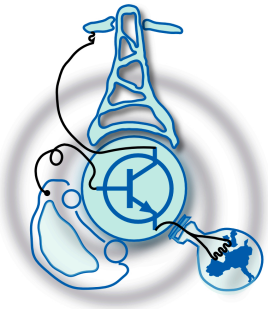
Operation System for Terminal Low Voltage Distribution Networks

by

Adrián Miranda Paz



Submitted to the Department of Electrical Engineering, Electronics, Computers and Systems
in partial fulfillment of the requirements for the degree of
Electrical Energy Conversion and Power Systems
at the
UNIVERSIDAD DE OVIEDO
July 2020

Author .....

Certified by .....
Pablo Arboleya Arboleya
Associate Professor
Thesis Supervisor

Certified by .....
Lucía Suarez Ramón
Head of Infrastructure of the Remote Management System, EDP
Thesis Supervisor

Operation System for Terminal Low Voltage Distribution Networks

by

Adrián Miranda Paz

Submitted to the Department of Electrical Engineering, Electronics, Computers and Systems
on July 22, 2020, in partial fulfillment of the
requirements for the degree of
Electrical Energy Conversion and Power Systems

Abstract

The electric distribution utilities are facing a change of paradigm in the recent years. This change is being caused by the increase of energy consumption, inclusion of new renewable technologies in the grid and the appearance of electric vehicles and storage systems in the customers end. To adapt the distribution grid to this new scenario, utilities implemented new measurement technologies that allow them to monitor the grid. However, the high data volume generated by the grid, as well as the complex structure of the grid make the classical computational techniques not appropriated for the data management of low voltage networks. In this thesis, it is developed a graph based approach for the grid's data storage and processing for the real time operation of an european type low voltage network. The proposed solution is developed for the terminal low voltage grid of EDP in Asturias. Analysis algorithms that take advantage of the redundancies given by the PLC communications infrastructure are developed and tested with a real scenario dataset.

Thesis Supervisor: Pablo Arboleya Arboleya
Title: Associate Professor

Thesis Supervisor: Lucía Suarez Ramón
Title: Head of Infrastructure of the Remote Management System, EDP

Acknowledgments

I would like to thank my supervisors Pablo Arboleya and Lucía Suárez for all your help, advices, cheerings and patience.

Thanks to all people from EDP in project Marte, for all your time and efforts. I've learned a lot from you. I should like to extend my gratitude to Pepe, for being a mate, a teacher, a supervisor and a friend.

Thanks to LEMUR people for making the lab such a nice place. I hope all of us can come back soon.

Thanks to my family, for their unconditional support. They are the reason why I am here.

Thank you Alba, for your love and support during these years. A part of this is yours. I love you.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Electric distribution system is experimenting one of the biggest revolutions in the industry. In last years, the introduction of distributed energy resources (DERs), electric vehicles (EVs) and energy storage devices (ESDs) is changing the role of costumers in the grid, turning them into load prosumers or controlled generators. This change is causing a forced evolution of the utilities paradigm, that now has to face a rapidly change on the behavior of the actors in the electric grid and the appearance of new bussiness models and new players in the market.

The electrification of the transportation is being boosted in the last years, being forecasted to represent the 58% of vehicle sales by 2040 [5]. This impact will be accelerated by the fast growing of buses and electric delivery vans, promoted by many utilities [1]. But not only the transportation, other sectors are being electrified, as the cooling systems that are being replaced by heat pumps in a big part of Europe. These are clear examples of the huge change that electrification is causing on the society, which will produce an increase of electric energy deman in following years.

In this new environment, the utilities need to face the capacity problems that appear when all these technologies are introduced in the grid. Moreover, the effective inclusion of DERs in the grid will also require from advances in technology and regulatory frameworks.

Some utilities evaluated the reinforcement of the grid as an option in order to support the big growth of power in the grid. However, this solution demonstrated to require from high investments. In the United States it is being invested about US$ 130M per utility, which implies more than US$ 1 billion[10]. This solution is not economically eficient, specially taken into account the reduction of remuneration margins driven by european regulators[1].

Some utilities propose the implementation of transactive energy models as the solution for this problem, in such a way that the prosumers and EVs can effectively participate in the electric markets [1]. This way, the system can takes advange of the new generation capability. From the distribution perspective, the customers participation on the grid will represent an opportunity to increase the flexibility of the system. This new scenario radically changes the role of utilities from being resellers of power, to be operators dealing with bidirectional power and data flows [18]. This way, the correct management of the new technologies can allow the utilities to limit the investments on the grid and to access to new markets and bussiness oportunities.

1.2 Motivation

The biggest improvement recently introduced by utilities in Spain was the installation of a smart-metering infrastructure to monitor customers. This caused an exponential increase of data coming from the grid, that requires from big data techniques to manage such a big ammount of information. Under this scenario, the so-called meter data management systems (MDMS) were implemented as the basic solution to collect and store data from smart-meters. Up to the date, these new systems were mainly used to calculate the customers bill.

However, these new measurement systems offered the utilities new possibilities in order to improve the operation and planning works of the grid. Taking advantage of these capabilities geographical information systems (GIS), and supervisory control and data acquisition (SCADA) were implemented in order to put in context the monitored information. These systems allows the operator the identification of problems

and the planning of operation works. Nevertheless, in the near future, utilities will need to understand, control, and monitor all the information comming from the grid in real time, which is not possible with the current technology.

The smart metering infrastructure in Spain is based on Power Line Communications (PLC) technology, that uses the power lines as the physical channel to transmit the communication signals. The main drawback of this technology is its low bandwidth to transmit information, so the sampling time of measurements are limited. This limits the capability of performing real time monitoring on the grid, but also limited the needed computing capability required by the system. However, in the future, it is forecasted to have new monitoring techniques that allow a higher resolution of measurements in the power system. Moreover, the specific physical channel characteristics make necessary a specific management systems adapted to this type of communications in order to take the maximum advantage of the infrastructure.

Newer approaches propose the implementation of an Energy Management System (EMS), capable of managing forecasted and scheduled information of the grid in real time, as well as to perform online state estimation, power flow or contingency analysis algorithms. However, with the future reduction of sampling time and communication rates, the computing infractructure currently used is not prepared to perform the needed analysis. So new approaches for data technologies should be proposed and tested in order to achieve a future real time operation of low voltage distribution networks.

1.3 Objectives of the project

The main purpose of this thesis is the development of a system that allows the real-time operation and monitoring of low voltage distribution grids. This system should be capable of manage the complexity of the grid in the most possible efficient way in order to achieve the future needs of the real time operation of distribution networks. It will be analyzed for the specific case of european type distribution networks using PLC technologies as communication system. This thesis will be focused in the analysis

of the different techniques available to structure and store the grid information. It will be selected the best technology from the point of view of technical requirements for storage, loading and extraction processes from a data management system. The proposed method will be developed for the practical testing with the distribution grid of EDP in Asturias.

1.4 Thesis structure

This thesis is organized in six chapters:

- **Chapter 1**: introduces the topic and motivation for the present project.
- **Chapter 2**: analyzes the input data used for the proposed solution.
- **Chapter 3**: describes the background theory that will define the criteria for the technology selection.
- **Chapter 4**: the design of the data model is explained and developed.
- **Chapter 5**: describes the use of the selected tool and the final implementation of the solution.
- **Chapter 6**: describes the conclusions extracted from the present work, and future works on the topic.

Chapter 2

Input Data

The system presented in this thesis will be focused on the european low voltage distribution grids. In this chapter, the structure of these type of distribution networks will be explained, with all the elements that form the it and their role on the operation of the grid. It will be explained how each of the devices is desfined and which information is generated by them.

In the Figure 2-1 all elements taken into consideration in the distribution grid are shown, as well as its expected configuration. Among the power elements that can be found there are transformers, segments, nodes, fuses, low voltage switch-boards, breakers, switching boxes, connection points, customers and transformer stations. Besides, some communication devices as transformer station supervisors, Power Line Communications (PLC) concentrators, feeders' supervisors, routers or the smart meters.

2.1 Nodes

The nodes are defined as points where two or more segments connect one of their terminals. They will only be defined by their MSLINK (ID in the GIS system).

These elements are generally defined for any electrical connection, independently of its nature. However, as it can be noticed in Figure 2-1, and it will be lately explained in the Chapter 4, they are generally defined for the conducting part of the

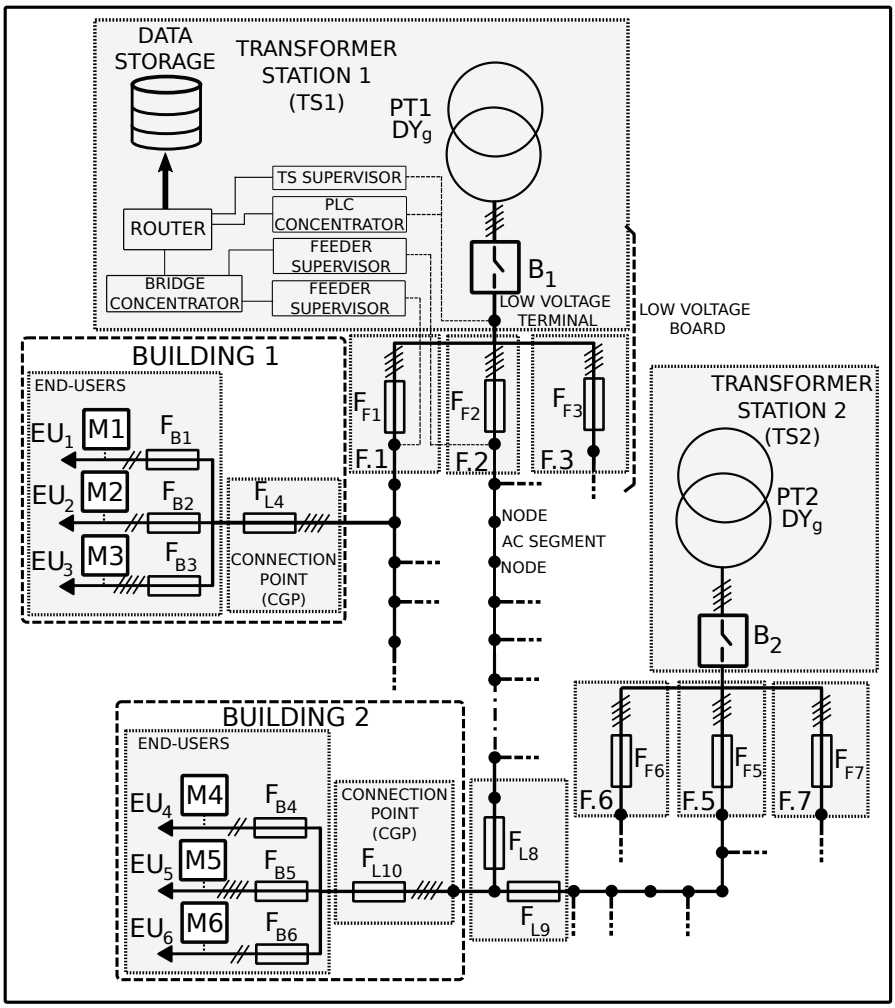


Figure 2-1: General configuration of the low voltage grid in Asturias.

network (the conductors).

2.2 Transformer station

This element plays the role of a power source to the low voltage part of the system. It must be highlighted that event though it is the central element to indicate the power source of the grid, it is actually an abstraction that groups a set of different elements. The transformer station is an *aggregation element* (lately explained in Section 4.2.3) that groups a set of power elements and communications and measurement systems

(see TS1 in Figure 2-1). Therefore, this instance only provides information about the greographical position and general setup of the station.

Three main setups are used for these stations can be outdoor, shed or private. In this last case, the configuration and characteristics of the downstream network is not known. Therefore, these elements must be removed from the low voltage operator scope, and they will be modeled as medium voltage loads instead of low voltage consumptions.

The stations will be defined with the following attributes:

- MSLINK (*unsigned integer*): it is the unique key to identify the element in the GIS system.

- BDI key(*string*): it is used by operator to identify the transformer station. Stands for *Base de Datos de Instalaciones* (Installations Database). It is formed by letters and numbers, and it also provides information about the configuration of the station. If it starts with "I" the station is an outdoor transformer, if it starts with "C" it is an indoor type station, and if it starts with "CTP" it is a private transformer.

- Municipality (*string*): name of municipality where the station is placed on.

- Description (*string*): name of the station and sometimes also the street and direction.

- Coordinates(X,Y) (*tuple(float,float)*): the geographical position in Universar Transverse Mercator (UTM) coordinate system for zone 30.

- State (*string*): indicates if the station is in service or out of service.

- Type (*string*): contains the information about the specific type of configuration of the transformer station. It can be INTEMPERIE (outdoor), EDIFICIO (building), CASETA (shed) or PREFABRICADO (premanufactured).

- Ownership (*string*): defines if the station belongs to the utility or to a private owner. This parameter is redundant since it can also be determined by the initial characters of the BDI key.

2.3 General breaker

This breaker is placed at the transformer output in order to isolate the output feeders from the medium voltage grid (B_1 and B_2 in Figure 2-1). It is used when some mainteinance is taking place at the transformer and it is out of service. In this case, isolating it, the feeders can be fed from a different transformer reconfiguring the feeders general fuses.

Taking as example the grid in Figure 2-1, both transformer stations are connected through F_{L8} and F_{L9}, placed in a fuses box. In normal operation case, when both transformers are on service, some of the fuses of the box will be open. But if the PT1 is out of service, and B_1 is open, the feeder F.1, F.2 and F.3 can be fed closing F_{L8} and F_{L9}. This way PT2 will feed temporarily the lines of TS1.

The needed characteristics to be define about the general breaker are:

- MSLINK(*unsigned int*) that in this case is used also as unique key in the database systems

- Status(*string*) to indicate the state of the breaker.

- Slot (*unsigned integer*): it is used to identify which transforme within the transformer station is connected to (see Figure 2-2).

Despite this element count with more electrical characteristics, since they will not be relevant for the real time operation, they will not be reflected in the data storage system.

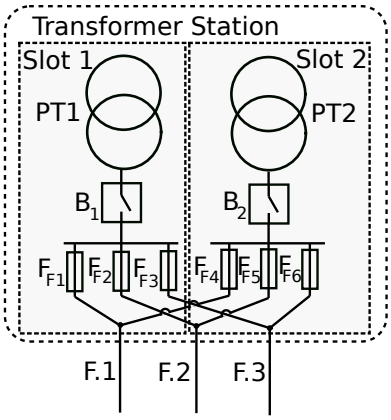


Figure 2-2: Power circuit of a transformer station with two slots and three feeders.

2.4 Transformer

It is a two windings transformer that connects the medium voltage to the low voltage network. The characteristics of all of them are similar, being all of them Dyg11 transformers (see Figure 2-1), with an output voltage of 400V. The main characteristics needed to identify and model a transformer are:

- Reference (*string*): will define the identifier of the transformer inthe database.
- Rated power (*unsigned integer*)
- Slot (*unsigned integer*): is is used to distinguish between transformers in the same transformer station when there are several in the same station (see Figure 2-2).

It is very common that the transformers have on-load tap changers (OLTC). This tap is operated manually and no information is received or registered about the tap position in the telecontrol system of the utility.

2.5 Low voltage switch-board

This is the panel that contains the breakers to connect/disconnect the feeders to the low voltage winding of one transformer. So the main target of this element is to distribute the power between the different feeders and to isolate them if it is needed.

The main identification characteristics are:

- MSLINK(*unsigned int*) to identify the element in the GIS system
- BDI key(*string*) as the main key in the internal system database
- Slot (*unsigned in*): this parameter is used to identify the different connections to the medium voltage system. It is the parameter that relates the low voltage board with the transformer (see Figure 2-2).
- State(*string*) to define if the output terminals are energized or not.

2.6 PLC Concentrator

It is the communication device that receives the information from smart meters of the customers. It is connected to the output of low voltage switch-board, and thus, it is the master device of all the meters fed within a subnetwork (network of one transformer station). As it is shown in Figure 2-1, the concentrator is connected to a four wires conductor. The concentrator receives information through any of the three phases, and can also inject by all of them at the same time. Some brands also offer the *selectable single phase* communication, where the concentrator can inject a signal to one of the phases[2]. In the distribution grid in Asturias, the main manufacturers of PLC concentrators are ZIV and Circutor [13].

The PLC communication protocol used for the transmission of the data is based on the PRIME system, mainly developed by Iberdrola. This protocol is right now the most used protocol in Spain for the deployment of the smart-meters. The communication protocol is the so-called STG-DC, specifically developed to fulfill the requirements from the Spanish gobernment.

A PRIME network is structured as shown in Figure 2-3. The concentrator acts as the Base Node, and the meters can act as Terminal or Switch. The Terminal communicates the measurements with another device, while the Switch communicates its measurements plus a set of received messages from other devices. This way, the network can be structured in depth layers.

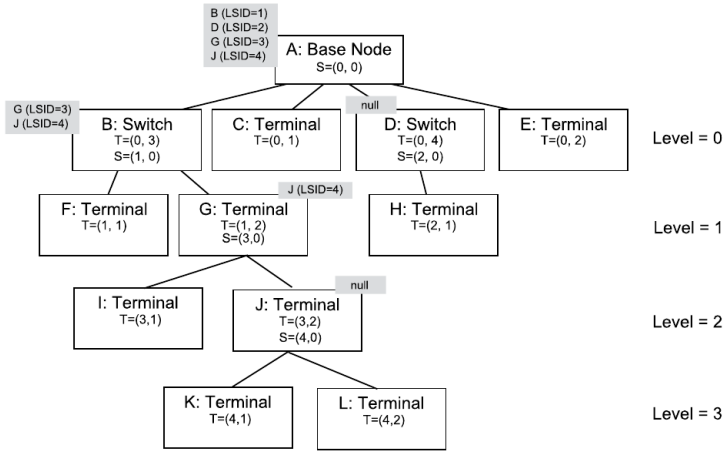


Figure 2-3: Example of a switching schema of a PRIME network [20].

The STG-DC is a protocol where the operator communicates only with the concentrator, and this device will be in charge of manage all the devices (smart-meters) connected to its sub-network. The STG is based on XML messages, and there are several standarized reports that the system can generate in order to transmit the required data.

These reports can be of three main types[13]:

- Asynchronous: the telegestion center requires the report, and the concentrator will require the information to the meters. When all the information is collected, the response is sent to the center.

- Synchronous: in this case the response is directly sended by the concentrator without any request to the smart meters.

- Events: they are espontaneous. The communication is started by the concentrator.

Base Node Information (S11) (Asynchronous): this report is generated by the concentrator.

- Prof.: defines the "depth" of the device within the network (Level in Figure 2-3).

- Switch Identifier (SID): defines the id of the device in the PRIME network (first value of parameter S in Figure 2-3).
- Local Node Identifier (LNID): local id of the device (second value of parameter T in Figure 2-3).
- State: it can be "Swith" or "Terminal" according to the role of the device in the PRIME network.
- Local Switch Identifier (SSID): defines the id of the swith device through which the meter is communicating with the base node (second value of T in Figure 2-3).
- Concentrator reference: reference of the base node device.

Smart Meters Communications Daily Statistics (G02) (synchronous): this report is generated by the concentrators in the PRIME network and it describes the communicacion network of each concentrator. It relates each concentrator with all the meters it is communicating and defines the characteristics of te sensed communication.

- Concentrator (CNC): Reference of the concentrator.
- Meter (CNT): Reference of the smart-meter.
- Date (FH): Date (by day) when each register was generated.
- Active Time Percentage (Atimeperc): defines what was the active time that smart meter was communicating with the concentrator. This is normally relatively high (above 70%), if this is not this way it could be caused by two main reasons:
  - The communication quality is poor, which imply that the information provided by the meter is not reliable.
  - The meter was disconnected.
  - Some operation in the network changed the concentrator to which the meter was reporting.

Many other STG-DC reports are available to request information about the devices in the network, possible intrusive devices, communications quality... Check the full list of reports in A.1.

2.7 Bridge concentrator

It works as the concentrator of the signals coming from the feeder supervisors. It is a communication device to transmit the measurements to the router of the station.

2.8 Router

It is the communication element that transmits the information from the bridge and PLC concentrators to telecontrol center. The connection can be 3G or with optical fiber and it is supported by telecommunications companies.

2.9 Feeder Supervisor

At the output of the switch-board there are current and voltage probes in order to measure the power to the feeder. This readings are normally used to compare with the aggregation of the meters. This way, these elements allow us to estimate the losses in the grid.

2.10 Feeder

Every transformer feeds a set of outgoing lines. There is a connection element between the low voltage switch-board and the line which main target is the thermal protection of the line in case of any fault(see F_{Fx} in Fig. 2-1). So this element is formed normally by a fuse element. However, the importance of these elements goes beyond the protection of the lines, being them used to operate the grid. Moreover, they

work as manual circuit breakers as well as signal elements for the maintenance and operation personnel.

