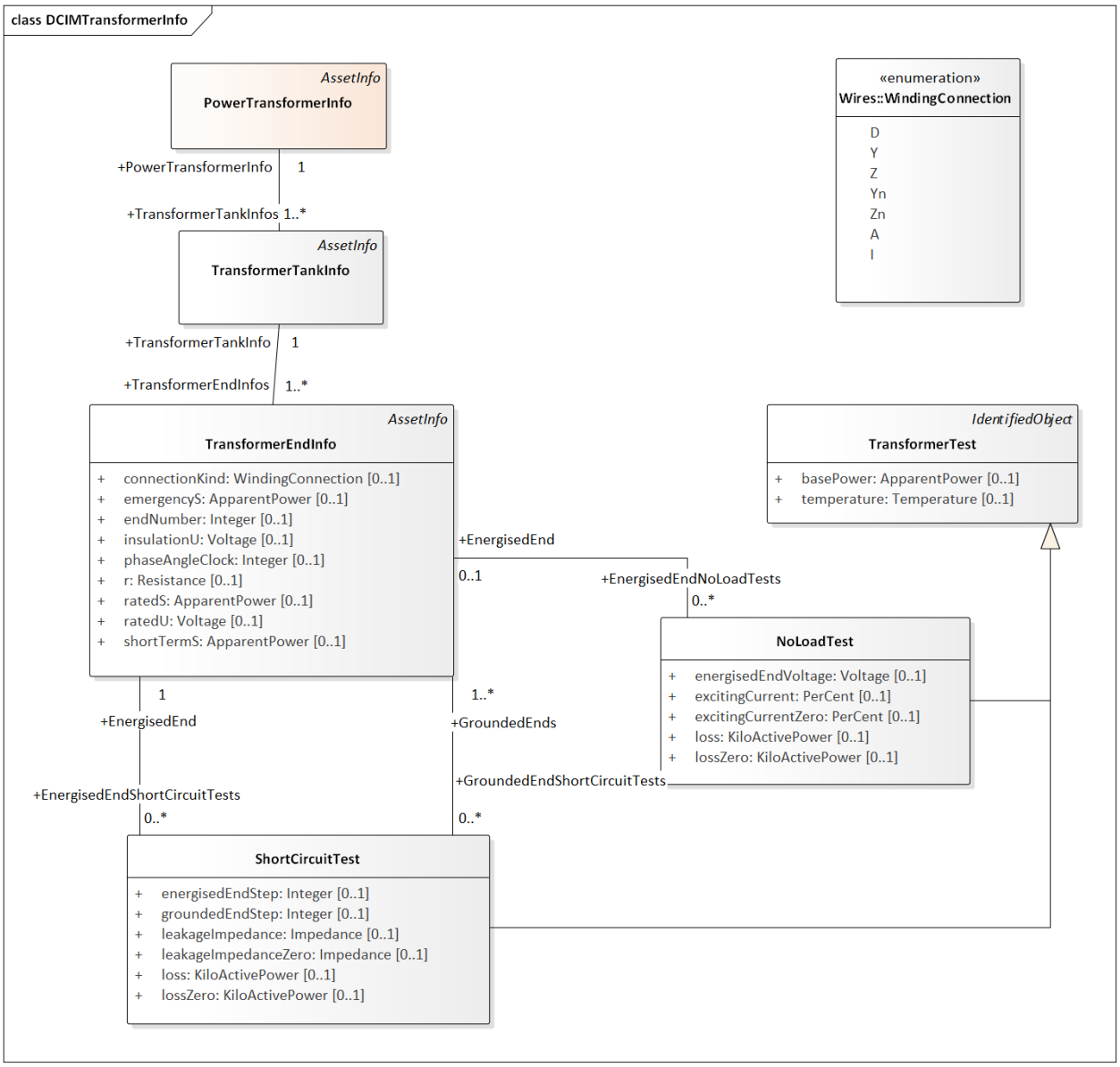


Figure 9: The catalog mechanism for transformers will associate a TransformerTank (Figure 6) with TransformerTankInfo (here), via the one-to-many mechanism described in Figure 8. The PowerTransformerInfo collects TransformerTankInfo by reverse association, but it does not link with PowerTransformer. In other words, the physical tanks are cataloged because transformer testing is done on tanks. One possible use for PowerTransformerInfo is to help organize the catalog. It's important that TransformerEndInfo:endNumber (here) properly match the TransformerEnd:endNumber (Figure 6). The shunt admittances are defined by NoLoadTest on a winding / end, usually just one such test. The impedances are defined by a set of ShortCircuitTests; one winding / end will be energized, and one or more of the others will be grounded in these tests.

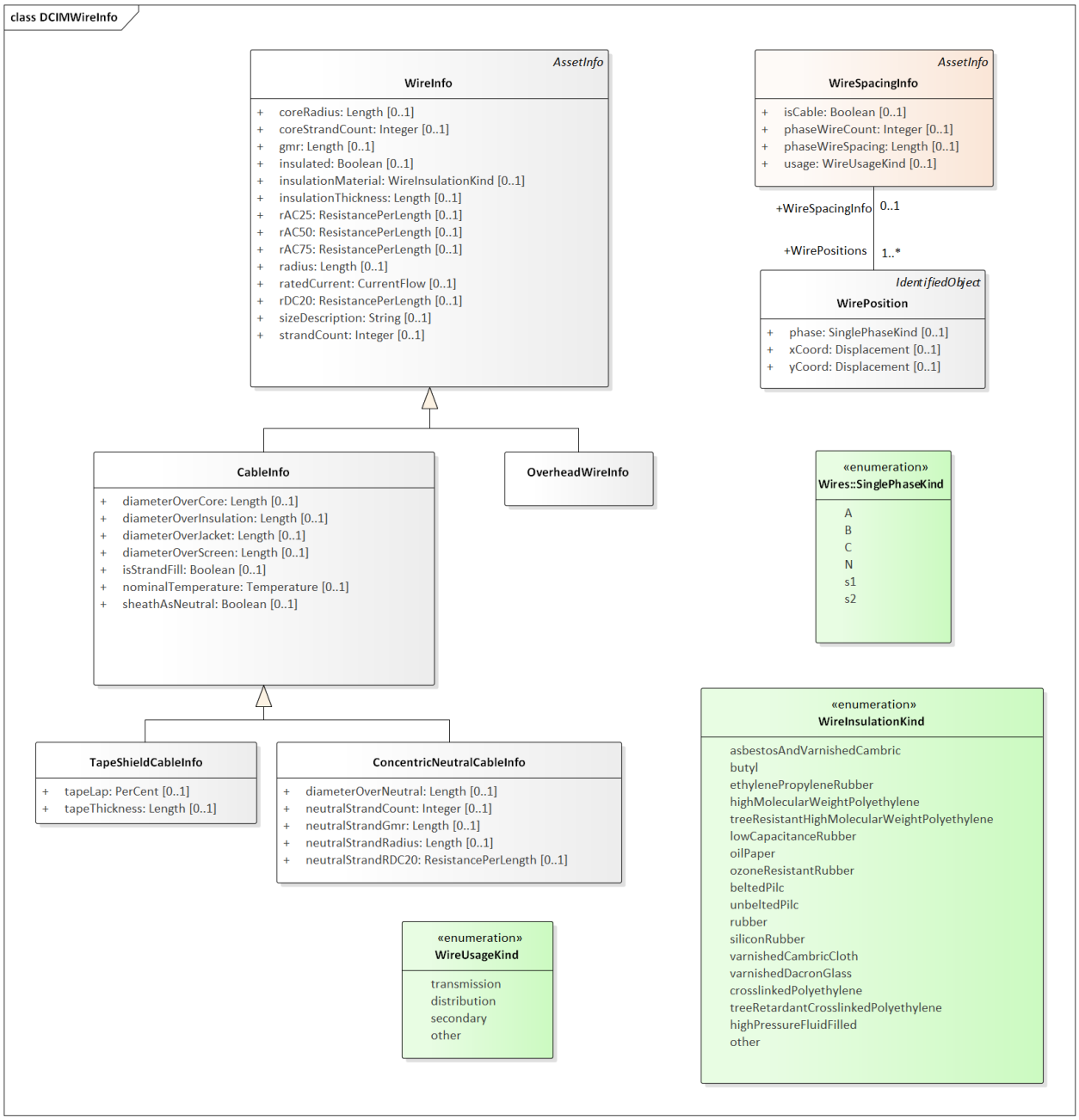


Figure 10: The catalog / library mechanism for ACLineSegment will have a WireSpacingInfo associated as in Figure 9. This will indicate whether the line is overhead or underground. phaseWireCount and phaseWireSpacing define optional bundling, so these will be 1 and 0 for distribution. The number of phase and neutral conductors is actually defined by the number of reverse-associated WirePosition instances. For example, a three-phase line with neutral would have four of them, with phase = A, B, C and N. On the right-hand side, concrete classes OverheadWireInfo, TapeShieldCableInfo and ConcentricNeutralCableInfo may be associated (as in Figure 9) to either ACLineSegment or ACLineSegmentPhase. The association to ACLineSegment only applies for three-conductor, three-phase lines all using the same wire data, or to supply just the ratedCurrent attribute. All other use cases would associate to ACLineSegmentPhase. It's the application's responsibility to calculate impedances from this data. In particular, soil resistivity and dielectric constants are not included in the CIM. Typical dielectric constant values might be defined for each WireInsulationKind.

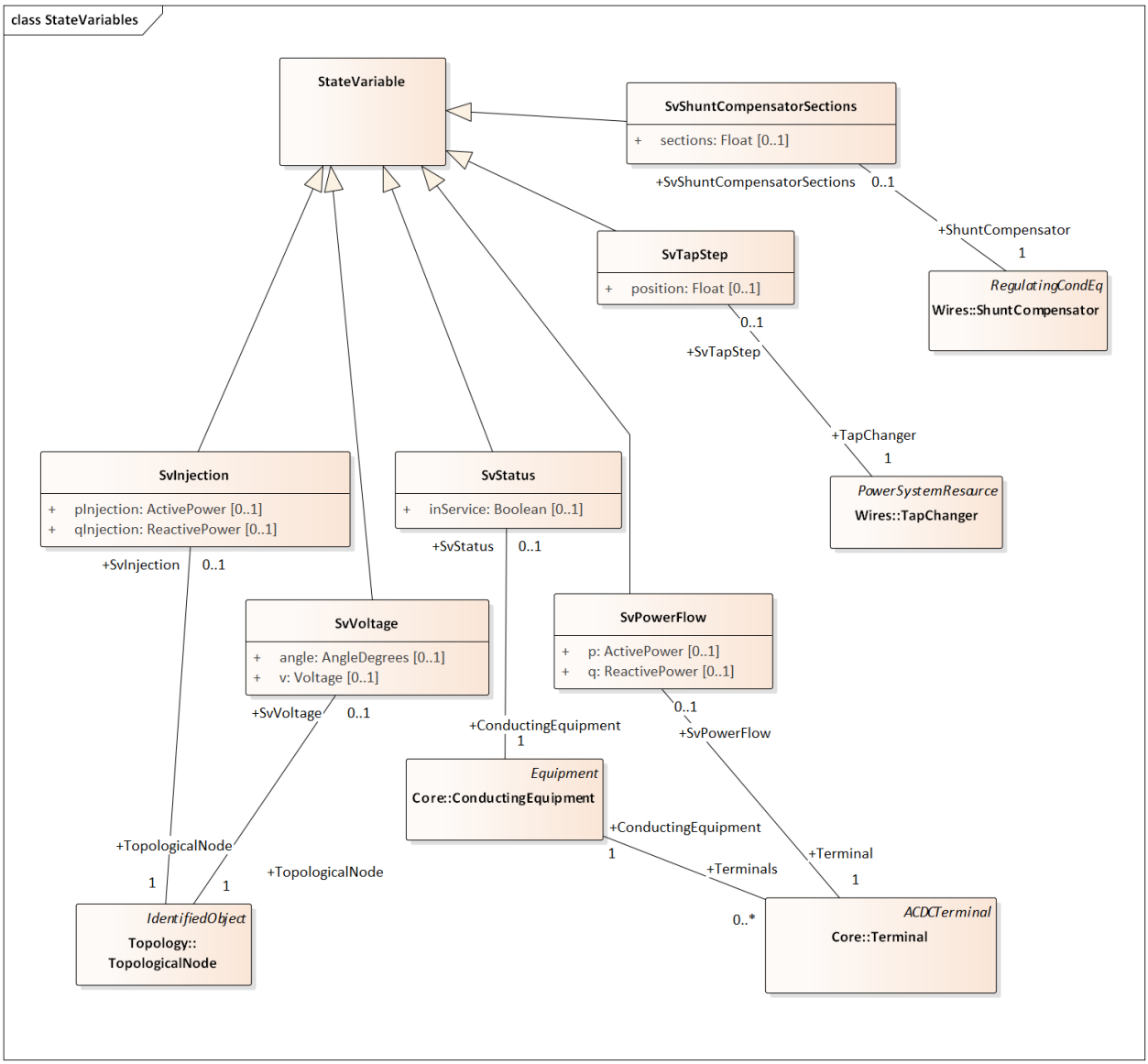


Figure 11: The CIM state variables package might be used to mimic sensor locations and values on the distribution system. Voltages are measured on TopologicalNodes, power flows are measured at Terminals, step positions are measured on TapChangers, status is measured on ConductingEquipment, and on/off state is measured on ShuntCompensators. The "injections" have been included here, but there may not be a use case for them in distribution. On the other hand, we would need an SvCurrent, which was probably not included in the CIM because of its transmission system heritage. Attributes for sensor characteristics would also have to be added in future versions of GridAPPS-D.

4.5.2 Typical Queries

These queries focus on requirements of the first volt-var application.

1. Capacitors (Figure 5, Figure 12, Figure 13, Figure 14)
   (a) Create a list of capacitors with bus name (Connectivity Node in Figure 1), kVAR per phase, control mode, target value and target deadband
   (b) For a selected capacitor, update the control mode, target value, and target deadband

2. Regulators (Figure 7, Figure 8, Figure 12, Figure 29)
   (a) List all transformers that have a tap changer attached, along with their bus names and kVA sizes
   (b) Given a transformer that has a tap changer attached, list or update initialDelay, step, subsequentDelay, mode, targetDeadband, targetValue, limitVoltage, lineDropCompensation, lineDropR, lineDropX, reverseLineDropR and reverseLineDropX
3. Transformers (Figure 6, Figure 9)
   (a) Given a bus name or load (Figure 3), find the transformer serving it (Figure 16, Figure 19)
   (b) Find the substation transformer, defined as the largest transformer (by kVA size and or highest voltage rating)
   (c) List the transformer catalog (Figure 9, Figure 20) with name, highest ratedS, list of winding ratedU in descending order, vector group (https://en.wikipedia.org/wiki/Vector_group used with connectionKind and phaseAngleClock), and percent impedance
   (d) List the same information as in item c, but for transformers (Figure 6) and also retrieving their bus names. Note that a transformer can be defined in three ways
      i. Without tanks, for three-phase, multi-winding, balanced transformers (Figure 16 and Figure 17).
      ii. With tanks along with TransformerTankInfo (Figure 9) from a catalog of "transformer codes", which may describe balanced or unbalanced transformers. See Figure 19 and Figure 20.
      iii. With tanks for unbalanced transformers, and TransformerTankInfo created on-the-fly. See Figure 19 and Figure 20.
   (e) Given a transformer (Figure 6), update it to use a different catalog entry (TransformerTankInfo in Figure 9)
4. Lines (Figure 2, Figure 10, Figure 12)
   (a) List the line and cable catalog entries that meet a minimum ratedCurrent and specific WireUsageKind. For cables, be able to specify tape shield vs. concentric neutral, the WireInsulationKind, and a minimum insulationThickness. (Figure 27)
   (b) Given a line segment (Figure 2) update to use a different linecode (Figure 10, Figure 26)
   (c) Given a bus name, list the ACLineSegments connected to the bus, along with the length, total r, total x, and phases used. There are four cases as noted in the caption of Figure 2, and see Figure 23 through Figure 26.
   (d) Given a bus name, list the set of ACLineSegments (or PowerTransformers and Switches) completing a path from it back to the EnergySource (Figure 3). Normally, the applications have to build a graph structure in memory to do this, so it would be very helpful if a graph/semantic database can do this.
5. Voltage and other measurements (Figure 1, Figure 11)
   (a) Given a bus, attach a voltage measurement point (SvVoltage, Figure 30)
   (b) List all voltage measurement points and their buses, and for each bus, list the phases actually present
   (c) For tap changer position (SvTapStep, Figure 31), attach and list measurements as in items a and b
   (d) For capacitor switch status (SvShuntCompensatorSections, Figure 32), attach and list measurements as in items a and b
6. Loads (Figure 3, Figure 28)
   (a) Given a bus name, list and total all of the loads connected by phase, showing the total p and q, and the composite ZIP coefficients
7. Switching (Figure 4, Figure 22)

(a) Given a bus name, trace back to the EnergySource and list the switches encountered, grouped by type (i.e. the leaf class in Figure 4). Also include the ratedCurrent, breakingCapacity if applicable, and open/close status. If SwitchPhase is used, show the phasing on each side and the open/close status of each phase.

(b) Given switch, toggle its open/close status.

4.5.3 Object Diagrams for Queries

This section contains UML object diagrams for the purpose of illustrating how to perform typical queries and updates. For those unfamiliar with UML object diagrams:

1. Each object will be an instance of a class, and more than one instance of a class can appear on the diagram. For example, Figure 12 shows two ConnectivityNode instances, one for each end of a ConductingEquipment.
2. The object name (if specified and important) appears before the colon (:) above the line, while the UML class appears after the colon. Every object in CIM will have a unique ID, and a name (not necessarily unique), even if not shown here.
3. Some objects may be shown with run-time state below the line. These are attribute value assignments, drawn from those available in the UML class or one of the class ancestors. The object may have more attribute assignments, but only those directly relevant to the figure captions are shown in the diagrams of this section.
4. Object associations are shown with solid lines, role names, and multiplicities similar to the UML class diagrams. One important difference is that only one way of navigating a particular association will be defined in the profile. For example, the lower left corner of Figure 1 shows a two-way link between TopologicalNode and ConnectivityNode in the UML class diagram. However, Figure 12 shows that only one direction has been defined in the profile. Each ConnectivityNode has a direct reference to its corresponding TopologicalNode. In order to navigate the reverse direction from TopologicalNode to ConnectivityNode, some type of conditional query would be required. In other words, the object diagrams in this section indicate which associations can actually be used in GridAPPS-D.
5. In some cases, the multiplicities on the object diagrams are more restrictive than on the class diagrams, due to profiling. For example, Figure 12 reflects a one-to-one correspondence between ConnectivityNode and TopologicalNode in this profile.

The object diagrams are intended to help you break down the CIM queries into common sub-tasks. For example, query #1 works with capacitors. It's always possible to select a capacitor (aka LinearShuntCompensator) by name. In order to find the capacitor at a bus, say "bus1" in Figure 12, one would retrieve all Terminals having a ConnectivityNode reference to "bus1". Each of those Terminals will have a ConductingEquipment reference, and you want the Terminal(s) for which that reference is actually a LinearShuntCompensator. In this CIM profile, only leaf classes (e.g. LinearShuntCompensator) will be instantiated, never base classes like ConductingEquipment. There can be more than one capacitor at a bus, more than one load, more than one line, etc.

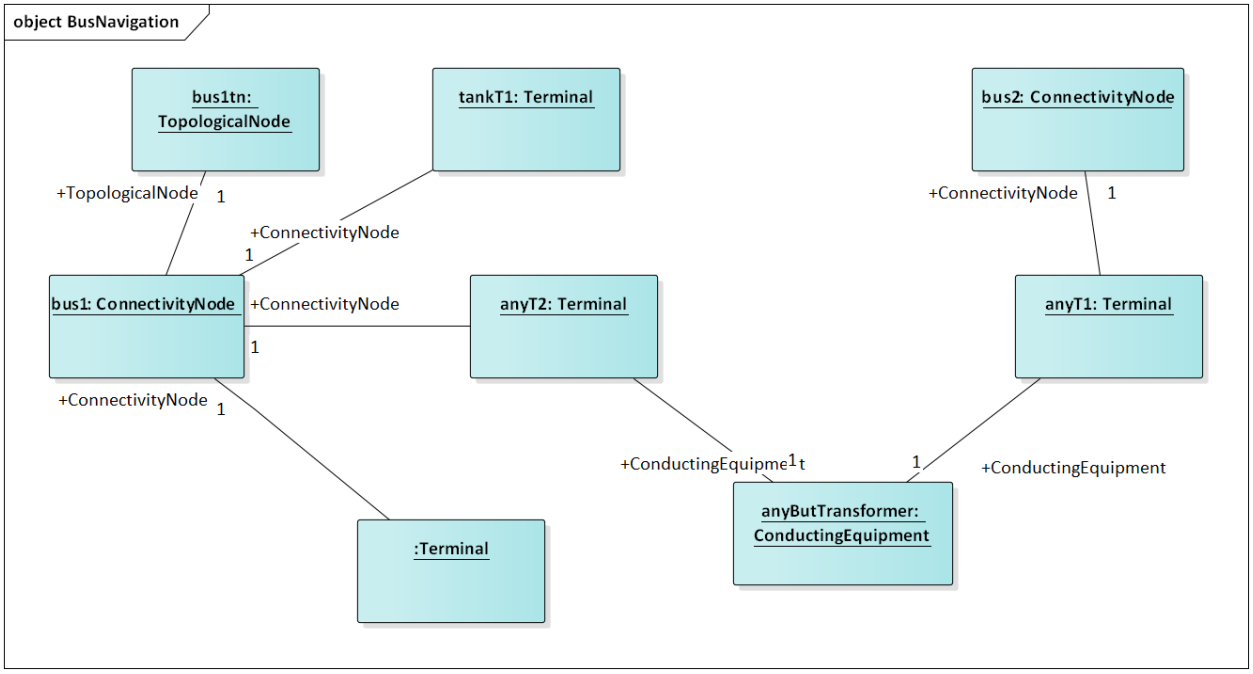


Figure 12: In order to traverse buses and components, begin with a ConnectivityNode (left). Collect all terminals referencing that ConnectivityNode; each Terminal will have one-to-one association with ConductingEquipment, of which there are many subclasses. In this example, the ConductingEquipment has a second terminal referencing the ConnectivityNode called bus2. There are applications for both Depth-First Search (DFS) and Bread-First Search (BFS) traversals. Note 1: the Terminals have names, but these are not useful. Some Terminal names have been shown above, just to illustrate there is no useful implication of sequencing or ordering. Note 2: in this version of GridAPPS-D, we have one-to-one association of TopologicalNode and ConnectivityNode, but all searches should visit ConnectivityNodes. Note 3: transformers are subclasses of ConductingEquipment, but we traverse connectivity via transformer ends (aka windings). This is illustrated later.

In order to find capacitors (or anything else) associated with a particular "feeder", Figure 13 shows that you would query for objects having EquipmentContainer reference to the feeder's Line object. In GridAPPS-D RC1, we only use Line for equipment container in CIM, and this would correspond to one entire GridLAB-D model. There is also a BaseVoltage reference that will have the system nominal voltage for the capacitor's location. However, in order to work with equipment ratings you should use ratedS and ratedU attributes where they exist, particularly for capacitors and transformers. These attributes are often slightly different than the "system voltage". Most of the attribute units in CIM are SI, with a few exceptions like percent and kW values on transformer test sheets (i.e. CIM represents the test sheet, not the equipment).

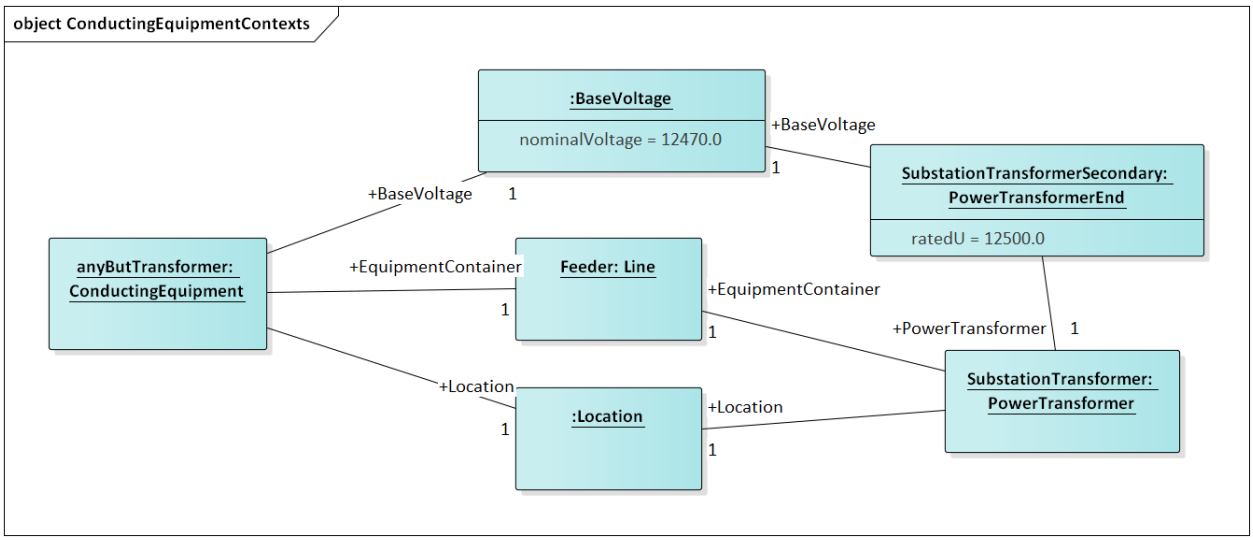


Figure 13: All conducting equipment lies within an EquipmentContainer, which in GridAPPS-D, will be a Line object named after the feeder. It also has reference to a BaseVoltage, which is typically one of the ANSI preferred system voltages. Power transformers are a little different, in that each winding (called "end" in CIM) has reference to a BaseVoltage. Note that equipment ratings come from the vendor, and in this case ratedU is slightly different from nominalVoltage. All conducting equipment has a Location, which contains XY coordinates (see Figure 1). The Location is useful for visualization, but is not essential for a power flow model.

Completing the discussion of capacitors, Figure 14 provides two examples for single-phase, and three-phase with local voltage control. As shunt elements, capacitors have only one Terminal instance. Loads and sources have one terminal, lines and switches have two terminals, and transformers have two or more terminals. Examples of all those are shown later. In Figure 14, the capacitor's kVAR rating will be based on its nameplate ratedU, not the system's nominalVoltage.

Often, the question will arise "what phases exist at this bus?". There is no phasing explicitly associated with a ConnectivityNode or Terminal in CIM. To answer this question, we'd have to query for all ConductingEquipment instances having Terminals connected to that bus, as in Figure 12. The types of ConductingEquipment that may have individual phases include LinearShuntCompensators (Figure 14), ACLineSegments, PowerTransformers (via TransformerEnds), EnergyConsumers, and descendants of Switch. If the ConductingEquipment has such individual phases, then add those phases to list of phases existing at the bus. If there are no individual phases, then ABC all exist at the bus. Note this doesn't guarantee that all wiring to the bus is correct; for example, you could still have a three-phase load served by only a two-phase line, which would be a modeling error. In Figure 14, we'd find phase C at Bus611 and phases ABC at Bus675. Elsewhere in the model, there should be ACLineSegments, PowerTransformers or Switch descendants delivering phase C to Bus611, all three phases ABC to Bus675.