These elements can be defined by:

- MSLINK (*unsigned integer*): unique identification number of the device within the database.

- MSLINK_CTM (*unsigned int*): unique identification number in the database of the transformer station the feeder belongs to.

- Real status (*string*): defines if the element is closed or open. It can be "C" (closed) or "A" (open).

- Common status (*string*): defines the normal status of the feeder. If common and real status don't match, it means that any abnormal case is taking place, and it required a reconfiguration of this feeder.

- Voltage (*string*): it can be B1 or B2. The B1 lines have 135V phase to neutral voltage. They are normally old lines that still have these lower voltage technology. In these cases, the loads are connected between two phases. The B2 lines are the most common ones, being 400V phase to phase voltage and 230 phase to neutral voltage.

- Line number (*unsigned integer*): identifier used to the numbering of the lines outgoing from the same transformer.

- Status (*string*): normally the low voltage switch-boards have standard number of outputs (normally 8). So it is very common that not all of them are needed, so some of the feeders will not be used, being their status "Out of service". In the rest of cases, the state will be "On service".

- Coordinates(X,Y) (*tuple(float,float)*): approximate geographical position of the feeder only to representation purposes in UTM 30 system.

2.11 Breakers

The breakers are the elements that allow the operation and reconfiguration of the grid. There are two main types of breakers: in-line breakers and connection point breakers.

In the case of in-line breakers they are placed in between two AC segments, and its main target is the reconfiguration of the grid. These type of elements can change their state and normally are grouped in "fuse boxes" (see F_{L8} and F_{L9} in Fig. 2-1).

The characteristics that define these elements are:

- MSLINK (*unsigned integer*) as unique ID in the database.
- Real Status (*string*): it defines if the breaker is open or closed.
- Common Status (*string*): describes the planned status of the breaker. I allows the operator to distinguish between temporal status of the breaker (comparing this value with the Real Status).
- Box (*unsigned integer*): defines the code of switching box where the breaker is placed (see F_{L8} and F_{L9} in Fig. 2-1).
- Type (*string*): defines the purpose of the device. It can be *Maniobra*, *PL*, *Acoplamiento* or *Unknown*.

2.12 AC Segments

The major part of the grid will be formed by lines. These elements are formed by the conductors that connect the different elements that form the grid. In the spanish low voltage systems they have a dual role: transmission of power and communication signals.

At the time of defining them it is important to split their information in two natures: the connectivity information and the geographical information.

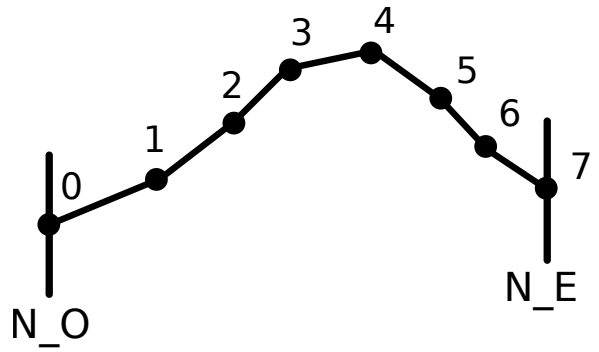


Figure 2-4: Example of a segment definition between arbitraries buses N_O and N_E with eight geogrhaphical vertices.

The connectivity information is the essential information that defines the topology of the grid. To do this is only necessary the specification of the two terminals where the segment is connected, the conductor characteristics and the length.

On the other hand, the geographical information is formed by the coordinates of the vertices that define the segment's path. This second set splits each segment in multiple straight segments in order to create an accurate graphical representation of the shape of the lines over a map (see Figure 2-4).

It is important to keep both datasets clearly separated, since the geographical information includes a higher number of topological vertices. The use of these vertices for the analysis (topological or electrical) will imply a huge increase in the order of processing time of the algorithms due to the high number of additional elements.

The main attributes used to define the connectivity information are:

- MSLINK (*unsigned int*): this element is used to link the GIS system with the rest of databases and also to relate the connectivity and geographical information.

- Voltage level (*string*): it can be B2, B1 or unknown.

- Length (*float*): it defines the length of the segment in meters.

- UNE (*string*): contains the name of the cable type in UNE (Spanish norm). From this attribute the electrical parameters of the conductor can be deter-

mined. It is also specified in this field the type of installation (underground or aerial).

- N_O (*unsigned int*): defines the unique identifier of the origin node of the segment.
- N_E (*unsigned int*): defines the unique identifier of the ending node of the segment.
- State (*string*): indicates if the segment is in service or out of service. The segments, since they are not operation devices, they will be "out of service" only in case of any maintenace works.

For the definition of the geographical information, a sort of additional vertices are defined:

- MSLINK (*unsigned int*): this attribute is used to link the geographical information with the connectivity information and with the GIS system.
- Vertex order (*unsigned int*): this parameter contains a number used to sort the "subsegments" in the correct way (see Figure 2-4).
- X (*float*): defines the longitude of the vertex in UTM30 system.
- Y (*float*): defines the latitude of the vertex in UTM30 system.
- Type (*string*): defines if the segment is undergrounded or if it is aerial. Normally all the "subsegments" within the same segment are the same type.

Each conductor may have different characteristics since the manufacturer and conditions of each of them are different. So depending on the section and the type (underground or aerial) the resistance and inductance can be determined.

2.13 Connection points

These elements define the last protection element that the company install between the costumer and the grid. It is physically formed by a fuses box. To this element

it can be connected one or more costumers (e.g. in the case of a building), see F_{L4} and F_{L10} in the Figure. 2-1. In the database they are defined by CGP acronym in Spanish (*Caja General de Protección*), and their main properties are:

- MSLINK (*string*)
- BDI key (*string*)
- Type (*string*): it follows the UNE naming and it defines the type of element was used to perform the connection: sigle/three phase, base voltage, maximum current, type of box...

2.14 Meters

They are the devices in charge of measure the consumption of the final customers as well as other electrical parameters. The smart meters that are currently installed have the capability of disconnecting the installation from the grid in case of fault detection. Moreover, they also include the needed systems to send the readings to the concentrators (normally placed at the transformer station). So in this case it is a device that comprises communication, measurement, protection and conducting equipment.

The consumptions from the client side are transmitted using a STG-CT report:

Daily load profile (S02): this report contains the active and reactive components with a samplig time of one our in most of cases. This sampling time can be decreased until the range of minutes, but normally not less than 15 minutes. The fields of the package are:

- Datetime: sampling timestamp.
- Incoming Active Power (ACTIVA_E)
- Outgoing Active power (ACTIVA_S)
- Reactive 1, 2, 3 and 4: the reactive power absorved by quadrant.

The active and reactive power are defined by quadrants according to a load convenium:

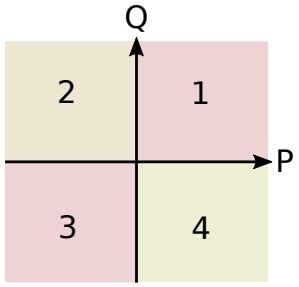


Figure 2-5: Definition of quadrants in the S02 report.

In such a way that reactive 1 and 2 are absorved (inductive) when active power is absorved or generated respectively. In the same manner, reactives 3 and 4 represent the generated reactive cases (capacitive).

The STG-DC protocol also allows the generation of other reports about statistics of communications, data about the device model, firmware updates... The reports of STG-DC are shown in A.1.

Chapter 3

Design of Data Management System

In this chapter it is explained the method used for the modelling of a general electrical network, specifying some characteristics of the distribution grid. For that, the basics of graph theory are rapidly explained, exposing the main types of graphs and their characteristics. Finally, the methods used in computer science to work with these type of structures are explained and compared, showing their main strengths and weaknesses.

3.1 Data generated by power systems

As it was exposed in Chapter 2, the data associated to different elements can be of different nature. Basically, in this project, the information will be divided in three natures: static, dynamic and spontaneous.

1.- The static data is formed by the definition of the elements that don't suffer any change along time. This kind of data is generally associated with the topology definition of the grid, as well as the characteristics of the elements that form the network.

2.- The dynamic data is generally associated to a timestamp, this is, the data that

needs to be periodically updated. Normally this data is associated with the consumptions, electrical variables evolution or communications status.

3.- The events that are related with a change of anything in the grid at a determined instant. The main difference with the so-called dynamic data is that they are not expected to be generated periodically and generally they are updated with very low frequency.

In the actual grid, all the information is actually not static at all since all the elements in the network are subjected to suffer any change. A clear example of this is the substitution of the equipment on the client side or the repowering of segments. So a better definition of the static data is the data that changes with very low frequency (typically in the order of years).

There is also some elements that change with low frequency, but not as fast as the dynamic data. These elements are the operation elements: generally asociated to the switching devices. The majority of these elements are operated manually, so the capability of utilities to operate them is limited. Tipically the operation frequency is in the range of weeks or months. These kind of changes in the network imply important differences, so they must be taken into account. Moreover, the time window between events is not wide enough to ignore their evolution along time (consider them part of the static data).

3.2 Topology modelling in power systems

This project will be focused in the management system used to store and extract the needed information about the topology of the distribution grid. This implies the design of an efficient storage system for static data and events.

The power system can be described as a set of assets that are electrically interconnected. A simple representation of these systems was proposed by Gustav Kirchoff in 1847 using graphs[14].

A graph can be defined as an ordered pair of two elements:

$$G = (V, E) \tag{3.1}$$

where:

V is a finite set of elements (called vertices or nodes)

E is a finite set of 2 subsets of V (known as edges or arcs)

The vertices in a graph are normally associated with some kind of information, being the main element that stores the details of the modelled system. On the other hand, the main functionality of the edges is to set the connections and relationships between nodes.

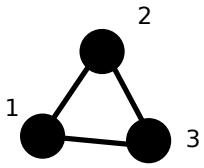


Figure 3-1: Simple graph with three nodes and three edges.

In Figure 3-1 it is represented a simple graph. This representation corresponds to the graph defined by:

$$V = \{1, 2, 3\} \tag{3.2}$$

$$E = \{\{1, 2\}, \{2, 3\}, \{3, 1\}\} \tag{3.3}$$

The manner of defining how large is a graph, is by its size and order. The order is defined by the number of vertices, while the size is defined as the number of edges. Hence, the graph defined on 3.2 and 3.3 is of size 3 and order 3.

3.2.1 Multidimensional graph model

The last definition is enough when all the vertices included in the graph are from the same nature. From a storage system perspective this means that they will be defined

by the same tags or attributes. As it is shown in Figure 3-2a, even the most simple electric power system is formed by very different elements. This requires the creation of some different type of vertices or dimensions. For the case of Figure 3-2b, it was required to use 4 different type of elements.

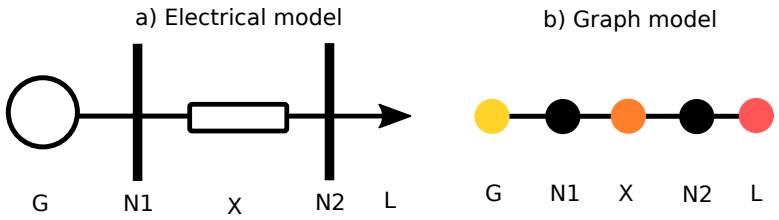


Figure 3-2: Representation of a simple electrical system using a multidimensional graph model.[Yellow:generator, black:bus, orange:line, magenta:load].

In this case, the mathematical definition must be changed in order to define the different subsets of nodes in the graph. The graph in Figure 3-2 can be expressed as:

$$V = \{Generator, Bus, Line, Load\} \quad / \quad Generator = \{G\}, Bus = \{N1, N2\}, \\ , Line = \{X\}, Load = \{L\} \quad (3.4)$$

$$E = \{\{G, N1\}\{N1, X\}\{X, N2\}, \{N2, L\}\} \quad (3.5)$$

Notice that the edges connect nodes across all dimensions in the graph.

3.2.2 Weighted graph models

Until this point, the only functionality defined for the edges was to connect the vertices in the graph. These type of graphs are known as unweighted graphs. In some applications, the graphs are used to study the possible delivery paths (routes of edges to follow within a graph) between two vertices. In these cases, the use of weighted graphs is the most convenient solution.

In the weighted graphs solution, the edges have a weight assigned in order to model the cost of using each edge. In power systems, this model is also convenient

for the representation of the conducting networks. For this case, the cost can be modelled by the impedance of the conductor. The model explained in Figure 3-2 can be converted in the one in Figure 3-3. As it can be seen, this allows a more direct model of the delivery cost of the electrical energy, as well as we can reduce the number of dimensions of the graph from 4 to 3.

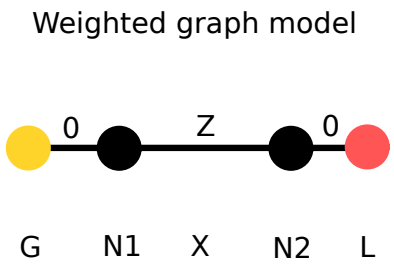


Figure 3-3: Representation of a simple electrical system using a weighted graph model.

For this case, the graph definition can be rewritten as:

$$V = \{Generator, Bus, Load\} \quad / \quad Generator = \{G\}, Bus = \{N1, N2\}, \\ , Load = \{L\} \quad (3.6)$$

$$E = (\{G, N1\}\{N1, N2\}, \{N2, L\}) \quad (3.7)$$

and it has to be added the weights vector:

$$W = (0, X, 0) \quad (3.8)$$

Notice that in this case the order of edges is not mutable in order to assign weights. Moreover, the graph is more compact since the order was reduced from 5 to 4, and the size from 4 to 3.

3.2.3 Directed graph models

In some cases, it is needed to model a single direction connection. In power systems this is the case when we model a pure generator or a pure load, the load only admits incoming power, while the generator only admits outgoing power. Furthermore, it is common the modelling of conductors as directed graphs in order to define the sign convention. In such manner, if power flows in the defined direction, its sign is possitive, while if the power flows in the opposite direction, it will be negative. Nevertheless this characteristic must not be defined from the data storage system, but from the mathematical solver.

The directinal characteristic can be added to the graph, obtaining the so called directed graphs. The previously exposed example is expressed in Figure 3-4 with a directed graph. Notice that an undirected connection can be expressed as two directed edges with the same weight.

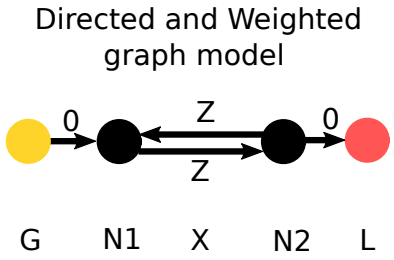


Figure 3-4: Representation of a simple electrical system using a directed and weighted graph model.

$$V = \{Generator, Bus, Load\} \quad / \quad Generator = \{G\}, Bus = \{N1, N2\}, \\ , Load = \{L\} \quad (3.9)$$

$$E = ((G, N1), (N1, N2), (N2, N1), (N2, L)) \quad (3.10)$$

As expressed in equation (3.10), the sets of vertices that define the edges are not permutable for the directed case. Moreover, the weights should be expressed in a

matricial form since two possible directions can be defined for each edge. Notice that this matrix is also possible to be defined in the undirected case, and the matrix will be symmetrical.

$$W = \begin{array}{c} \\ G \\ N1 \\ N2 \\ L \end{array} \begin{array}{c} \begin{array}{cccc} G & N1 & N2 & L \end{array} \\ \begin{pmatrix} 0 & 0 & \infty & \infty \\ \infty & 0 & Z & \infty \\ \infty & Z & 0 & 0 \\ \infty & \infty & \infty & 0 \end{pmatrix} \end{array}$$

If the graph of Figure 3-4 is modified in such a way that only the conducting elements are represented, this is, removing the generator and the load, the weights matrix can be reformulated as equation (3.11). It can be noticed that this result in the impedance matrix of the circuit.

$$W = \begin{bmatrix} 0 & Z \\ Z & 0 \end{bmatrix} \tag{3.11}$$

3.2.4 Graph families

The graph families are sets of graphs that contain some specific properties. Some of the most important are listed below:

1.- Bipartite graph: is a graph whose vertex set can be partitioned in two subsets V_1 and V_2 such that every edge $\{u, v\}/u \in V_1 \wedge v \in V_2$. It is generally named as $K_{n,m}$, being n the vertices number in V_1 and m the vertices number in V_2.

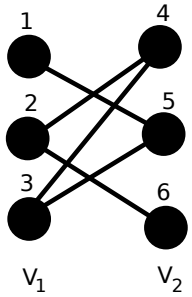


Figure 3-5: Example of a bipartite graph $K_{3,3}$.

2.- Star graph: it is a special subtype of the bipartite graph. In this case one of the vertices subsets only have one vertex.

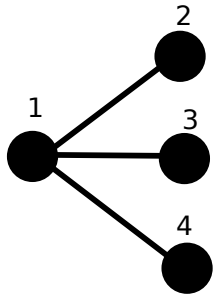


Figure 3-6: Example of a star graph $K_{1,3}$.

3.- Path graph: it is a graph where the vertices can be arranged in a given sequence. It is generally named as P_n, being n the number of vertices of the graph.

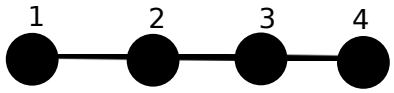


Figure 3-7: Example of a path graph P_4.

$$V = (V_1, V_2, V_3, ..., V_n) \tag{3.12}$$

$$E = \{\{V_i, V_{i+1}\} \quad / \quad i = 1, ..., n-1\} \tag{3.13}$$

4.- Cycle graph: It is a graph that can be arranged in a cyclic sequence.

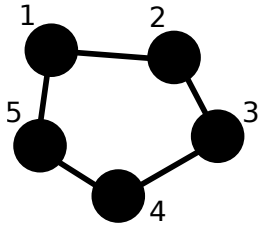


Figure 3-8: Example of a cycle graph C_5.

$$V = (V_1, V_2, V_3, ..., V_n) \tag{3.14}$$

$$E = \{\{V_i, V_{i+1}\} \quad / \quad i = 1, ..., n-1\} \cup \{V_1, V_n\} \tag{3.15}$$

5.- Trees: it is generally defined as a graph where there exist one single path between any two nodes in the graph.

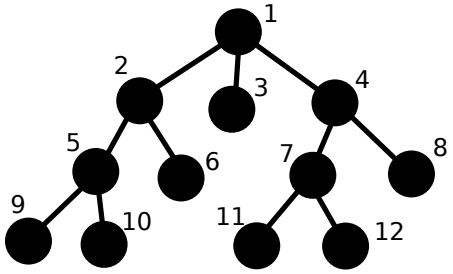


Figure 3-9: Example of a tree.

An arbitrary graph can be composed by a set of different graphs that could belong to different families. A decomposition of a graph G is a family Γ of edge-disjoint subgraphs of G such that:

$$\bigcup_{F \in \Gamma} E(F) = E(G) \tag{3.16}$$

Following this definition, it is possible to set that a tree can be decomposed in a series of star graphs. Moreover, it is also true that a tree mustn't contain a cycle.

An important characteristic of these type of structures is that it is possible to arrange these star graphs in hierarchical structures as shown in Figure 3-9. There are several posibilities depending on which vertex is defined as the root of tree. For a given tree with a defined root is possible to define for a vertex its "parent" and "child" nodes. Being the parents the closest neighbors to the root vertex, and the children vertices the furthest from the root.

In distribution systems, the network is operated in such a way that the topology of the grid is tried to keep radial. This means that the structure of the grid can be modelled as a tree. Because of this, the most important processes to run in the data system will be searching algorithms along tree structures.

3.2.5 Graph exploration algorithms in tree structures

Given a graph and a structure to work with it, it is needed to define the algorithms to explore the graph. Since the most computing intensive part of our system will

defined by tree graphs, the most convenient algorithms are the hierarchical traversal algorithms. These algorithms exploit the above mentioned hierarchical characteristic of trees in order to explore them. Moreover, their target is to explore the graph in such a way that each vertex is only analyzed once. There are two main options: Breadth-First Search (BFS) and Depth-First Search (DFS).

Breadth-First Search Algorithm

The BFS is one of the most used algorithms in graph exploration. Because of this, this is one of the common algorithms used by benchmarks to measure the performance of computers with this type of data. These algorithms are used also as kernel for other more complex algorithms as connected components analysis or centrality analysis[21].

The fundamental idea behind the algorithm is to perform the exploration in a "layered" way. This is, starting from a vertex set, neighbors of the vertices are analyzed. This way, the exploration is permormed by the analysis of layers of vertices in the same "depth" from the initial node. This allows the possibility of parallel execution[21], but the sequential version can also be implemented as shown in Algorithm 1. This variant is also called Level Order Traversal.

Algorithm 1 Top-Down Breadth-First Search Algorithm

Input: origin vertex ($\mathbf{ID}_{init}$), Graph

```
1: Initialize frontier as the vertex set to analyze.
2: Add ID_init to frontier.
3: Initialize list of already analyzed vertices: parents.
4: Add frontier to parents.
5: while frontier is not empty do
6:     for n in frontier do
7:         Remove vertex n from frontier
8:         Match neighbors of n
9:         for neighbor in n do
10:            if neighbor was not already seen then
11:                Add neighbor to frontier
12:            end if
13:        end for
14:    end for
15: end while
```

Depth-First Search Algorithm

The most common and simple variant of DFS is the Preorder Traversal Algorithm. In this case, the visiting order gives preference to reach the deepest vertex of each subtree. In this case, an order of visit is conditioned by the discovery order of neighbors of a vertex. This algorithms is shown in Algorithm 2.

Algorithm 2 Depth-First Search Algorithm

Input: origin vertex ($\mathbf{ID}_{init}$), Graph

```
1: Initialize n as the vertex defined by ID_init.
2: Initialize list of already analyzed vertices: parents.
3: Add n to parents.
4: Initialize neighbors to the set of neighbors of ID_init
5: while neighbors is not empty do
6:     n equals to the first element of neighbors
7:     Remove n from neighbors set
8:     Store neighbors of n in neighbors_temp
9:     for neighbor in neighbors_temp do
10:        if neighbor was not already seen then
11:            Add neighbor to first element of neighbors
12:        end if
13:    end for
14: end while
```

As it can be noticed, this algorithm requires from the direct access from the id (see line 6 of Algorithm 2). Besides, the storage sequence of the neighbors will impact the performance of the algorithm.

3.3 Graphs and Database Management Systems

In electrical utilities, the databases have to store these graph-based structures using database systems. A database is a more complex system than a graph since not only the structure is needed to be stored, a sort of attributes must be stored in order to correctly characterize the elements in the system. This way, when a query is executed, the engine of the database should find the requested information according some rules (the structure of the graph).

Big O Notation

The Big O notation is a simplified manner of express how the processing time of an algorithm increases as the size of input data grows [19]. In software engineering is a common tool to compare the efficiency of different algorithms. It is important to highlight that this notation doesn't express the exact time of computation that an algorithm will require: to calculate this time, it must be taken into account the machine used to run it, its conditions, the initialization times of different modules... The Big O notation will define the trend of the function with large ammounts of input data.

This way it can be defined the most common operation orders as:

- $O(1)$ is used to represent algorithms where a fixed number of operations are performed. This means, almost **constant time** of execution.

- $O(logn)$ is used to represent an algorithm that grows in a **logarithmic** way. It is very common when optimizing searches using binary trees, and the computation time with large inputs is near to constant.

- $O(n)$ is the **linear** growing algorithms. It generally appears with "brute force" operations where all elements (n elements) of a set are processed sequentially.

- $O(n^2)$ represents the **quadratic** algorithms. They are generally associated with two nested iterative structures that perform "brute force" operations.

- $O(e^n)$ represents the **exponential** algorithms. It generally represents the worst case, and its behavior with big inputs is very slow.

In Figure 3-10 it can be noticed that a change btween two of the defined orders can lead to a huge improvement in terms of execution time. Moreover, it explains why the exponential order algorithms will be avoided in big data systems.

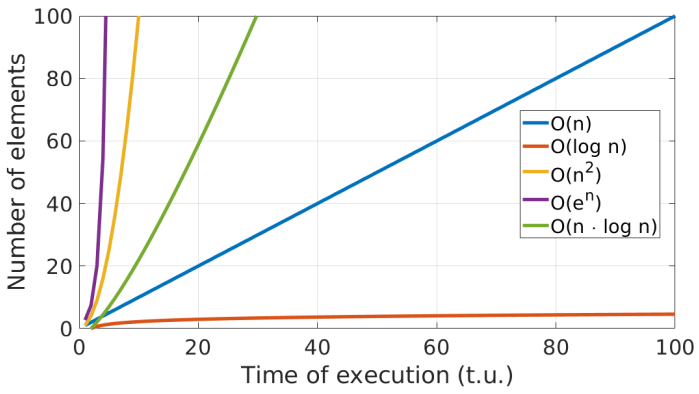


Figure 3-10: Evolution of exetucion time (in time units) with different order algorithms.

3.3.1 Classical graph storage in Relational Database

The classical database systems are the so-called Relational Database Management Systems (RDBMS) or SQL (Structured Query Language) Databases. The data is organized in tables and relationships between tables.

According to classical RDBMS, in order to guarantee certain principles, the redundancy of data should be avoided. This way, some characteristics as atomicity or consistency are ensured. The atomicity is the principle that avoid to treat the transactions as single units, if some data is repeated in different tables this cannot be easily implemented. The consistency is the principle that ensures that there will not be different values for the same instance, so repetition of values should also be avoided. The common way of define a graph ensuring these characteristis is using two type of tables: tables of vertices and tables of edges.

An example of this is shown in Tables 3.2 and 3.1 for a unidimensional graph of order n and size m.

Table 3.1: Example of vertex definition table in RDBMS

Node ID	**Attributes**
1	Attrs. Node 1
2	Attrs. Node 2
...	...
n	Attrs. Node n

Table 3.2: Example of edge definition table in RDBMS

Edge ID	**From**	**To**
1	1	2
2	1	3
3	1	4
...	...	...
m	n-k	n

Notice that in order to add more dimensions to the graph example of tables 3.2 and 3.1, it will only be needed to add more types of edge tables. Moreover, it is also possible to define different types of relationships using different types of edge tables.

In RDBMS the tables are implemented as hashing tables. This means that ID used in the tables is directly related with the memory direction. So the access with the ID can be asumed a constant time operation. As reference, it can be taken the basic iterative part of the BFS algorithm (lines 6 to 14 of Algorithm 1), that can be sumarized in the following steps:

1.- Access to the selected vertex

2.- Look for edges connected to the selected vertex

As the tables are modelled as hashing tables, this first step will be constant time order. However, the second step is a filtering process by an attribute of the edges. As the order of the segments doesn't follow any rule, the entire list of attributes should be read in order to find the desired rows (edges). This process is repeated as many times as the number of vertices. So the order of this exploration will be:

$$O(BFS)_{SQL} = O(m \cdot n) \tag{3.17}$$

If we approximate the graph as a big tree:

$$m = n - 1 \tag{3.18}$$

So the order of a BFS in a RDBMS will be cuadratic respect to the order of the graph:

$$O(BFS)_{SQL} = O(n \cdot (n-1)) = O(n^2) \tag{3.19}$$

In distribution systems, an electrical node is commonly a simple terminal where two or more segments are connected. This means that no additional information is related to them appart from the connectivity inherited by the relationships stablished in the tables. So the Table 3.1 can be added or not to the database.

Under the assumption that the network is formed by a single tree structured graph, it is possible to improve the performance of the algorithm under the assumption that the graph is static. This implies that no changes can be performed on the connections between nodes (no switching operations in the grid). To set the "from" vertex as an ID of the edge will reduce the linear operation to a constant time operation (step 2), but this is not possible since several edges can have the same "from" vertex.

The id number of the nodes where segments are connected to can be stored in a structure specially designed for the fast searching of nodes. A common structure for this purpose was proposed by Georgy Adelson-Velsky and Evgenii Landis in 1962, with the known AVL trees[9]. This structure is based on a searching tree, where the values are sorted from left to right. Moreover, the maximum "height" of the tree is reduced, in such a way that the maximum "depth" of the tree is minimized. If the edges are stored in an AVL tree structure according to the "from" node identifier, the order of step 2 can be converted in logarithmic order [15]. This improvement will reduce the order to (3.20).

$$O(BFS)_{SQL-AVL} = O(m \cdot log(n)) = O(n \cdot log(n)) \tag{3.20}$$

This improvement can result in remarkable time reductions (see Figure 3-10), but the assumption is not feasible for a real distribution system. The actual distribution grid is formed by several transformer stations. Normally, each feeder is connected between two or three transformer stations, in such a way that with the breakers

arrangement, the transformer where each segment is connected to, can be changed. This means that the "from" and "to" buses of each segment cannot be predefined since their nature is changing. So generally, a graph exploration with RDBMS responds to a quadratic order (3.19).

3.3.2 NoSQL database technologies for graph structured data

Despite the RDBMS demonstrated to be the best general solution for data storage in software solutions, they have a sort of drawbacks that make them not to be the best solution for some specific applications. The main condition imposed by the SQL database is the need of having a fixed structure (predefined number of columns and tables), that can make the definition of data not as direct as needed to decrease the order of the processing algorithms. Because of this, and specially in high performance processing applications (as big data), it is common the use of the so called NoSQL(Not-Only SQL) databases that modify the principles of the SQL databases in order to be better suitable for specific applications.

As the graph structures are one of the most used in software engineering, an specific database technology arise specifically focused in graph structured data. These are the Graph-based Databases (GDB). In these databases, the above mentioned way to define a graph is modified, and a table of vertices (that in the previous case was not estrictly necessary) is added with the direct definition of the direction of edges connected to them (see Tables 3.3 and 3.4).

Table 3.3: Example of vertex definition table in GDB

Node ID	**Attributes**	**Edge ID**	**Edge ID**	**...**
1	Attrs. Node 1	1	2	3
2	Attrs. Node 2	1	3	-
...	...	...	...	...
n	Attrs. Node n	m	-	-

With this modification on the structure, it is added redundancy to the database: the relations vertex-edge are defined twice. This will cause the increase of memory

Table 3.4: Example of edge definition table in GDB

Edge ID	From	To
1	1	2
2	1	3
3	1	4
...	...	...
m	n-k	n

required to store the same amount of data and the risk of loosing the consistency in the database.

There is no fixed structure since the number of edges associated to a vertex is not fixed. This implies that the data structure will be more complex to implement.

But this way the exploration algorithm's order will be modified. As last case, taking as reference the iterative process of BFS (lines 6 to 14 of Algorithm 1):

1.- The access to a selected vertex from its ID is an operation of $O(1)$ working with hashing tables.

2.- The search for the attached edges of a vertex is an operation of a complexity of $O(1)$. This is because the IDs of the segments are directly stored in the vertex information.

So the order of a tree exploration will be linear depending on the number of edges of the tree (3.21).

$$O(BFS)_{GDB} = O(m) = O(n-1) = O(n) \tag{3.21}$$

As it can be deduced on Figure 3-10, the linear behavior results on a huge improvement in terms of computational time.

Nowadays there are a wide number of databases with really good performance, and also able to ensure ACID(Atomicity, Consistency, Isolation and Durability) transactions[19].

- Atomicity: this characteristics ensures that each transaction is made as a unique operation. So if it is interrupted, or failed, all operations are rolled out.

- Consistency: the database must be "sound". Hence, after transactions the data must not have different values on different places.

- Isolation: each transaction must run completly independent from the rest. The DBMS is in charge of setting the sequentiation of transactions in case of contentious access to a state.

- Durability: the information after each transaction is permanent.

Moreover, despite the headers are bigger than in the SQL case, the information needed to define the grid is not memory intensive (in the order of GB) and it doesn't grow with time (or it does very slow). The main requirement of systems that process grid is the fast response of the system. Thus, the GDB technology was decided to be used for the topology management module design.

3.4 Graph database selection

The GDB technology is available from many software providers. Among the available databases, the most known ones are Amazon Neptune, CosmosDB, Giraph or Neo4j, among others.

The selected graph database should be native. This means that the internal query algorithms used to access to data are based on the concepts exposed in Section 3.3.2. Some SQL databases allow the user to use of a graph oriented query language, being its internal working as the SQL case instead. These type of databases must be avoided. Moreover, for a good performance solution it will also be required the support of ACID transactions.

For the development of the project, two database solutions were taking in consideration: Neo4j and TigerGraph.

Neo4j is the most popular solution for GDB [7], having a longer trajectory, since it appeared in 2007. Because of this, a huge documentation is available, as well as a strong community with lots of cases of use. Moreover, this database is available for

most of operative systems (OS X, Windows, Linux or Solaris), and it has application programming interfaces (APIs) for a wide list of tools and languages.

A newer approach to GDB is proposed by TigerGraph[3], which allows the parallel computation of graph structured data using Bulk Synchronous Parallel (BSP) technique. This database was launched to market in 2017, being its community more reduced that in the case of Neo4j. The environment for this solution is more restricted, being only available for Linux systems and with basic APIs and supported languages. Despite its shorter trajectory, the community is growing fast due to the high possibilities that this technology offers. In [11], it is shown the high potential of parallel computing capability to develop a high performance Energy Management System (EMS) for a transmission network in China.

Table 3.5: Main features comparison between neo4j and TigerGraph

Feature	**Neo4j**	**TigerGraph**
Parallel Computation Capability	NO	YES
Native Graph Database	YES	YES
Documentation	GOOD	GOOD
Case studies with power grid	YES	YES
Operative System	WINDOWS/LINUX	LINUX
Python API	YES	UNDER DEVELOPMENT
Built-in graph algorithms	YES	YES
Community	HUGE	MEDIUM
GSQL	NO	YES
Schema-free	YES	NO

The main strength of Neo4j is the *schema-optional* solution they offer. This capability allows the user to change the structure of data at any moment without any reset of the system and with no impact on the running processes. On the other hand, TigerGraph requires from a fixed schema that must be respected. This can be restrictive, specially during development works, but it is a common restriction to on-production applications.

The query language for graph databases is not standarized, being different for each of the solutions in the market. The most interesting proposal is made by TigerGraph,

where they combine a programming language with a query language using GSQL Language. This allows the direct implementation of algorithms within the database engine, making more efficient the processing of graph structured data.

Due to the high potential that parallel computing shows in EMS applications, this project will be implemented using TigerGraph solution. However, in spite of the high performance of parallel computing capability, this project will not be focused on the use of this feature for the development of the proposed system.

Chapter 4

Graph-based Data Model of Electric Distribution Network

A graph database implementation requires from the definition of a graph structure where the data must be mapped. This database should be designed in such a way that models the power system with all the information about the elements, and at the same time as simple as possible in order to accelerate the processing time. To do so, it is important to take into account how the information will be extracted and introduced to the database.

4.1 Common Information Model (CIM)

The CIM models are standards used in software industry in order to methodically define the characteristics of the software systems, using an object-oriented approach. The main purpose of these standards is the definition of a common structure of data in order to make compatible different systems that need to exchange any kind of information.

The CIM model were adopted by power utilities in order to define an standard model to allows the information transmission between different utilities or different systems within the same utility [16]. This approach was essential for achieving a seamless operation of power system applications and interoperability between different

systems.

The CIM defines the files that the utility system should admit as input and should produce as output from their internal system. The generation and consumption of these files are required also by some regulations in electric industry in order to stablish a standarized way of representing the power system information. The main problem that arised when these models were introduced is that in most of cases the data defined in the CIM doesn't match with the structure of the internal utility's database. To solve this problem, an interface module is needed to the importation and exportation of data to adapt the file to CIM. Hence, software engineers started to design the databases taking as reference CIM standards. This way, it is ensured that the database structure will be similar to the structure to import/export, making these processes easier and more efficient.

The CIM specification for power systems was developed by the Electrical Power Research Institute (EPRI). Then, this standard was adopted by the International Electrotechnical Commission (IEC). The CIM defined by the IEC is divided in three sets of standards:

- IEC 61970: the two main parts are the 301 and 302. In this project, it will be described part of 301, mainly dedicated to the definition of network model. The 302 part is focused in energy scheduling, financial and reserve model.

- IEC 61968: this set of standards is focused in this case in the representation of meters, assets and works.

- IEC 62325: in this document, the main purpose is the description of the markets organization.

- IEC 61850: it is a set of standards that specify the structure and requirements of the communication signals within the power systems.

4.1.1 IEC 61970:301

Each part in IEC 61970 is divided in different packages. In Figure 4-1 there are represented the dependecies between diferent packages defined in the part 301 of IEC 61970. The meaning of packages is simply the grouping of similar elements.

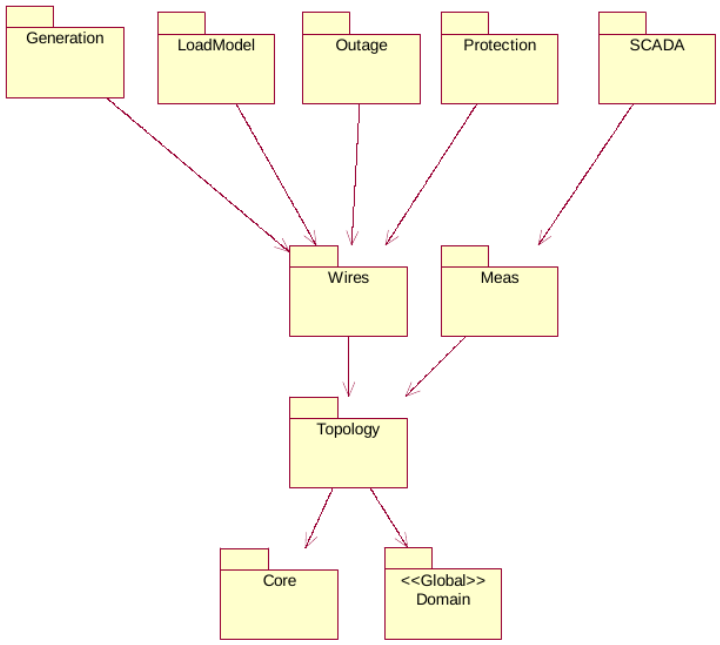


Figure 4-1: CIM Part 301 Package Diagram [6].

The main packages needed to model the static data of a distribution network are the following:

1.- **Core** : This package is the general abstract model for any element included in the system. Its entities are shared by all the applications: *Naming*, *PowerSystemResource*, *EquipmentContainer* and *ConductingEquipment*.

2.- **Topology**: this package is intended to the model of how equipment in the network is connected. In this package two main parts are defined: the topology model and the connectivity model.

3.- **Wires**: this package's target is the definition of the electrical characteristics of elements that form electrical grids (as transmission or distribution). It is

an extension of the Topology package and the information modelled in this package is intended to feed State Estimation, Load Flow or Optimal Power Flow applications.

4.2 CIM/UML

The Unified Modeling Language (UML) is used to define system architecture in software engineering. The UML is an Object Oriented Model intended to divide the software in different parts that have a set of characteristics and interact between them. This way, this methodology allows to design software engineering applications with an abstraction level appart from the source code or the specific details of the system.

This language can be used to model a wide variety of components as data structures, system interactions or use cases [8]. This language is generally used by CIM specifications in order to difine a generic approach to the model, not tied to any technology or programming language.

The main entities that compound UML are the ones described in this section.

4.2.1 Classes

This is the main entity, used to represent the objects or object-oriented programming paradigm. The classes mainly represent the characteristics that define a specific type of object. So these entities are defined by a name, a set of attributes and the relationships that relate them with the rest of classes.

In the case of a circle (in Figure 4-2), it can be defined as the name of the class and its characteristics: x, y of the center position, and its radius. This means that every object of class "circle" will need to contain these characteristics in order to be defined within the model.

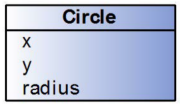


Figure 4-2: Example of class representation for the case of a circle [8].

4.2.2 Inheritance

It is an abstract type of class, used to define sub-classes within the model. A sub-class is a concretion of another general class, and the objects defined within the paradigm of it will inherit all the attributes of the "parent" class. The inheritance relationship between classes is generally represented with an arrow.

Taking this into account, we can differentiate between abstract and concrete classes if they are expected to be instantiated or not. To illustrate this, it is possible to redefine the previous case of the circle (see Figure. 4-3). In this case the x and y attributes can be abstracted to a class shape, which is the parent of different sub-classes with different additional characteristics. In this example, the class "shape" is abstract since it is not expected to be instantiated.

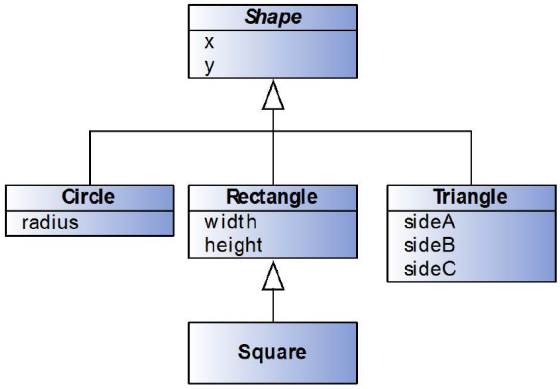


Figure 4-3: Example of inheritance with the case of different geometric shapes [8].

Moreover, it is possible to define "square" subclass of the "rectangle" class since all the squares are rectangles but not all rectangles are squares. This unidirectional definition characteristic is also common with inheritance characteristics (one to many relationship). When this is not this way, it is possible to merge both classes as in the previous case of the circle definition in Figure 4-2.

4.2.3 Aggregation

This relationship is intended to define containers. This is used to define classes which functionality within the model will be to group other classes of the model. In Figure 4-4 is shown an example of aggregation of different shapes in layers.