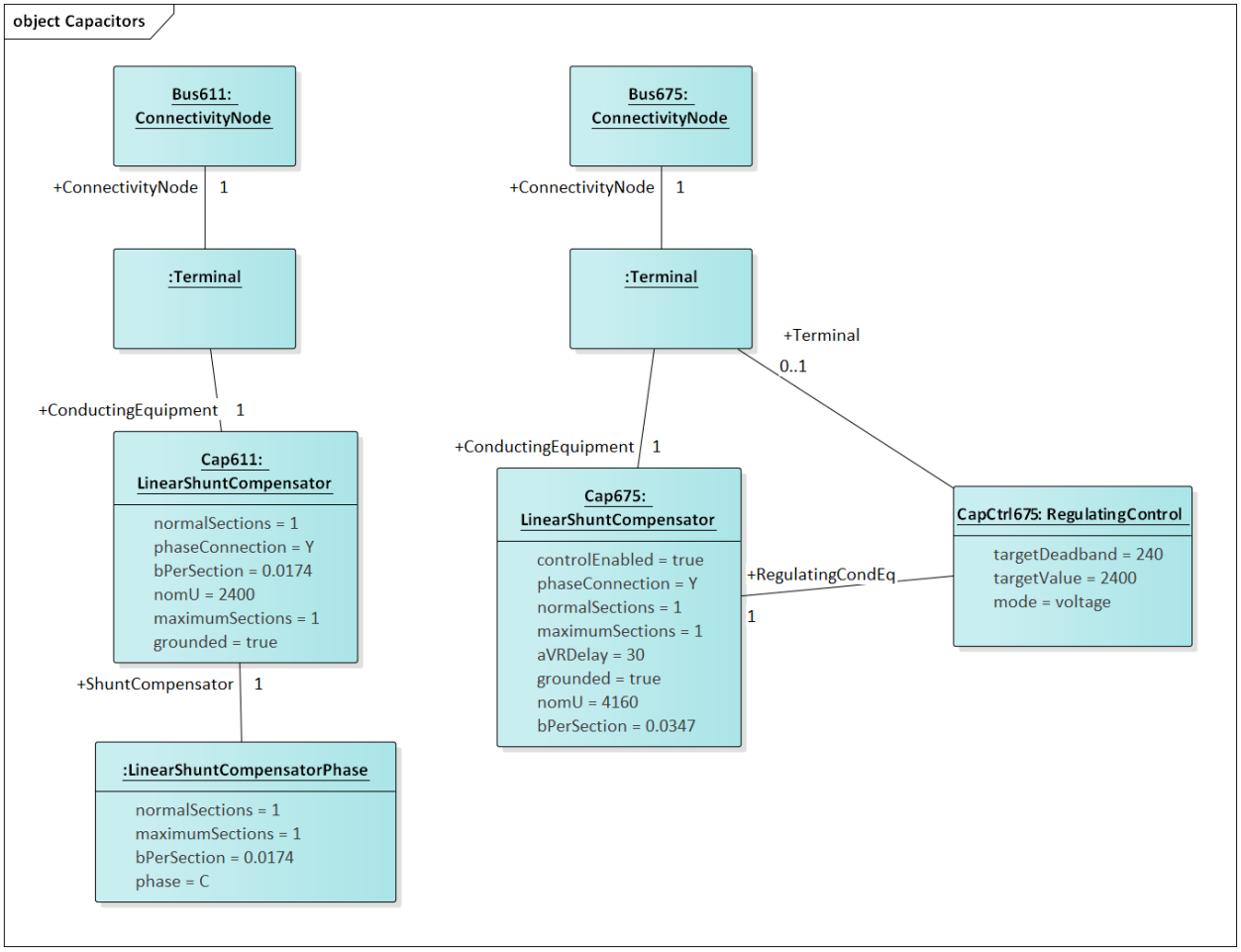


Figure 14: Capacitors are called LinearShuntCompensator in CIM. On the left, a 100 kVAR, 2400 V single-phase bank is shown on phase C at bus 611. bPerSection = 100e3 / 2400^2 [S], and the bPerSection on LinearShuntCompensatorPhase predominates; these values can differ among phases if there is more than one phase present. On the right, a balanced three-phase capacitor is shown at bus 675, rated 300 kVAR and 4160 V line-to-line. We know it's balanced three phase from the absence of associated LinearShuntCompensatorPhase objects. bPerSection = 300e4 / 4160^2 [S]. This three-phase bank has a voltage controller attached with 2400 V setpoint and 240 V deadband, meaning the capacitor switches ON if the voltage drops below 2280 V and OFF if the voltage rises above 2520 V. These voltages have to be monitored line-to-neutral in CIM, with no VT ratio. In this case, the control monitors the same Terminal that the capacitor is connected to, but a different conducting equipment's Terminal could be used. The control delay is called aVRDelay in CIM, and it's an attribute of the LinearShuntCompensator instead of the RegulatingControl. It corresponds to "dwell time" in GridLAB-D.

Figure 15 through Figure 20 illustrate the transformer query tasks, plus Figure 29 for attached voltage regulators. The autotransformer example is rated 500/345/13.8 kV and 500/500/50 MVA, for a transmission system. The short circuit test values are Z_{HL}=10%, Z_{HT}=25% and Z_{LT}=30%. The no-load test values are 0.05% exciting current and 0.025% no-load losses. These convert to r, x, g and b in SI units, from and , where S_{rated} and U_{rated} are based on the "from" winding (aka end). The same base quantities would be used to convert r, x, g and b back to per-unit or percent. The open wye – open delta impedances are already represented in percent or kW, from the test reports.

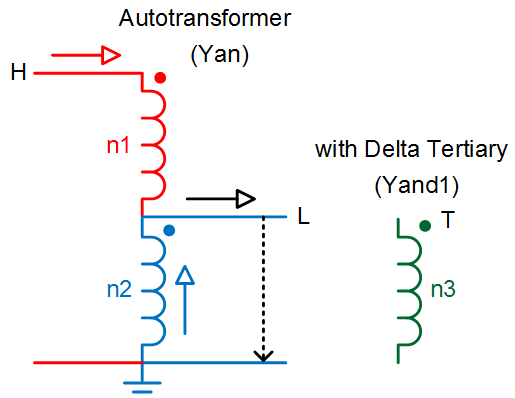


Figure 15: Autotransformer with delta tertiary winding acts like a wye-wye transformer with smaller delta tertiary. The vector group would be Yynd1 or Yyd1. For analyses other than power flow, it can be represented more accurately as the physical series (n1) – common (n2) connection, with a vector group Yand1. In either case, it's a three-winding transformer.

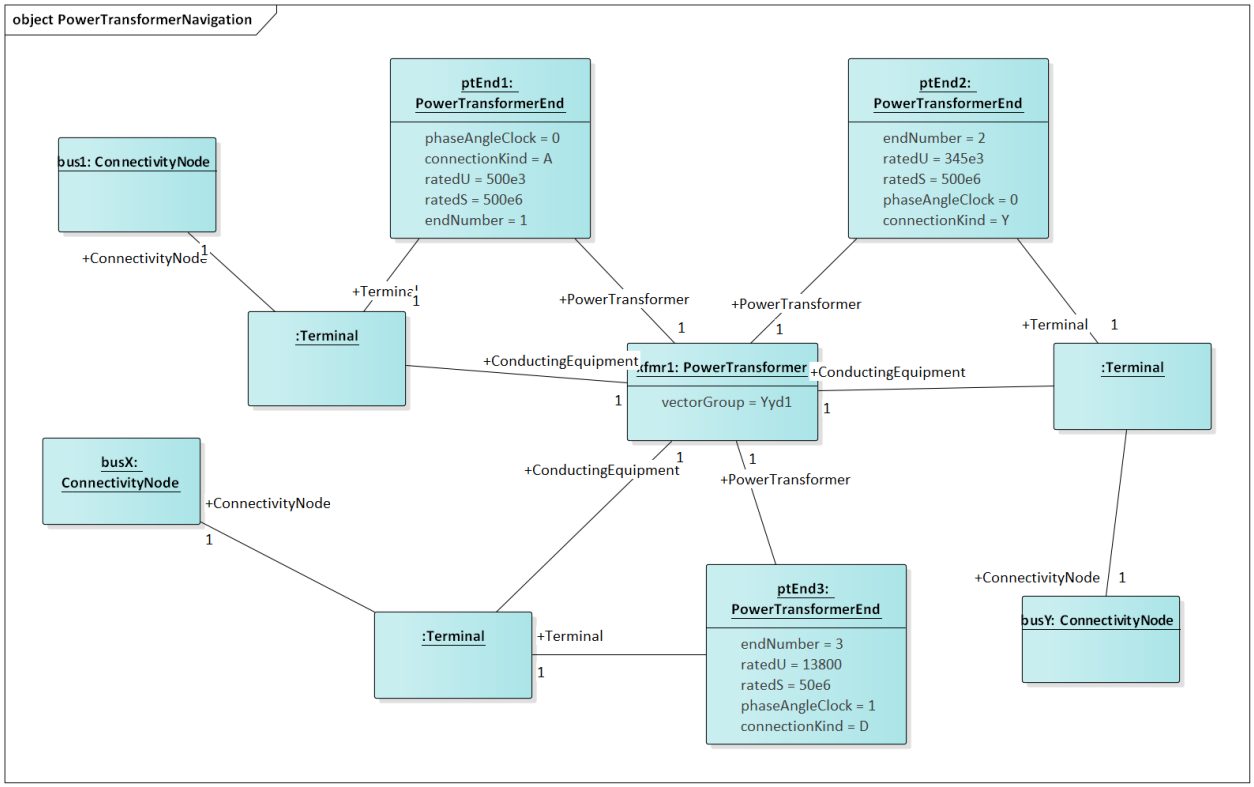


Figure 16: A three-winding autotransformer is represented in CIM as a PowerTransformer with three PowerTransformerEnds, because it's balanced and three-phase. The three Terminals have direct ConductingEquipment references to the PowerTransformer, so you can find it from bus1, busX or busY. However, each PowerTransformerEnd has a back-reference to the same Terminal, and it's own reference to BaseVoltage (Figure 13); that's how you link the matching buses and windings, which must have compatible voltages. Terminals have no sequence number, so the endNumber is important for correct linkage to catalog data as discussed later. By convention, ends with highest ratedU have the lowest endNumber, and endNumber establishes that end's place in the vectorGroup.

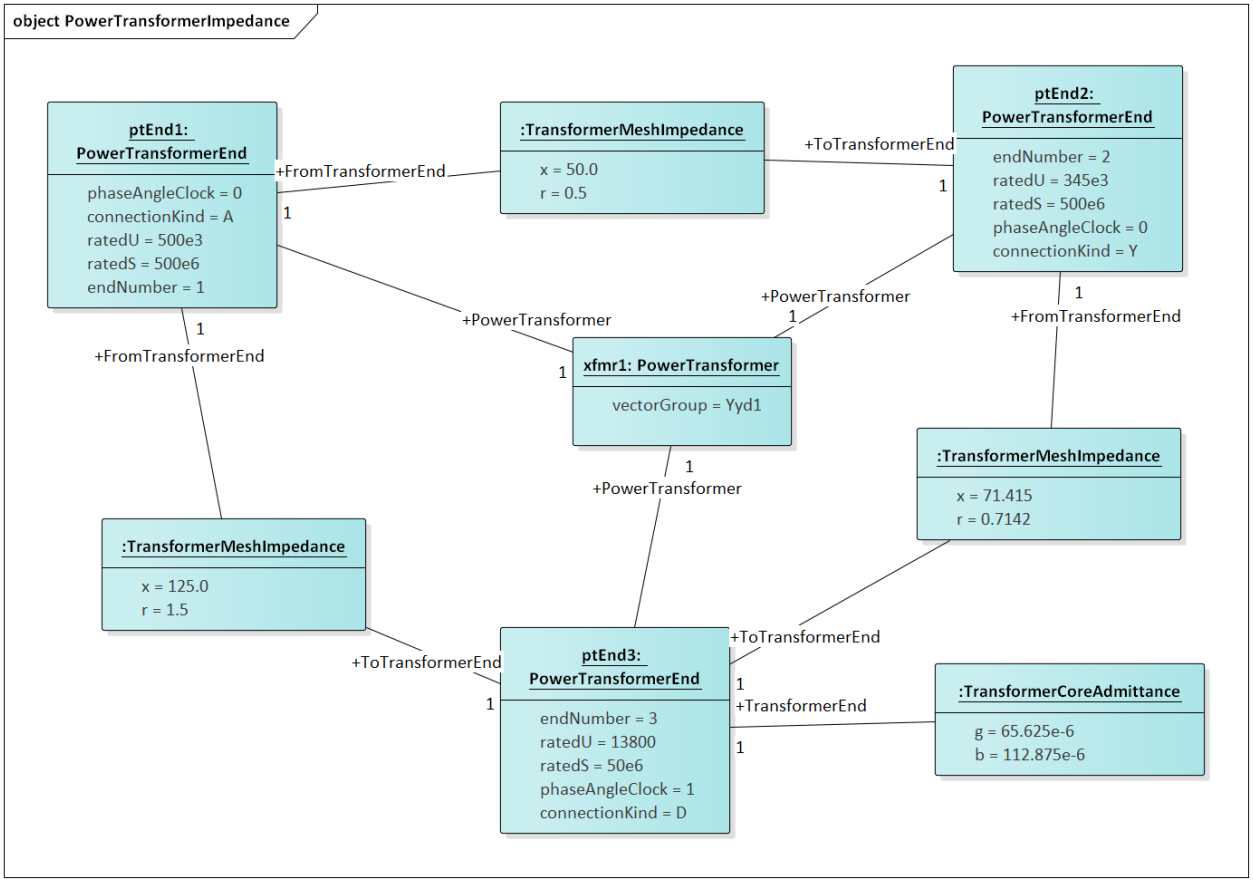


Figure 17: Power transformer impedances correspond to the three-winding autotransformer example of Figure 15 and Figure 16. There are three instances of TransformerMeshImpedance connected pair-wise between the three windings / ends. The x and r values are in Ohms referred to the end with highest ratedU in that pair. There is just one TransformerCoreAdmittance, usually attached to the end with lowest ratedU, and the attribute values are Siemens referred to that end's ratedU.

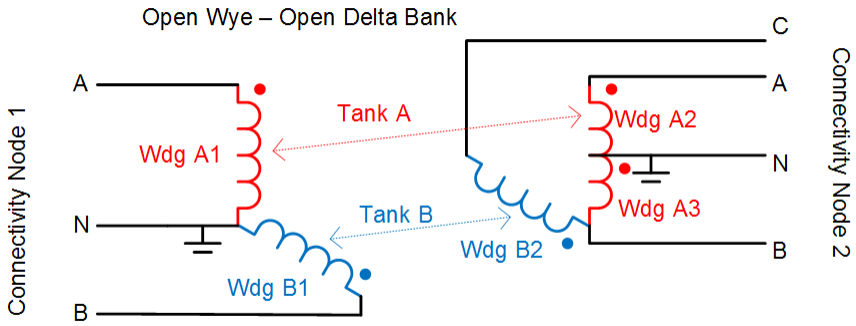


Figure 18: Open wye - open delta transformer banks are used to provide inexpensive three-phase service to loads, by using only two single-phase transformers. This is an unbalanced transformer, and as such it requires tank modeling in CIM. Physically, the two transformers would be in separate tanks. Note that Tank A is similar to the residential center-tapped secondary transformer, except the CIM phases would include s1 and s2 instead of A and B.

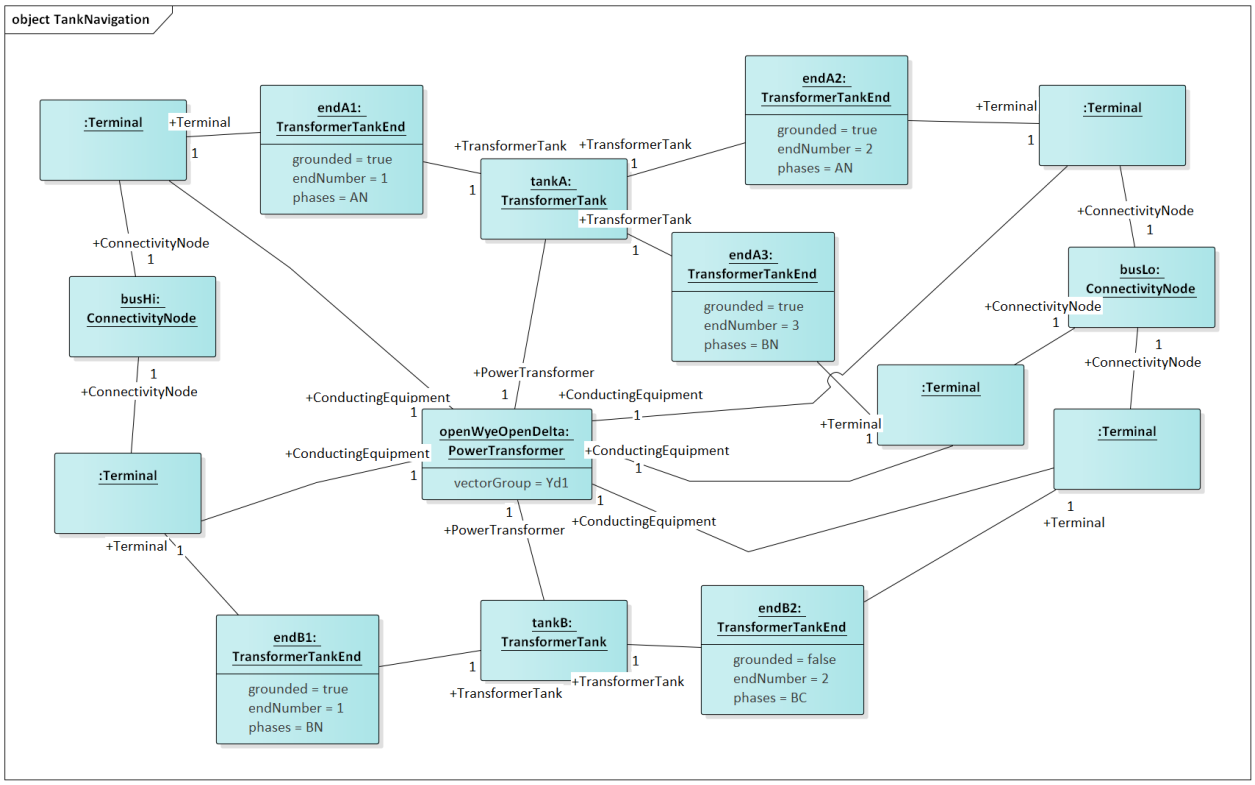


Figure 19: Unbalanced PowerTransformer instances comprise one or more TransformerTanks, which own the TransformerTankEnds. Through the ends, busHi collects phases ABN and busLo collects phases ABCN. Typically, phase C will also exist at busHi, but this transformer doesn't require it. We still assign vectorGroup Yd1 to the supervising PowerTransformer, as this is the typical case. The modeler should determine that. By comparison to Figure 19, there is a possible ambiguity in how endA3 represents the polarity dot at the neutral end of Wdg A3. An earlier CIM proposal would have assigned phaseAngleClock = 6 on endA3, but the attribute was removed from TransformerTankEnd. It may not be possible to infer the correct winding polarities from the vectorGroup in all cases. There is a phaseAngleClock attribute on TransformerTankEndInfo, but that represents a shelf state of the tank, not necessarily connections in the field. Therefore, it may be necessary to propose the phaseAngleClock attribute for TransformerTankEnd.

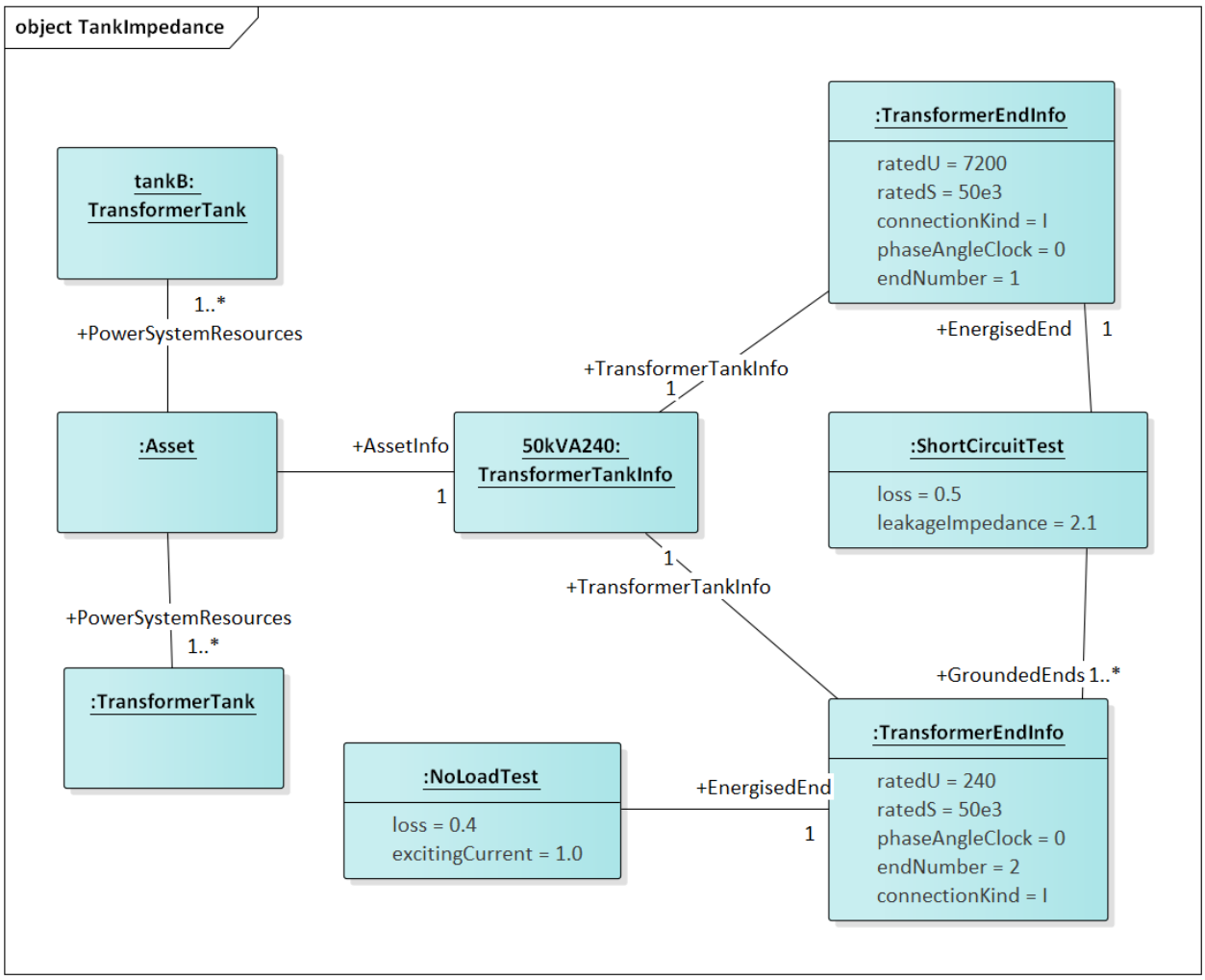


Figure 20: This Asset catalog example defines the impedances for Tank B of the open wye – open delta bank. This is a 50 kVA, 7200 / 240 V single-phase transformer. It has 1% exciting current and 0.4 kW loss in the no-load test, plus 2.1% reactance and 0.5 kW loss in the short-circuit test. A multi-winding transformer could have more than one grounded end in a short-circuit test, but this is not common. The catalog data is linked with one or more TransformerTanks via the Asset instance, shown to the left. This Asset instance won't exist without such links (i.e. the catalog data is actually used), so cardinalities are 1 for AssetInfo and 1..* for PowerSystemResources. Furthermore, endNumber on the TransformerEndInfo has to match endNumber on the TransformerTankEnd instances associated to Tank B. Instead of catalog information, we could have used mesh impedance and core admittance as in Figure 17, but we'd have to convert the test sheets to SI units and we could not share data with other TransformerTank instances, both of which are inconvenient.

Figure 21 through Figure 27 illustrate the query tasks for ACLineSegments and Switches, which will define most of the circuit's connectivity. The example sequence impedances were based on $Z_1 = 0.1 + j0.8$ Ω/mile and $Z_0 = 0.5 + j2.0$ Ω /mile. For distribution systems, use of the shared catalog data is more common, either pre-calculated matrix (Figure 25) or spacing and conductor (Figure 26 and Figure 27). In both cases, impedance calculation is outside the scope of CIM (e.g. GridLAB-D internally calculates line impedance from spacing and conductor data).

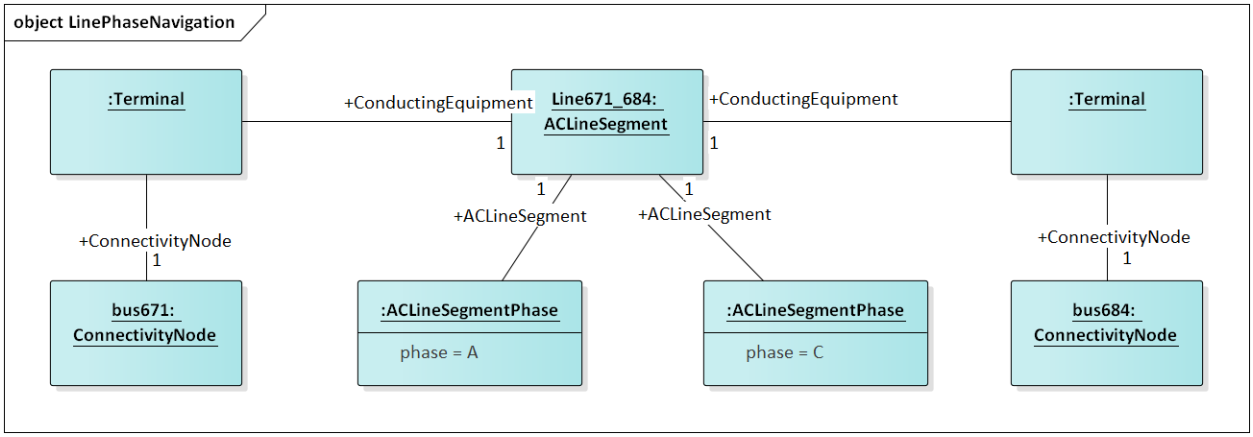


Figure 21: An ACLineSegment with two phases, A and C. If there are no ACLineSegmentPhase instances that associate to it, assume it's a three-phase ACLineSegment. This adds phases AC to bus671 and bus684.

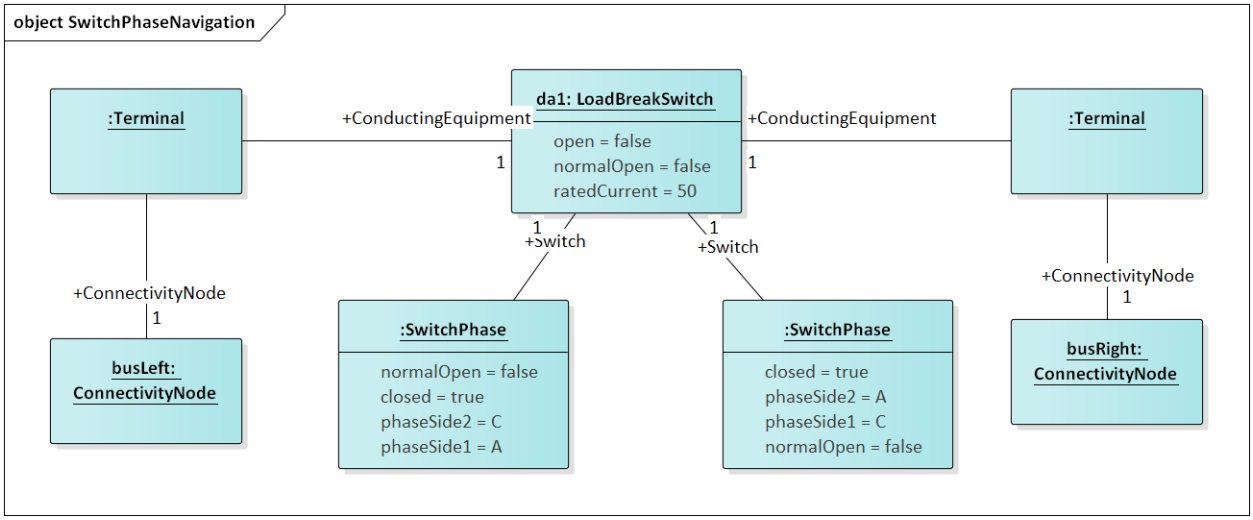


Figure 22: This 50-Amp load break switch connects phases AC between busLeft and busRight. Without associated SwitchPhase instances, it would be a three-phase switch. This switch also transposes the phases; A on side 1 connects with C on side 2, while C on side 1 connects with A on side 2. This is the only way of transposing phases in CIM. Note the ambiguity in side 1 and side 2, because Terminal.sequenceNumber was subsequently removed from the CIM. This needs to be addressed in a future version of the CIM. Also note that LoadBreakSwitch has the open attribute inherited from Switch, while SwitchPhase has the converse closed attribute. In order to open and close the switch, these attributes would be toggled appropriately. See Figure 4 for other types of switch.

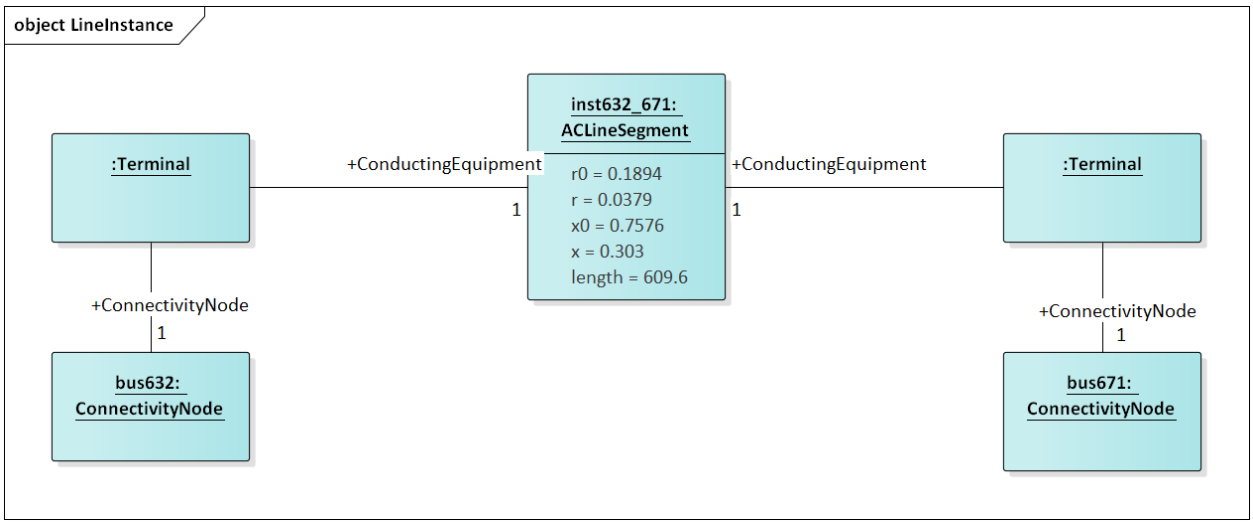


Figure 23: This is a balanced three-phase ACLineSegment between bus632 and bus671, 2000 feet or 609.6 m long. Sequence impedances are specified in ohms, as attributes on the ACLineSegment. This is a typical pattern for transmission lines, but not distribution lines.

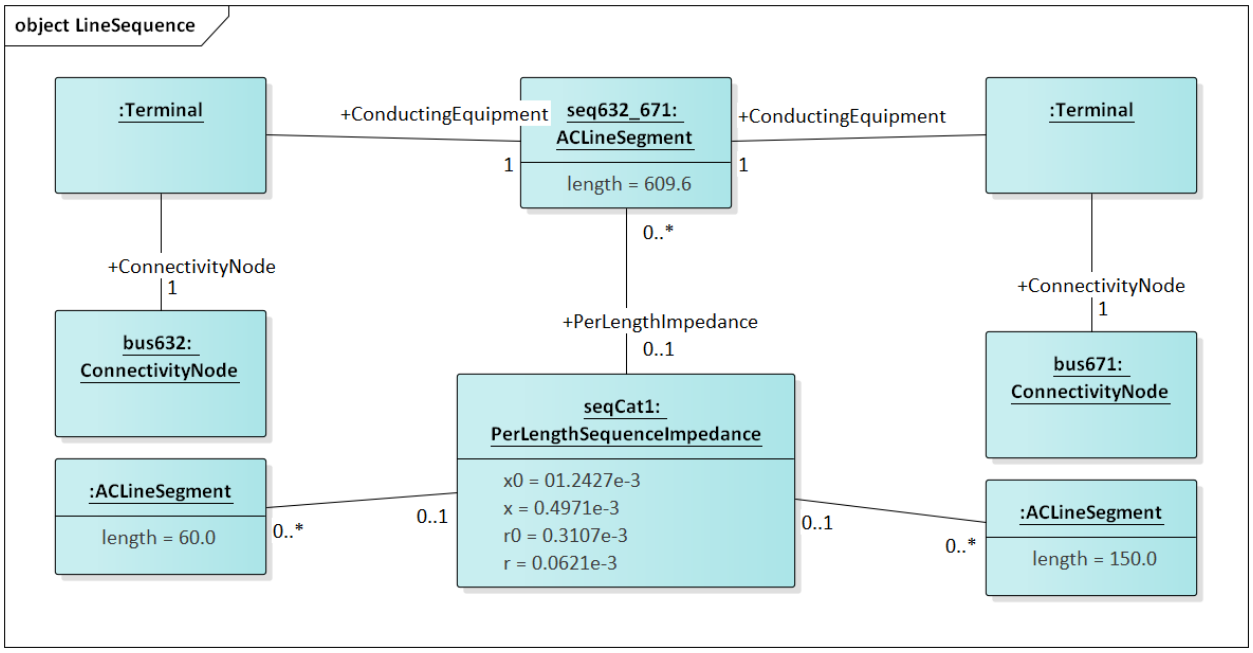


Figure 24: The impedances from Figure 23 were divided by 609.6 m, to obtain ohms per meter for seqCat1. Utilities often call this a "line code", and other ACLineSegment instances can share the same PerLengthImpedance. A model imported into the CIM could have many line codes, not all of them used in that particular model. However, those line codes should be available for updates by reassigning PerLengthImpedance.

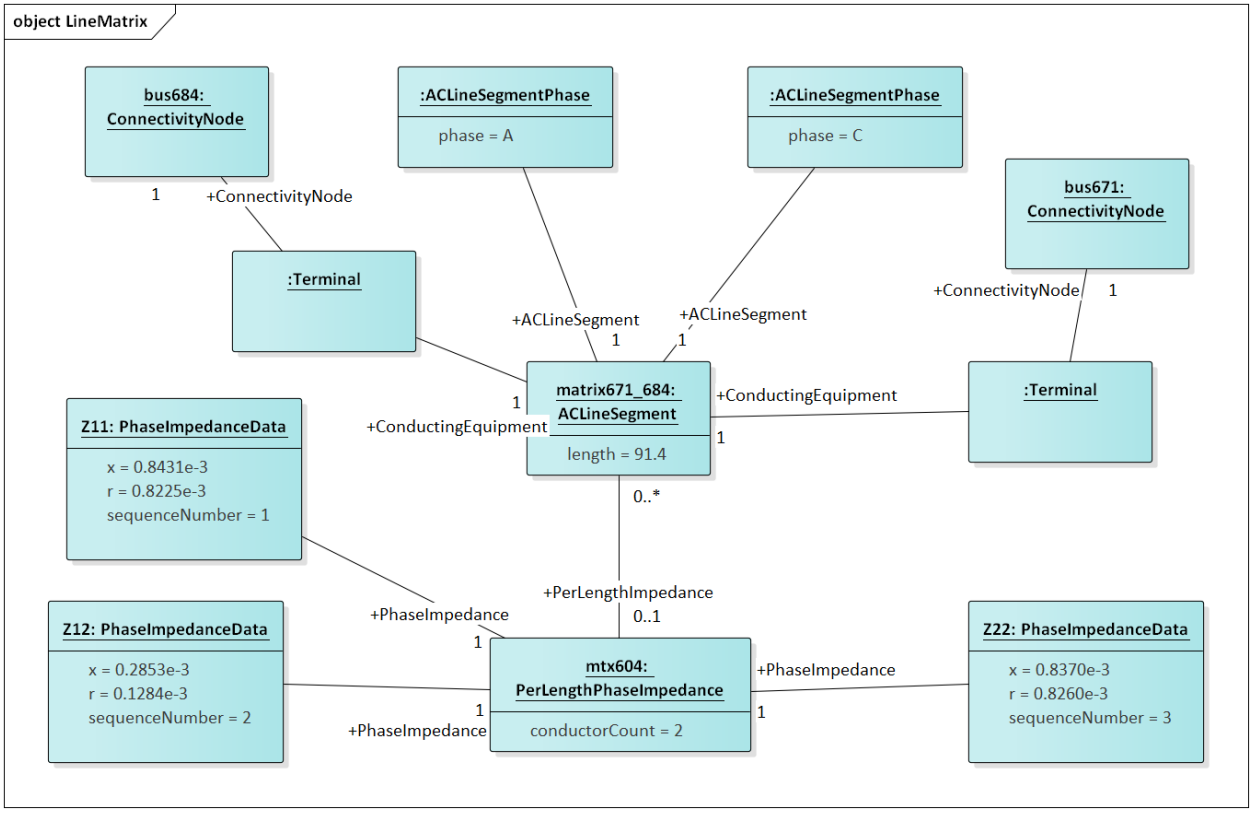


Figure 25: This is a two-phase line segment from bus671 to bus684 using a line code, which has been specified using a 2x2 symmetric matrix of phase impedances per meter, instead of sequence impedances per meter. This is more common for distribution than either Figure 23 or Figure 24. It's distinguished from Figure 24 by the fact that PerLengthImpedance references an instance of PerLengthPhaseImpedance, not PerLengthSequenceImpedance. The conductorCount attribute tells us it's a 2x2 matrix, which will have two unique diagonal elements and one distinct off-diagonal element. The elements are provided in three PhaseImpedanceData instances, which are named here for clarity as Z11, Z12 and Z22. However, the sequenceNumber is most significant, as the elements must be numbered in lower triangular form. Finally, note that Z11 and Z22 are slightly different. The matrix row numbers must correspond to the phases present in ABC order. CIM doesn't provide a way of transposing matrix row assignments, so in order to swap phases A and C, we'd have to create a second instance of PerLengthPhaseImpedance, with Z11 and Z22 swapped. The GridAPPS-D CIM importer will create these automatically, which expands the set of line codes. As presented here, mtx604 can apply to phasing AB, BC or AC.

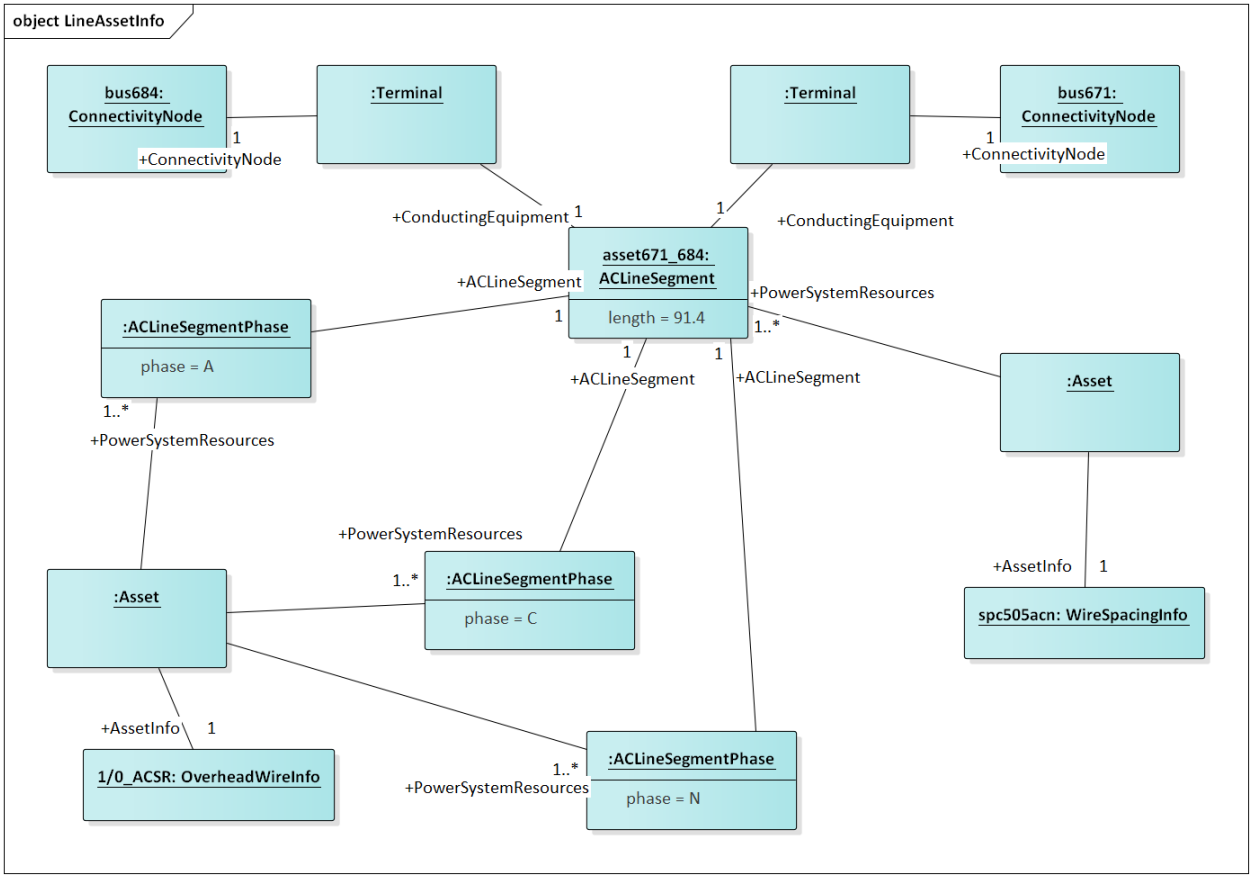


Figure 26: The two-phase ACLineSegment impedance defined by sharing wire and spacing data from a catalog. Each ACLineSegmentPhase links to an OverheadWireInfo instance via the Asset instance. If the neutral (N) is present, we have to specify its wire information for a correct impedance calculation. In this case, ACN all use the same wire type, but they can be different, especially for the neutral. Similarly, the WireSpacingInfo associates to the ACLineSegment itself via a separate Asset instance. These Asset instances only exist when the catalog data is used, so cardinalities are 1 for AssetInfo and 1..* for PowerSystemResources.

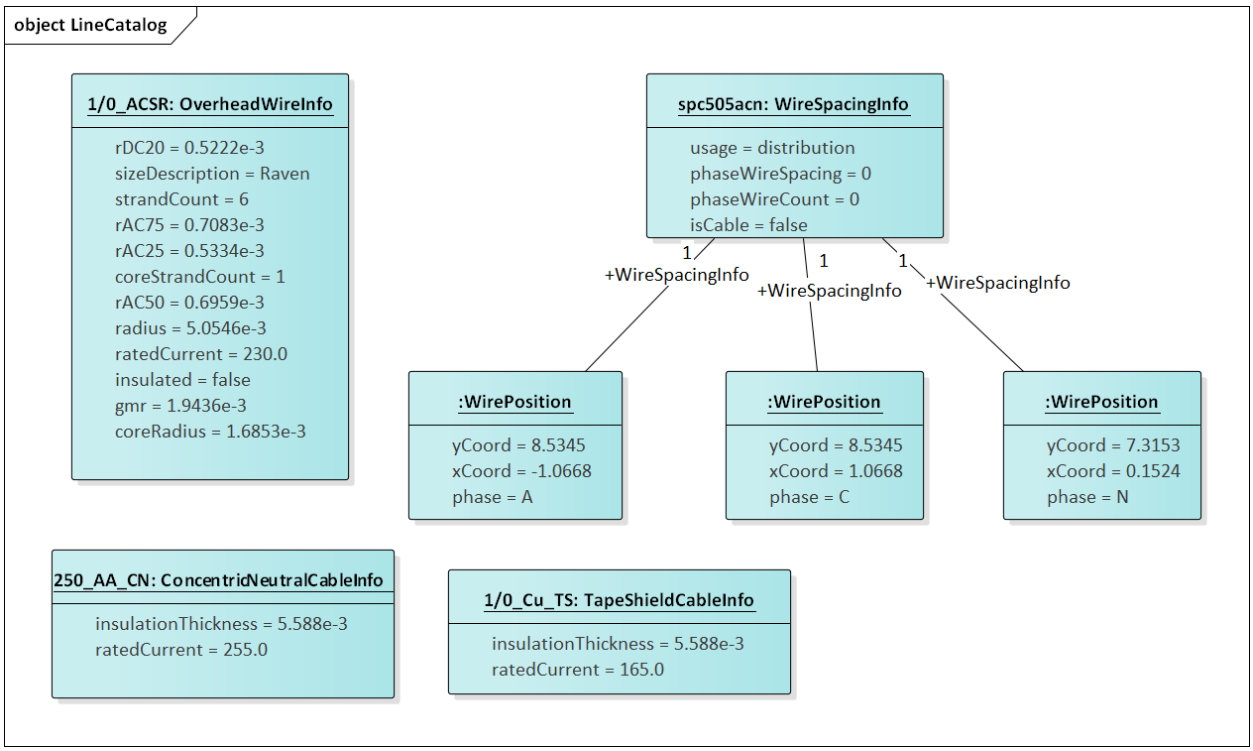


Figure 27: The upper five instances define catalog attributes for Figure 26. The WirePosition xCoord and yCoord units are meters, not feet, and they include explicit phase assignments to match ACLineSegmentPhase. This removes any ambiguity, but it's still necessary to create copies for phase transposition. The phaseWireSpacing and phaseWireCount attributes are for sub-conductor bundling on EHV and UHV transmission lines; bundling is not used on distribution. The number of WirePositions that reference spc505acn determine how many wires need to be assigned, and the phase attributes in those WirePosition instances determine how many phases and neutrals there are. Eliminating the neutral, this would produce a 2x2 phase impedance matrix. Although the pattern appears general enough to support multiple neutrals and transmission overbuild, the CIM doesn't actually have the required phasing codes. When isCable is true, the WirePosition yCoord values would be negative for underground depth. To find overhead wires of a certain size or ampacity, we can put query conditions on the ratedCurrent attribute. To find underground conductors, we query the ConcentricNeutralCableInfo or TapeShieldCableInfo instead of OverheadWireInfo. All three inherit the ratedCurrent attribute from WireInfo. Cables don't have a voltage rating in CIM, but you can use insulationThickness as a proxy for voltage rating in queries. Here, 5.588 mm corresponds to 220 mils, which is a common size for distribution.

Figure 28 illustrates the loads, which are called EnergyConsumer in CIM. The houses and appliances from GridLAB-D are not supported in CIM. Only ZIP loads can be represented. Further, any load schedules would have to be defined outside of CIM. Assume that the CIM loads are peak values.

Figure 29 illustrates the voltage regulator function. Note that GridLAB-D combines the regulator and transformer functions, while CIM separates them. Also, the CIM provides voltage and current transducer ratios for tap changer controls, but not for capacitor controls.

Figure 30 through Figure 32 illustrate how measurements required for RC1 can be attached to buses or other components. Individual phase measurements for voltage and capacitor status have to be added.

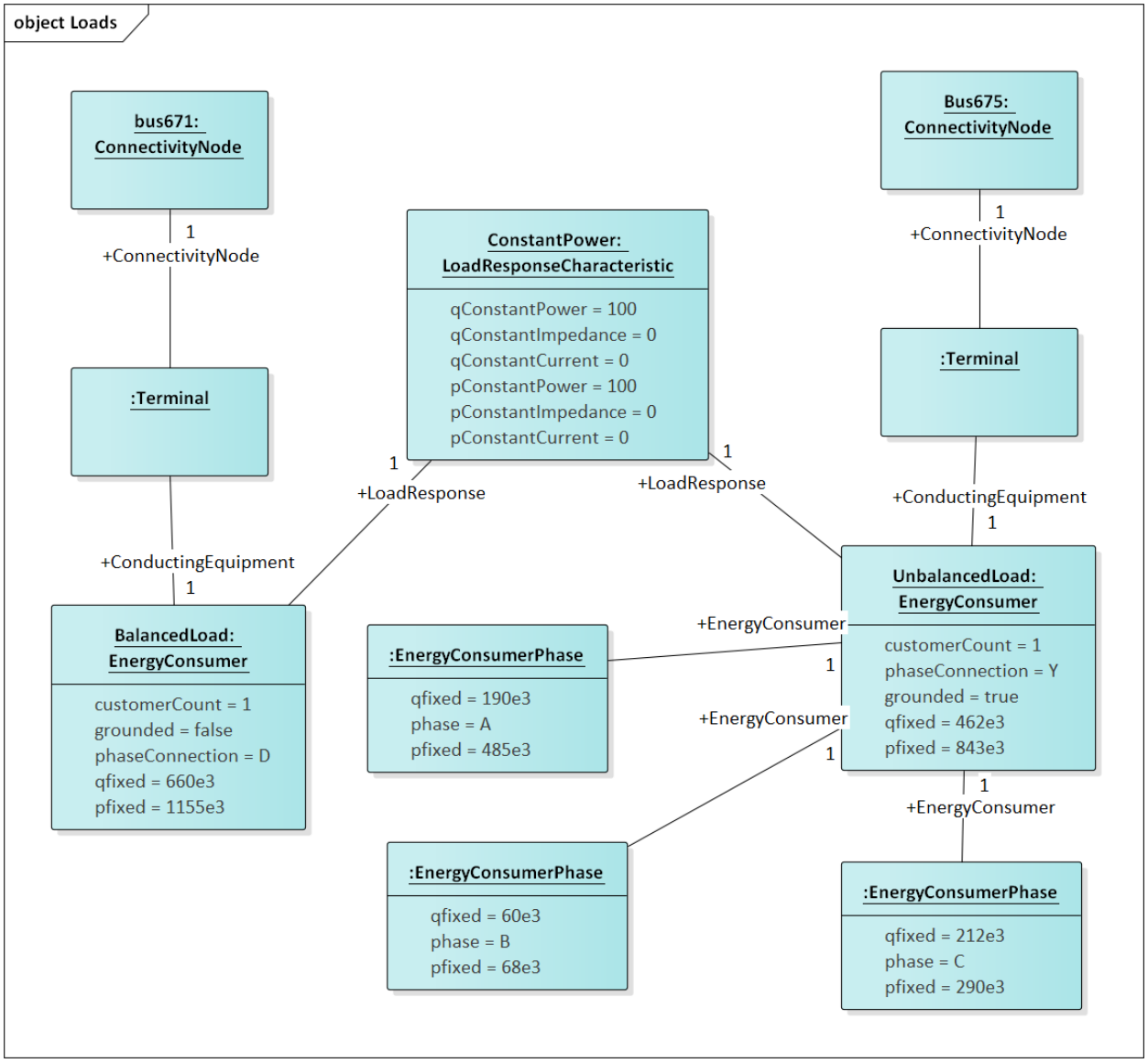


Figure 28: The three-phase load (aka EnergyConsumer) on bus671 is balanced and connected in delta. It has no ratedU attribute, so use the referenced BaseVoltage (Figure 13) if a voltage level is required. On the right, a three-phase wye-connected unbalanced load on bus675 is indicated by the presence of three EnergyConsumerPhase instances referencing UnbalancedLoad. For consistency in searches and visualization, UnbalancedLoad.pfixed should be the sum of the three phase values, and likewise for UnbalancedLoad.qfixed. In power flow solutions, the individual phase values would be used. Both loads share the same LoadResponse instance, which defines a constant power characteristic for both P and Q, because the percentages for constant impedance and constant current are all zero. The two other most commonly used LoadResponseCharacteristics have 100% constant current, and 100% constant impedance. Any combination can be used, and the units don't have to be percent (i.e. use a summation to determine the denominator for normalization).