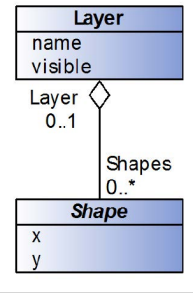


Figure 4-4: Example of shapes aggregation by layers [8].

4.2.4 Composition

A special case of aggregation is the composition. This relationship defines an aggregation when the contained class is a part of the fundamental aggregator. This is specially important in creation and remove processes since all the parts and container should be created and removed simultaneously since their individual definition makes no sense.

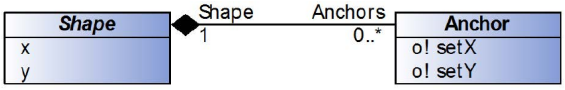


Figure 4-5: Example of definition of anchor composition by shapes [8].

4.2.5 Association

This characteristic is intended to define relationships between classes of any nature different from the previously defined.

An extension of the previous examples is shown in Figure 4-6. The relation is defined with roles on the endings of the line. Each association has two roles. In

this example, the "1" number means that each shape must have only one style class associates. On the other side "0..*" means that each style can be associated to 0 or more shapes.

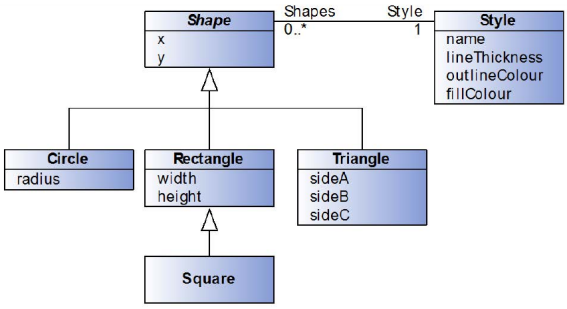


Figure 4-6: Example of association relationship between classes "shape" and "style" [8].

4.3 CIM model for the input static data

At the time of defining the power system, there are two different models to take into account: the assets model and the connectivity model. The first one is in charge of representing all needed characteristics of elements in the power system. And the connectivity model is focused on the definition of the relationships between the assets.

4.3.1 Defining devices

In Figure 4-7 it is represented the main power devices that form the distribution system. These systems are fuses, breakers, AC segments and customers.

- **IdentifiedObject (IO)**: this class is inherited by all elements in the system. Its main purpose is the identification of each element.
  Attributes:
  - mRID: unique identifier of the element.
  - name: it is a text that gives a readable name easier to manage by humans for the device.

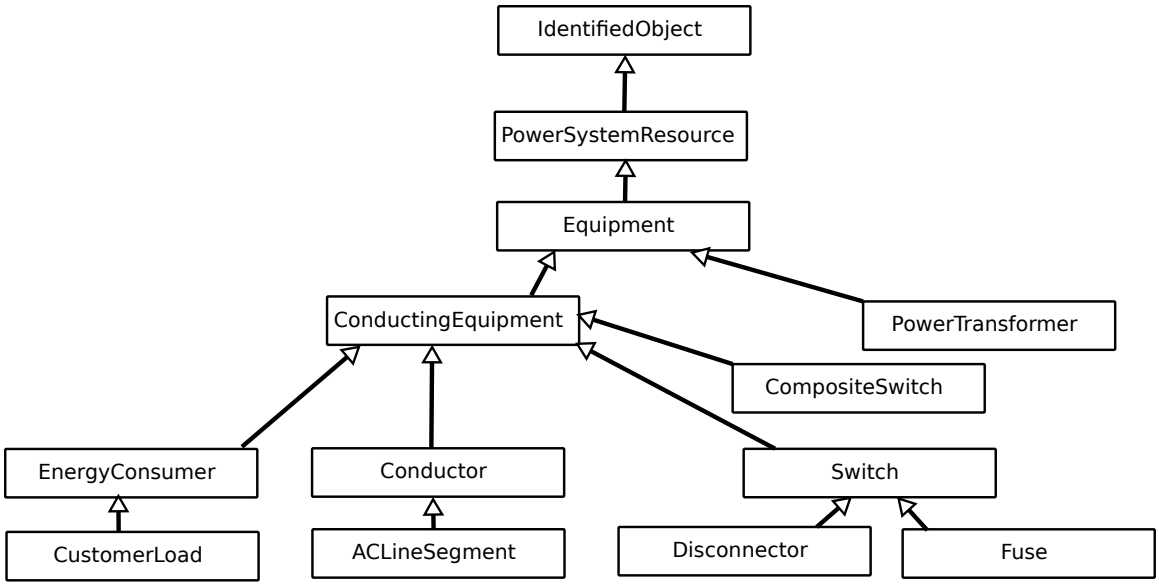


Figure 4-7: UML equipment definition of main elements used from IEC 61970.

- localName : identifier used to distinguish the object within a group. In this project, only unique identifiers are used, so local names will not be defined.
- pathName: gives the name of subgroups where the element is defined. As in this model only simple devices are introduced, this attribute will be removed.
- aliasName: a free text, normally shorter than the name. In this project the utility doesn't define any alias for the elements in the grid.
- description: it is a free text that gives a description of the object.

Roles:

- ModelingAuthoritySet: it identifies the authority that provided the information for modelling. In our case, as it is modelled an isolated grid for the company E-REDES, there is a single ModelingAuthority, so this class will not be included in the database and this role will not be used.

- **PowerSystemResource (PSR)**: it will define any physical equipment in the

system as well as equipment containers. It only adds a set of roles to the models. Roles:

  - OperatedBy_Companies: it defines which is the company in charge of the operation. As in the case of ModelingAuthority, it will be only one, so this role will be removed.
  - Contains_Measurements: it relates the resource with any measurement made over this element.
  - OutageSchedule: the outages management will not be developed in this project, so it will also be removed.

- **Equipment (EQ)**: models the physical devices. It only adds one role:
  - MemberOf_EquipmentContainer: relates the equipment with its container. E.g. in the case of transformers there is one container that allocates all elements as the windings.

- **ConductinEquipment (CEQ)**: this class is inherited by all parts of the power systems which role is the carry of current. Attributes:
  - phases: this attribute defines a phase code with the information of phases of the conducting equipment. The possible values are: ABCN, ABC, ABN, ACN, BCN, AB, AC, BC, AN, BN, CN, A, B, C, N.
  - type: this attribute is not defined by IEC 61970, and it corresponds to a specific adaptation in distribution systems in Spain. B2 means that the phase voltage is 230V, so single phase devices will be phase to ground connected. B1 stands for grids where the phase voltage is 120V, so single phase devices are phase to phase connected.

  Roles:
  - Terminals: relates the equipment with its terminals, needed for associating the conductor to the connectivity model.

- BaseVoltage: allows the association of the conducting equipment with a voltage level. In our case, the voltage level will be 400 V for all the system, so this role will not be defined.
- ClearanceTags: this role is related with field work, that is not analyzed in this project.
- ProtectionEquipments: it allows the relationship with some equipment whose role is the protection of this conducting equipment, i.e. a fuse. In our case, despite the breakers and fuses in the grid are added, as the monitoring of the protection system performance is not on real time, the system will be simplified avoiding its modelling.

- **EnergyConsumer (ECons)**: it models any element in the grid that uses energy. Attributes:
  - customerCount: number of customers that conform this load.
  - pFexp: Exponent of per unit frequency effecting real power.
  - pfixed: active of the load that is fixed.
  - pfixedPct: percentage of fixed active power within the load group.
  - pnom: nominal active power.
  - pnomPct: percentage of nominal active power within the load group.
  - powerFactor: nominal power factor
  - pVexp: Exponent of per unit voltage effecting real power.
  - qFexp: Exponent of per unit frequency effecting reactive power.
  - qfixed: fixed reactive component.
  - qfixedPct: percentage that represents fixed reactive power respect to the load group.
  - qnom: nominal reactive power
  - qnomPct: percentage of nominal reactive power respect to the load group.

  - qVexp: Exponent of per unit voltage effecting reactive power.

- **Conductor (Cond.)**: Attributes:

  - b0ch: zero sequence shung susceptance.
  - bch: possitive sequence shunt susceptance.
  - g0ch: zero sequence shunt conductance.
  - gch: possitive sequence shunt conductance.
  - length: length of the conductors.
  - r: possitive sequence series resistance of the conductor.
  - r0: zero sequence series resistance of the conductor.
  - x: possitive sequence series reactance of the conductor.
  - x0: zero sequence series reactance of the conductor.

  Roles:

  - ConductorType: it relates the conductor with a ConductorType that defines the section of the conductor.

- **Switch (SW)**: it is defined as a device that is able to open, close or both. Attributes:

  - normalOpen: when no measurement about the state of the switch is being performed, the status of the switch is spected to be the one stored in this attribute. As in the distribution system there will not be any sensing on the switches states, this attribute will store the represented switch status in the database.
  - switchOnCount: number of operations since last reset.
  - switchOnDate: datetime of last operation.

  Roles:

  - CompositeSwitch: sometimes the switches are places in the same cabinet, in such those cases the switches will be related with a common CompositeSwitch class.
  - SwitchingOperations: the switch can be related with a given schedule comming from the operation.

- **CompositeSwitch (CSW)**: aggregates switches placed in the same cabinet. Attributes:

  - compositeSwitchType: reference code that allows the identification of the schema that defines the cabinet.

  Roles:

  - Switches: it relates the CompositeSwitch with the contained devices.
  - MemberOf_Substation: this relationship will be applied for the definition of the switches boards in substations.

- **CustomerLoad**: it models a meter in the grid which target is the measurement of the energy consumed by a costumer connected to the network. It only specifies the type of energy consumer, and it doesn't add any role or attribute to the model.

- **ACLineSegment**: it is just a definition of a subclass of a conductor with no additional roles or attributes.

- **Fuse (FS)**: It adds the rating of the fuse to the attributes:

  - ampRating: defines the maximum current that the fuse admits in Amps.

- **Disconnector (BR)**: it is manually operated switching device. No additional attributes or roles.

4.3.2 Defining connectivity

In Figure 4-8 it is represented the connectivity model definition in IEC 61970. As it can be seen three additional classes are added: *Terminal*, *ConnectivityNode* and *TopologicalNode*.

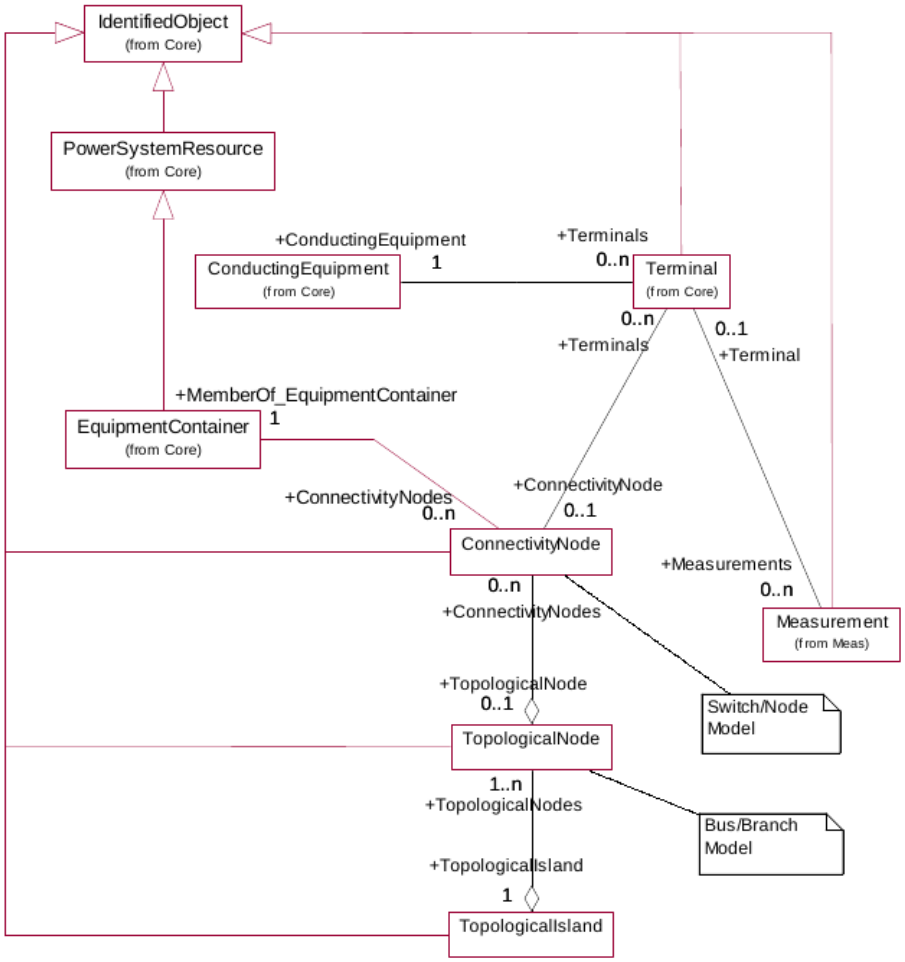


Figure 4-8: Connectivity model definition in IEC 61970[6].

In CIM model the electrical components are not directly connected with each other. Each electrical element will be related with a Terminal class that will define how many output/input connections the element has. This way a terminal can have three main roles: to a conducting equipment in charge of carry currents, to a Measurement class or to a ConnectivityNode class.

The ConnectivityNode class is used to define an electrical connection with no impedance. As it can be noticed, role on the terminal's side define that a single connectivity node can be connected to n terminals. This is the main role of the terminal's class and it is intended to define the physical connectivity between devices.

A simple power system is shown in Figure 4-9 in order to illustrate the CIM modelling procedure for connectivity definition. This system has two substations ({N1,N2} and {N3,N4,}) and three lines connecting them. Each substation can isolate two busbars to feed different loads. In the represented state, the N3/N4 substation is splited in two busbars, and line L3 is out of service.

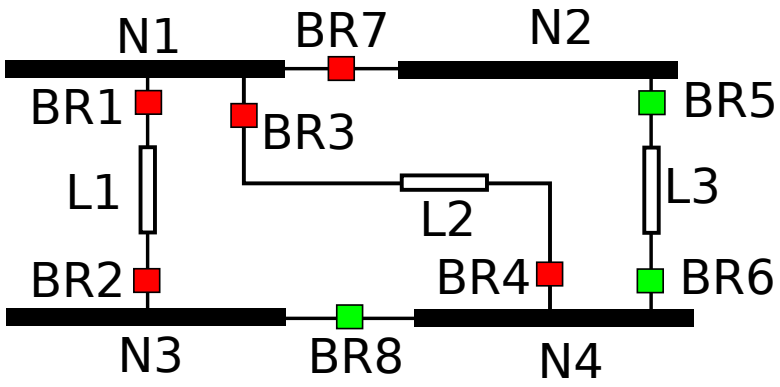


Figure 4-9: Node-breaker model of a simple power system. [Switch status: red= closed, green=open].

The detailed model of the previous defined system is shown in Figure 4-10. As it can be seen, this model contains all the elements in the system and the connectivity between them. The connectivity of elements is defined by the joint of terminals using connectivity nodes. The terminals are generally defined by conducting equipment, in the example case the breakers and lines(ACLineSegment).

The node-breaker model shows all the devices in the network, no matter what the state of the devices is. This model is specially usefull for operation works. However, for planning purposes, the approach is a bit different since for simulation it is only needed the actual status of the breakers and the actual system (the operation status). For this reason an additional class TopologicalNode is also defined.

The topological nodes are aggregation classes that join ConnectivityNodes when any closed switching device is connecting them. As the status of switching devices

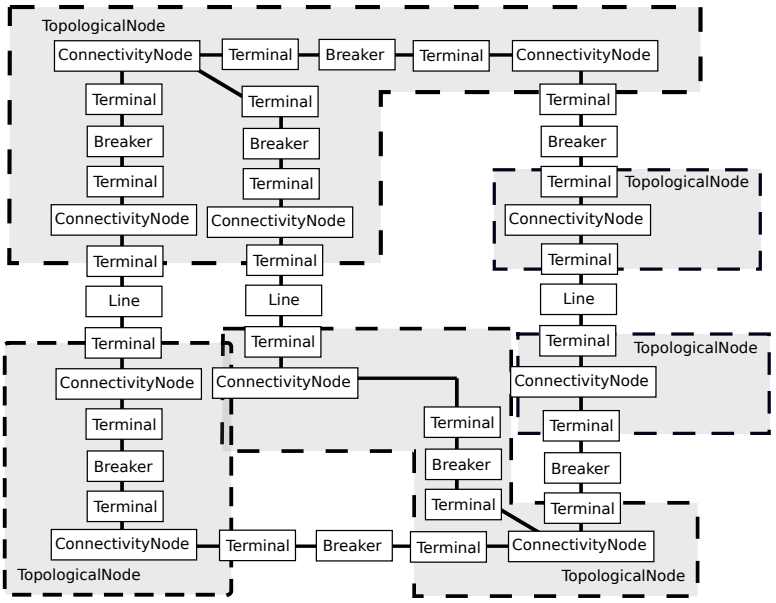


Figure 4-10: Connectivity/topology model's CIM instances to define the example's power system.

changes, the topological nodes also do.

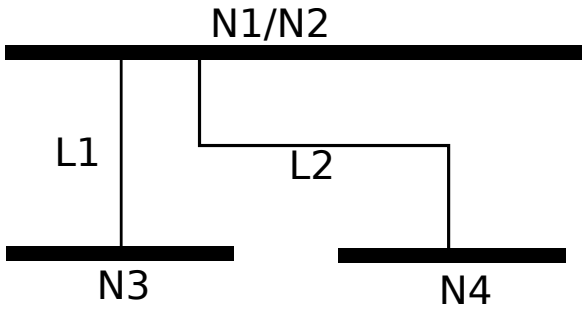


Figure 4-11: Bus-branch model of a simple power system.

This way, using TopologicalNodes and ignoring the switching devices it is possible to reach the so-called bus-branch model (see Figure 4-11) [12]. This model only includes the actual active elements in the grid and it models the switching devices as zero impedance connections.

4.3.3 Model of distribution transformer stations

In last sections it was explained how is implemented the conducting equipment in CIM. These models match quite well with the utilitie's internal model used for

databases, so they can be adapted. But this is not the case with transformer stations.

The system to model is the low voltage system, so higher voltage networks will not be included in the model. This way, the transformers will be model as generators to the system. This generation element needs to store attributes about a transformer, so none of CIM defined classes will match with this. A new no-CIM class will be created for this purpose.

Something similar happens with the station itself. A distribution transformer station can be defined as a composition class formed by conducting equipment, self-defined generator (the transformer) and communication devices.

The CIM model of communication for power systems is defined in IEC 61850 standard. But as the communication devices also behave as information sinks from the point of view of the power system, a simplified model will be used instead of the CIM standarized.

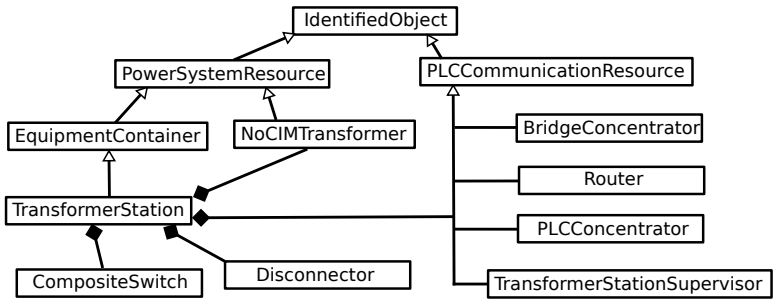


Figure 4-12: UML architerture of new elements added to CIM needed to model distribution transformer stations.

In Figure 4-12 is represented the structure used for the definition of all these new classes. The trasformer was modeled as a subclass of PowerSystemResource and the communication devices were included in a new subclass PLCCommunicationResource. The transformer station class was model in a similar way to a substation CIM class, as a EquipmentContainer subclass that acts as a composition class. This means that the intiantiation of a TransformerStation class will imply the creation of the rest of attached classes. Notice that the classes CompositeSwitch and Disconnector were also attached to the station in order to model the low voltage board and the general breaker of the transformer, explained in sections 2.3 and 2.5.

- **EquipmentContainer (EQC)**: it is a root class for Equipment classes. It adds the roles to connect with ConnectivityNodes and some Equipment, in this case it will also be added a role to connect it to PLCCommunicationResource classes.

- **TransformerStation (TS)**: it provides general information about the transformer.
  Attributes added (see Section 2.2):

  - Type
  - Status
  - Municipality
  - X
  - Y

- **NoCIMTransformer (TF)**: it will define the main characteristics of the transformer.
  Attributes:

  - RatedPower: defines the nominal power of the transformer in MVA.
  - ModelRef: it is a text that defines the transformer model.

- **PLCCommunicationResource (Com.)**: it is a class that groups communication devices. It provides a role of being associated by a TransformerStation.
  Attributes added:

  - IPLAN: it stores the local IP address within the local network of the transforer station.
  - ModelRef: defines the model of the equipment installed.