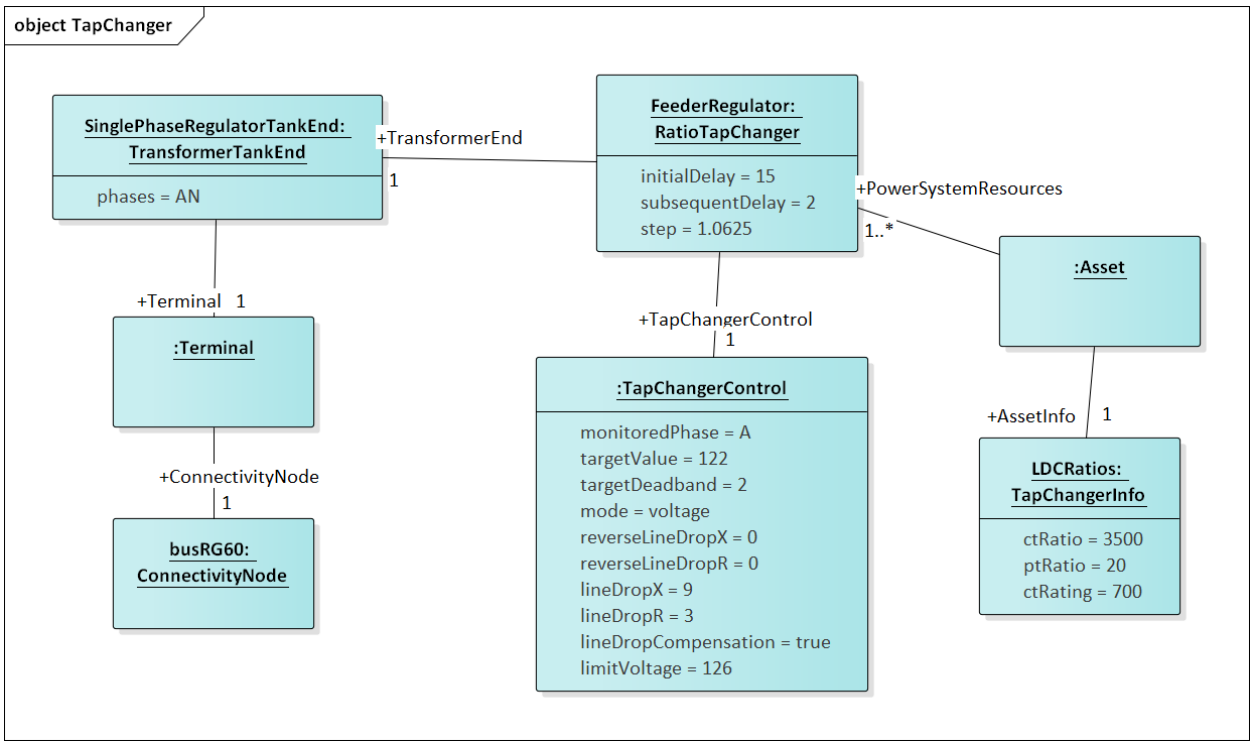


Figure 29: In CIM, the voltage regulator function is separated from the tap-changing transformer. The IEEE 13-bus system has a bank of three independent single-phase regulators at busRG60, and this example shows a RatioTapChanger attached to the regulator on phase A, represented by the TransformerTankEnd having phases=A or phases=AN. See Figure 19 for a more complete picture of TransformerTankEnds, or Figure 16 for a more complete picture of PowerTransformerEnds. Either one can be the TransformerEnd in this figure, but with a PowerTransformerEnd, all three phase taps would change in unison (i.e. they are "ganged"). Most regulator attributes of interest are found in RatioTapChanger or TapChangerControl instances. However, we need the Asset mechanism to specify ctRatio, ptRatio and ctRating values. These are inherent to the equipment, whereas the attributes of RatioTapChanger and TapChangerControl are all settings per instance. For the IEEE 13-bus example, there would be separate RatioTapChanger and TapChangerControl instances for phases B and C.

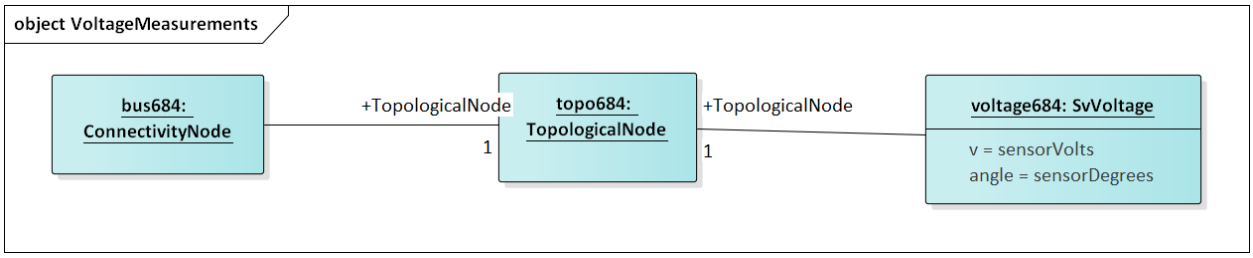


Figure 30: In CIM, the voltage measurement attaches to TopologicalNode, which we can find from the ConnectivityNode in GridAPPS-D. Positive sequence or phase A measurement is implied, so we must add a phase attribute on SvVoltage for GridAPPS-D. Physically, a voltage sensor is more closely associated with a Terminal or ConnectivityNode.

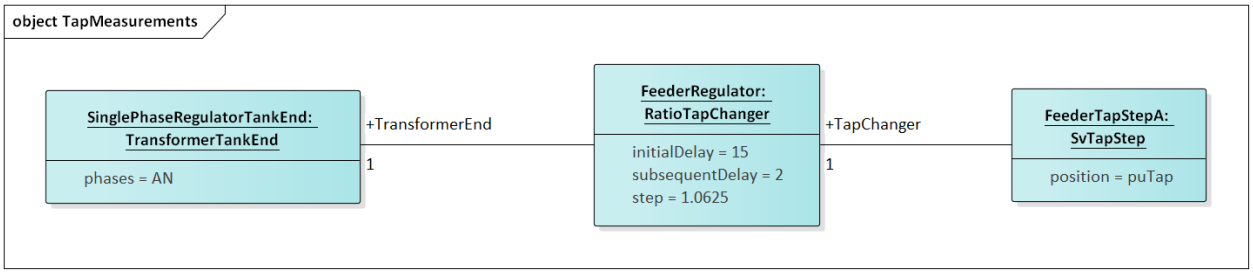


Figure 31: SvTapStep links to a TransformerEnd indirectly, through the RatioTapChanger. There is no phasing ambiguity because TransformerTankEnd has its phases attribute, while PowerTransformerEnd always includes ABC. Units for SvTapStep.position are per-unit.

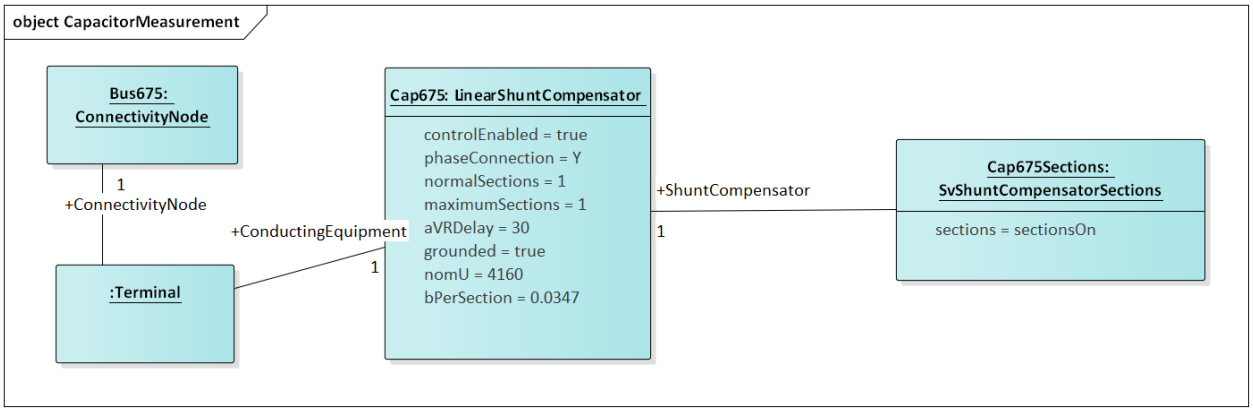


Figure 32: The on/off measurement for a capacitor bank attaches directly to LinearShuntCompensator, but there is no phasing support. That needs to be proposed as a CIM extension.

4.5.4 Metering Relationship to Loads in the CIM

These UML class relationships in Figure 33 through Figure 35 have not been planned for implementation in RC1, but in a future version of GridAPPS-D, they can be used to link automated meter readings with loads in the distribution system model.

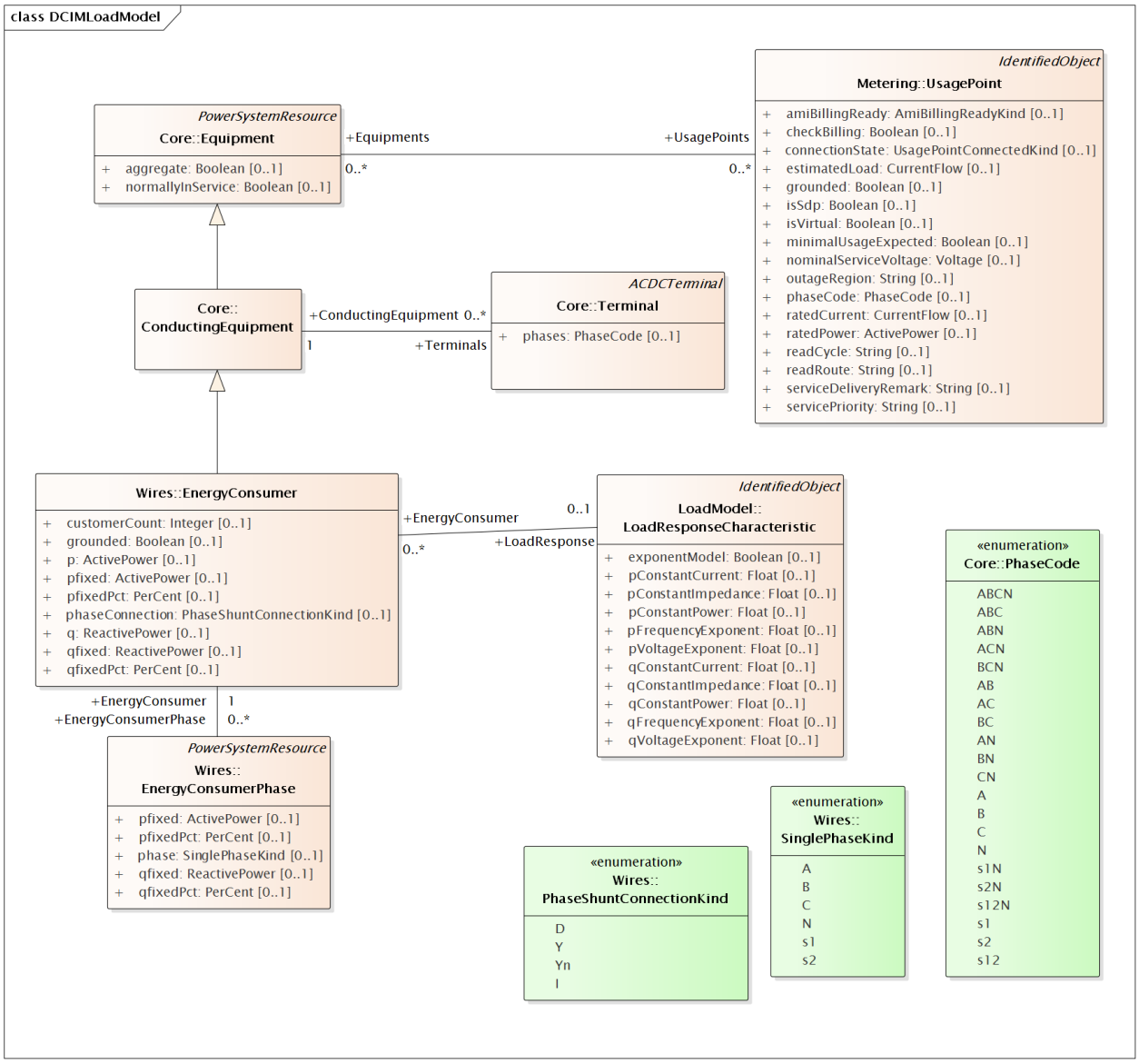


Figure 33: Energy Consumers are associated to Metering Usage Points

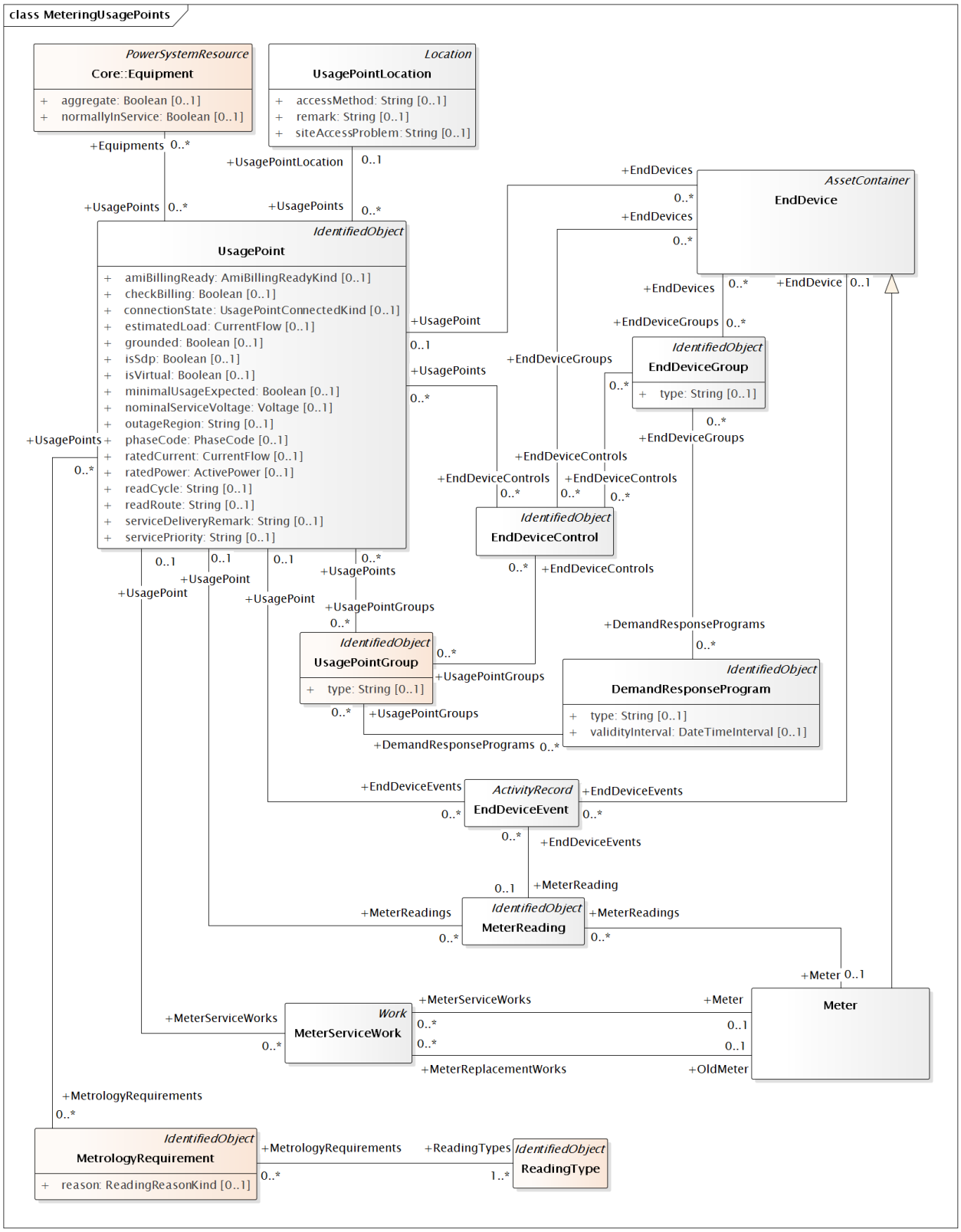


Figure 34: Metering Usage Points have one or more EndDevices (i.e. Meters)

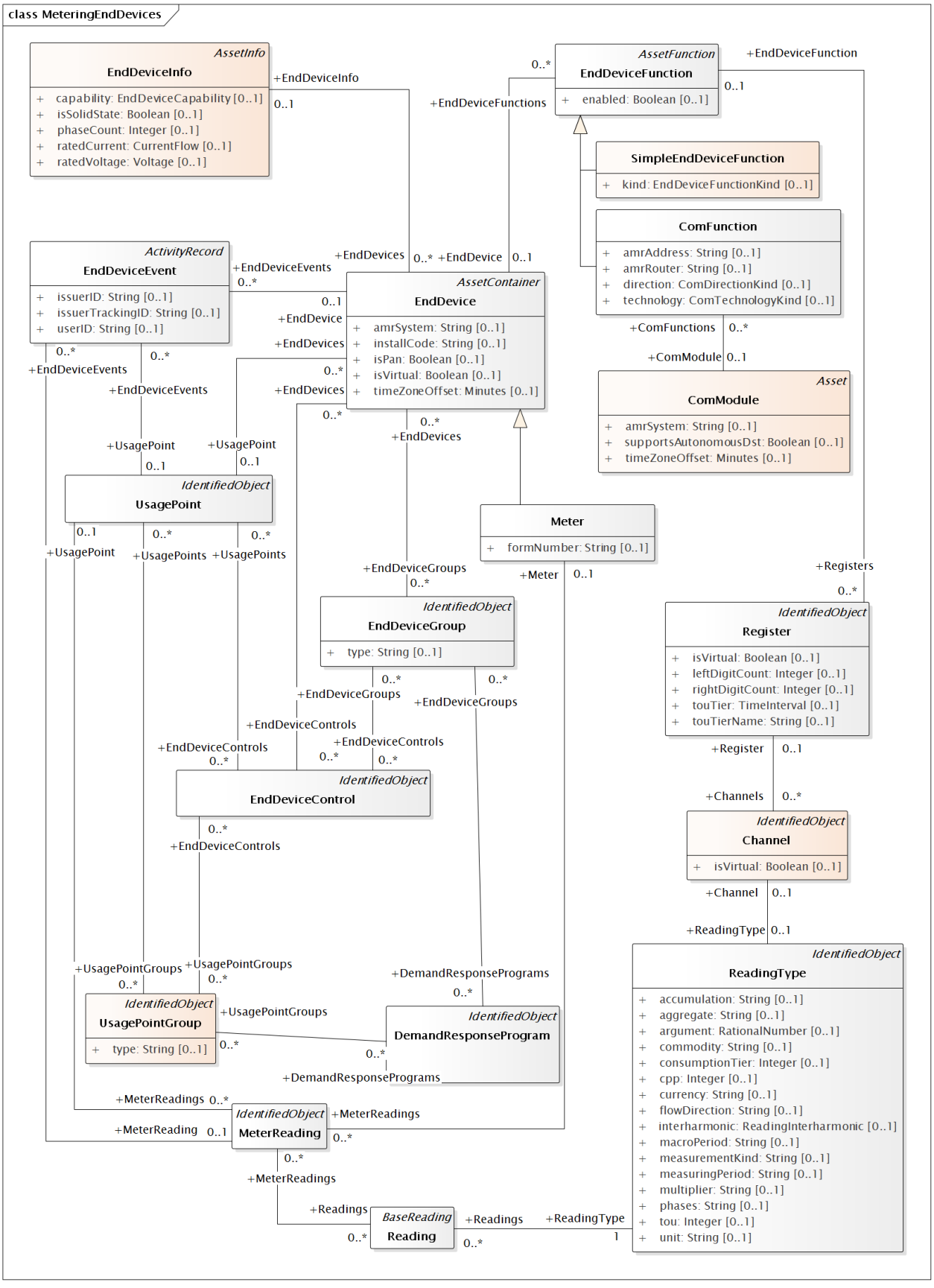

class MeteringEndDevices
AssetInfo
EndDeviceInfo
+ capability: EndDeviceCapability [0..1]
+ isSolidState: Boolean [0..1]
+ phaseCount: Integer [0..1]
+ ratedCurrent: CurrentFlow [0..1]
+ ratedVoltage: Voltage [0..1]
+EndDeviceInfo
0..1
AssetFunction
EndDeviceFunction
+ enabled: Boolean [0..1]
0..*
+EndDeviceFunctions
+EndDeviceFunction
0..1
SimpleEndDeviceFunction
+ kind: EndDeviceFunctionKind [0..1]
ComFunction
+ amrAddress: String [0..1]
+ amrRouter: String [0..1]
+ direction: ComDirectionKind [0..1]
+ technology: ComTechnologyKind [0..1]
+ComFunctions
0..*
+ComModule
0..1
Asset
ComModule
+ amrSystem: String [0..1]
+ supportsAutonomousDst: Boolean [0..1]
+ timeZoneOffset: Minutes [0..1]
ActivityRecord
EndDeviceEvent
+ issuerID: String [0..1]
+ issuerTrackingID: String [0..1]
+ userID: String [0..1]
+EndDeviceEvents
0..*
+EndDevices
0..*
+EndDevice
0..1
AssetContainer
EndDevice
+ amrSystem: String [0..1]
+ installCode: String [0..1]
+ isPan: Boolean [0..1]
+ isVirtual: Boolean [0..1]
+ timeZoneOffset: Minutes [0..1]
+EndDevices
+EndDevice
+UsagePoint
0..1
IdentifiedObject
UsagePoint
+UsagePoints
Meter
+ formNumber: String [0..1]
+Meter
+EndDeviceGroups
IdentifiedObject
EndDeviceGroup
+ type: String [0..1]
+Registers
IdentifiedObject
Register
+ isVirtual: Boolean [0..1]
+ leftDigitCount: Integer [0..1]
+ rightDigitCount: Integer [0..1]
+ touTier: TimeInterval [0..1]
+ touTierName: String [0..1]
+Register
+Channels
IdentifiedObject
Channel
+ isVirtual: Boolean [0..1]
+Channel
+ReadingType
IdentifiedObject
ReadingType
+ accumulation: String [0..1]
+ aggregate: String [0..1]
+ argument: RationalNumber [0..1]
+ commodity: String [0..1]
+ consumptionTier: Integer [0..1]
+ cpp: Integer [0..1]
+ currency: String [0..1]
+ flowDirection: String [0..1]
+ interharmonic: ReadingInterharmonic [0..1]
+ macroPeriod: String [0..1]
+ measurementKind: String [0..1]
+ measuringPeriod: String [0..1]
+ multiplier: String [0..1]
+ phases: String [0..1]
+ tou: Integer [0..1]
+ unit: String [0..1]
+EndDeviceControls
IdentifiedObject
EndDeviceControl
+UsagePointGroups
IdentifiedObject
UsagePointGroup
+ type: String [0..1]
IdentifiedObject
DemandResponseProgram
+DemandResponsePrograms
+MeterReadings
+MeterReading
IdentifiedObject
MeterReading
+Readings
BaseReading
Reading

Figure 35: EndDevices associate to meter readings, functions and channels.

4.5.5 CIM Enhancements for RC2

Possible CIM enhancements to support volt-var feeder modeling:

1. Different on and off delay parameters for RegulatingControl (Figure 5)
2. Phase modeling for EnergySource (Figure 3)
3. Current ratings for PerLengthImpedance (Figure 2). At present, some users rely on associated WireInfo, ignoring all attributes except currentRating.
4. Transducers for RegulatingControl (Figure 5)
5. Dielectric constant and soil resistivity (Figure 10)
6. Current flow and switch open/closed measurements (Figure 11)
7. Individual phase measurements for voltage and capacitor state (Figure 11)
8. Clock angles for TransformerTankEnd (i.e. move phaseAngleClock from PowerTransformerEnd to TransformerEnd (Figure 6)
9. Clarify side1 and side2 for switch phase modeling (Figure 4)

4.5.6 CIM Profile in CIMTool

CIMTool was used to develop and test the profile for RC1, because it:

1. Generates SQL for the MySQL database definition
2. Validates instance files against the profile

The CIMTool developer will not be able to support the tool in future, so eventually we will use the new Schema Composer feature in Enterprise Architect.

In order to view the profile, import the archived Eclipse project *OSPRREYS_CIMTOOL.zip* into CIMTool. Please see the CIM tutorial slides provided by Margaret Goodrich for user instructions.

Four instance files were validated against the profile in CIMTool. In order to generate them, we use a current version of OpenDSS with the *Export CDPSMcombined* command on four IEEE test feeders that come with OpenDSS:

1. **~/src/opendss/Test/IEEE13_CDPSM.dss** is the IEEE 13-bus test feeder with per-length phase impedance matrices and a delta tertiary added to the substation transformer.
2. **~/src/opendss/Test/IEEE13_Assets.dss** is the IEEE 13-bus test feeder with catalog data for overhead lines, cables and transformers. Capacitor controls have also been added.
3. **~/src/opendss/Distrib/IEEETestCases/8500-Node/Master.dss** is the IEEE 8500-node test feeder with balanced secondary loads.
4. **~/src/opendss/Distrib/IEEETestCases/8500-Node/Master-unbal.dss** is the IEEE 8500-node test feeder with unbalanced secondary loads.

Either the 3rd or 4th feeder will be used for the volt-var application. The 1st and 2nd feeders are used to validate more parts of the CIM profile used in RC1. In all four cases, CIMTool reports only two kinds of validation error:

1. **Isolated connectivity node**: CIMTool expects two or more Terminals per ConnectivityNode, but dead ended feeder segments will have only one on the last node. This is not really an error, at least for distribution systems.

2. **Minimum cardinality**: For TapChangerControl instances, the inherited RegulatingControl.RegulatingCondEq association is not specified. This is not really an error, as the association is only needed for shunt capacitor controls. Figure 36 shows that RegulatingCondEq was not selected for TapChangerControl in the profile, so this may reflect a defect in the validation code. Efforts to circumvent it were not successful.

With these caveats, the profile and instances validate against each other, for feeder models that solve in OpenDSS.

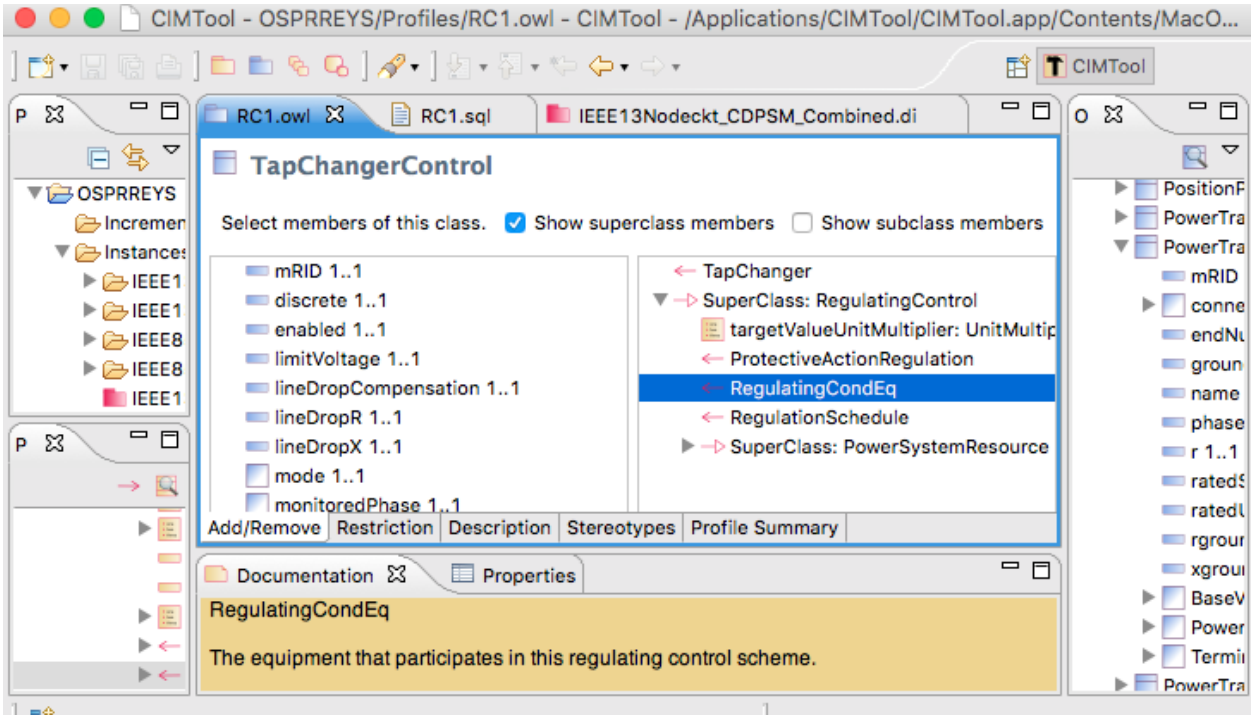


Figure 36: Profiling TapChangerControl in CIMTool; the inherited RegulatingCondEq is not included.

4.5.7 Creating Data Definition Language (DDL) for MySQL

As shown at the top of Figure 36, CIMTool builds *RC1.sql* to create tables in a relational database, but the syntax doesn't match that required for MySQL. The following manual edits were made:

1. Globally change **CHAR VARYING(30)** to **varchar(50)** with a blank space pre-pended before the varchar
2. Globally change **“** to **‘**
3. In foreign keys to enumerations, change the referenced attribute from **mRID** to **name**
4. In foreign keys to **EquipmentContainer** or **ConnectivityNodeContainer**, change the referenced table to **Line**
5. In foreign keys to **ShuntCompensator**, change the referenced table to **LinearShuntCompensator**
6. In foreign keys to **TapChanger**, change the referenced table to **RatioTapChanger**.
7. The CIM UML incorporates several polymorphic associations, which can't be implemented directly in SQL. Base parent class tables were added for:
   (a) **AssetInfo**, which can be referenced via the Parent attribute from ConcentricNeutralCableInfo, TapeShieldCableInfo, OverheadWireInfo, WireSpacingInfo, TapChangerInfo and TransformerTankInfo
   (b) **TransformerEnd**, which can be referenced via the Parent attribute from PowerTransformerEnd and TransformerTankEnd

(c) **PerLengthImpedance**, which can be referenced via the Parent attribute from PerLengthSequenceImpedance and PerLengthPhaseImpedance

(d) **Switch**, which can be referenced via the SwtParent attribute from Breaker, Fuse, Sectionaliser, Recloser, Disconnector, Jumper and LoadBreakSwitch.

(e) **ConductingEquipment**, which can be referenced via the Parent attribute from ACLineSegment, EnergySource, EnergyConsumer, LinearShuntCompensator, PowerTransformer, and all of the Switch types.

8. The catalog data mechanism in Figure 8 required two new tables, one for polymorphic associations and another for many-to-many joins:

   (a) **PowerSystemResource**, which can be referenced via the PSR attribute from ACLineSegment, ACLineSegmentPhase, RatioTapChanger and TransformerTank.

   (b) **AssetInfoJoin**, which references AssetInfo and PowerSystemResource. This table actually supplants the Asset class in Figure 8.

9. The ShortCircuitTest in Figure 9 has a one-to-many association to TransformerEndEnfo, and we need to implement the many side by adding:

   (a) **GroundedEndJoin**, which references TransformerEndInfo and ShortCircuitTest.

10. The ToTransformerEnd association in Figure 6 is one-to-many, so CIMTool did not export it to SQL. Rather than create a join table, a ToTransformerEnd attribute was added to TransformerMeshImpedance. This supports only one-to-one association, which is justified because the one-to-many case is very rare, and GridLAB-D cannot model transformers having the one-to-many association. This restriction may be removed in future versions having a semantic or graph database.

    Except for the first two items, all of these adjustments arose from the absence of inheritance or polymorphism in SQL. These adjustments will make the updates, queries and views more complicated. However, they allow referential integrity to be enforced, which is one of the most important reasons to use SQL and relational databases. Other types of data store could be a more natural fit to the CIM UML, but they may not have the performance of a relational database.

    In GitHub:

1. *RC1.sql* is the manually adjusted SQL export from CIMTool
2. *LoadRC1.sql* will **re-create the GridAPPS-D database in MySQL**, incorporate *RC1.sql*, and finally document the foreign keys. It should run without error.

4.6 Platform UML Diagrams

4.6.1 UML from the Functional Specification

This section presents a selection of GridAPPS-D domain (class) diagrams to supplement the *OSPRREYS Functional Specification* document. The purpose is to enhance understanding of the functional specification, by providing graphical walkthroughs of some important use cases. The reader should be familiar with definitions in the functional specification, and with Universal Modeling Language (UML) diagrams.

GridAPPS-D is organized as a suite of internal function managers, twelve of them composing the Platform Manager as shown in Figure 1. All GridAPPS-D functions and interactions are mediated by one (or more) of these function managers. When running, the GridAPPS-D 413 Platform Manager will be composed of one (and only one) of each internal manager numbered 401 – 412. These internal managers work together to accomplish various GridAPPS-D functions.

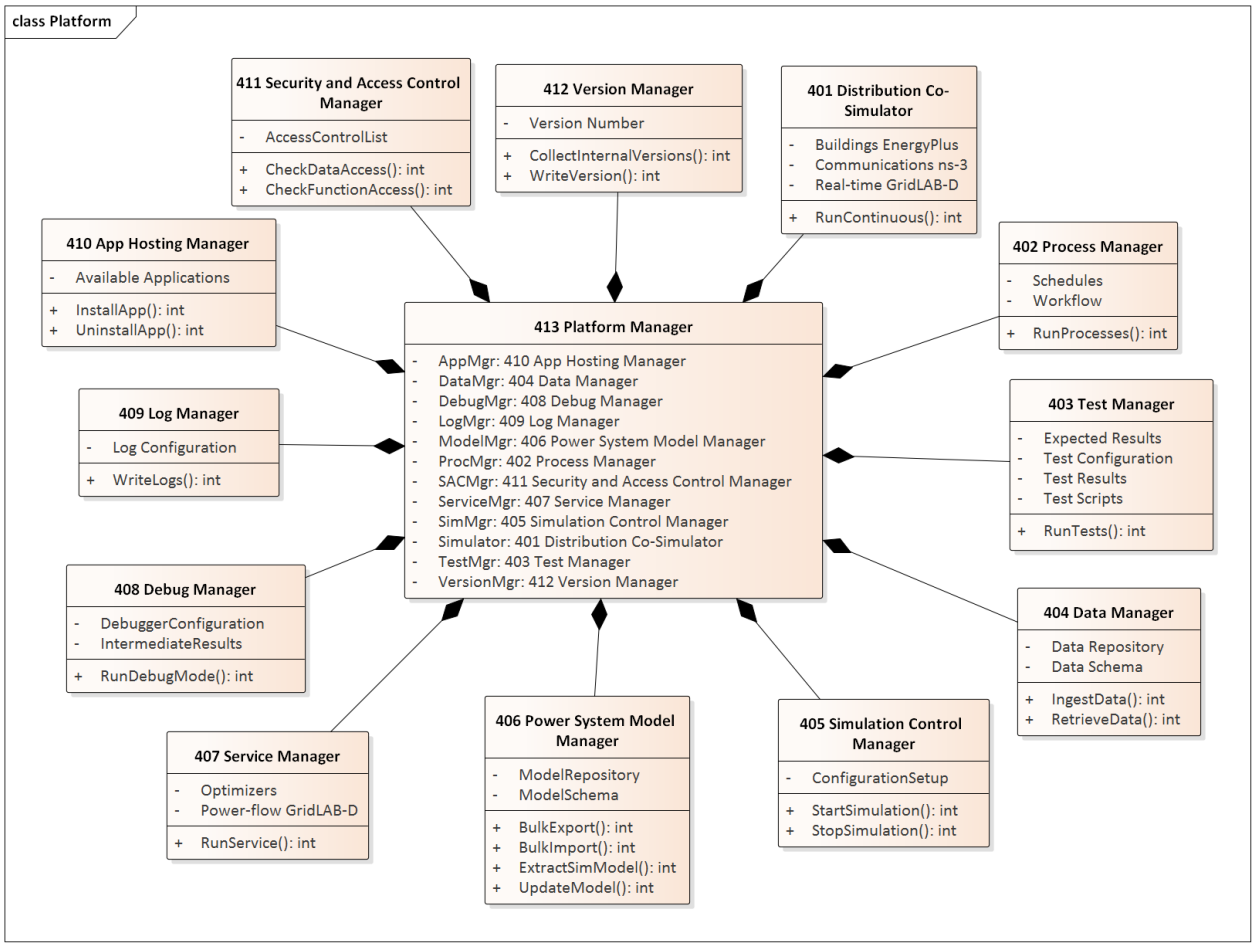


Figure 1: Composition of the GridAPPS-D Platform Manager

Within each class block, some top-level attributes are listed with (-) signs in the middle division, and some top-level methods are listed with (+) signs in the lower division. For example, we already know that 401 Distribution Co-Simulator will need component simulators (i.e. attributes) for buildings (open-source EnergyPlus), communications (open-source ns-3), and the electric power distribution grid (open-source GridLAB-D running in a real-time mode). It will also need at least one method that runs the suite of simulators in a mode emulating continuous real-time operation. Taking another example, 407 Service Manager also contains an attribute for GridLAB-D to provide power flow calculations, but run as a service to applications.

As the design evolves, classes in Figure 1 will acquire many more attributes and methods. The attributes themselves may reference complicated classes and data structures. Therefore, the UML model will expand each class into layer and sub-layer diagrams to more clearly show these evolving details. We can still use the top-level diagrams to make sure that the major components are in place for the important use cases.

Figure 2 illustrates the case of a user executing an application, in the role of EF7 from the functional specification. We initially focused on volt-var optimization (VVO), and then added a more complicated demand response (DR) application that fits the same basic pattern. As a prerequisite, some entity has provided both applications to GridAPPS-D for registration and hosting, in a process detailed later. For now, we assume the application(s) have been installed and will focus first on running VVO.

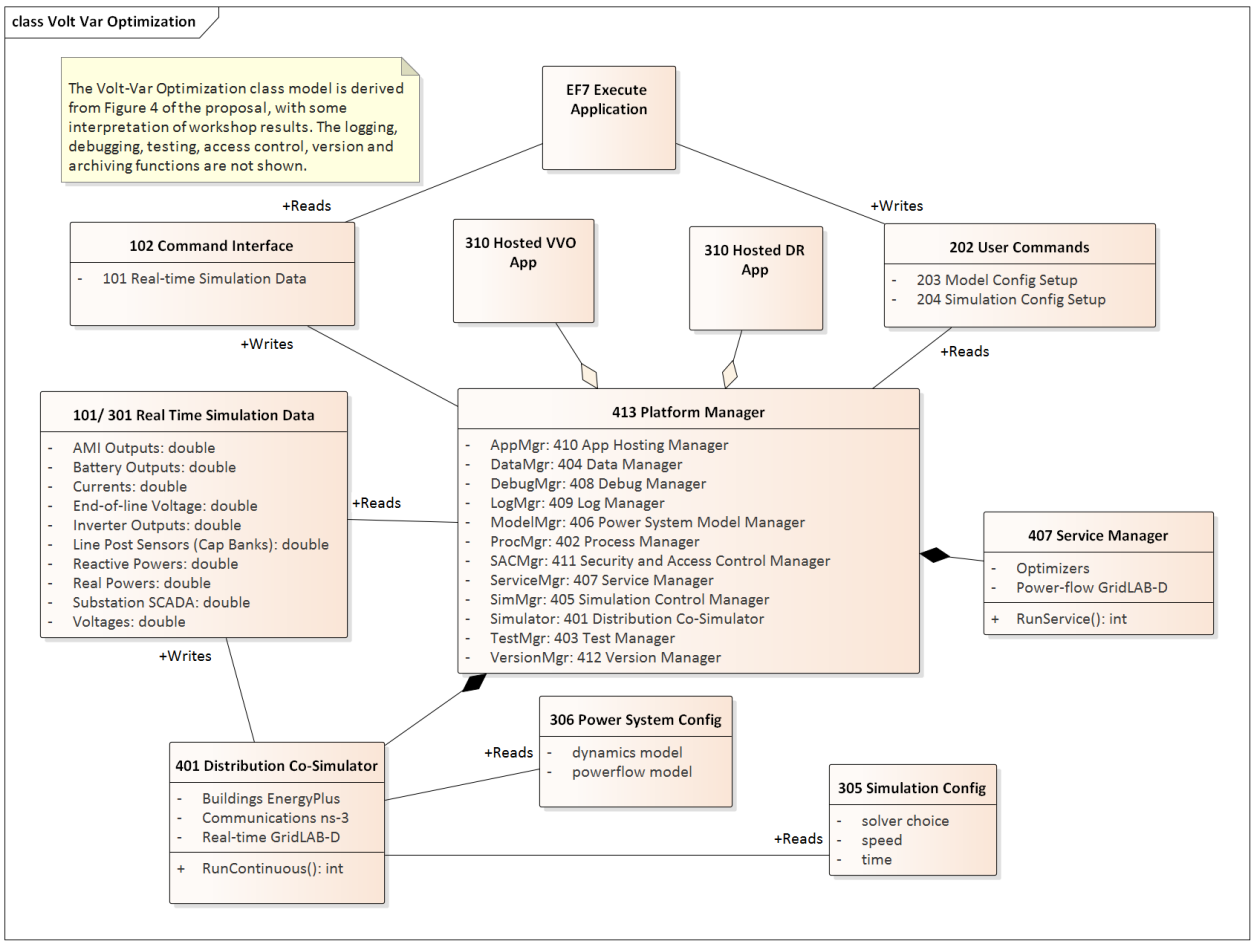


Figure 2: Executing an application

All user interaction with GridAPPS-D occurs through a command interface, numbered 202 when the user writes commands to GridAPPS-D, and numbered 102 when the user gets data from GridAPPS-D. To run VVO, the user will issue 203 Model Configuration Setup and 204 Simulation Configuration Setup to GridAPPS-D, which then delegates the commands to various internal function managers (see Figure 1). The 203 Setup will probably extract the feeder model of interest, set load and weather data, etc. The 204 Setup will probably tell 401 to run GridLAB-D for a certain time period, but not to run ns-3 or EnergyPlus. The exact composition of 203 and 204 Setups will be determined later in the design process. In a process described later, internal functions 405 (Simulation Control Manager) and 406 (Power System Model Manager) will transform 201, 203 and 204 into 305 and 306, which 401 can then read and run from directly.

When it runs, 401 will generate streams of data that mimic real-time operation of the system, and these streams pass to the other parts of GridAPPS-D as 301 Real-time Simulation Data. Some of the data streams may also output to the user as 101 Real-time Simulation Data. The 310 VVO Application can act on this data to make decisions (e.g. switch capacitor banks, change regulator taps, change solar inverter settings). In this process, 310 VVO could invoke power flow calculations in GridLAB-D via 407 Service Manager, but this is different from the way 401 Co-Simulator runs. The application may use 407 services to explore alternatives or run contingency analysis, which could change the power system model, but the 401 real-time simulations always take priority and always use the "real" model.

When we considered adding the second and more complicated application, 310 DR, the structure of Figure 2 didn't change very much. The open-headed diamond symbols indicate that GridAPPS-D can host several applications, which is UML aggregation. These applications may interact via the GridAPPS-D command interface, if the applications and their command sets have been designed for it. For example, the DR application may use VVO to check and mitigate voltage limits.

A DR application is more likely than VVO to need EnergyPlus and ns-3 in the co-simulation. In response, we added those attributes to 401, and will add supporting attributes to 201, 203 and 204 as the design evolves. It should also be recognized that more sophisticated VVO applications might incorporate communications (ns-3) if available.

Figure 3 depicts the process of managing power system models, including the schema and repository within 201 Distribution System Model. Because it's based on standards (e.g. IEC 61968) and open-source tools (e.g. MySQL), the model can be created and maintained from outside GridAPPS-D, directly by EF 21, the Model Manager. This is shown at the top of Figure 3. This process is out of GridAPPS-D scope but within project scope, and it can leverage existing tools like Cimphony, Cimdesk, EA, etc.

For use by and within GridAPPS-D, all model configuration commands will pass from EF21 through the command interface to function 406, the Power System Model Manager. This function reads the base power system model data from 201, and configures it into a three-phase load flow model for solution in 106/306. The Distribution Co-Simulator uses 306, but the user might want 106 for off-line use. Working with 404 Data Manager, the 406 Power System Model Manager may also write additional data (i.e. not used in the load flow calculation) to 104/304. In this case, the 102 Model Output function will collect that data from both 104 and 106 for reporting to the user, EF7, via the command interface. Note that the base data, in 201, is not modified through this process. Instead, the base data is treated as input to GridAPPS-D.

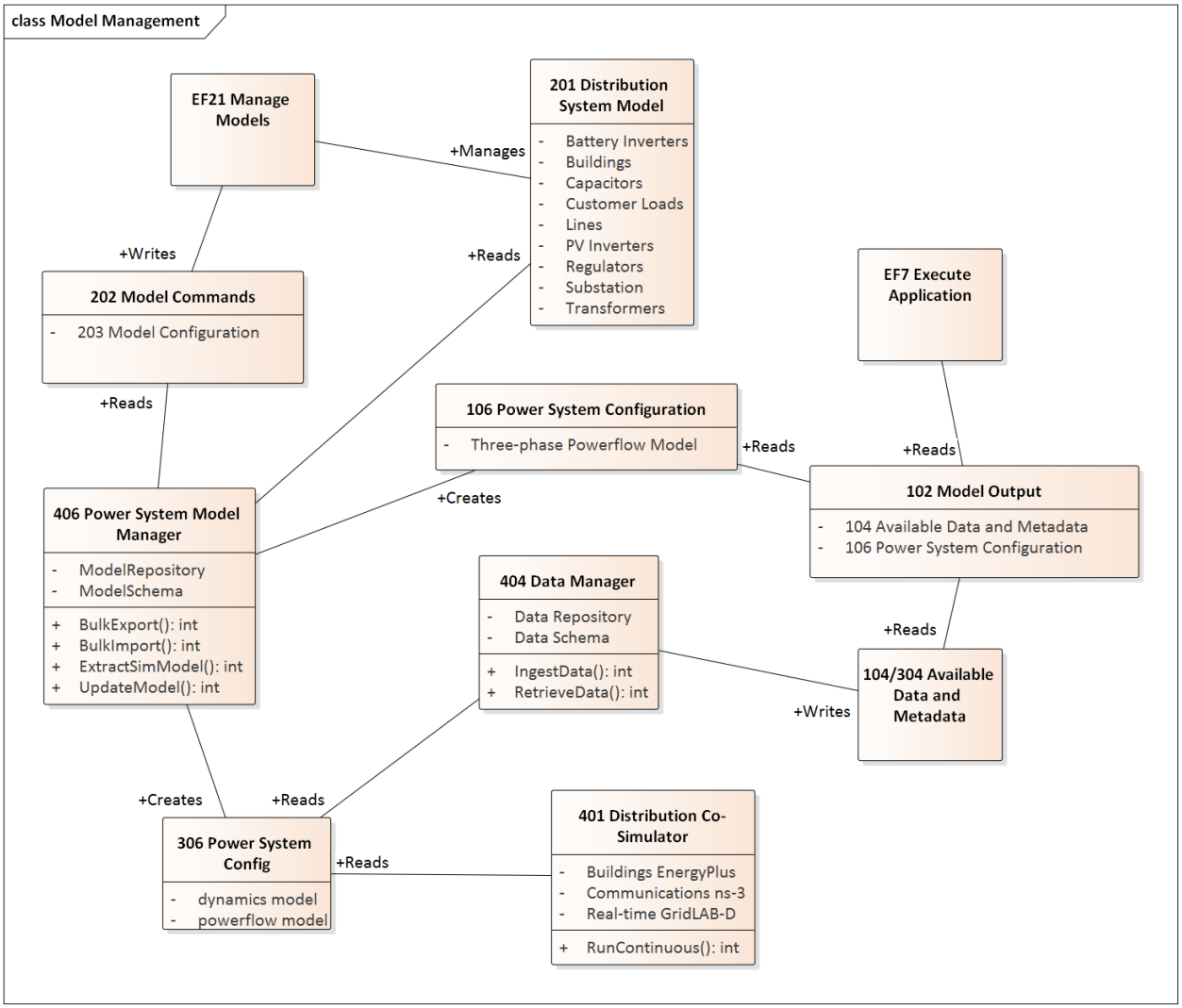


Figure 3: Internal model management

Figure 4 shows the internal Platform Manager flow when running application tests. Compared to the case of normal

usage in Figure 2, this example shows additional control and output for testing. The test commands include 203 and 204, as in Figure 2, but they also include:

- 205 Test Scripts, for the sequence of steps to perform
- 206 Test Configuration Setup, including initial conditions, etc.
- 207 Expected Results, for comparison to the actual output
- 210 Application Metadata, for information to run and instrument the application

The 403 Test Manager orchestrates the steps to run the application and collect results. As part of 103 Test Results, it will compare the real-time data (101/301) to the expected results in 207. If the testing user, EF8, requested logging, then the 409 Log Manager will create 109/309 System Logs for collection by 403 Test Manager. Logging is optional, and should have been requested as part of the 206 Test Config Setup or 204 Model Config Setup (this is not spelled out in the functional specification).

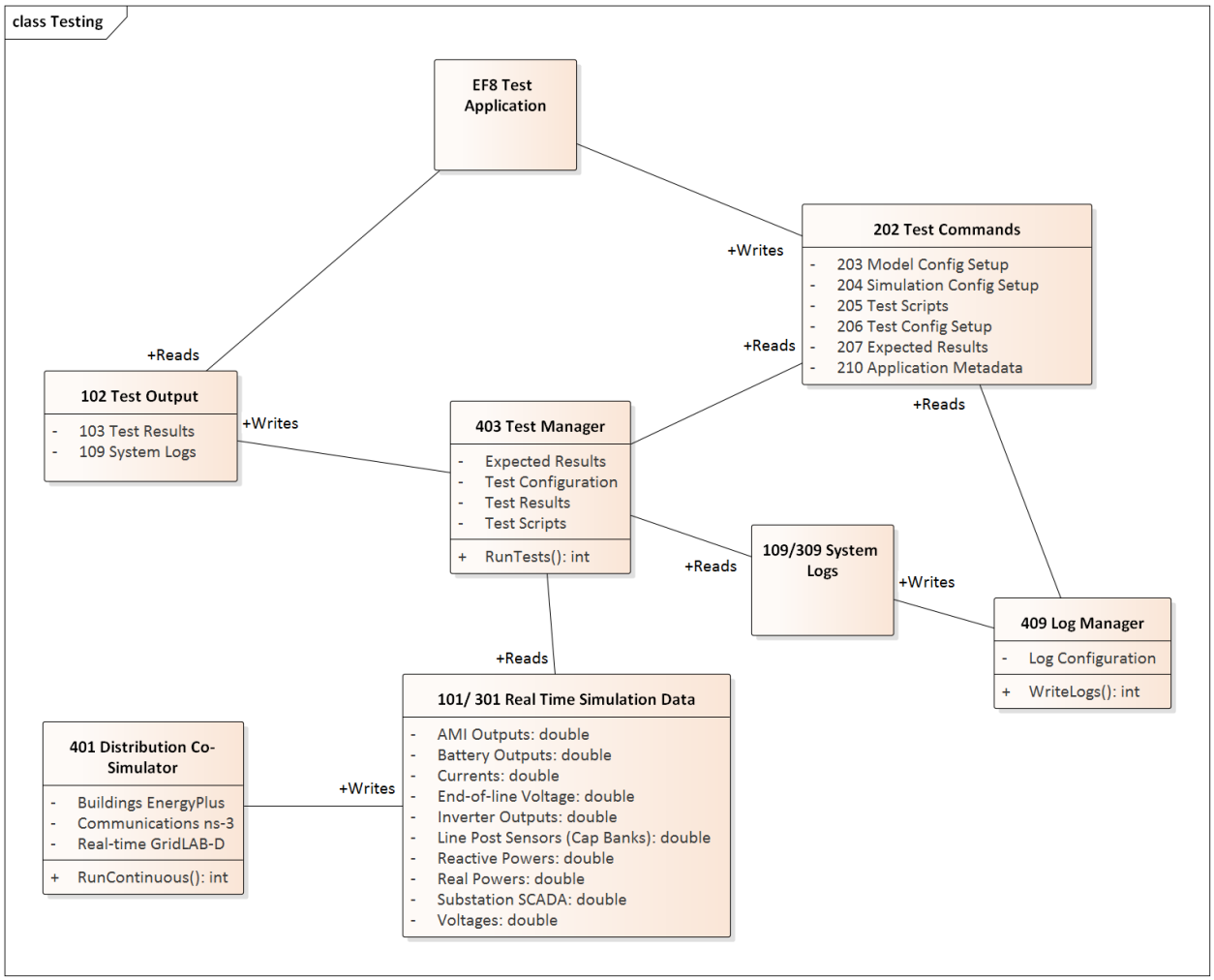


Figure 4: Testing an application or the platform

Figure 5 shows some of the internal 413 Platform Manager detail when a user, EF7, runs an application in debugging mode. Compared to Figure 2, there is much more internal output. The 212 Debug Configuration will include such things as breakpoints, watch variables, and logging requests. When run in debug mode, the 408 Debug Manager will collect the internal inputs and intermediate results from a variety of GridAPPS-D modules, including the simulator, services in use, model data, and access violations. The 404 Data Manager mediates most of this data collection (and with a change to the specification it could also mediate 101/301). The 408 Debug Manager combines this into 108

Intermediate Results, with 109 System Logs, for output to the user via the command interface. Depending on the implementation of GridAPPS-D, interactive debugging may also be supported, but is not shown in Figure 5.