- **BridgeConcentrator (BC)**: it models the equipment in charge to collect the measurements performed in the transformer station. No additional attributes or roles were added to this project.

- **Router**: it is the element that transmits the information to the telecontrol center.
  Added attributes:
  - Network provider: this attribute stores the name of the company that provides the service to access to the Internet.
  - IP: defines the IP address to access to the Internet.

- **PLCConcentrator**: it models the element that acts as the central node in the Prime Network (the transformer's subgrid). It only adds a Prime address that defines its direction in the PLC network.

- **TransformerStationSupervisor (TSS)**: it is the system that generates the measurements in the transformer station to transmit to the BridgeConcentrator.

4.4 Mapping of CIM model into a GDB structure

Once the model of main components was reviewed and gaps in the standard were explained, it was needed to define the data model of the designed database system. To do this, it is needed to map the UML architecture into a graph structure. In [17] it is explained how to convert a SQL oriented CIM database structure to a GDB data structure. The process consist on represent all kind of relationships as edges, except in case of inheritance. The inheritance must be abstracted, being the characteristics of all types of nodes dependant on the inherent nature of overy type. In this work the author indicates this inheritance relationship using tags, a feature that is allowed in any graph database managers. In Figure 4-13 it is shown the topology model of IEC61970 (see Figure 4-8).

4.4.1 Simplified model of low voltage customers

The model of customers is defined in CIM as previously explained in the section 4.3.1. Despite this definition is valid for a general customer connected to a grid, when

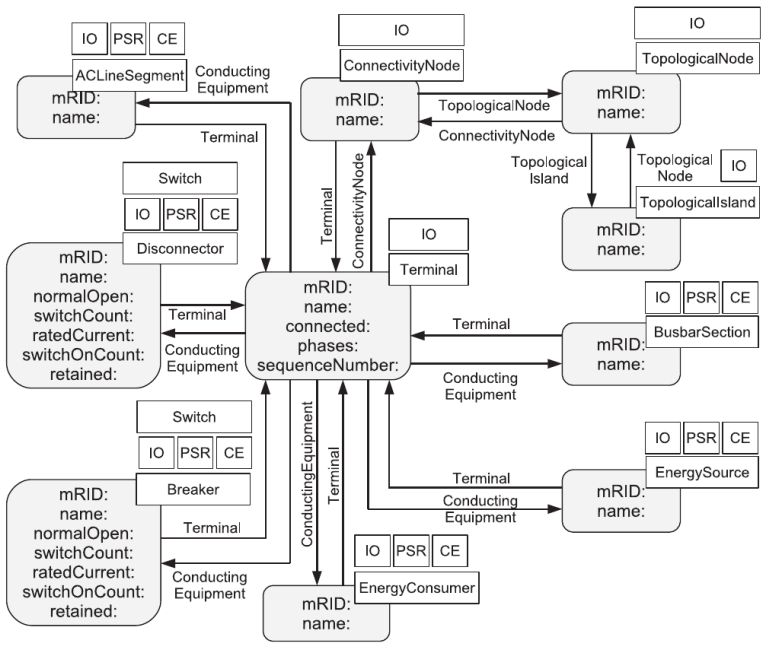


Figure 4-13: Mapping of CIM topology model to a graph structure[17].

defining low voltage customers, many of required parameters are not defined by the utility.

On the one hand the customers are defined individually, installing a single metering infrastructure for each of them. There is somethig similar to this defined aggregation in the CIM, that is the connection point. This connection point is mainly formed by fuses, which role is the protection of the installations. Thus, this element will be formed by two classes: the connection point, as the aggregation class, and a set of Fuses. Due to this nature of the connection point of aggregating protection elements, it is created as a subclass of CompositeSwitch.

Respect to the CompositeSwitch class, the ConnectionPoint will add the type of connection. For the distribution system in Spain, as this element is in the customer end, it is needed the specification of the type of system (check Voltage Level in Section 2.12).

Besides, to this connection point there are connected a sort of customers. Two elements are modelled on the customers' end: the smart meters and the individual fuses that each user uses to connect to the connection point. Hence, as shown in Figure

4-15, the end users are modelled using three classes: a Fuse, an EnergyConsumer (that represents the smart meter) and an End User aggregation class that joins both elements and stores all the needed information about customers.

The aggregations of customers are not used in this model, since the individual definition of customers is required. Thus, all parameters defined in CIM devoted to the aggregation of customers are not applicable. These attributes are *conformingLoadFlag*, *customerCount*, *pfixedPct*, *pnomPct*, *qfixedPct* and *qnomPct*.

Moreover, in low voltage loads, the impact of a single load in the whole power system is so low. The loads generally are not able to affect the frequency or voltage in the nodes of the system. So the parameter used to model this impact are not defined by the company and will also be removed from the model. These attributes are *pFexp*, *pVexp* and *qVexp*.

The main difference between the smart meters in the distribution grid and the CustomerLoad class is the measurement and communication capabilities included in the smart meters. So the Smart-Meter class will be created as a subclass of CustomerLoad, adding the parameters about communications, in this case the direction of the smart-meter (IP).

4.4.2 Conductions model simplification

As it was shown in Figure 4-10, even for a simple model, many instances are needed. Most of these needs are due to the connectivity definition requirement. This definition is made from a general perspective in order to allow any possible real equipment configuration (see Figure 4-14-a), but makes the model so complex for simple cases. Since in the distribution system the conducting equipment forms the majority of the elements, a simplification of the model was used.

In Figure 4-14-b the CIM/E model simplification, proposed in [22]. This model was proposed in order to reduce the ammount of data and speed up the response of the database system, for a transmission application.

For this project, as in distribution systems the weight of the ACLineSegment elements is huge, it was proposed the direct implementation of bus-branch model of

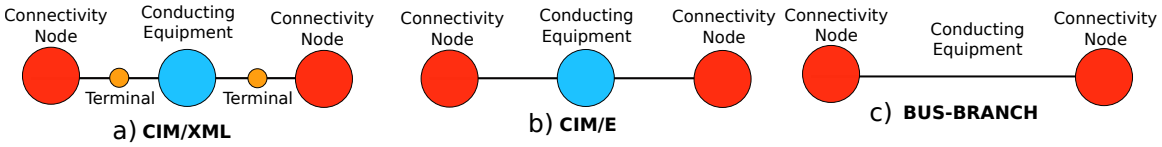


Figure 4-14: Different model modifications for modeling conducting equipment.

the conducting equipment (see Figure 4-14-c). This design requires the association of some information (set of attributes) with an edge. This is normally avoided, and has some implications:

1.- The model will be unmutable, being the maximum and minimum amount of terminals of the conducting equipment fixed to 2.

2.- The ConductingEquipment class cannot have any role other than the relation with a ConnectivityNode. This imply that the modelling of some works or changes on this equipment along time is not possible. This is, the historical evolution of the conducting element represented cannot be included. I.e. if the conductor type is changed, the attributes of edge will be updated, loosing past information about the conductors.

Despite their drawbacks, it will allow the model of the conducting equipment as weighted graphs. This will be particularly convenient for the path analysis algorithms, since the impedance can be used as the weighting variable of the edge (see Section 3.2.2).

4.4.3 Events registration

As explained in Section 3.1, the main purpose of this project is the creation of a system that analyzes the topology of the grid in real time. This mainly implies the model of the static part of the data, devoted in its major part to the connectivity and assets definition. However, there is information related with time evolution of the grid that must be included in order to have a full picture of the ”static” part of the grid along time.

The information about time evolution of the grid is mainly related with events. As showed in Section 3.3.2, the graph databases are not efficient storing big ammounts of data. Thus, the timestamped data should be avoided in these type of database systems. So two new class called Switching and Event will be created for the storage of this information.

- Event (EV): it is the parent class that models the events to include in the topology storage system. It defines a temporal state of an element so it will add two main attributes:
  - InitialTime: initial instant of the state defined in the event. A 0 value means that the state was given from the creation of the database.
  - EndTime: indicates the time when the event was given.
- Switching: this class models the change on the status of a Switch class. It adds one attribute:
  - Status: describes the status of the Switch during the time specified by the Event.

  It add one role:
  - Event: relates the Switching class with the Switch.

4.4.4 Feeders model extension

From each transformer station, a sort of feeders depart by means of a fuse connection. These fuses are joined in the same CompositeSwitch container, that models the Low Voltage Switch-Board to the station. But each of these fuses are supervised by a measurement system that is the FeederSupervisor. Hence, somehow these two classes should be aggregated by a single class. To do this, it is created the OutgoigFeeder class, that adds the needed roles to join the feeder supervisor and the fuses. Moreover, this class adds roles that allow the transformer station to aggregate all feeders.

4.4.5 Geographical information

The database will feed two type of applications: the visualization module and the EMS. The main feature of the visualization module is the geographical representation of the grid. While the EMS will only require the electrical characteristics and connectivity in order to create the electrical model of the grid. The CIM model is mostly intended to model the electrical characteristics, and no visualization modules are included in it.

To solve this problem, the coordinates will be added to the following classes:

- Feeder, ConnectivityNode, ConnectionPoint and TransformerStations are normally defined at a single geographical point, so the X and Y coordinates will be added as attributes.
- ACLineSegment: these elements will be defined by two arrays in order to define a path according to the method defined in section 2.12

4.4.6 Complete model

In Figure 4-15 it is depicted the data model designed for the implementasion of a topology management system for the distribution grid.

As it is indicated in the Figure, there are four main parts: the transformer station, the feeders output, the conductions and the end-users domain. Each of these elements contain at least one element as aggregator of all elements that form each domain. These elements are the Trasformer Station (TS), Outgoing Feeder and the End User. The only exception to this rule is the conductions domain. This part is formed mainly by ConnectivityNodes and ACSegments, and the main function of this domain is the relationship definition of the rest ot the domains.

Moreover, the representation shows the duality of the distribution network. In the same system, there is a measurements/communications layer and a power layer. The first one is represented by the vertices above the aggregation elements, while the power elements below them. The exception to this rule is formed by the conductions domain, where the elements represented belong to both layers.

Besides, there are additional roles (edges) included within the TransformerStation and Feeder domains. The main purpose of this, is to join all the elements between them with specific edge-types. This way, at the programming time, it is easier to select specific paths rather than in the case of having general edge-types (as the case of Connection edge type). The increase of the number of edges in these parts of the database will not be translated into a big increase of the data volume, since the number of these domains is limited (specially in the case of transformer stations). Moreover, the selection of these edges should be done taking into account how the information will be requested by the user.

For instance, it is quite possible that the user will require the number of feeders outgoing from a station. Hence, if a direct connection between these two elements is provided, the number of vertices to read is limited to the minimum, reducing this way the processing time.

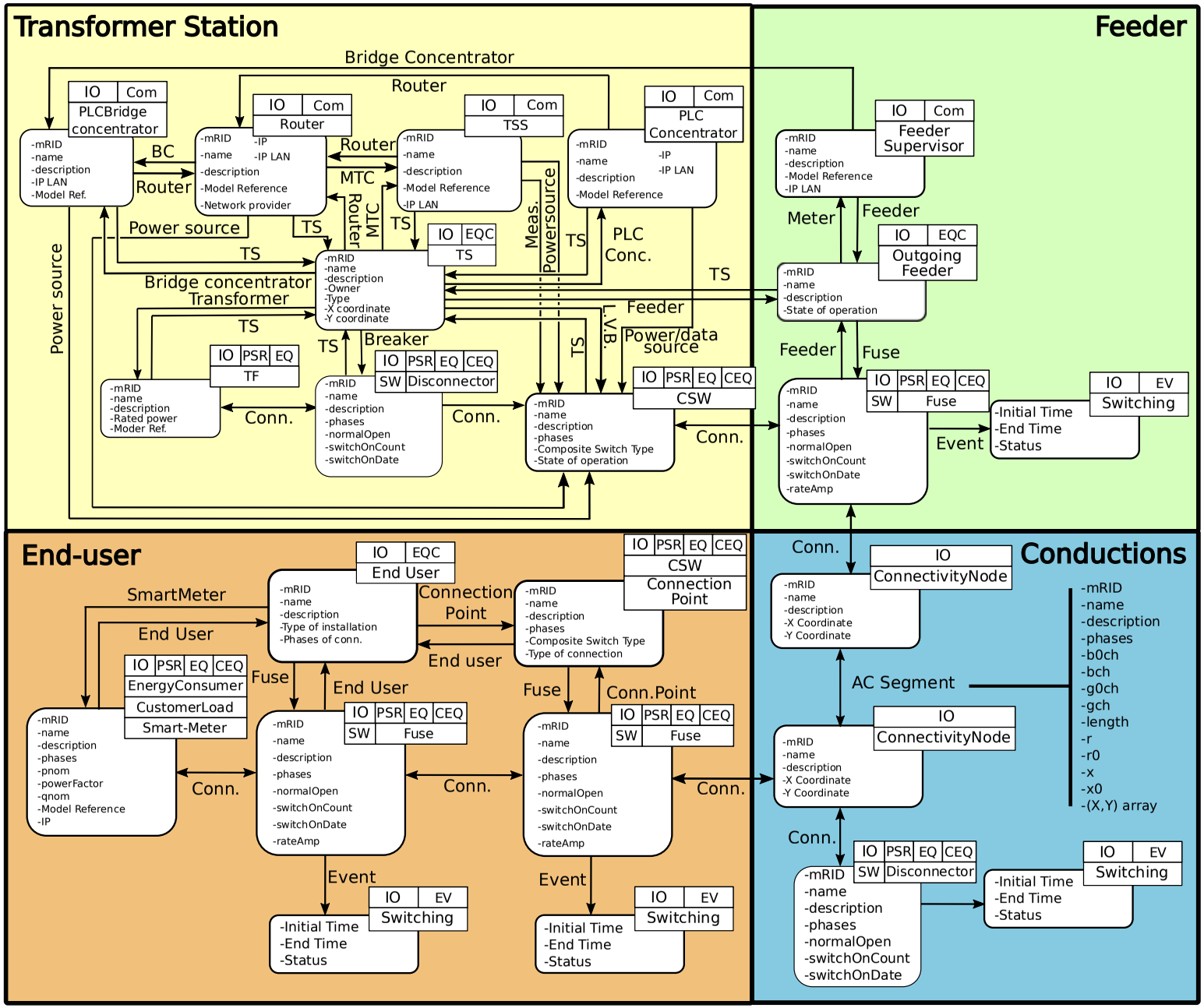


Figure 4-15: Designed graph-oriented data model for distribution system.

Chapter 5

Practical Implementation

According to the criteria defined in last chapters, it will be created a data management system for the grid of Asturias. In this chapter, it will be explained the basics of TigerGraph for the database creation and graph analysis algorithms implementation, used for the processing of grid topology. Finally, a web application interface is created to check the performance of the system.

5.1 TigerGraph

TigerGraph is a graph-based database platform designed to run on Linux environments. In Figure 5-1 the main components and architecture of the platform are represented. TigerGraph's platform uses a Nginx server to run (orange square in Figure).

The core components of this system are the Graph Storage Engine (GSE) and the Graph Processing Engine (GPE), both implemented in C++ and in charge of the main processing works in the database. Moreover, there are included five additional modules: graph processing interpreter (GSQL), Graph Visualization (GraphStudio application), application programming interfaces (APIs), and standard and custom User Define Functions (UDFs).

TigerGraph platform will be used with three main tools: RESTPP API, GSQL and GraphStudio.

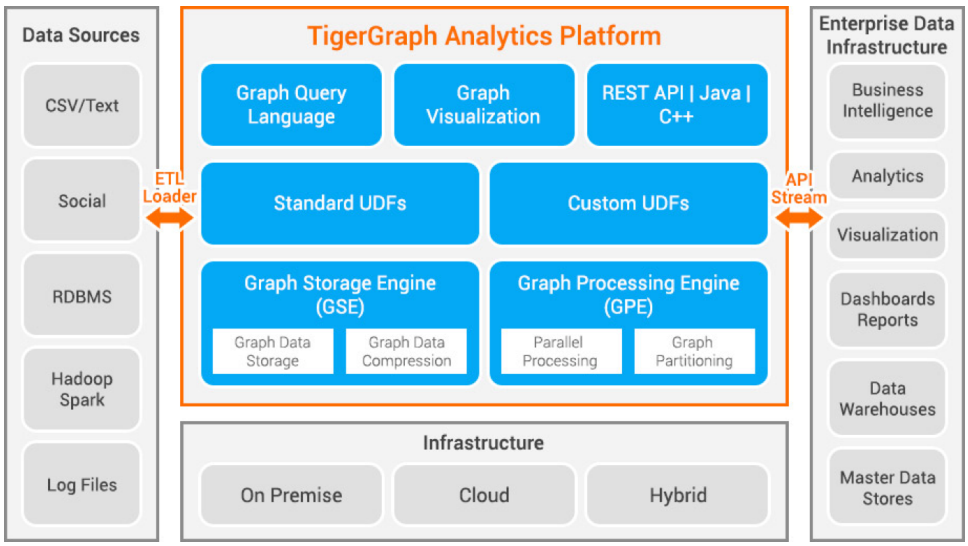


Figure 5-1: Architecture of TigerGraph platform[3].

The main method to communicate with the internal engines of the database will be the RESTPP (or REST++) API. As it can be seen in Figure 5-2, this API is the only one that directly communicates with the Message Queuing application of Tigergraph. Therefore, for enterprise applications, this will be the preferred method to implement the interactions with the database. This API provides several HTTP endpoits that trigger pre-installed C++ commands that perform operations with the graph database.

GSQL is the functionality that interprets the queries programmed by the user. These operations can be used to define the schema of the graph, to load/update data or to create queries. This client uses a language also called GSQL, and it will be the main method to install new commands to RESTPP API.

The GraphStudio provides an User Interface(UI) that allows the user to perform the basic operations also available with GSQL, but in a graphical way. Moreover, it provides a platform that allows the user to query the graph basic structure and the data loaded to it. This tool will be specially useful during debugging processes.

TigerGraph also includes a tool called TigerGraph System Service State (TS3) to monitor the database system. The data measured by this tool can be checked by the user using GraphStudio or GSQL.

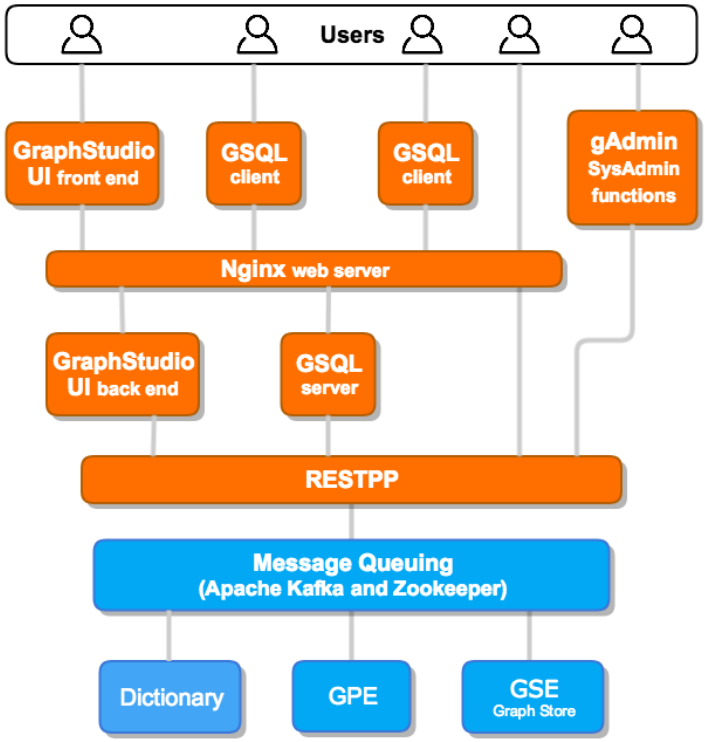


Figure 5-2: Internal architecture of main Tigergraph tools[3].

Infrastructure

The TigerGraph platform is available for cloud, on premise or hybrid implementation. The application developed in this project involves sensitive data about customers and the company, so special regulations apply to these kind of applications, requiring on premise implementation. This will be the approach followed in this thesis.

5.1.1 GSQL Language

The query language used by GSQL client is a bit different from the rest of query languages for databases. The main difference between this language and the one of the rest of engines is that it includes control flow features as the structures IF-THEN-ELSE, WHILE, FOREACH... This feature is specially usefull when performing graph exploration algorithms.

As it was explained in Section 3.2.5, the algorithms are formed by at least one iterative structure that is broken when some situation is given (i.e. no new vertices

found from the frontier set). Thus, the GSQL capability to implement this iterative loops on the query engine avoids external flow control applications that would slow down the overall performance of the query system.

Define a graph schema

This will be the first step on creating a database, and its main target is to stablish the different types of edges and vertices and how are them related between each other. This process will be equivalent to the tables definition in a SQL database.

```
1  /* EXAMPLE OF VERTEX TYPE CT CREATION */
2  CREATE VERTEX CT(PRIMARY_ID MSLINK UINT, MSLINK UINT, CLAVE_BDI
       STRING, DESCRIPTION STRING, REGION STRING, MUNICIPALITY STRING,
       PROPERTY STRING, STATUS STRING, TYPE STRING)
3
4  /* CREATION OF THE CONNECTIVITYNODE TYPE VERTEX */
5  CREATE VERTEX CONNECTIVITYNODE(PRIMARY_ID ID UINT, ID UINT, X FLOAT,
        Y FLOAT)
6
7  /* CREATION OF FUSES TYPE VERTEX */
8  CREATE VERTEX FUSE(PRIMARY_ID MSLINK UINT, MSLINK UINT, REAL_STATUS
       STRING, NORMAL_STATUS STRING, TYPE STRING)
9
10 /* EXAMPLE OF EDGE TYPE "OF_CT" CREATION */
11 CREATE DIRECTED EDGE OF_CT(FROM *, TO CT)
12
13 /* CREATION OF EDGE THAT CONNECTS A CT DIRECTLY TO A FUSE */
14 CREATE DIRECTED EDGE WITH_FUSE(FROM CT, TO FUSE) WITH REVERSE_EDGE="
       OF_CT_2"
15
16 /* CREATION OF SEGMENTS TYPE EDGE */
17 CREATE UNDIRECTED EDGE ACLINESEGMENT(FROM CONNECTIVITYNODE, TO
       CONNECTIVITYNODE, LENGTH FLOAT, MSLINK UINT, X STRING, Y STRING,
       COUNTER FLOAT, UNE STRING)
18
19 /* CREATION OF THE GRAPH */
20 CREATE GRAPH DistributionGridGraph(CT, CONNECTIVITYNODE, OF_CT,
       ACLINESEGMENT, FUSE, WITH_FUSE)
```

Figure 5-3: Example of simple graph schema creation.

In the case of vertices creation it is only needed to define the name of the vetex and the list of attributes as well as the type of data they will store (see the creation of CT and CONNECTIVITYNODE in Figure 5-3). For the case of edges is similar, but it is necessary to define if they are directed or indirected and the type of vertices they

will be connected to (see the creation of "ACLINESEGMENT" edge in Figure 5-3). It can also be noticed in the last example that TigerGraph allows to set attributes to the edges. In the case of definition of an edge that can link several types of vertices it can be used '*' to define it (see the creation of "OF_CT" edge in Figure 5-3). Finally, the previously created structure elements should be assigned to a graph in order to create the full structure.

Load data to graph

Once the structure of the database is defined, it is necessary to load data according to that structure. TigerGraph provides some methods to do it (see Figure 5-1). In this project the data will be loaded from Comma Separated Values (CSV) files. The columns of these files will be mapped to the attributes of the different vertices and edges defined in the graph schema.

As it can be seen in Figure 5-4, the load of data is done with the so-called "LOADING JOB" function. This function is formed by two main steps: the definition of paths to data files and the mapping of data files to the data structure.

The first step only sets the path to the file. This path can be global or relative to the prompt path.

The mapping of columns with attributes can be made in two ways. On the one hand it can be defined the name of column taken for each attribute, defined in the header of the file. On the other hand it is possible to define the column number (starting from 0) instead of the name. The first method allows the modification of the columns order, while in the second one the columns order must not be changed. The first method will be generally preferred. The mapping of attributes is performed with the characteristics header to True (not to take the first row), and the separator is set to be comma.

Query language

The query language allows the consulting of the information in the database.

The basic structure used in GSQL is the SELECT statement. An example of its

```
/* CREATE THE LOADING JOB (TO A SPECIFIC GRAPH) */
CREATE LOADING JOB Load_Simple_Data FOR GRAPH DistributionGridGraph{
    /* THE PATH TO THE FILES THAT CONTAIN THE INFORMATION MUST BE
       DECLARED */
    DEFINE FILENAME file_cts = "<path>/ct.csv";
    DEFINE FILENAME file_segments = "<path>/segments.csv";
    DEFINE FILENAME file_nodes = "<path>/nodes.csv";
    DEFINE FILENAME file_fuses = "<path>/fuses.csv";
    /* IT IS SPECIFIED HOW THE DATA WILL BE MAPPED TO THE PREVIOUSLY
        DEFINED GRAPH STRUCTURE */
    LOAD file_cts
            TO VERTEX CT VALUES($"MSLINK", $"MSLINK", $"CLAVE_BDI",
               $"DESCRIPTION", $"REGION", $"MUNICIPALITY", $"
               PROPERTY", $"STATUS", $"TYPE") USING header="true",
               separator=",";

    LOAD file_segments
            TO EDGE ACLINESEGMENT VALUES($"NO" NUDO, $"NE" NUDO, $"
               LONGITUD", $"MSLINK", $"X", $"Y", $"count", $"UNE")
               USING header="true",separator=",";

    LOAD file_nodes
            TO VERTEX CONNECTIVITYNODE VALUES($0, $0, $1, $2) USING
               header="true", separator=",";

    LOAD file_nodes
             TO VERTEX FUSE VALUES($0, $0, $1, $2, $3, $4)
             TO EDGE WITH_FUSE VALUES($5,$0) USING header="true",
                 separator=",";
}

/* SELECT THE GRAPH WHERE THE DATA WILL BE LOADED TO */
USE GRAPH DistributionGridGraph
/* LOAD DATA TO GRAPH */
RUN LOADING JOB Load_Simple_Data
```

Figure 5-4: Example of loading job creation for a simple graph.

use is shown in Figure 5-5 where the variable *ResultSet* is used to store the vertices contained in *SourceSet*.

```
/* Basic selection from a Starting set */
ResultSet = SELECT id FROM <SourceSet>:id;
```

Figure 5-5: Example of loading job creation for a simple graph.

GSQL Language also allows the request of related data through the graph exploration. This functionality is ilustrated in the Figure 5-6. There are several query

possibilities, being able to filter vertices by the edge type, if it is directed/undirected, or by vertex type.

```
1 /* SELECTION OF NEIGBORS OF TYPE ToType LINKED WITH UNDIRECTED EDGES
     of edgeType */
2 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)-ToType:t;
3 /* SELECTION OF NEIGBORS OF TYPE ToType LINKED WITH DIRECTED EDGES
     of edgeType */
4 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)->ToType:t;
5 /* SELECTION OF NEIGBORS LINKED WITH UNDIRECTED EDGES*/
6 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)-ANY:t;
7 /* SELECTION OF NEIGBORS OF TYPE ToType LINKED WITH ANY UNDIRECTED
     EDGE */
8 ResultSet = SELECT t FROM FromSet:f-(:e)-ToType:t;
```

Figure 5-6: Examples of SELECT statement in 1-hop queries.

The SELECT statement also admits a sort of clauses with additional actions to extend the functionalities of the statment.

The WHERE clause is used to filter by attributes the sets involved in the statement (see example in Figure 5-7). To do so, it is defined a function that returns a boolean value, so the SELECT statement will store the sets where the boolean function returns a *True* value. It admits conditions over the from vertex set, the to vertex set or the edges set.

```
1 /* SELECTION OF NEIGBORS OF TYPE ToType LINKED WITH UNDIRECTED EDGES
     of edgeType */
2 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)-ToType:t WHERE f.
     NAME == "someName" AND f.PROPERTY=="someOwner";
```

Figure 5-7: Examples of WHERE clause use case.

The ACCUM and POST-ACCUM clauses are devoted to the aggregation of data accross the query. They will be the central element to store the results accross the graph exploration algorithms programmed. The information that these clauses will collect will be stored in special variables called accumulators. These accumulators must be prototyped at the beggining of every query.

There are several type of accumulators that aggregate different types of information, and in different ways. The main aggregators used in this project are listed

below:

1.- **AndAccum**: these accumulators store boolean values. The aggregation procedure corresponds to an AND operation in such a way that the value stays *True* until any *False* value is added to the aggregation.

2.- **OrAccum**: these accumulators store boolean values. The aggregation process corresponds to an OR operation in such a way that the value stays *False* until any *True* value is added to the aggregation.

3.- **ListAccum**: these accumulator store any type of data in an array. The information is ordered as it is being introduced in the accumulator.

4.- **MapAccum**: this accumulator type admits any type of data. Its behavior is similar to an object, mapping the values to a given instance (generally a name (string)).

5.- **MinAccum**: this accumulator is devoted to store numerical type variables. It stores a single value with the minimum value introduced to the aggregation.

6.- **MaxAccum**: it is devoted to store numerical type values, storing the maximum value added to the aggregation.

7.- **SumAccum**: it is devoted to store numerical type values. Is saves the summation of all numbers added to the aggregation.

The accumulators can be declared in two ways:

- **Global Accumulators**: these accumulators represent a single global variable that can be modified at any moment during the query. These accumulators are represented by a name preceded by "@@". An example is shown in Figure 5-8.

- **Local Accumulators**: they act as temporal attributes to the vertices in the graph. So each vertex acts as an individual storage unit. These accumulators are represented by a name preceded by "@". This will be specially useful to

```
1 /* Declaration of the accumulator */
2 ListAccum<STRING> @@ListNames;
3 /* Example of use of a global accumulator to store the names of the
     found vertex set. */
4 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)-ToType:t ACCUM
     @@ListNames+=t.NAME;
```

Figure 5-8: Example of use of global accumulator(using a ListAccum).

```
1 /* Declaration and initialization of the accumulator */
2 OrAccum @Seen=False;
3 /* Example of use of local accumulators to mark the already seen
     nodes */
4 ResultSet = SELECT t FROM FromSet:f-(edgeType:e)-ToType:t ACCUM t.
     @Seen+=True;
```

Figure 5-9: Example of use of local accumulator (using an OrAccum).

"mark" the vertices accross the exploration process. An example is shown in Figure 5-9.

The main difference between ACCUM and POST-ACCUM is that ACCUM is devoted mainly for accumulating during the searching process while POST-ACCUM performs the accumulation after finalizing the searching process. This means that ACCUM can stores information of the "from vertex set", to "to vertex set" or of the edges at the same time, while POST-ACCUM can only accumulates information about the "from vertex set" or the "to vertex set", but not both at the same time.

As it was explained, the RESTPP API will be the main method used to perform the communication with the database from any external application. Using the previously explained GSQL Language, it will be programmed algorithms that perform any analysis to the database. To do so, it will be used parametrized queries. This queries allow the introduction of parameters that the user can use to run the same query (or analysis of the graph) taking the source of the searching process at different parts of the database. These starts will be introduced as parameter, therefore, the queries will act like functions.

An example of a simple parametrized query is shown in Figure 5-10. This query is divided in three main parts. The first part will be dedicated to the prototyping of

```
1  /* CREATION OF THE QUERY */
2  CREATE QUERY CountLines(STRING owner) FOR GRAPH
       DistributionGridGraph{
3      /* ---------- PART 1 ------------*/
4      /* DEFINITION OF CUSTOM TUPLES */
5      TYPEDEF TUPLE <UINT FUSE_KEY, STRING STATUS, STRING TYPE>
           FUSE_DATA_TYPE;
6      /* DECLARATION OF ACCUMULATORS AND INITIALIZATION */
7      SumAccum<INT> @@number_of_ct_of_owner = 0;
8      ListAccum<FUSE_DATA_TYPE> @@closedFusesInformation;
9      OrAccum @Seen=False;
10     /* ---------- PART 2 ------------*/
11     /* SELECTION OF A DATASET OF A WHOLE GROUP OF VERTEX TYPE */
12     Start = {CT.*};
13     /* LOOK FOR THE STATIONS THAT BELONG TO THE OWNER INTRODUCED BY
           PARAMETER
14         IT IS USED A GLOBAL ACCUMULATOR TO STORE THE COUNT */
15     Selected_ct = SELECT ct FROM Start:ct
16                             WHERE ct.PROPERTY==ct_name
17                             POST-ACCUM @@number_of_ct+=1;
18     /* LOOK FOR THE CLOSED FUSES DIRECTLY CONNECTED TO THE CTS THAT
           BELONG TO THE SELECTED OWNER
19         IT IS USED A LOCAL ACCUMULATOR TO MARK THEM
20         A GLOBAL ACCUMULATOR IS USED TO STORE THE SELECTED
               CHARACTERISTICS OF THEM */
21     ClosedFuses = SELECT f
22                   FROM Selected_ct:ct-(WITH_FUSE)->FUSE:f
23                   WHERE f.STATUS == "C"
24                   ACCUM @@closedFusesIDs+=f.MSLINK, f.@Seen=True;
25     /* LOOK FOR THE FUSES THAT WHERE NOT SEEN (THE OPEN ONES)*/
26     OpenFuses = SELECT f FROM Selected_ct:ct-(WITH_FUSE)->FUSE:f
27                          WHERE f.@Seen==False;
28     /*----------- PART 3 ------------*/
29     /* RETURN THE RESULT OF THE QUERY */
30     PRINT @@number_of_ct_of_owner, @@closedFusesInformation,
           OpenFuses;
31 }
```

Figure 5-10: Example of loading job creation for a simple graph.

variables and accumulators and their initialization. The second part will be dedicated to the expression of the algorithm needed to execute the required query. Finally, it is stated the variables to return from the created query.

5.1.2 TigerGraph graphical user interface

TigerGraph includes a web User Interface (UI) that runs by default on the port 14240 of the machine where TigerGraph is intalled. This web interface includes two main

applications: GraphStudio and Admin Portal.

The first one is the graphical tool that allows the user to perform all the previous explained works from a graphical environment (see Figure 5-11). GraphStudio includes five main tools: *Design Schema*, *Map Data to Graph*, *Load Data*, *Explore Graph* and *Write Queries*.

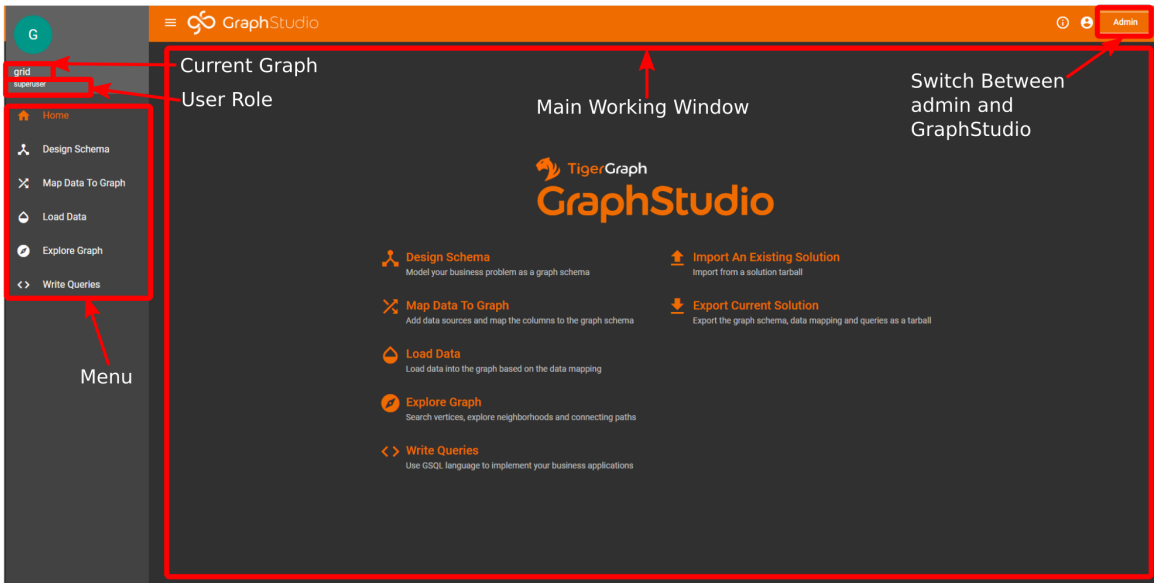


Figure 5-11: Snapshot of home window of GraphStudio.

Define Schema Tool

This tool is devoted to the definition of the graph structure. The application is shown in Figure 5-12. This process will be equivalent to the CREATE GRAPH command of GSQL.

Load data to graph

In order to load data to the designed graph it is necessary the use of two tools: the Map Data to Graph and Load Data.

In the Map Data to Graph two actions are made (see Figure 5-13). First, the files than contain the data should be uploaded to the server. In a second stage, it is necessary to define the relationship between files with vertices and edges. Once

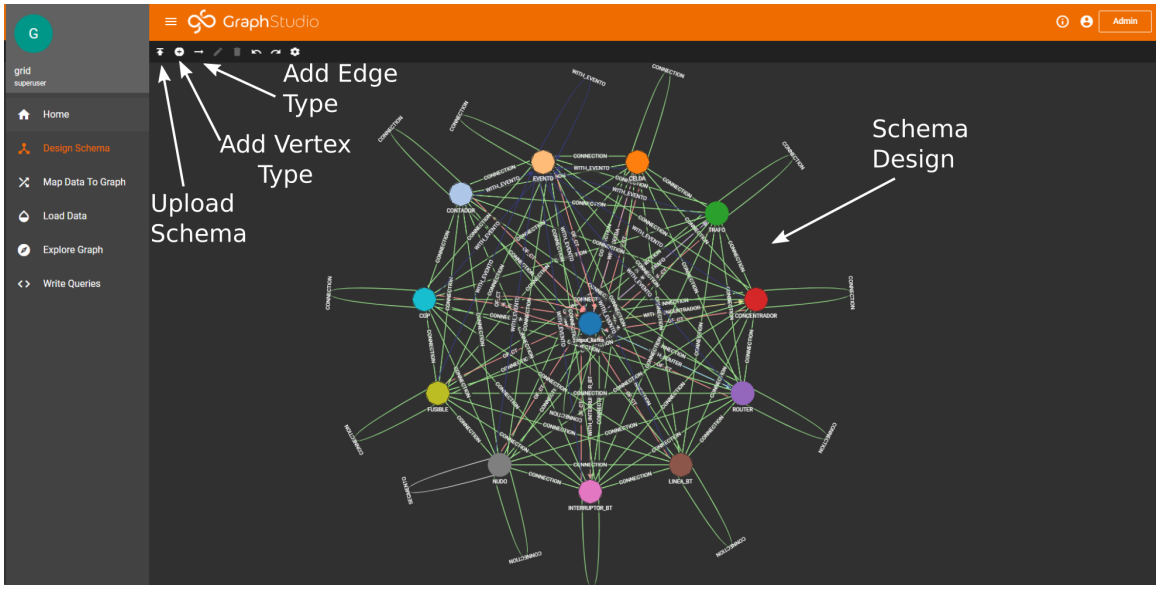


Figure 5-12: Snapshot of Design Schema tool of GraphStudio.

the affected edges and vertices by a file are defined it is needed the definition of the mapping rule, this is, relationship between the columns of the file and the attributes. This process is equivalent to the loading job creation using GSQL (see Figure 5-4).

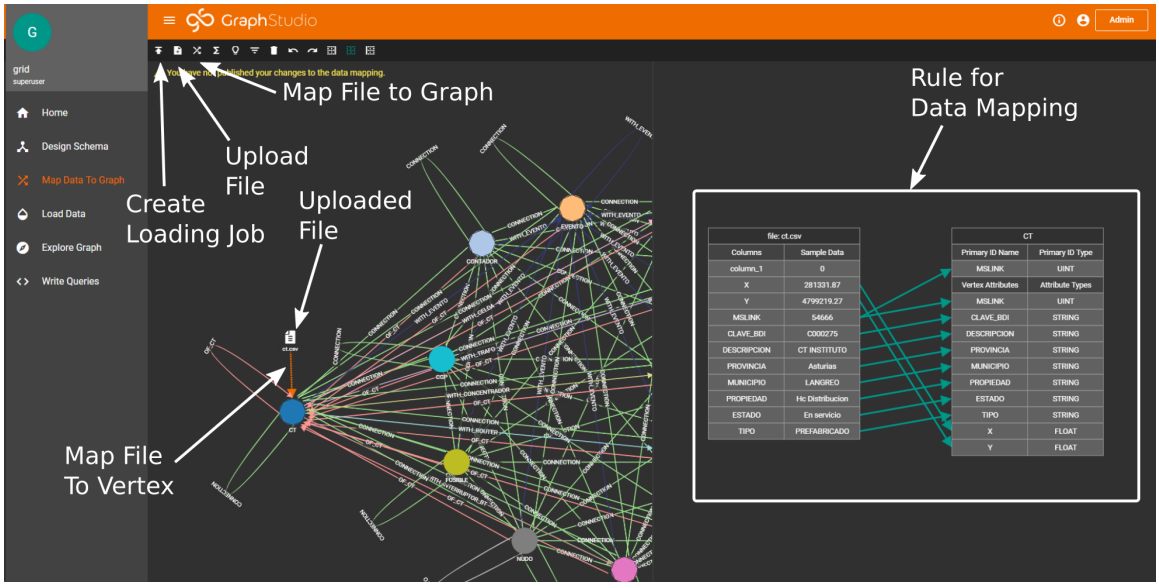


Figure 5-13: Example of data mapping functionality of GraphStudio for the case of a transformer station.