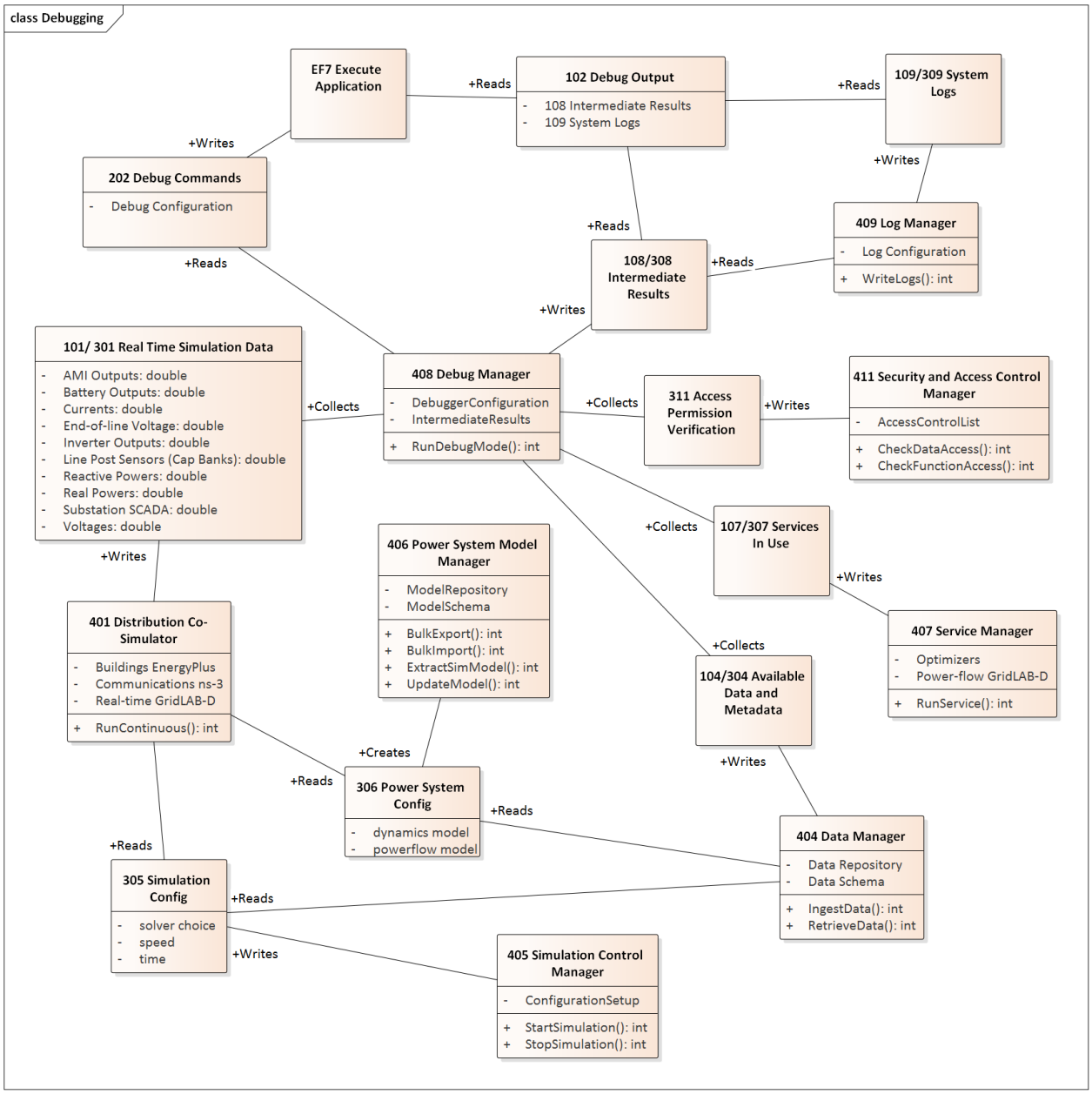


Figure 5: Debugging an application

Figure 6 shows the process of registering or updating an application to use with GridAPPS-D. The developer, in the role of EF13, must provide the application itself (211) along with the application data schema (208) and metadata (210). The data schema includes input and output parameters. The metadata includes a user-friendly name, description, calling parameters, command syntax, API functions used, etc. Using this information, 410 Application Hosting Manager will install and register the application, and its data, with 407 Service Manager and 404 Data Manager. After completing these steps, 412 Version Manager will output the current version information via the command interface; the current version includes information about which applications are installed along with the application versions.

In order to perform application management, EF13 also needs to provide user credentials to be checked against the 209 Access Control List. If these credentials are valid, the 411 SAC Manager will create 311 Access Permission

Verification for all of the internal Platform Manager components. In Figure 6, the 410 Application Hosting Manager can pass 311 to 404, 407 and 412 as needed. Although not shown earlier, SAC is actually incorporated into all GridAPPS-D processes this way.

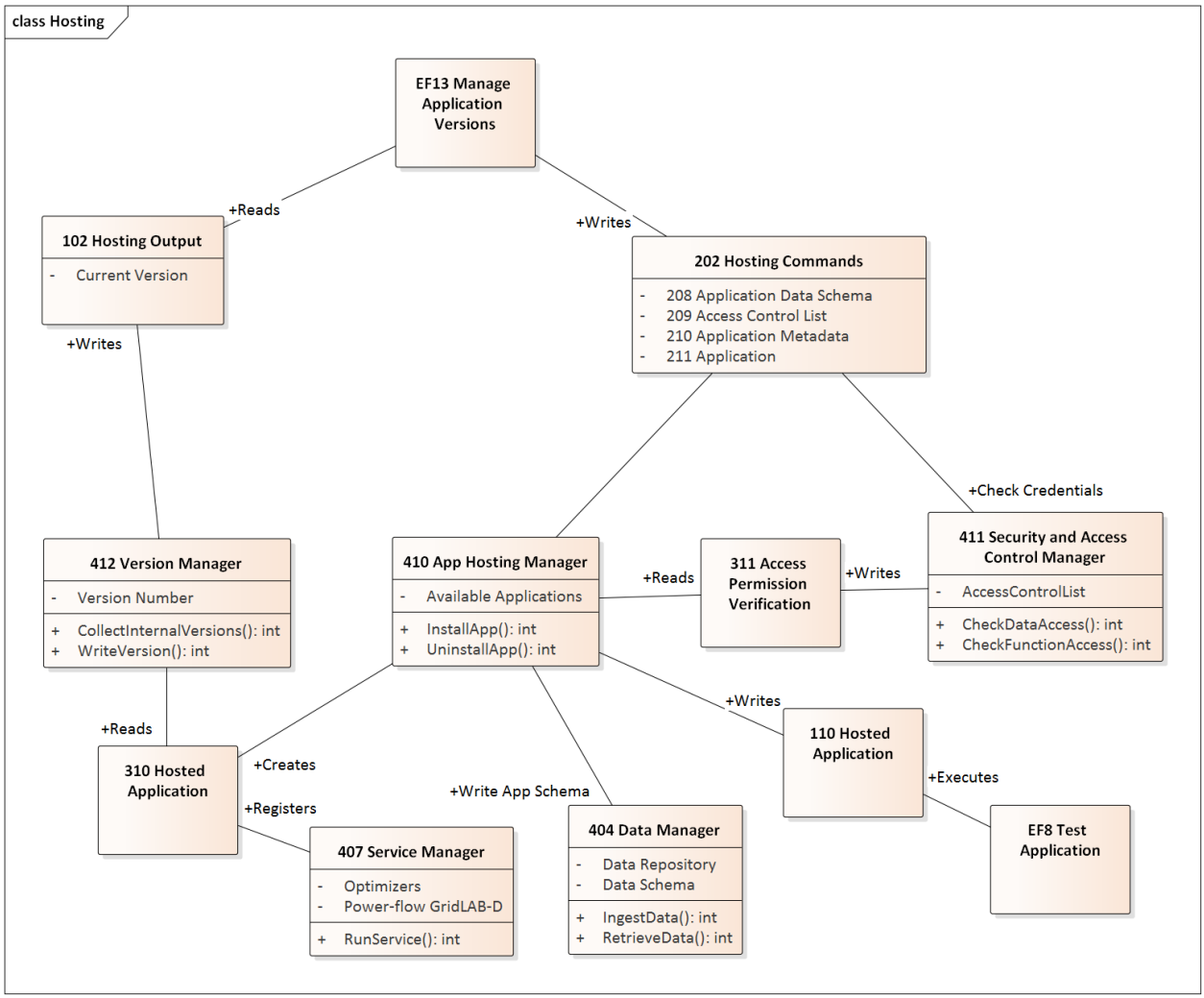


Figure 6: Hosting an application

4.6.2 UML for Release Cycle 1

Our objective is to demonstrate useful functionality, which is standards-compliant, by the end of March 2017. A simple heuristic VVO application will be running in GridAPPS-D. In terms of the Functional Requirements, we will be implementing:

- 102/202 Command Interface
- 301 Real-time Simulation Data
- 310 Hosted Application, but short-cutting the registration process
- 401 Distribution Co-Simulator (partial)
- 402 Process Manager (partial)
- 404 Data Manager (partial)

- 405 Simulation Manager (partial)
- 406 Power System Model Manager (partial)
- 413 Platform Manager (encapsulating 401 and 403-406)

This represents five out of twelve Internal Functions from the Functional Requirements, in partial form. The deadline leaves four months for detailed design and implementation, plus two months for documentation and testing. Therefore, we have chosen a minimal set of functions that can show end-to-end use of GridAPPS-D at the first milestone.

In developing the work breakdown structure (WBS), we noted that real-time simulation data is published with no time lags or errors in Release 1. However, data flow in a real DMS is affected by sensor and communication system performance, and also by the action of other subsystems. In Release 2, this might be addressed through some combination of:

- Communication and sensor models in the Distribution Co-Simulator
- Adding MDM and SCADA service attributes to the 407 Service Manager
- Filters on 301 Real-time Simulation Data

These decisions, and many others affecting Release 2 and Release 3, can be deferred until we gain experience developing Release 1.

Figure 1 shows the software components planned for Release 1. Most of these correspond to internal functions from the Functional Requirements, with some relatively minor re-factoring. The Power System Model Manager functionality has been split. The data store management and the creation of a complete GridLAB-D model appear at the bottom. Once the simulator is running, incremental changes are posted to the messaging bus.

Most of the "pink" components in Figure 1 are assigned to one task, except:

- The 310 VVO is a sub-task of the Command Interface, due to the close coupling of those efforts. The team on this task needs both power system and software skills.
- A separate task has been added for some project-level items.

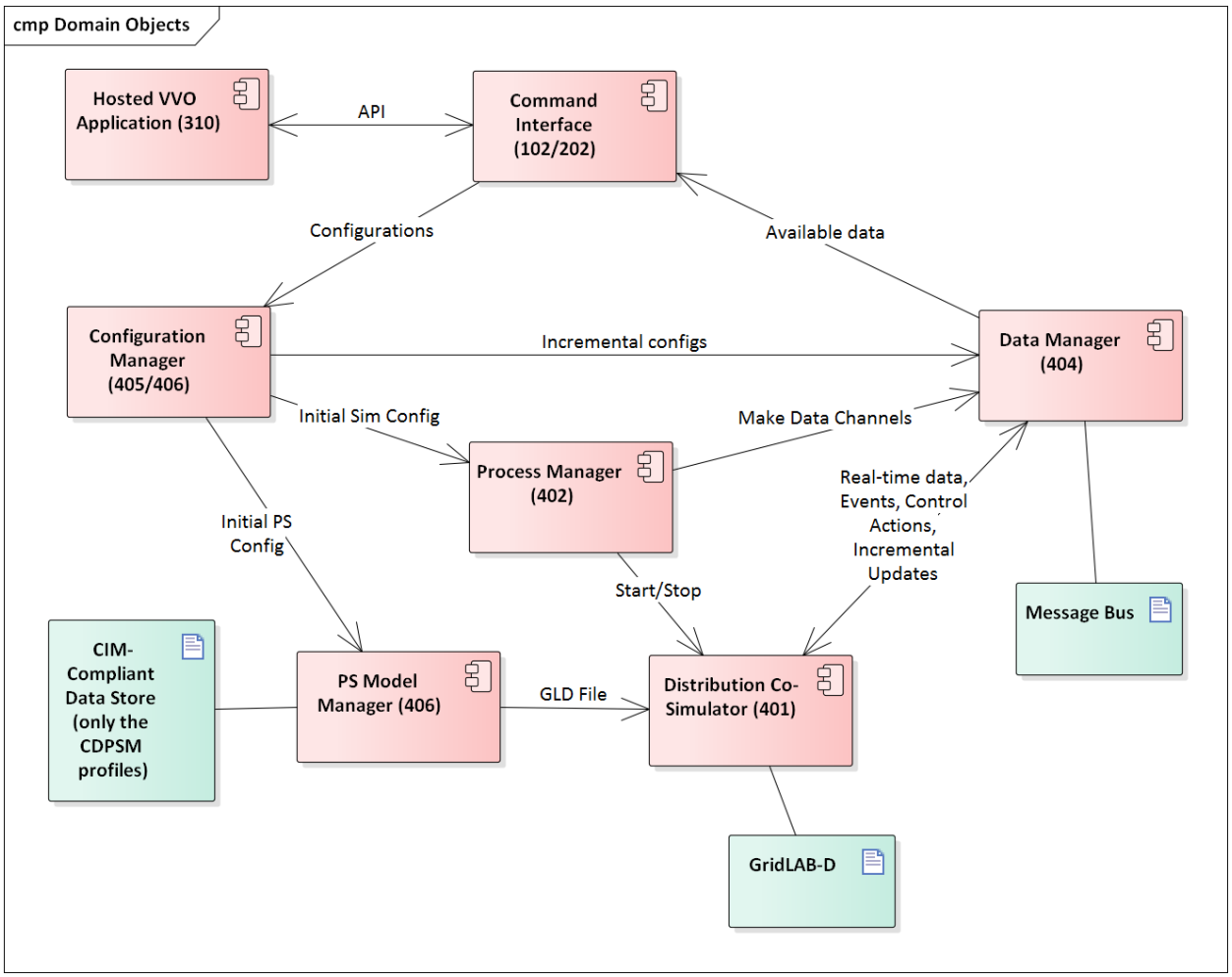


Figure 1: Component Diagram for GridAPPS-D Release 1

4.6.3 Initial Work Breakdown for Release Cycle 1

The Release 1 work breaks down into seven tasks, listed below. Three **critical items must be completed first**; these are highlighted in red. There are other inter-task dependencies that have not yet been called out. We plan to sequence the work over eight two-week "sprints" within the four months allocated for detailed design and development, using an agile process (Kanban).

1. Project-level Elements
   (a) **Identify a power system model (note: IEEE-13 is already in CIM/CDPSM)**
   (b) Design data store schema
   (c) Manually ingest power system models
2. Command Interface
   (a) Design APIs
      i. For all configurations in Task 4
      ii. For power system control actions (e.g. open/close switch)
   (b) **Select one language binding** (e.g. Python, Java, C++, MATLAB) and implement

   (c) Develop a heuristic volt-var application (VVO) in the bound language
   (d) Integrate VVO into GridAPPS-D
3. Messaging and Data Manager
   (a) **Select a messaging framework (eg. ZeroMQ)**
   (b) Create communication APIs
   (c) Receives real-time data from simulator
   (d) Receives power system control actions
   (e) Handle communication between GridAPPS-D managers
   (f) Log messages to file
4. Configuration Manager (both Power System Config & Simulation Config)
   (a) Receive configurations from command interface over message bus
   (b) Translate configurations to native GridLAB-D
   (c) Translate and publish incremental update messages
   (d) Send configurations to Process Manager for simulation start
5. Process Manager
   (a) Receives configurations from the Configuration Manager
   (b) Send configuration to the Distribution Co-Simulator
   (c) Start Co-Simulation Process
   (d) Create simulation data channels and inform application
   (e) Stop simulation process
6. Distribution Co-Simulator (wraps GridLAB-D)
   (a) Accepts configurations from Process Manager
   (b) Start simulation
   (c) Produce and publish data in real time
   (d) Accept changes in real time (e.g. capacitor switching) via message bus
7. Power System Model Manager
   (a) Access the power system model in data store
   (b) Create native GridLAB-D file for initial loading into the simulator

CHAPTER 5

Data Model

5.1 IEEE 8500-Node Test Feeder

An IEEE Working Group specified a set of distribution test circuits *[CIT1]* and we have chosen the largest one of these as a sample circuit for GridAPPS-D *[CIT2]*. The 8500-Node test feeder operates at 12.47 kV and has a peak load of about 11 MW, including approximately 1100 single-phase, center-tapped transformers with triplex service drops. Loads are *balanced* between the two center-tapped windings.

The circuit includes 4 shunt capacitor banks and 4 voltage regulator banks, making it a reasonable test for solving voltage problems and for applying volt-var optimization (VVO). The circuit is also relatively lossy at peak load.

The model in GridAPPS-D came from the IEEE 8500-Node input files distributed with OpenDSS, exported to CIM from OpenDSS, and then imported to the GridAPPS-D data manager. In this automated process, four changes were implemented:

1. **Use constant-current load models, rather than constant-power load models**. This is necessary for the solution to converge at peak load. Voltages at peak load are low, and a constant-power load will draw more current under those conditions. Holding the current magnitude constant allows GridLAB-D to achieve convergence under a variety of operating conditions. This is an appropriate compromise in accuracy for real-time applications, which need to be robust through wide variations in voltage and load. In contrast, planning applications usually need more accurate load models, even at the possible expense of re-running some non-converged simulations.
2. **Disable automatic regulator and capacitor controls**. The volt-var application, described below, will supersede these settings. If a developer or user is testing the GridLAB-D model outside of GridAPPS-D, these control settings should be re-enabled in order to solve the circuit at peak load. That requires manual un-commenting edits to the GridLAB-D input file.
3. **Substitute a variable called VSOURCE for the SWING bus nominal voltage**. This needs to be set at 1.05 per-unit of nominal on the 115-kV system (i.e. 69715.065) in order to solve at peak load. Other conditions may require different source voltage values.
4. **Use a schedule for the loads** so they can vary with time during GridAPPS-D simulation. The file should be named *ziploads_schedule.player*.

CHAPTER 6

Integrated Applications

6.1 Volt-var Optimization (VVO)

The sample VVO application is a Python implementation of a heuristic method that PNNL has investigated before *[CIT3]*, *[CIT4]*, *[CIT5]*. There are more advanced VVO methods that could be implemented in future applications.

6.2 Visualization

We have created a web-based visualization of the sample VVO application. The visualization displays the topology of the IEEE 8500-Node system as an interactive graph. Capacitors and regulators are highlighted in the graph and displayed alongside tables with current values for capacitor status (OPEN or CLOSED), regulator voltage, and feeder power.

6.3 PNNL Applications (Release Cycle 2)

CHAPTER 7

State Estimator

7.1 Objectives

State estimation is widely used in transmission system operations but is less common in distribution system operations due to a relatively limited value in traditional distribution systems, additional computational complexity, and a lack of sensors. Advanced distribution management platforms like GridAPPS-D provide access to model and sensor data that can be leveraged to overcome barriers to adoption and open the door to distribution system state estimators that are fast and accurate enough to be useful in utility operations.

A distribution system state estimator computes the most likely state given a set of present and/or past measurements. The full state of a distribution system consists of either the full set of complex bus voltages or the full set of complex branch currents; given the system model (admittance matrix), the remaining system parameters can be computed given the full system state.

7.1.1 Use Cases

- Assist power factor optimization: Utility objective is unity power-factor at the substation.
- Assist voltage optimization (planning): Utility objective is 1 p.u. voltage at last house primary.
- Real-time state estimation for advanced applications: applications can access the state estimate at a sufficient resolution to capture e.g. insolation variation caused by clouds.

7.1.2 Distribution System State Estimation Algorithms

State estimation uses system model information to produce an estimate of the state vector x given a measurement vector z. The measurement vector is related to the state vector and an error vector by the measurement function, which may be non-linear.

$$z = h(x) + e$$

Multiple formulations of the distribution system state estimation problem are possible:

1. *Node Voltage State Estimation (NVSE):* The state vector consists of node voltage magnitudes and angles for each node in the system (one reference angle can be eliminated from the state vector). This formulation of the state estimation problem is general to any topology and it is the standard for transmission system state estimation.
2. *Branch Current State Estimation (BCSE):* Radial topology and assumptions about shunt losses create a linear formulation of the state estimation problem. The state vector contains branch currents and, for a fully-constrained problem, requires one state per load, which can be less than the number of branches in the system.

Different algorithms provide different advantages for distribution system state estimation. A subset of the state estimation algorithms below will be used to achieve these goals.

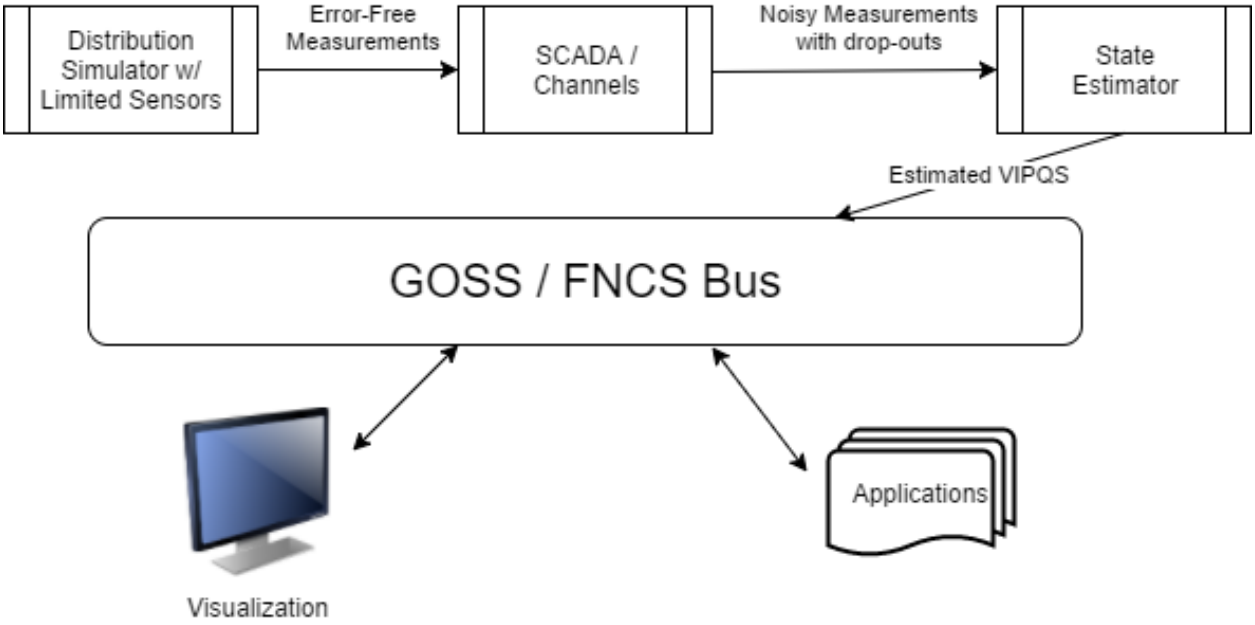
1. *Weighted Least Squares Estimation (WLSE)*: a concurrent set of measurements are used to find a state vector that minimizes the weighted least squares objective function. The algorithm is memoryless with respect to previous solutions and measurements should be synchronized.
2. *Kalman Filter Estimation (KFE) and Extended Kalman Filter Estimation (EKFE)*: The Kalman filter provides a mechanism to consider past state estimates alongside present measurements. This provides additional noise rejection and allows asynchronous measurements can be considered individually. KFE is appropriate for linear BCSE and EKFE is compatible with nonlinear NVSE.
3. *Unscented Kalman Filter Estimation (UKFE)*: The unscented transform estimates the expected value and variance of the system state by observing the system outputs for inputs spanning the full dimensionality of the measurement space. Again, the Kalman filter provides a mechanism to consider past estimates.

7.1.3 TRL

The state estimator application will provide the capability to estimate the full system state using asynchronous measurement data. In addition a model order reduction technique will be implemented to greatly speed up the state estimation computation and to reduce the dependence on forecast-based pseudo-measurements. A paper (*Reduced-Order State Estimation for Power Distribution Systems with Sparse Sensing*) is targeted for IEEE Transactions on Power Systems.

7.2 Design

The state estimation service is being developed in c++. A modern c++ implementation allows the application to adapt to an evolving interface. The program architecture is shown below.



Topology Processor: initializes the measurement function and its Jacobian and determines the size of the measurement vector, the measurement covariance matrix, and the state vector.

Meter Interface: updates the measurement vector and the measurement covariance matrix as new measurement data comes available.

State Estimator: performs the state estimation operation according to the specified algorithm.

Output Interface: formats the state vector and any implicit states as an output stream.

7.2.1 Inputs:

Upon initialization, the topology processor will receive the Y-bus from the GridLAB-D service and will query contextual information and sensor locations from the CIM database.

Periodic measurement data, including any forecasts to be used a pseudo-measurements will be required as inputs.

A "terminate" command from the platform will end the state estimation process.

7.2.2 Outputs:

The output will include the full system state (node voltages and/or branch currents TBD).

7.3 Testing and Validation

7.3.1 Evaluation metrics

- State Error: compare state estimation output to "true" system state.
- Accuracy over baseline: compare state error of state estimator to state error of a QSTS load-flow model.
- Execution Time
- Bad Sensor Detection (binary)

7.3.2 Scenarios

- Full sensor deployment: verify that the true system state can be reproduced.
- Sparse sensor deployment: verify that the state estimator performs better than a QSTS load-flow model.
- Breaker trip: verify that switch state can be detected even when it is reported incorrectly.
- Bad sensor detection: verify that a sensor that is producing bad data can be identified.
- Dependent application support: verify that the state estimator can support e.g. the VVO application.
- Fault: for a radial system, determine the nearest common bus from multiple emulated customer calls.

7.4 Operating/Running

The state estimator will execute the topology processor at initialization and will enter a stat estimation loop. The state estimation loop will exit and the process will end upon receiving a 'terminate' command from the platform.

At initialization, a configuration file will be read for:

- State estimation mode (state vector and algorithm) selection
- Normalized residual threshold for bad measurement / sensor detection

7.5 References

[1] Abur and A. G. Exposito, *Power System State Estimation*, New York, NY: Marcel Dekker, Inc., 2004.

[2] M. E. Baran and A. W. Kelley, "A branch-current-based state estimation method for distribution systems," in *IEEE Transactions on Power Systems*, vol. 10, no. 1, pp. 483-491, Feb 1995.

[3] Z. Jia, J. Chen and Y. Liao, "State estimation in distribution system considering effects of AMI data," *2013 Proceedings of IEEE Southeastcon*, Jacksonville, FL, 2013, pp. 1-6.

[4] S. C. Huang, C. N. Lu and Y. L. Lo, "Evaluation of AMI and SCADA Data Synergy for Distribution Feeder Modeling," in *IEEE Transactions on Smart Grid*, vol. 6, no. 4, pp. 1639-1647, July 2015.

[5] M. Kettner; M. Paolone, "Sequential Discrete Kalman Filter for Real-Time State Estimation in Power Distribution Systems: Theory and Implementation," in *IEEE Transactions on Instrumentation and Measurement*, vol.PP, no.99, pp. 1-13, Jun. 2017.

[6] G. Valverde and V. Terzija, "Unscented kalman filter for power system dynamic state estimation," in *IET Generation, Transmission & Distribution*, vol. 5, no. 1, pp. 29-37, Jan.

CHAPTER 8

Model Validator

8.1 Objectives

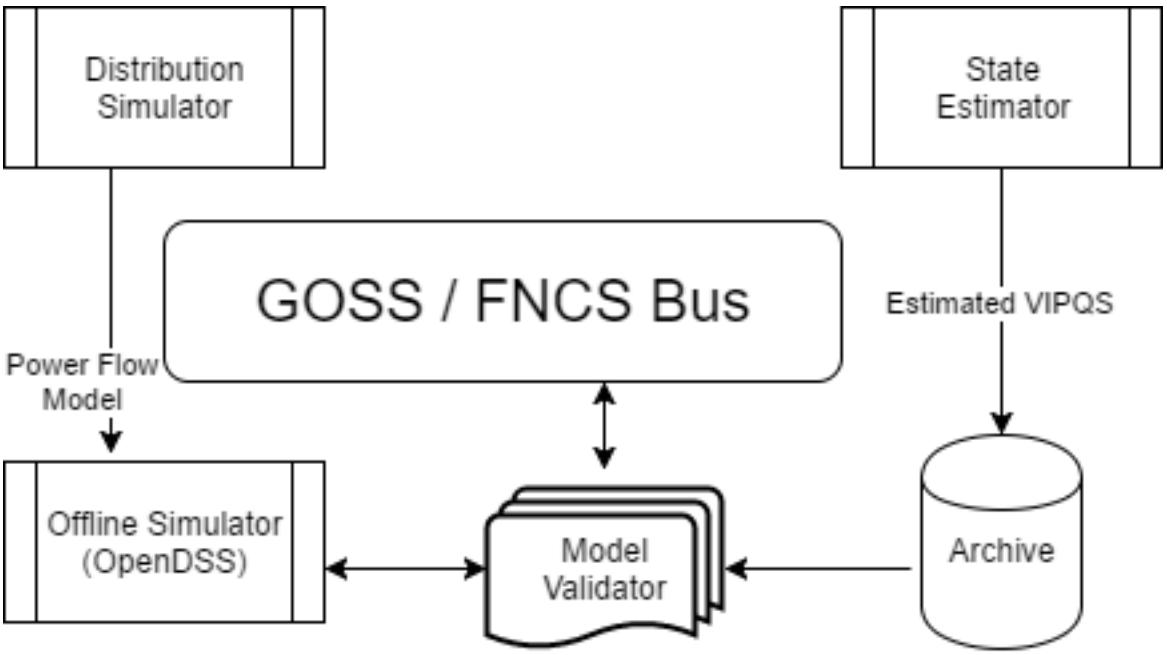
The model validator will detect and attempt to correct unreasonable component interconnections and network parameters.