The second tool will be used to execute the previously created loading job. It is

simpler than the previous process, and in GraphStudio is shown the statistics about the loading job. This is equivalent to the RUN LOADING JOB command in GSQL (see Figure 5-4).

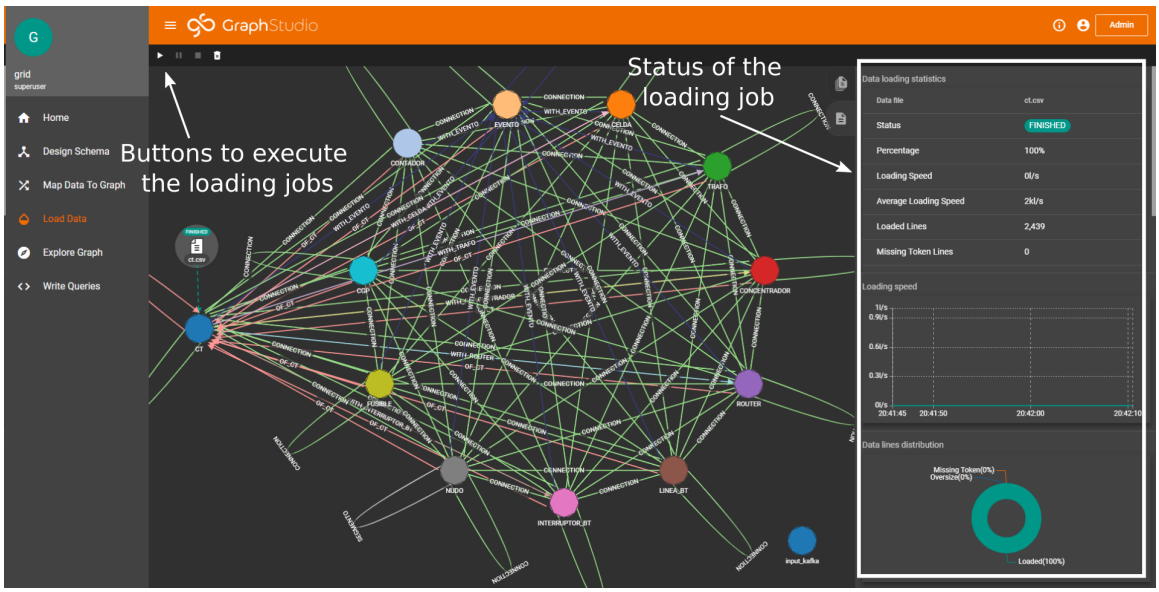


Figure 5-14: Load Data functionality in GraphStudio.

Graph exploration

This tool allows the graphical definition and running of simple queries. It is really usefull since it offers a tool to the "manual" exploration of the graph. This tool will be really usefull during the debugging process, the checking of the correct data upload process, searching for anomalies in the graph that may cause any problem with algorithms and so on.

In Figure 5-15 it is shown the part of the tool devoted to the search of vertices. This specific part is equivalent to the running of a SELECT statement in GSQL.

GSQL Query Creation

GraphStudio also includes a tool devoted to the creation and test of GSQL queries (see Figure 5-16). The main advantage of this tool is the highlighting of GSQL Language, the online semmantics check and automatic installation and optimization

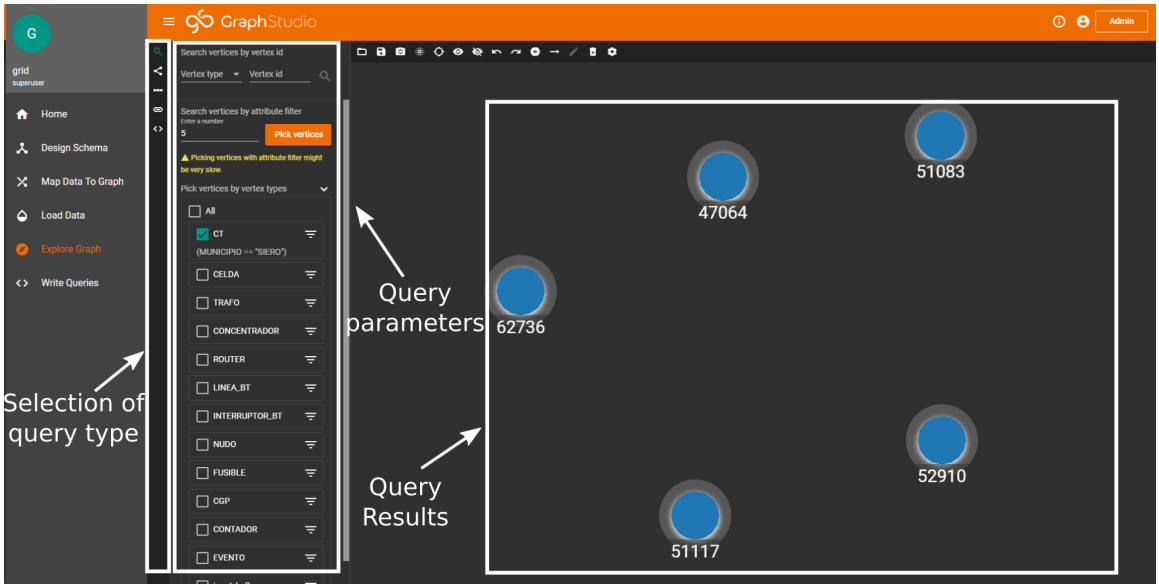


Figure 5-15: Exploration Graph functionality in GraphStudio.

to RESTPP API. The Write Queries tool also allows the run of the installed queries, returning a graphical representation of the result.

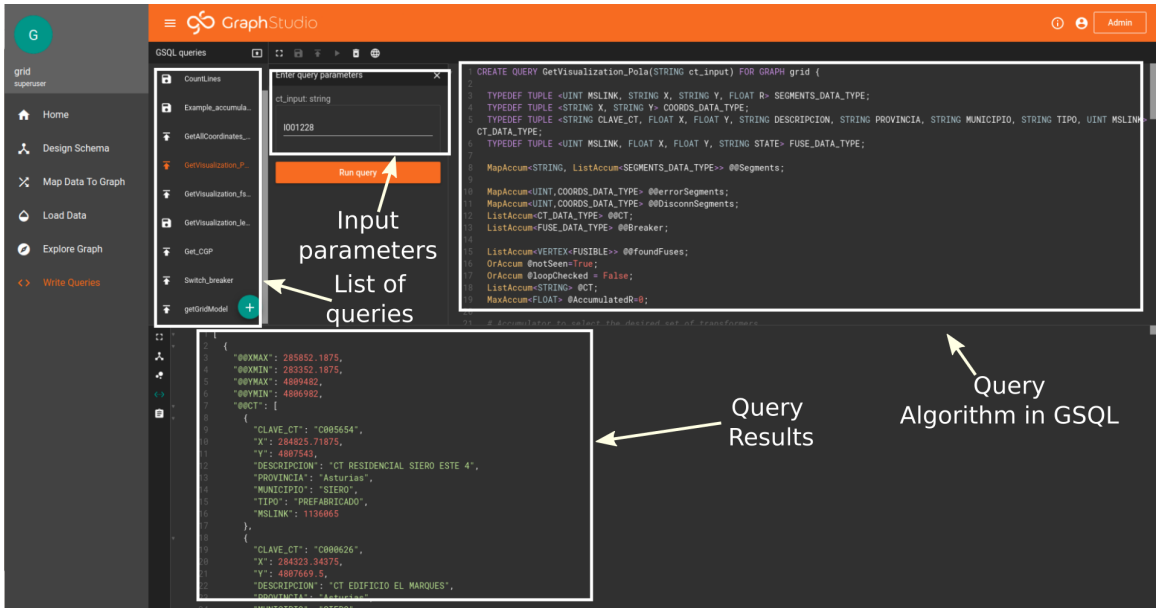


Figure 5-16: Write Queries tool.

5.2 Database creation

The structure shown in Figure 4-15 shows the most complete version of the schema to implement in TigerGraph. However, the data of all the shown elements was not available at the time of creating the database. Thus, a simpler version of the structure with the essential elements needed for the proposed analysis will be created.

Initially, the data was exported from the database currently on production in the utility (Oracle SQL) to CSV and Excel files. It was exported in 6 sets of files corresponding to the 6 areas in which the grid is divided: Gijón, Avilés, Oviedo, Valnalón, Occidente and Oriente. So the loading process was also performed in six stages with the same procedure.

1.- Pre-processing: The CSV files were adapted to the needed structure required by the database: files with vertices and files with edges. This stage was done using Python, with Pandas library. The adaptation of files of one area is shown in B.7.

2.- Loading Job creation and execution: Once the needed files were prepared, with all the required attributes and structure, a loading job was created to load this information to the database. This loading job should be adapted to the structure previously defined for the data files during the pre-processing stage. An example of loading job for the area of Gijón is shown in B.4.

5.3 Graph analysis

Once the database was created, with all the needed data mapped in a structure as similar as possible to the one defined in 4-15, it was designed and programmed the algorithms to perform the exploration and the data extraction of the grid.

Two algorithms were developed: the analysis based on registers and the analysis based on communications.

5.3.1 Topology exploration algorithm

As it was explained in section 3.2.5, the basic algorithm used to explore the graph will be BFS. This will be preferred since it will be the most suitable to exploit the parallelization capability of TigerGraph in future developments. So, algorithm 1 will be modified to this specific case, taking into account the structure of the graph (Figure 4-15).

The first consideration to take into account is that the BFS exposed was designed for an unidimensional graph, while in the current graph will be several types of vertices and edges. To analize this, the graph will be simplified. As the Transfomer Stations and End-User have a fixed structure, the control flow in these domains can be designed as a series of steps to query the different elements in a sequential manner. When reaching the interface elements to the network (fuses), the case is different. The exploration can only be performed with an iterative algorithm since the graph topology and number of steps to do is not defined. Hence, for the exploration algorithm it will only be taken into account the Fuse and ConnectivityNode vertex classes, limiting the dimensions of processed graph to two.

In Figure 5-17 it is represented a small part of the conductions of a distribution grid. As it is depicted, the source vertex of the algorithm will be a ConnectivityNode, and all the conductions accross ConnectivityNodes and Breakers must be explored.

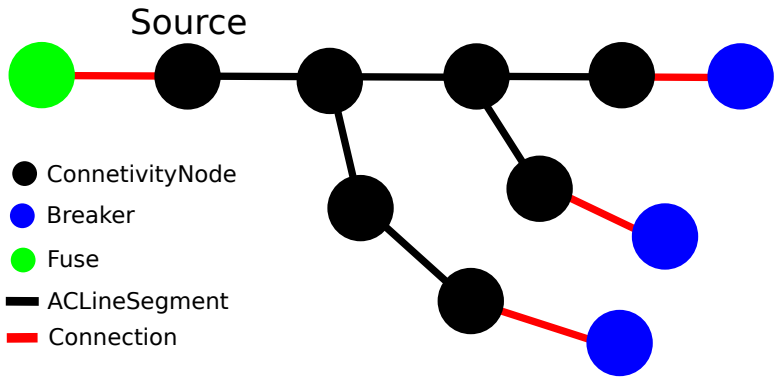


Figure 5-17: Representation of the considered graph for the graph exploration algorithm implementation.

Algorithm 3 Two Dimmensional Top-Down Breadth-First Search Algorithm for a single feeder.

```
Initialize frontierN as the ConnectivityNode set to analyze.
Define Source as ConnectivityNode connected to Fuse.
Add Source to frontierN.
Initialize list of already analyzed Breakers: parentsBR.
Initialize list of already analyzed ConnectivityNodes: parentsN.
Add frontier to parentsN.
while There are elments in frontierBR OR it is the first iteration. do
    while frontierN is not empty do
        for n in frontierN do
            Remove vertex n from frontierN
            Match neighborhood of n
            for neighbor in neighborhood do
                if neighbor was not already seen then
                    if neighbor is a ConnectivityNode then
                        Add neighbor to frontierN
                        Add neighbor to parentsN
                    else if neighbor is a Breaker then
                        Add neighbor to frontierBR
                    end if
                end if
            end for
        end for
    end while
    if frontierBR is not empty then
        for BR in frontierBR do
            Remove vertex BR from frontierBR
            if status of BR is "closed" then
                Match neighborhood of BR
                Add BR to parentsBR
                for neighbor in neighborhood do
                    if neighbor was not already seen then
                        Add neighbor to frontierN
                        Add neighbor to parentsN
                    end if
                end for
            else
                Continue
            end if
        end for
    end if
end while
```

The BFS algorithm can be used to return the connectivity of a given unidimensional graph, so it can be used to "compress" one of the dimensions of the graph. This way, it is possible to obtain the connectivity between Breakers (see Figure 5-18). Thus, it is possible to analyze a two dimensional graph using two nested BFS algorithms that iteratively analyze the ConnectivityNodes and the Breakers dimensions in a sequential manner.

This way, the Algorithm 1 of Section 3.2.5 can be modified in order to obtain the Algorithm 3. In this case there are two nested BFS algorithms: the first one, from lines 8 to 23, to explore the ConnectivityNodes dimension; and the second one, from lines 7 to 41, to explore the Breakers dimension. This way, it is possible to see that in this case the ConnectivityNode's dimension will be the "lower" dimension.

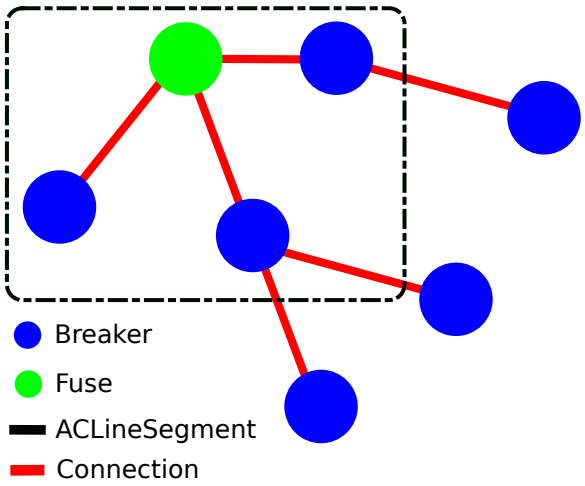


Figure 5-18: Representation of the breakers' dimension of a radial network. The dashed square delimits the network represented in Figure 5-17.

As it can be noticed, both loop structures are similar, with two differences:

1.- The inner BFS loop should collect the found vertices from both dimensions. This way, when this loop stops (because the inmediate "lower" dimension was already finished), the found Breakers' frontier is already available for the outer BFS.

2.- The Breakers' BFS should check the status of the device in order to filter the open ones.

3.- The Breakers' BFS do not collect vertices from the breakers dimension. This is a simplification specific for a distribution grid application, since no breakers are connected in series without a ConnectivityNode between them.

The Algorithm 3 was implemented in GSQL using code shown in the appendix B.1 (lines from 74 to 120).

5.3.2 Loop detection

As it was explained in section 3.2.4, the distribution grid is operated in a radial way, and the operator tries to keep it in a tree configuration. But sometimes this is difficoult to check due to the high number of breakers interconnecting the different stations as well as parallel electrical paths that appear along lines. The presence of these kind of structure makes difficoult to manage the communication infrastructure since two PLC concentrators are available from the same point. Moreover, this situation implies a security issue since single elements are fed from two different points, being necessary several operations to disconnect a selected part of the grid. For these reasons, a functionality was developed to detect the cyclic configurations. These cyclic configurations are commonly known as "loops".

The main difference between a path or tree structure and a cycle structure is that the last one contains an edge that connects one of the vertices "upstream" in the graph (see 3.2.4).

It is possible to detect two different kind of loops:

- Connection between two transformer stations. This kind of loops is the most dangerous since the secondary windings of two transformers are interconnected. An example of this is shown in Figure 5-19 in the loop caused by BR1 (loop 1).

- Connection between two elements fed by the same transformer station. Despite this type of configuration is avoided, it is not as dangerous as the previous one. This can be seen in Figure 5-19 in the loop cases caused by breakers BR3 or BR2 (loops 2 and 3).

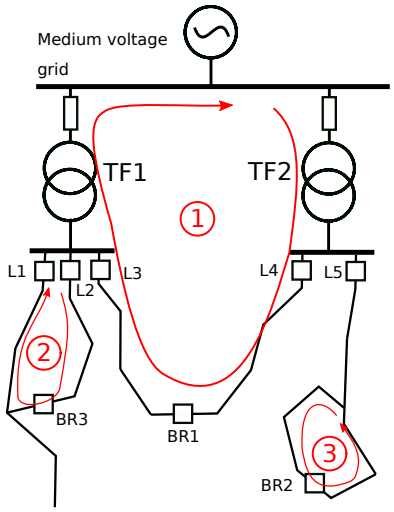


Figure 5-19: Different loop possibilities.

The implementation of this functionality will be easier with the use of accumulators in Tigergraph. To do so, it will be added local accumulators (ListAccums) to all ConnectivityNodes. These accumulators will store the transformer station mRID they are connected to. To implement it, the mRID will be copied to the next frontier vertices in each of the "hops" of the BFS algorithm (lines 15 and 17 of Algorithm 3).

Finally, during the BFS over the Breakers dimension, the if statement (line 31 of Algorithm 3) will be modified. It will be added an *elsif* statement to check the case when the ConnetivityNode was already seen but the mRID in the transformer list of the vertex doesn't match with the one comming from the other side of the breaker.

This way, if some loop between transformer stations appears, the list of mRIDs in the ConnectivityNodes will contain more than one element. Within the same if statement, when this situation is detected, it can be stored and stoped the BFS algorithm of that branch for a later analysis.

Finally, after finishing the BFS algorithm execution, the list of ConnectivityNodes that were detected to be in a loop configuration can be analyzed. In the present project case, the part of the grid affected by these loops was collected using again a BFS algorithm to lately represent them in a different color to notify such situation to the operator.

Notice that this method will only allow us to detect the loops of type 1 (see

Figure 5-19). The loops of types 2 and 3 cannot be represented since both sides of the breakers (BR3 and BR2) are fed from the same transformer station.

5.3.3 Disconnected segments detection

The main purpose of topology exploration algorithm is to collect the elements that are connected the same transformer station. However, this group doens't necessarily include the whole network since it is possible to have electrical islands in the network.

The network can have isolated segments if the segment doesn't have any connection to any transformer, or if the elements of connection are "open". In B.1, a local accumulator @notSeen is used in order to mark all the nodes where the algorithm has "passed". Hence, this accumulator can also be used to detect the segments of the grid that are disconnected from transformers (see lines from 124 to 129 of B.1).

5.3.4 Communications analysis

The main problem with the operation of the low voltage grid is that the majority of the breakers, as well as other elements as tap chagers are analog. This means that the only way to store their status is with the communication by the maintenance crew at the time of performing any change in the devices. This method is not reliable since sometimes the crews doesn't communicate the operations to the control center, the communicated information is wrong, or it is not correctly loaded to the system.

In Figure 4-15 is shown that the grid is formed by two main layers: the communication layer and the power layer. Both layers represent two networks that share the conducting equipment. This characteristic is unique for the PLC managed grids and, in this case, if the operator is capable to determine the topology of one of them, this topology must be common to both networks.

Due to this, it was implemented an application that tries to get the configuration of the switching devices of the grid only taking into account the information comming from the PLC network. If a smart-meter and a concentrator are communicating each other, it is possible to conclude that there is a physical connection between them.

This means that all the breakers along the electrical path between the smart-meter and the concentrator (or the transformer station) are closed.

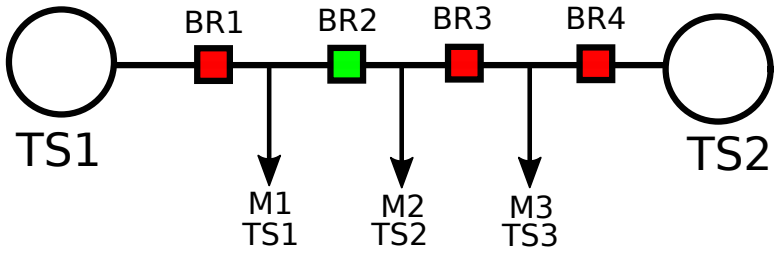


Figure 5-20: Simple example about the inference of breakers status using communications information.

An example of this is shown in Figure 5-20 where a set of smart meters (M1) are communicating with TS1, while the sets M2 and M3 are communicating with TS2. This way, it is easy to determine that the breakers BR1, BR3 and BR4 are closed, while the breaker BR2 must be open.

In Figure 5-20 it is shown the ideal case, and no doubt appears at the time of determining the configuration of the breakers. But there are two specific situations where the connectivity cannot be determined:

1.- Connectivity of a segment where there are no smart meters connected to. In this case, there isn't any device connected to the segment that ensures the physical communication between the elements. However, this situation gives some information. Taking as example the Figure 5-21 case, it is possible to determine that s1 and s3 are isolated from each other, using F1 and F2, so at least one of them must be disconnected (or both), but it is not possible to determine which one of them.