8.1.1 Use Cases

- Valid transformer size and orientation (Utility): orientation is not captured explicitly in their GIS system.
- Discover secondary line impedance parameters (Utility) conductor type and line length are currently based on generic assumptions.
- Sanity check or estimate transformer size and impedance.
- Verify that the nominal voltage of nodes matches the base voltage of the segment: generally the winding voltage of the upstream transformer or swing bus voltage.
- Sanity check conductor sizes and line current ratings.
- Validate and fill in regulator and capacitor control settings.
- Check phase continuity (GridLAB-D may not model phase discontinuities)

8.2 Design

The model validation application will be implemented in Python.



8.2.1 Inputs:

The model validator will have access to the CIM database and archived data from the state estimator.

8.2.2 Outputs:

The model validator will one or both of the following outputs:

- Model status: log file or GUI pipe for identified issues.
- Model correction: CIM updates to correct identified issues.

8.3 Testing and Validation

8.3.1 Evaluation metrics

- Ability to detect known issues.

8.3.2 Scenarios

- Utility merger: models with different format may be interpreted differently, creating issues a CIM model.
- Data entry issue: model update does not match upgrade performed in the field

8.4 Operating/Running

The model validator script will execute once when called by the platform.

At initialization, a configuration file will be read for:

- Mode (status, quiet, verbose; see outputs section)
- Selectable validation items (use cases)

Given a perfect and complete set of voltage magnitude and angle measurements, along with a detailed and accurate power system model, one could calculate the real power, or any other electrical variable of interest, anywhere in the system. In practice, measurements have errors, time delays, and may even be missing. State estimation refers to the process of minimizing the errors and filling in gaps[1]. One state estimation method is called "weighted least squares", and it's analogous to drawing the best-fit line through a set of scattered points. Other methods may perform better[2]. Also, on distribution systems, it may be better to estimate branch currents instead of node voltages, but the principle is the same. In GridAPPS-D, the visualizations and applications ought to use the best available state estimator outputs, instead of raw SCADA values, for both accuracy and consistency. Therefore, the state estimator is not an application but a service in GridAPPS-D, sitting between emulated SCADA and the GOSS bus.

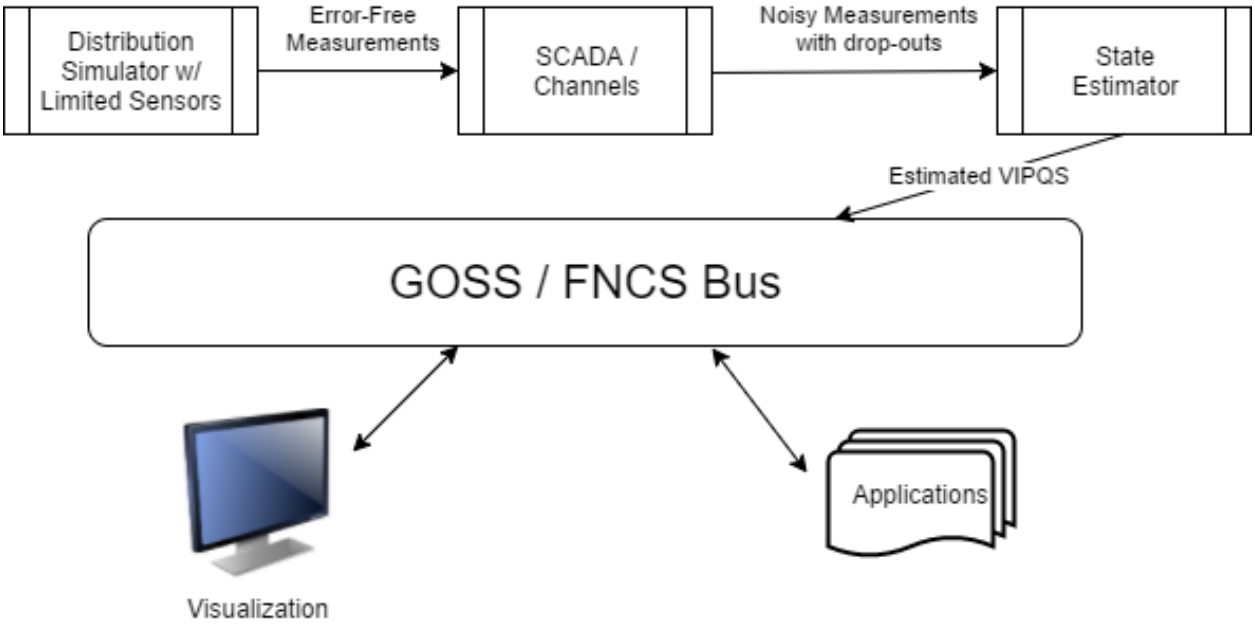


Figure 1: The state estimator processes noisy and incomplete measurements, then posting estimated voltage (V), current (I), real power (P), reactive power (Q) and switch status (S) values onto the GridAPPS-D message / data bus.

In Figure 1, the power system model (upper left) will include a limited number of sensors, corresponding to actual voltage and current transformers, line post sensors, wireless sensors, etc. In some scenarios, smart meters can also be sensors. Each such sensor will have different performance characteristics (e.g. precision, accuracy, sampling rate). Distribution systems typically do not have enough sensors to make the system observable, so there will be measurement gaps in the topology. The state estimator might fill these gaps with interpolation and graph-tracing methods on the power system model.

The supervisory control and data acquisition (SCADA) system in Figure 1 introduces more errors and failure points. Eventually, GridAPPS-D may simulate these impacts by federating ns-3 as a co-simulator. Until then, a placeholder module could be used to insert variable errors, time delays and dropouts in each measurement, whether due to sensor characteristics or the communication system. The output represents data as it would come into an operations center, and feeds the state estimator. Internally, the data flows between simulator, SCADA and state estimator might be

[1]

20. (a) McDermott, "Grid Monitoring and State Estimation," in *Smart Grid Handbook*, ed: John Wiley & Sons, Ltd, 2016.

[2]

1. Abur and A. Gomez Exposito, *Power system state estimation : theory and implementation*. New York, NY: Marcel Dekker, 2004.

implemented with FNCS, but this is an implementation detail. The state estimator will provide two outputs to the GOSS bus used by all GridAPPS-D applications:

1. At a time step configured by the platform, publish the best-estimate VIPQS values wherever sensors actually exist in the model, with quality attributes that still have to be established. Sensor locations delineate circuit segments, and note that all VIPQS values will be estimated at the boundaries, even if the sensor measures only V or I, for example.
2. Upon request by another application or service, publish the estimated VIPQS values for all nodes and components in the model, even at locations where no sensors exist. A variant is to publish the estimates only for selected nodes and components.

As indicated in Figure 1, other applications need to obtain estimated VIPQS values from the GOSS bus. Switch open/close states are a special case; they might be considered known values, but in practice the switch state is a measurement, which could lead to topology errors in the model. For GridAPPS-D, switch state estimates need to be a point of emphasis. Given that most distribution systems lack redundant measurements, It would be possible for an application to query these VIPQS values directly from the simulator or SCADA, bypassing the state estimator, but this is "cheating" in most situations. However, in the application development process, idealized VIPQS values could be obtained through a combination of two methods:

1. Add more sensors to the power system model
2. Set the sensor and channel errors to zero

Because the sensor outputs in GridAPPS-D come from a power flow solution that enforces Kirchhoff's Laws, the state estimator will produce ideally accurate values whenever the sensor and channel errors have been specified to be zero. The state estimator may still exhibit interpolation errors between sensor locations, but that is readily mitigated for testing purposes by adding more sensors.

With reference to RC1, the visualization and VVO applications should now subscribe to VIPQS values from the state estimator, not from the distribution simulator. They may also use or display quality metrics on the estimated values.

The state estimator basically attempts to fit measured data to a power flow model, usually assuming that the model is correct. However, a model attribute (e.g. line impedance) could also be estimated by minimizing its error residual in the state estimator's power flow solution. This process works best when applied to just one or a few suspect attributes, and/or when an archive is available to provide enough redundant measurements. The Model Validation Application will use these state estimator features off-line to help identify and correct the following types of model errors:

1. Unknown or incorrect service transformer sizes
2. Unknown or incorrect secondary circuit lengths
3. Incorrect phase identification of single-phase components
4. Phase wiring errors in line segments and switches
5. Transformer connection errors, especially reversed primary and secondary
6. Primary conductor sizes that don't decrease monotonically with distance from the source
7. Missing regulator and capacitor control settings (i.e. supply defaults from heuristic rules)
8. More than one of these on the same pole: recloser, line regulator, capacitor
9. Substation transformer impedance and turns ratio

These types of errors often appear upon the initial model import from a geographic information system (GIS), or in periodic model updates from GIS. Other error types may be added later. Many utilities do not have their secondary circuits modeled at all, but this has an important impact on AMI data. The service transformers and secondary circuits insert significant impedance between AMI meters and the primary circuit, where most of the other sensors are installed. Therefore, the first two items will require AMI data, and also enable its more effective use.

As shown in Figure 2, the Model Validator integrates with GridAPPS-D as a hosted application on the GOSS bus. Internally, it will use some of the same algorithms as the State Estimator and may share some code or binary files, but this is an implementation detail. It will need to access an archive of state-estimated VIPQS data, which may include AMI data. It will also use or incorporate an off-line power flow model, not the same one running in the GridAPPS-D distribution simulator. This may be EPRI's OpenDSS simulator[3]; compared to GridLAB-D, it's more tolerant of model errors and provides more diagnostic information about model errors.

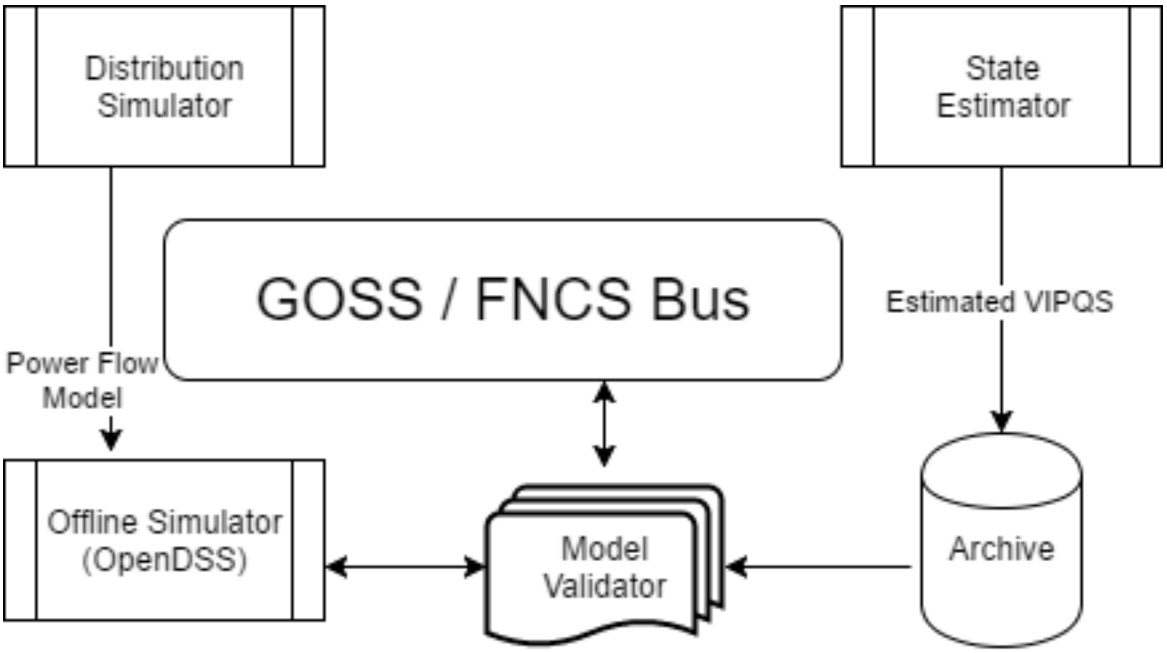


Figure 2: The Model Validator works with an archive from the state estimator, and an off-line power flow model.

Transactive energy is a method of controlling loads and resources on the distribution system, combining both market and electrical principles[4]. One reason for including this application in DOE-funded GridAPPS-D is that PNNL has made several technical contributions and led several demonstration projects in transactive systems, also funded by DOE[5].

Application structure

This transactive systems application is to be implemented as a modularized 2-layer 3-level structure, as seen from Figure 3. The layer decomposition helps the control of various groups, with limited information flow between different layers. With the predefined functions in each agent type (Agent A, B, and C) in each level, the existing transactive system related work can be conveniently integrated into the application, and the new control features can be added into specific control function in each type of the agent easily.

[3]

18. (a) Dugan and T. E. McDermott, "An open source platform for collaborating on smart grid research," in *Power and Energy Society General Meeting, 2011 IEEE*, 2011, pp. 1-7.

[4] Gridwise Architecture Council. (2017). *Transactive Energy*. Available: http://www.gridwiseac.org/about/transactive_energy.aspx

[5] Pacific Northwest National Laboratory. (2017). *Transactive Energy Simulation Platform (TESP)*. Available: http://tesp.readthedocs.io/en/latest/

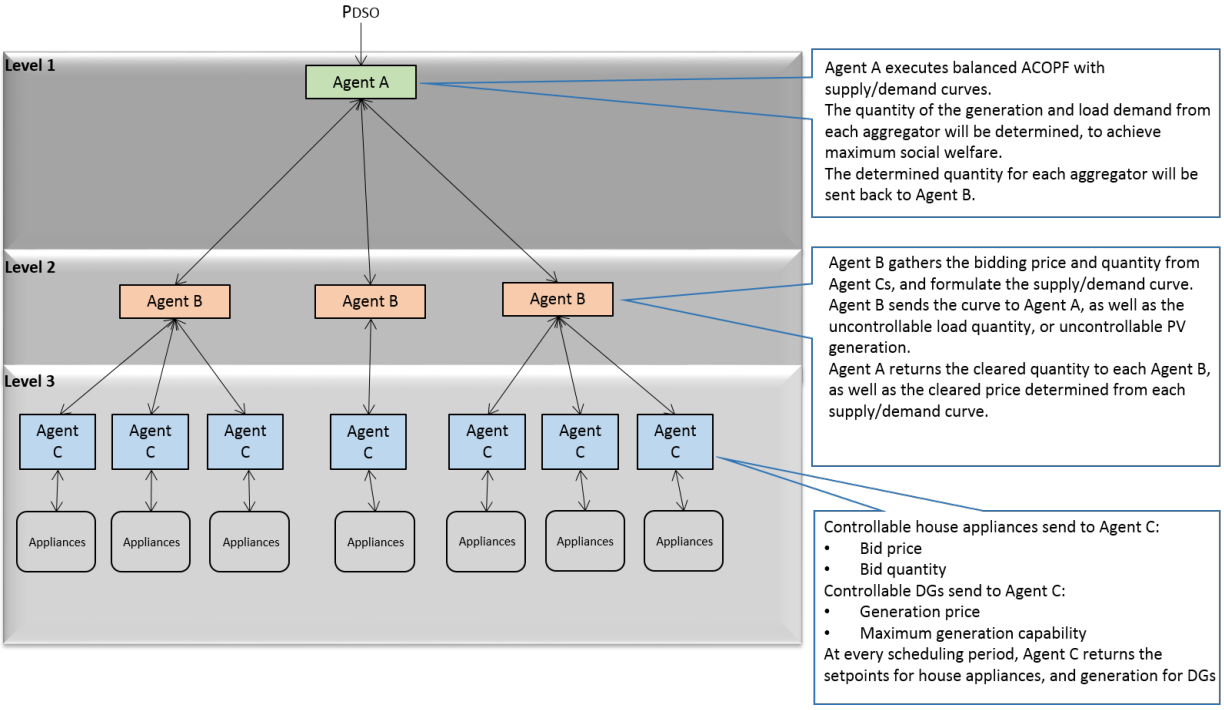


Figure 3: The structure of the modularized 2-layer 3-level transactive system application

The modularized agents opens the door for integrating different control mechanisms into the application. Users need to consider which level their control algorithm fits into, and fill in the control function of the Agent class in that level, without worrying about communications between the agents. In each level, the same type of the agent may have various control functions, which help combining benefits of different control schemes together.

Agent A, B and C will be implemented as VOLTTRON applications. VOLTTRON is an application platform for distributed sensing and control applications[6]. With the capability of hardware-in-the-loop (HIL) testing through VOLTTRON, the transactive systems application will be tested using the actual devices. A GOSS-VOLTTRON Bridge is to be implemented, for the communication between GridAPPS-D and the VOLTTRON agents in the transactive systems application.

Application test cases

The hierarchical control framework introduced in[7] for integrated coordination between distributed energy resources and demand response will be implemented into the application. In addition, [7] has not considered the power losses or power constrains, which will be taken into consideration in this test case. The two-layer control mechanism, including the coordination layer and device layer, fits the proposed structure of the application well. The control in each level will be implemented into corresponding function in each type of the agent. The IEEE 123-node test feeder built in GridLAB-D will be used for testing the application.

CIM extension for the Application

The latest versions of GridAPPS-D has used a reduced-order CIM to support feeder modeling. With transactive system application included into GridAPPS-D platform, more objects, such as house air conditioner and water heater, need to be defined in CIM. Before the definition in CIM, a simplified version of the house object and water heater object are to be implemented in GridLAB-D.

[6] 19. Katipamula, J. Haack, G. Hernandez, B. Akyol, and J. Hagerman, "VOLTTRON: An Open-Source Software Platform of the Future," *IEEE Electrification Magazine,* vol. 4, pp. 15-22, 2016.

[7] Di Wu, Jianming Lian, Yannan Sun, Tao Yang, Jacob Hansen, "Hierarchical control framework for integrated coordination between distributed energy resources and demand response," Electric Power Systems Research, pp. 45-54, May 2017.

8.5 NREL Applications (Release Cycle 2)

CHAPTER 9

Distribution Optimal Power Flow for Real-Time Setpoint Dispatch

9.1 Objectives

This application is designed to address the problem of optimizing the operation of aggregations of heterogeneous energy resources connected to a distribution system. We will focus on real-time optimization method and the power setting points of the distributed energy resources (DERs) will be updated on a second or subsecond timescale to maximize the operational objectives while coping with the variability of ambient conditions and noncontrollable energy assets [1]. In order to avoid massive measurements and overcome the limitation caused by model inaccuracy, this application will be implemented in a distributed manner, and only local measurements and a feedback signal from the substation aggregator are needed to determine the optimal setpoints for each controlled DER unit.

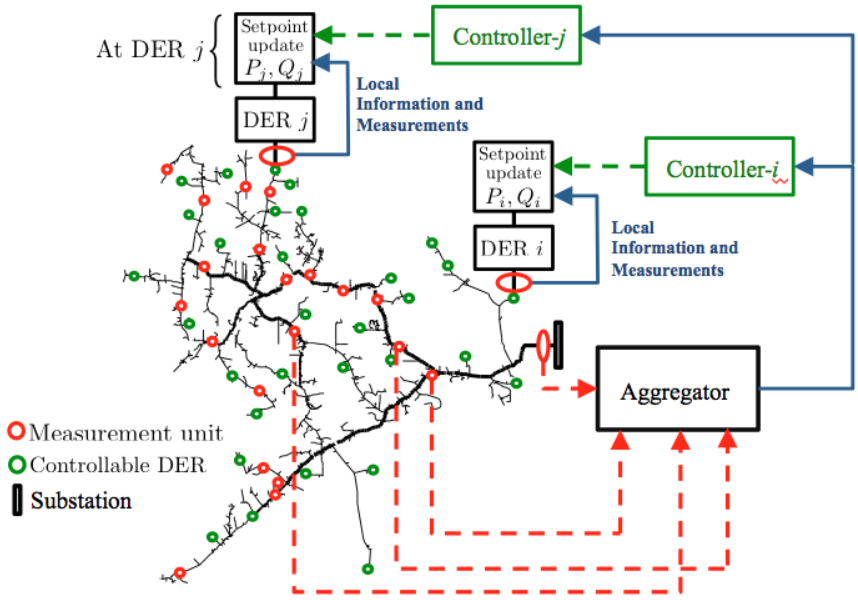


Figure 1 The conceptual framework of distribution OPF for real-time setpoint dispatch.

Figure 1 shows the conceptual framework of this application, and this application is targeting at TRL 3.

9.2 Design

Figure 2 describes the overall work flow of the application. Distribution OPF algorithm requires real-time measurements, distribution system model and power flow results, which will be obtained from GridAPPS-D platform through GOSS/FNCS message bus. The optimization problem formulation can be constructed using user-defined cost functions for different controllable devices. Finally the optimal setpoints for controllable devices will be solved based on the feedback information from system measurements. These setpoints will be sent back to GridLab-D grid model to update DER operations. Such a closed-loop control forms the control iteration for the studied time point, and new setpoints for the following time points will be determined in the same manner using the updated model and measurements.

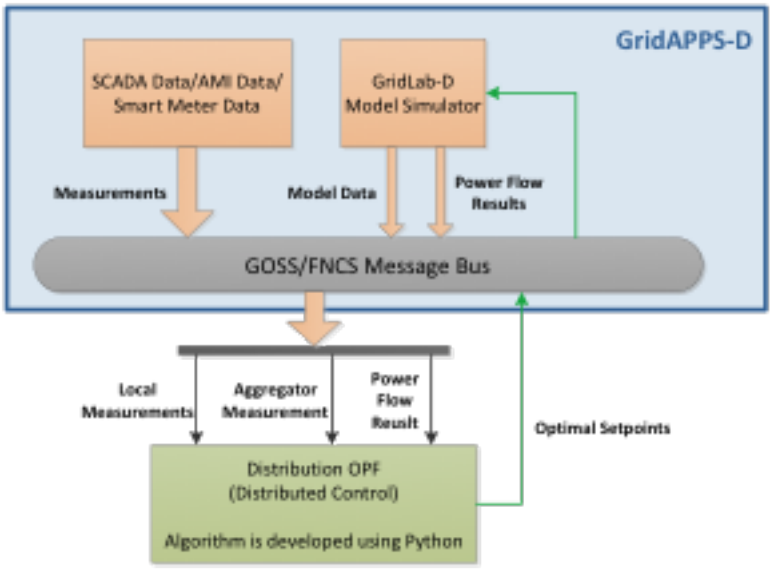


Figure 2 The workflow of real-time setpoint dispatch application and its interaction with GridApps-D.

Data requirements

Message schemas (UML) (Enterprise Architect software) Jeff will help draw the UML diagram.

9.3 Testing and Validation

Evaluation metrics of this application:

- Real/reactive power at the substation
- System loss
- Voltages across the entire distribution grid: voltage magnitude, voltage fluctuation, voltage unbalance.
- Legacy control device operations: total control actions of all capacitors and regulators

Scenarios:

- Optimal Dispatch for Distributed PV Systems
- Optimal Dispatch for Distributed PV + Energy Storage

- Etc. (will be added when implementing the application)

9.4 Operating/Running

This application will be developed using Python.

9.5 References

[1] E. Dall'Anese, A. Bernstein, and A. Simonetto, "Feedback-based Projected-gradient Method for Real-time Optimization of Aggregations of Energy Resources," IEEE Global Conference on Signal and Information Processing (GlobalSIP), Montreal, Canada, Nov. 2017.

CHAPTER 10

API Documentation

10.1 GridAPPS-D

10.2 GOSS

The GridOPTICS Software System (GOSS) manages the platform data and message bus; its overall design is described in *[CIT6]*.

10.3 FNCS

The Framework for Network Co-simulation (FNCS) manages the time clock and message traffic between platform simulators; its overall design is described in *[CIT7]*. For API documentation see https://github.com/FNCS/fncs/wiki .

10.4 VVO

10.5 GridLAB-D

GridLAB-D is the distribution grid simulator within the platform; its overall design is described in *[CIT8]*.

10.6 gov.pnnl.gridlabd.cim

This Java package converts CIM RDF to GridLAB-D format.

10.6.1 CDPSM_to_GLM

public class **CDPSM_to_GLM**
This class converts CIM (IEC 61968) RDF to GridLAB-D format

The general pattern is to retrieve iterators on the different types of objects (e.g. ACLineSegment) through simple SPARQL queries. Usually these iterators include just the mrID, or the mrID and name. Then Jena RDF model and resource functions are used to pull other properties from iterated objects, writing GridLAB-D input along the way. A future version will rely more heavily on SPARQL queries to do the selection and filtering, as the preferred pattern for developers working with CIM. In existing code, the EnergySource most closely follows this new preferred pattern.

Invoke as a console-mode program

Author Tom McDermott

See also: *CDPSM_to_GLM.main*, CIM User Group, CIM Profile and Queries for Feeder Modeling in GridLAB-D, GridLAB-D

Fields

baseURI

static final String **baseURI**
identifies gridlabd

mapNodes

static HashMap<String, *GldNode*> **mapNodes**
to look up nodes by name

mapSpacings

static HashMap<String, *SpacingCount*> **mapSpacings**
to look up line spacings by name

neg120

static final Complex **neg120**
Rotates a phasor -120 degrees by multiplication

nsCIM

static final String **nsCIM**
namespace for CIM; should match the CIM version used to generate the RDF

nsRDF

static final String **nsRDF**
namespace for RDF

pos120

static final Complex **pos120**
Rotates a phasor +120 degrees by multiplication

ptBaseNomV

Property **ptBaseNomV**

ptEqBaseV

Property **ptEqBaseV**

ptEquip

Property **ptEquip**

ptLevBaseV

Property **ptLevBaseV**

Methods

AccumulateLoads

static boolean **AccumulateLoads**(*GldNode nd*, String *phs*, double *pL*, double *qL*, double *Pv*, double *Qv*, double *Pz*, double *Pi*, double *Pp*, double *Qz*, double *Qi*, double *Qp*)
Distributes a total load (pL+jqL) among the phases (phs) present on GridLAB-D node (nd)

Parameters

- **nd** – GridLAB-D node to receive the total load
- **phs** – phases actually present at the node
- **pL** – total real power
- **qL** – total reactive power
- **Pv** – real power voltage exponent from a CIM LoadResponseCharacteristic
- **Qv** – reactive power voltage exponent from a CIM LoadResponseCharacteristic
- **Pz** – real power constant-impedance percentage from a CIM LoadResponseCharacteristic
- **Qz** – reactive power constant-impedance percentage from a CIM LoadResponseCharacteristic
- **Pi** – real power constant-current percentage from a CIM LoadResponseCharacteristic
- **Qi** – reactive power constant-current percentage from a CIM LoadResponseCharacteristic
- **Pp** – real power constant-power percentage from a CIM LoadResponseCharacteristic
- **Qp** – reactive power constant-power percentage from a CIM LoadResponseCharacteristic

Returns always true

Bus_ShuntPhases

static String **Bus_ShuntPhases** (String *phs*, String *conn*)
appends N or D for GridLAB-D loads and capacitors, based on wye or delta connection

Parameters

- **phs** – from CIM PhaseCode
- **conn** – contains *w* for wye connection and *d* for delta connection

Returns phs with N or D possibly appended

CFormat

static String **CFormat** (Complex *c*)

Parameters

- **c** – complex number

Returns formatted string for GridLAB-D input files with 'j' at the end

Count_Phases

static int **Count_Phases** (String *phs*)
from the phase string, determine how many are present, but ignore D, N and S

Parameters

- **phs** – the parsed CIM PhaseCode

Returns (1..3)

FindBaseVoltage

static double **FindBaseVoltage** (Resource *res*, Property *ptEquip*, Property *ptEqBaseV*, Property *ptLevBaseV*, Property *ptBaseNomV*)
Returns the nominal voltage for conduction equipment, from either its own or container's base voltage. For example, capacitors and transformer ends have their own base voltage, but line segments don't.

Parameters

- **res** – an RDF resource corresponding to a ConductingEquipment instance; we need to find its base voltage
- **ptEquip** – an RDF property corresponding to the EquipmentContainer association
- **ptEqBaseV** – an RDF property corresponding to a possible BaseVoltage association on the equipment itself
- **ptLevBaseV** – an RDF property corresponding to the EquipmentContainer's BaseVoltage association
- **ptBaseNomV** – an RDF property corresponding to the nominalVoltage attribute of a CIM BaseVoltage

Returns the nominal voltage as found from the equipment or its container, or 1.0 if not found

FindConductorAmps

static String **FindConductorAmps** (Model *mdl*, Resource *res*, Property *ptDataSheet*, Property *ptAmps*)
needs to return the current rating for a line segment 'res' that has associated WireInfo at 'ptDataSheet', which in turn has the current rating at ptAmps

TODO - this is not implemented; emitted syntax is for OpenDSS and the function call (below, in main) needs review

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to a CIM ACLineSegment
- **ptDataSheet** – an RDF property corresponding to CIM AssetDatasheet attribute
- **ptAmps** – an RDF property corresponding to CIM ratedCurrent attribute

Returns unusable OpenDSS input

FirstPhase

static String **FirstPhase** (String *phs*)

Parameters

- **phs** – a parsed CIM PhaseCode

Returns the first phase found as A, B, or C

GLDCapMode

static String **GLDCapMode** (String *s*)
translate the capacitor control mode from CIM to GridLAB-D

Parameters

- **s** – CIM regulating control mode enum

Returns MANUAL, CURRENT, VOLT, VAR

GLD_ID

static String **GLD_ID** (String *arg*)
parse the GridLAB-D name from a CIM name, based on # position

Parameters

- **arg** – the CIM IdentifiedObject.name attribute, not the mrID

Returns the compatible name for GridLAB-D

GLD_Name

static String **GLD_Name** (String *arg*, boolean *bus*)
convert a CIM name to GridLAB-D name, replacing unallowed characters and prefixing for a bus/node

Parameters

- **arg** – the root bus or component name, aka CIM name
- **bus** – to flag whether *nd_* should be prepended

Returns the compatible name for GridLAB-D

GetACLineParameters

static String **GetACLineParameters** (Model *mdl*, String *name*, Resource *r*, double *len*, double *freq*, String *phs*, PrintWriter *out*)
for a standalone ACLineSegment with sequence parameters, find GridLAB-D formatted and normalized phase impedance matrix

TODO - this is always three-phase, so we don't need all 7 variations from GetSequenceLineConfigurations

-

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **name** – the root name of the line segment and its line_configuration
- **r** – an RDF resource corresponding to a CIM ACLineSegment
- **len** – the length of the ACLineSegment in feet
- **freq** – frequency in Hz for converting susceptance to capacitance
- **phs** – phasing for the written line_configuration (one of 7 variations) that needs to be referenced
- **out** – the PrintWriter instance opened from the main program, passed here so that we can share code in GetSequenceLineConfigurations

Returns the name of the written line_configuration

GetBusName

static String **GetBusName** (Model *mdl*, String *eq_id*, int *seq*)
finds the bus (ConnectivityNode) name for conducting equipment

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file*
- **eq_id** – the CIM mrID of the conducting equipment
- **seq** – equals 1 to use the first terminal found, or 2 to use the second terminal found

Returns the GridLAB-D compatible bus name, or *x* if not found. As Terminals no longer have sequence numbers, the ordering of seq is unpredictable, so if there are two we can get bus 1 - bus 2 or bus 2 - bus 1

GetBusPositionString

static String **GetBusPositionString** (Model *mdl*, String *id*)
for a bus (ConnectivityNode), search for X,Y geo coordinates based on connected Terminals and equipment

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **id** – name of the bus to search from

Returns X,Y coordinates in comma-separated value (CSV) format

GetCableData

static String **GetCableData** (Model *mdl*, Resource *res*)
needs to return underground_line_conductor data in GridLAB-D format

TODO - this is not implemented; the emitted syntax is actually for OpenDSS

-

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to a CIM CableInfo (not a leaf/concrete class)

Returns unusable OpenDSS input

GetCapControlData

static String **GetCapControlData** (Model *mdl*, Resource *rCap*, Resource *ctl*)

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rCap** – an RDF resource corresponding to a CIM LinearShuntCompensator (aka capacitor)
- **ctl** – an RDF resource corresponding to the CIM RegulatingControl that was found attached to the LinearShuntCompensator

Returns the embedded capacitor control data for a GridLAB-D capacitor object

GetEquipmentType

static String **GetEquipmentType** (Resource *r*)
find the type of monitored equipment for controlled capacitors, usually a line or the capacitor itself

Parameters

- **r** – an RDF resource, will have a CIM mrID, should be a LinearShuntCompensator, ACLineSegment, EnergyConsumer or PowerTransformer

Returns cap, line, xf if supported in GridLAB-D; NULL or ##UNKNOWN## if unsupported

GetGldTransformerConnection

static String **GetGldTransformerConnection** (String[] *wye*, int *nwdg*)

Map CIM connectionKind to GridLAB-D winding connections. TODO: some of the returnable types aren't actually supported in GridLAB-D

Parameters

- **wye** – array of CIM connectionKind attributes per winding
- **nwdg** – number of transformer windings, also the size of wye

Returns the GridLAB-D winding connection. This may be something not supported in GridLAB-D, which should be treated as a feature request

GetImpedanceMatrix

static String **GetImpedanceMatrix** (Model *mdl*, String *name*, Property *ptCount*, Resource *r*, boolean *bWantSec*)

Convert CIM PerLengthPhaseImpedance to GridLAB-D line_configuration

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **name** – root name of the line_configuration(s), should be the CIM name
- **r** – an RDF resource, will have a CIM mrID, should be PerLengthPhaseImpedance
- **ptCount** – an RDF property for the PerLengthPhaseImpedance.conductorCount
- **bWantSec** – flags the inclusion of triplex, true except for debugging

Returns the GridLAB-D formatted impedance matrix for a line configuration. We have to write 3 of these in the case of 1-phase or 2-phase matrices. If (by name) it appears to be triplex and bWantSec is false, nothing will be returned.

GetLineSpacing

static String **GetLineSpacing** (Model *mdl*, Resource *rLine*)

needs to return the line_spacing and wire/cncable/tscable assignments for this rLine in GridLAB-D format

TODO - this is not implemented, the emitted syntax is actually for OpenDSS

-

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rLine** – an RDF resource corresponding to a CIM ACLineSegment that should have an associated AssetInfo

Returns unusable OpenDSS input

GetMatIdx

static int **GetMatIdx**(int *n*, int *row*, int *col*)
converts the [row,col] of nxn matrix into the sequence number for CIM PerLengthPhaseImpedanceData (only valid for the lower triangle) *

Parameters

- **n** – 2x2 matrix order
- **row** – first index of the element
- **col** – second index

Returns sequence number

GetPowerTransformerData

static String **GetPowerTransformerData**(Model *mdl*, Resource *rXf*)

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to CIM PowerTransformer; it should have mesh impedance data

Returns transformer and transformer_configuration objects in GridLAB-D format

GetPowerTransformerTanks

static String **GetPowerTransformerTanks**(Model *mdl*, Resource *rXf*, ResIterator *itTank*, boolean *bWantSec*)
writes a PowerTransformer in GridLAB-D format, in the case where individual tranformer tanks that are connected together in a bank. GridLAB-D supports only 2-winding banks with same phasing on both sides, or single-phase, center-tapped secondary transformers.

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to a CIM PowerTransformer that uses tank modeling
- **itTank** – a Jena iterator on the tanks associated with rXf, known to be non-empty before this function is called
- **bWantSec** – usually true, in order to include single-phase, center-tapped secondary transformers, which would come to this function

Returns transformer object in GridLAB-D format; the transformer_configuration comes from calling GetXfmrCode

GetPropValue

static String **GetPropValue**(Model *mdl*, String *uri*, String *prop*)
unprotected lookup of uri.prop value, to be deprecated in favor of SafeProperty

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **uri** – an RDF resource, currently only an EquipmentContainer is used, and it should always exist
- **prop** – currently only IdentifiedObject.name is used, and it should always exist

Returns the name of the CIM object

GetRegulatorData

static String **GetRegulatorData**(Model *mdl*, Resource *rXf*, String *name*, String *xfGroup*, String *bus1*, String *bus2*, String *phs*)

Connects a regulator in GridLAB-D format between bus1 and bus2; should be called from GetPowerTransformerTanks. In CIM, a regulator consists of a transformer plus the ratio tap changer, so if such is found, should call GetRegulatorData instead of just writing the transformer data in GetPowerTransformerTanks. Any impedance in the regulating transformer will be lost in the GridLAB-D model. Should be called from PowerTransformers that have RatioTapChangers attached, so we know that lookup will succeed

TODO: implement regulators for tank transformers

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **rXf** – an RDF resource corresponding to a CIM PowerTransformer that has a RatioTapChanger associated
- **name** – the name of the PowerTransformer (already looked up before calling this function)
- **xfGroup** – the PowerTransformer's IEC vector group (already looked up before calling this function)
- **bus1** – first bus (ConnectivityNode) on the regulator (already looked up before calling this function)
- **bus2** – second bus (ConnectivityNode) on the regulator (already looked up before calling this function)
- **phs** – phases that contain A, B and/or C (already looked up before calling this function)

Returns regulator and regulator_configuration objects in GridLAB-D format

GetSequenceLineConfigurations

static String **GetSequenceLineConfigurations**(String *name*, double *sqR1*, double *sqX1*, double *sqC1*, double *sqR0*, double *sqX0*, double *sqC0*)

For balanced sequence impedance, return a symmetric phase impedance matrix for GridLAB-D. We have to write 7 variations to support all combinations of 3, 2 or 1 phases used.

Parameters

- **name** – is the root name for these 7 variations
- **sqR1** – positive sequence resistance in ohms/mile
- **sqX1** – positive sequence reactance in ohms/mile
- **sqC1** – positive sequence capacitance in nF/mile
- **sqR0** – zero sequence resistance in ohms/mile

- **sqX0** – zero sequence reactance in ohms/mile
- **sqC0** – zero sequence capacitance in nF/mile

Returns text for 7 line_configuration objects

GetWdgConnection

static String **GetWdgConnection** (Resource *r*, Property *p*, String *def*)
parse the CIM WindingConnection enumeration

Parameters

- **r** – an RDF resource, will have a CIM mrID, should be a transformerEnd
- **p** – an RDF property, will be a CIM attribute, should be connectionKind
- **def** – default value if property is not found, such as Y

Returns D, Y, Z, Yn, Zn, A or I

GetWireData

static String **GetWireData** (Model *mdl*, Resource *res*)
needs to return overhead_line_conductor data in GridLAB-D format; res is the CIM OverheadWireInfo instance

TODO - this is not implemented; the emitted syntax is actually for OpenDSS

-

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **res** – an RDF resource corresponding to CIM OverheadWireInfo

Returns unusable OpenDSS input

GetXfmrCode

static String **GetXfmrCode** (Model *mdl*, String *id*, double *smult*, double *vmult*, boolean *bWantSec*)
Translates a single TransformerTankInfo into GridLAB-D format. These transformers are described with short-circuit and open-circuit tests, which sometimes use non-SI units like percent and kW, as they appear on transformer test reports.

TODO: smult and vmult may be removed, as they should always be 1 for valid CIM XML

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **id** – CIM mRID corresponding to a CIM TransformerTankInfo
- **smult** – scaling factor for converting winding ratings to volt-amperes (should be 1)
- **vmult** – scaling factor for converting winding ratings to volts (should be 1)
- **bWantSec** – usually true to include single-phase, center-tapped secondary tranformers, which come to this function

Returns transformer_configuration object in GridLAB-D format

GldPrefixedNodeName

static String **GldPrefixedNodeName** (String *arg*)
prefix all bus names with *nd_* for GridLAB-D, so they "should" be unique

Parameters

- **arg** – the root bus name, aka CIM name

Returns nd_arg

MergePhases

static String **MergePhases** (String *phs1*, String *phs2*)
accumulate phases without duplication

Phase_Kind_String

static String **Phase_Kind_String** (String *arg*)
parses a single phase from CIM SinglePhaseKind

Parameters

- **arg** – CIM SinglePhaseKind enum

Returns A, B, C, N, s1 or s2

Phase_String

static String **Phase_String** (String *arg*)
parses the phase string from CIM phaseCode

Parameters

- **arg** – CIM PhaseCode enum

Returns some combination of A, B, C, N, s1, s2, s12

SafeBoolean

static boolean **SafeBoolean** (Resource *r*, Property *p*, boolean *def*)
look up Jena boolean value

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns boolean value, or default if not found

SafeDouble

static double **SafeDouble**(Resource *r*, Property *p*, double *def*)
look up Jena double value

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns double value, or default if not found

SafeInt

static int **SafeInt**(Resource *r*, Property *p*, int *def*)
look up Jena integer value

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns integer value, or default if not found

SafePhasesX

static String **SafePhasesX**(Resource *r*, Property *p*)
look up Jena phase property

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute

Returns phases in string format, or ABCN if not found

SafeProperty

static String **SafeProperty**(Resource *r*, Property *p*, String *def*)
look up Jena string property

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns the property (or default value) as a string

SafeRegulatingMode

static String **SafeRegulatingMode** (Resource *r*, Property *p*, String *def*)
parse the CIM regulating control mode enum

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns voltage, timeScheduled, reactivePower, temperature, powerFactor, currentFlow, userDefined

SafeResName

static String **SafeResName** (Resource *r*, Property *p*)
for components (not buses) returns the CIM name from r.p attribute if it exists, or the r.mrID if not, in GridLAB-D format

Parameters

- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute

Returns a name compatible with GridLAB-D

SafeResourceLookup

static String **SafeResourceLookup** (Model *mdl*, Property *ptName*, Resource *r*, Property *p*, String *def*)

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **ptName** – should be the IdentifiedObject.Name property of the resource we are looking for
- **r** – an RDF resource, will have a CIM mrID
- **p** – an RDF property, will be a CIM attribute
- **def** – default value if property is not found

Returns the GridLAB-D formatted name of a resource referenced by r.p

Shunt_Delta

static boolean **Shunt_Delta** (Resource *r*, Property *p*)
for loads and capacitors, returns true only if CIM PhaseShuntConnectionKind indicates delta

Parameters

- **r** – an RDF resource, will have a CIM mrID, should be LinearShuntCompensator or EnergyConsumer
- **p** – an RDF property, will be a CIM attribute for phaseConnection

Returns true if delta connection

WirePhases

static String **WirePhases**(Model *mdl*, Resource *r*, Property *p1*, Property *p2*)

Returns GridLAB-D formatted phase string by accumulating CIM single phases, if such are found, or assuming ABC if not found. Note that in CIM, secondaries have their own phases s1 and s2. *

Parameters

- **mdl** – an RDF model (set of statements) read from the CIM imput file
- **r** – an RDF resource, will have a CIM mrID, should be something that can have single phases attached
- **p1** – an RDF property, will be a CIM attribute, should associate from a single phase back to r
- **p2** – an RDF property, will be a CIM attribute, should be the single phase instance's phase attribute

Returns concatenation of A, B, C, s1 and/or s2 based on the found individual phases

main

public static void **main**(String[] *args*)

Reads command-line input for the converter

Parameters

- **args** – will be CDPSM_to_GLM [options] input.xml output_root

Throws

- **java.io.FileNotFoundException** – if the CIM RDF input file is not found

Options:

-l={0..1} load scaling factor, defaults to 1

-t={y|n} triplex; y/n to include or ignore secondaries. Defaults to yes. Use no for debugging only, as all secondary load will be ignored.

-e={u|i} encoding; UTF-8 or ISO-8859-1. No default, so this should be specified. Choose 'u' if the CIM file came frome OpenDSS.

-f={50|60} system frequency; defaults to 60

-v={1|0.001} multiplier that converts CIM voltage to V for GridLAB-D; defaults to 1

-s={1000|1|0.001} multiplier that converts CIM p,q,s to VA for GridLAB-D; defaults to 1

-q={y|n} are unique names used? If yes, they are used as unique GridLAB-D names. If no, the CIM mrID is de-mangled to create a unique GridLAB-D name, but this option is only implemented for ACLineSegments as written to some earlier GIS profiles.

-n={schedule_name} root filename for scheduled ZIPloads (defaults to none)

-z={0..1} constant Z portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

-i={0..1} constant I portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

-p={0..1} constant P portion (defaults to 0 for CIM-defined LoadResponseCharacteristic)

Example: java CDPSM_to_GLM -l=1 -e=u -i=1 ieee8500.xml ieee8500

Assuming Jena and Commons-Math are in Java's classpath, this will produce two output files:

1. **ieee8500_base.glm** with GridLAB-D components for a constant-current model at peak load. This file includes an adjustable source voltage and manual capacitor/tap changer states. It should be invoked from a separate GridLAB-D file that sets up the clock, solver, recorders, etc. For example, these two GridLAB-D input lines set up 1.05 per-unit source voltage on a 115-kV system:
   - #define VSOURCE=69715.065 // 66395.3 * 1.05
   - #include "ieee8500_base.glm"

   If there were capacitor/tap changer controls in the CIM input file, that data was written to ieee8500_base.glm as comments, which can be recovered through manual edits.
2. **ieee8500_busxy.glm** with bus geographic coordinates, used in GridAPPS-D but not GridLAB-D

Cautions: this converter does not yet implement all variations in the CIM for unbalanced power flow.

1. AssetInfo links to WireSpacing, OverheadWireInfo, ConcentricNeutralCableInfo and TapeShieldCableInfo
2. PerLengthSequenceImpedance has not been tested
3. Capacitor power factor control mode - not in GridLAB-D
4. Capacitor user-defined control mode - not in GridLAB-D
5. Capacitor controlled by load (EnergyConsumer) - need to name loads
6. Line ratings for PerLengthImpedance
7. Dielectric constant (epsR) for cables - not in CIM
8. Soil resistivity (rho) for line impedance - not in CIM
9. Multi-winding transformers other than centertap secondary-not in GridLAB-D
10. Unbalanced transformer banks - not in GridLAB-D
11. Autotransformers have not been tested
12. schedule_name implemented for secondary loads only, primary loads to be done
13. Fuse not implemented
14. Breaker not implemented
15. Jumper not implemented
16. Disconnector not implemented

Throws

- **java.io.UnsupportedEncodingException** – if the UTF encoding flag is wrong

See also: *CDPSM_to_GLM*

10.6.2 CDPSM_to_GLM.GldNode

static class **GldNode**

Helper class to accumulate nodes and loads.

All EnergyConsumer data will be attached to node objects, then written as load objects. This preserves the input ConnectivityNode names

TODO - another option is to leave all nodes un-loaded, and attach all loads to parent nodes, closer to what OpenDSS does

Fields

bDelta

public boolean **bDelta**
will add N or D phasing, if not S

bSecondary

public boolean **bSecondary**
if bSecondary true, the member variables for phase A and B loads actually correspond to secondary phases 1 and 2. For GridLAB-D, these are written to phase AS, BS or CS, depending on the primary phase, which we find from the service transformer or triplex.

bSwing

public boolean **bSwing**
denotes the SWING bus, aka substation source bus

name

public final String **name**
root name of the node (or load), will have *nd_* prepended

nomvln

public double **nomvln**
this nominal voltage is always line-to-neutral

pa_i

public double **pa_i**
real power on phase A or s1, constant current portion

pa_p

public double **pa_p**
real power on phase A or s1, constant power portion

pa_z

public double **pa_z**
real power on phase A or s1, constant impedance portion

pb_i

public double **pb_i**
real power on phase B or s2, constant current portion

pb_p

public double **pb_p**
real power on phase B or s2, constant power portion

pb_z

public double **pb_z**
real power on phase B or s2, constant impedance portion

pc_i

public double **pc_i**
real power on phase C, constant current portion

pc_p

public double **pc_p**
real power on phase C, constant power portion

pc_z

public double **pc_z**
real power on phase C, constant impedance portion

phases

public String **phases**
ABC allowed

qa_i

public double **qa_i**
reactive power on phase A or s1, constant current portion

qa_p

public double **qa_p**
reactive power on phase A or s1, constant power portion

qa_z

public double **qa_z**
reactive power on phase A or s1, constant impedance portion

qb_i

public double **qb_i**
reactive power on phase B or s2, constant current portion

qb_p

public double **qb_p**
reactive power on phase B or s2, constant power portion

qb_z

public double **qb_z**
reactive power on phase B or s2, constant impedance portion

qc_i

public double **qc_i**
reactive power on phase C, constant current portion

qc_p

public double **qc_p**
reactive power on phase C, constant power portion

qc_z

public double **qc_z**
reactive power on phase C, constant impedance portion

Constructors

GldNode

public **GldNode** (String *name*)
constructor defaults to zero load and zero phases present

Parameters

- **name** – CIM name of the bus

Methods

AddPhases

public boolean **AddPhases** (String *phs*)
accumulates phases present

Parameters

- **phs** – phases to add, may contain ABCDSs

Returns always true

ApplyZIP

public void **ApplyZIP** (double *Z*, double *I*, double *P*)
reapportion loads according to constant power (Z/sum), constant current (I/sum) and constant power (P/sum)

Parameters

- **Z** – portion of constant-impedance load
- **I** – portion of constant-current load
- **P** – portion of constant-power load

GetPhases

public String **GetPhases** ()

Returns phasing string for GridLAB-D with appropriate D, S or N suffix

HasLoad

public boolean **HasLoad** ()

Returns true if a non-zero real or reactive load on any phase

RescaleLoad

public void **RescaleLoad** (double *scale*)
scales the load by a factor that probably came from the command line's -l option

Parameters

- **scale** – multiplying factor on all of the load components

10.6.3 CDPSM_to_GLM.SpacingCount

static class **SpacingCount**
helper class to keep track of the conductor counts for WireSpacingInfo instances

Number of Conductors is the number of phases (1..3) plus neutrals (0..1)

Constructors

SpacingCount

public **SpacingCount** (int *nconds*, int *nphases*)
construct with number of conductors and phases

Parameters

- **nconds** – number of phases plus neutrals (1..4)
- **nphases** – number of phase conductors (1..3)

Methods

getNumConductors

public int **getNumConductors** ()

Returns accessor to number of conductors

getNumPhases

public int **getNumPhases** ()

Returns accessor to number of phases

10.6.4 SPARQLcimTest

public class **SPARQLcimTest** extends Object
This class runs an example SQARQL query against CIM XML

Future versions of GridAPPS-D will rely more heavily on SPARQL queries to do the selection and filtering, as the preferred pattern for developers working with CIM. This example uses several triples to execute a query on LinearShuntCompensators (aka capacitors).

Invoke as a console-mode program

Author Tom McDermott

See also: *SPARQLcimTest.main*

Fields

baseURI

static final String **baseURI**
identifies gridlabd

nsCIM

static final String **nsCIM**
namespace for CIM; should match the CIM version used to generate the RDF

nsRDF

static final String **nsRDF**
namespace for RDF

Methods

GLD_Name

static String **GLD_Name** (String *arg*, boolean *bus*)
convert a CIM name to GridLAB-D name, replacing unallowed characters

main

public static void **main** (String[] *args*)
Reads command-line input for the converter

Parameters

- **args** – will be SPARQLcimTest [options] input.xml

Options: -e={uli} encoding; UTF-8 or ISO-8859-1; choose u if input.xml came from OpenDSS

CHAPTER 11

License

1. Battelle Memorial Institute (hereinafter Battelle) hereby grants permission to any person or entity lawfully obtaining a copy of this software and associated documentation files (hereinafter the Software) to redistribute and use the Software in source and binary forms, with or without modification. Such person or entity may use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and may permit others to do so, subject to the following conditions:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Other than as used herein, neither the name Battelle Memorial Institute or Battelle may be used in any form whatsoever without the express written consent of Battelle.

1. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL BATTELLE OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

General disclaimer for use with OSS licenses

This material was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the United States Department of Energy, nor Battelle, nor any of their employees, nor any jurisdiction or organization that has cooperated in the development of these materials, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness or any information, apparatus, product, software, or process disclosed, or represents that its use would not infringe privately owned rights.

Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Bibliography

[CIT1] W. H. Kersting, “Radial distribution test feeders,” in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, 2001, pp. 908-912 vol.2.

[CIT2] R. F. Arritt and R. C. Dugan, “The IEEE 8500-node test feeder,” in *IEEE PES T&D 2010*, 2010, pp. 1-6.

[CIT3] M. E. Baran and H. Ming-Yung, “Volt/VAr control at distribution substations,” in *IEEE Transactions on Power Systems,* vol. 14, pp. 312-318, 1999.

[CIT4] V. Borozan, M. E. Baran, and D. Novosel, “Integrated volt/VAr control in distribution systems,” in *2001 IEEE Power Engineering Society Winter Meeting. Conference Proceedings (Cat. No.01CH37194)*, 2001, pp. 1485-1490 vol.3.

[CIT5] K. P. Schneider and J. C. Fuller, “Voltage control devices on the IEEE 8500 node test feeder,” in *IEEE PES T&D 2010*, 2010, pp. 1-6.

[CIT6] I. Gorton et al., “GridOPTICS(TM) A Novel Software Framework for Integrating Power Grid Data Storage, Management and Analysis,” in *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, 2013, pp. 2167-2176.

[CIT7] S. Ciraci, J. Daily, J. Fuller, A. Fisher, L. Marinovici, and K. Agarwal, “FNCS: a framework for power system and communication networks co-simulation,” in *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, Tampa, Florida, 2014, pp. 1-8.

[CIT8] D. P. Chassin, J. C. Fuller, and N. Djilali, “GridLAB-D: An agent-based simulation framework for smart grids,” in *Journal of Applied Mathematics*, vol. 2014, no. 492320, pp. 1-12, 2014.

Index

P

Q

R

S

W