2.- Boxes with fuses or breakers. This case is similar to the previous one since in the conductors of the internal circuit of the switching boxes there is not any device that gives information about the communication status. This is a very common case along the grid. Taking the example of Figure 5-22, it is possible to determine that one of the outputs of the box is connected to trasformer B, while two of them are connected to transformer A. So the crossed breakers and

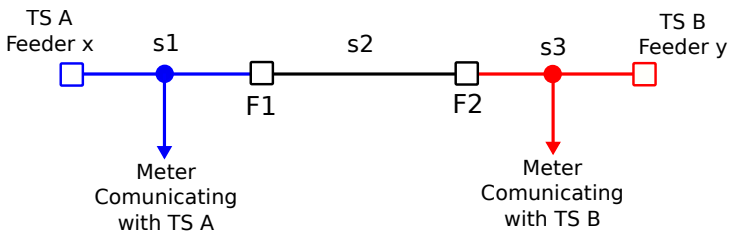


Figure 5-21: Structure of the grid of a line between two breakers with no connection points in it.

the not crossed breakers in the figure must be in complementary status, but it is not possible to determine if they are open or closed.

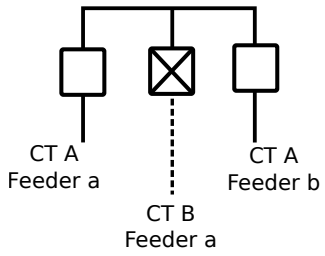


Figure 5-22: Fuse box between segments fed from two different transformer stations.

3.- Loops. When having loops of types 2 or 3 (see Figure 5-19), it is not possible to determine its presence, and the estimated positions of the breakers will be random. When having loops of type 1, the estimated configuration of the grid will not be accurate. In these cases the meters will have access to two or more concentrators, and normally they will report to one of them randomly. So the communications anlysis algorithm will determine the physical connections according to the communications, not being able to determine a presence of a loop configuration.

The algorithm to analyze the communications will not take into account the information stored about the breakers status. In this case the BFS algorithm on the breakers dimension will not contain the filtering of the open breakers. Instead of that, it will be used the aggregated information about the connection points found along the BFS on the ConnectivityNodes. The relationship that will be added to the me-

ters attributes will be extracted from the G02 report comming from the concentrator standard reports (see Section 2.6).

This procedure will be implemented using two boolean indicators, implemented as accumulators in TigerGraph.

- **@@stillConnected**: this indicator will be set to True while the found meters in the network report to one of the expected concentrators (the ones installed in the station where the algorithm is started).

- **@@anyConnected**: this indicator is set to True in any of the found meters is communicating with one of the expected concentrators.

So the BFS exploration on the ConnectivityNodes can be implemented as shown in Algorithm 4, adding the communications indicators (lines from 6 to 13).

Algorithm 4 Breadth-First Search Procedure for ConnectivityNodes with communications analysis.

```
 1: procedure BFSCN_Com(frontierN)
 2:     while frontierN is not empty and @@stillConnected do
 3:         for n in frontierN do
 4:             Remove n from frontierN
 5:             Match Connection Points connected to n
 6:             for CP in the found Connection Points do
 7:                 Get concentrator reference reported in CP: CNC_CP
 8:                 if CNC_CP matches with ref_CNC or is in @@ref_Port  then
 9:                     @@anyConnected = True
10:                 else
11:                     @@stillConnected = (@@stillConnected OR False)
12:                 end if
13:             end for
14:             Add ConnectivityNodes connected to n to frontierN
15:             Add Breakers connected to n to frontierBR
16:         end for
17:     end while
18:     return frontierBR, @@stillConnected. @@anyConnected
19: end procedure
```

This BFS will be run over each of the conducting parts of the networks between the breakers. When this procedure finishes there are four possible cases:

1.- @@stillConnected and @@anyConnected are True. In this case, there were meters found connected to the segments and they are communicating with the expected station.

2.- @@stillConnected is False and @@anyConnected is True. This means that some of the found meters are communicating with the expected station, but some others not. This situation corresponds to any inconsistency in the information. In these cases it will be necessary to use a specific handler to decide how to interpret this situation.

3.- @@stillConnected and @@anyConnected are both False. This means that all the found meters are not communicating with the expected concentrator. In this cases, the analyzed segment is not connected to the analyzed station's network, so that part of segments will be removed and the previous breaker will be assumed as open. This situation will also indicate the end of the feeder, being used to break the loop.

4.- @@stillConnected is True and @@anyConnected is False. This case is given when no meters were found in the segment. It is the case shown in Figure 5-21, segment s2. In this project, as it cannot be determined, it will be assigned to the first transformer station from where it is analyzed. So in practice it will be randomly assigned to any station.

Due to the different possibilities, at the time of collecting the ridden segments, it will be needed to create a temporary variable to store them. When BFS of ConnectivityNodes is finished, it is decided if this variable is copied to the returned one or not. This method is expressed in the Algorithm 5.

Wrangling of found inconsistencies.

Among the possible situations found in the communication status along a line, the most problematic part is formed by the number 2.

Algorithm 5 BFS exploration algorithm of the distribution grid using communications data for a single transformer station.

Input: mRID of CT, Graph

```
Initialize @@stillConnected to True
Initialize @@anyConnected to False
Initialize frontierBR and frontierN
Select CT as the transformer station with the input mRID
Store the reference of the main concentrator to refCNC
Store the references of the portable concentrators to @@refPort
Request the lines outgoing from CT and store them to AllLines
for line_id in AllLines do
    Select the first ConnectivityNode of line_id to StartN
    Initialize frontierN
    Store StartN to frontierN
    while frontierN is not empty or it is the first iteration do
        [frontierBR,@@stillConnected,@@anyConnected]←BFSCN_Com(frontierN)
        if @@stillConnected then
            for BR in frontierBR do
                Request ConnectivityNodes connected to BR
                for CN in the found ConnectivityNodes do
                    if CN was not already seen then
                        Add CN to frontierN
                    end if
                end for
            end for
        else if @@anyConnected and not @@stillConnected then
            Run error handling procedure for unconsistent data.
        else
            Clear frontierBR and stop analyzing the feeder.
        end if
    end while
end for
```

This situation can be given with a loop configuration of the grid, in such a way that some of meters communicate with one station and the rest to the other, despite all of them are connected to both.

These cases are also given when there are switching boxes between several stations. In the boxes, as the conductors are near from each other, it appears magnetic coupling between conductor, which can cause the communication transfer between transformer stations.

But in these situations, the most common case is that the inconsistencies come

from an error in the registers of the connection points installations. In this case, the error could be given by the wrong information about the topology of the grid. Moreover, sometimes the relationship between client and smart meter is wrong, being the smart meter actually placed in a different part of the grid. This last case is the most usual, so it will be the analysis implemented by default as handling procedure for inconsistent data.

This analysis will be based on the information provided by the Base Node Information report (S11) (see Section 2.6). As it was explained, the smart meters can communicate through another smart meter that is closer to the concentrator (called switch), acting this one as repeater. This is a common situation, and only the smart meters closer to the station communicate directly to the concentrator. From field works, it was determined that the switch is geographically near to the position of the meter. So the procedure is normally to look for the switch position, and a crew is moved to the surroundings in order to determine the exact position of the smart-meter.

The Algorithm 6 shows the implementation of the inconsistencies handler according to the described criteria.

Algorithm 6 Procedure to handle the inconsistencies in communications data.

1: **procedure** $UnconsistenciesHandler(ErrorNodes)$
2: Create object to store the found position of meters according to registers and the position according PRIME reports: *positions*
3: **for** n in $ErrorNodes$ **do**
4: $Meters \leftarrow$ Get the meters connected to n.
5: $Concentrator \leftarrow$Get the concentrator reference from $Meters$
6: $Switch_ID \leftarrow$Get the switch reference from $Meters$
7: $Switches \leftarrow$ Look for all the meters communicating with $Concentrator$ and whose STATE is "Switch".
8: $ActualSwitch \leftarrow$Get the meter in $Switches$ where the SSID is $Switch\_ID$
9: Store the found node and the node where $ActualSwitch$ is connected to.
10: **end for**
11: **end procedure**

5.4 Visualization module

The visualization module will be built as a web application implemented using JavaScript and HTML to create a simple interface in order to represent the output of the previous explained analysis.

TigerGraph will be used for the topology processing of the grid using queries shown in B.1 and B.2. As explained, these queries are installed in the RESTPP API of TigerGraph, in such a way that they can be executed using this API. This API runs over port 9000 of the machine that alocates the TigerGraph service.

These queries return the grid's geographical information grouped according to the result of the connectivity analysis. The result will be returned into a JSON format with five keys: *@@Segments*, *@@errorSegments*, *@@CT*, *@@Breaker*, *@@SegmentsPortatiles* and *@@DisconnSegments*.

@@Segments

This element will contain the segments grouped by transformer station. Within each station key entry, it will be a list of object with the MSLINK and the coordinates of the vertices of each segment connected to the station.

```
{"I001095":[{'MSLINK': 917353,
  'X': '270167.08|270166.36',
  'Y': '4813011.82|4813012.8'},
 {'MSLINK': 2467008,
  'X': '270166.36|270158.46|270143.47',
  'Y': '4813012.8|4813023.55|4813017.76'},
 {'MSLINK': 2467033,
  'X': '270143.47|270142.39|270141.73|270142.54',
  'Y': '4813017.76|4813019.76|4813019.51|4813017.4'},
 ...],
 ...}
```

Figure 5-23: Example of segments information coming from the topology analysis module in TigerGraph.

@@SegmentsPortatiles

This element has the same structure as @@Segments. It is only present in the output of the communications analysis algorithm in order to identify the segments that communicate with portable concentrators.

@@errorSegments

The segments that are detected to be in a loop (in the case of the analysis according registers) or in a segment with inconsistent data (in the case of the communications analysis), they will be stored in this entry. The main role of the entry is to store the elements that will be highlighted in any manner in order to indicate the operator that some issue is happening with the elements.

The format of this element will be similar to *@@Segments*, but in this case it will be a dictionary where the MSLINK of each segment is the key. Each element contains the arrays of the coordinates.

```
1 {2823503:{'X': '265320.85|265320.52|265319.97', 'Y': '
    4804053.31|4804051.11|4804050.77'},...}
```

Figure 5-24: Example of an error segment information comming from the topology analysis module in TigerGraph.

@@CT

This element will be a list of objects with three attributes: the key of the station, and the X and Y coordinates of its geographical position.

```
1 [{'CLAVE_CT': 'I000206', 'X': 281909.4375, 'Y': 4805080.5},...]
```

Figure 5-25: Example of CT information returned by the TigerGraph query.

@@DisconnSegments

This object is structured in the same way as @@errorSegments. In this case it will contain the segments that are detected not to be connected to any station.

```
{ 2754417: {'X': '267163.49|267169.73267163.49|267169.73',
            'Y': '4804676.9|4804684.284804676.9|4804684.28'},
      ...}
```

Figure 5-26: Example of a disconnected segment information returned by the TigerGraph query.

@@Breaker

This object will contain a list of objects with the MSLINK of all the Breakers in the grid, X and Y coordinates, and the status of the breaker.

```
[{'MSLINK': 12044, 'X': 269683.34375, 'Y': 4805135.5, 'STATE': 'C'
   },...]
```

Figure 5-27: Example of breakers information returned by the TigerGraph query.

The output of the topology processor will be the geographical data of the selected transformer stations, grouped by subnetwork. This way, each breaker and segment will be related with a transformer station.

In order to represent it in a visual manner, it will be created a web application that reads the output information from TigerGraph, and converts it into a SVG interactive image. To implement it, a simple HTML file (shown in B.5) was created with the basic structure to place the SVG canvas in the middle of the screen and execute the JavaScript code that makes the query and plots the response.

Once the structure of the document is defined, the JavaScript code was created. The main functions added in this file are: the query to REST API used as the interface to TigerGraph, the post-processing of the JSON response, and the creation of SVG image that represents the grid. This file is shown in B.6. For the creation of the interactive SVG representation of the grid, it was used the library d3js [4].

5.4.1 Topology according registers visualization

In Figure 5-28 it is shown the visualization module for the representation of the main elements. The positions of the transformer stations will be marked using circles. The breakers will be represented as squares with the color depending on its status

according registers. The selected color code for the breakers is green for the open ones, and red for the closed ones. Next to each of the transformer stations it will be indicated the BDI key of the station in order to identify it.

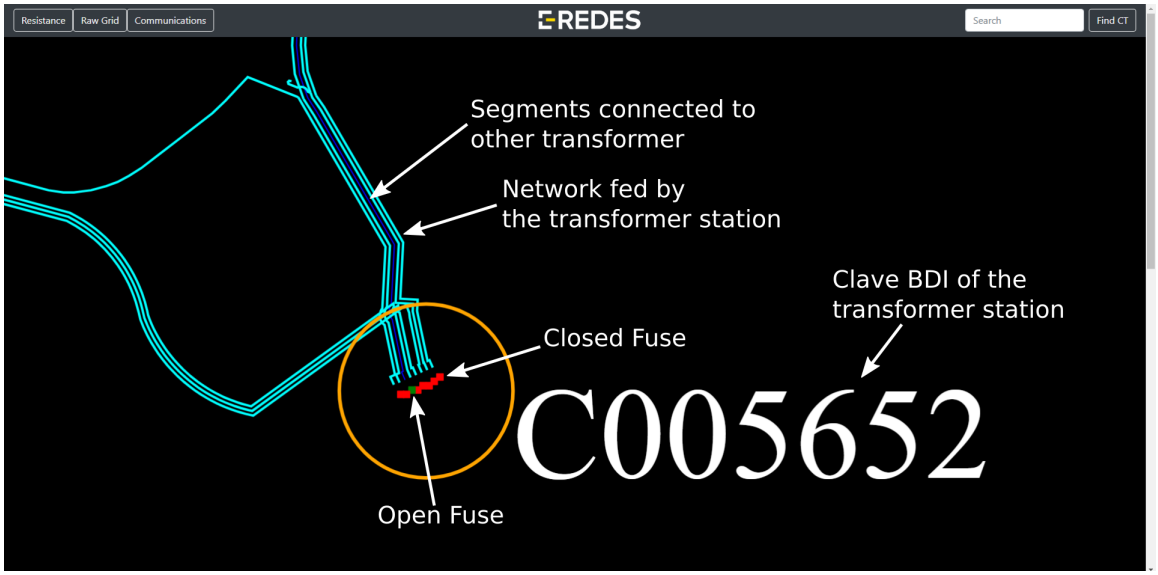


Figure 5-28: Representation of main elements of a selected transformer station in the visualization module.

Initially the segments will be represented in dark blue. In order to check the connectivity, it was added a mouse-over functionality that looks for all the "path" elements in the canvas and changes its color to cyan. This way it can be distinguish the part of the network connected to the same transformer station (see Figure 5-29). In Figure 5-28 it is shown how this functionality allows us to distinguish between the lines that are connected to a specific transformer station and which not, and how this information matches with the defined fuses status.

The information of @@errorSegments (the loops) is represented in a different color in order to indicate the operator that a loop was detected in that part of the network. In Figure 5-29, it is shown a loop detected between transformer stations C000601 and C001635.

Finally, the disconnected segments (object @@DisconnSegments) is represented with grey lines (see Figure 5-29).

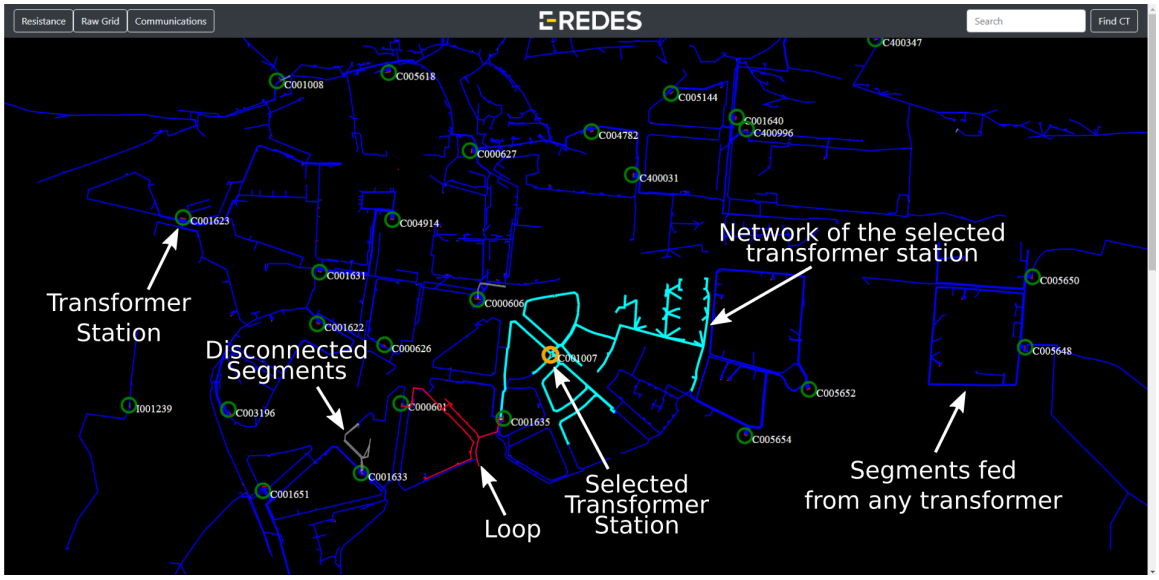


Figure 5-29: General grid representation according registers information.

5.4.2 Communications analysis visualization

For the visual representation of the communication analysis, a similar representation than the in register case was used. This query was only executed on the grid of Pola de Siero, since it is the only part where the PLC communications was provided in order to run the initial tests.

As it is depicted in Figure 5-30, more segments than in the previous case appear as "disconnected". This means that those segments are communicating with other transformers not available according to the provided topology. This can be caused by several reasons: bad representation of the grid, coupling of communications in the switching boxes, no available communications...

In this case, the segments that communicate with portable concentrators were highlighted in brown. These segments correspond to the parts of the network where there are communiction problems. This way the operator can easily identify them.

As it was explained, with communication analysis the loops cannot be detected. Because of this, the @@errorSegments object is used to indicate the segments where some unconsistent data was found. As shown in Figure 5-30, one was found connected to C000627. These unconsistencies were detected and handled during the algorithm

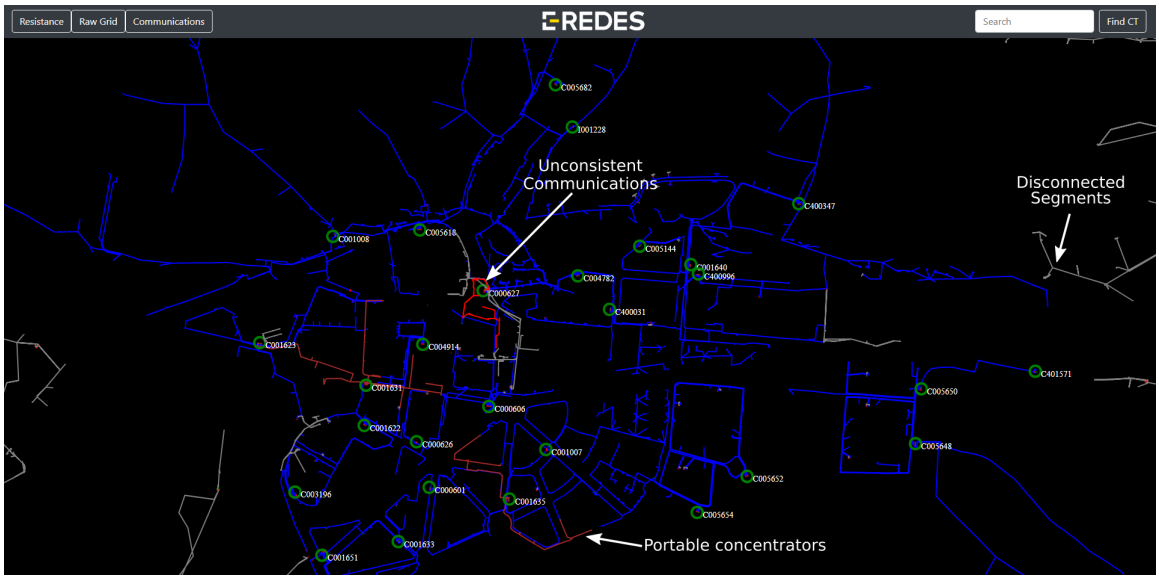


Figure 5-30: Visualization of the grid according communications.

execution (see code of B.2) in order to get the position of the problematic connection point and the position of the PRIME switch of the problematic meters. In the visualization tool, a mouseover function was added to the error segments in order to show these calculated positions (see Figure 5-31).

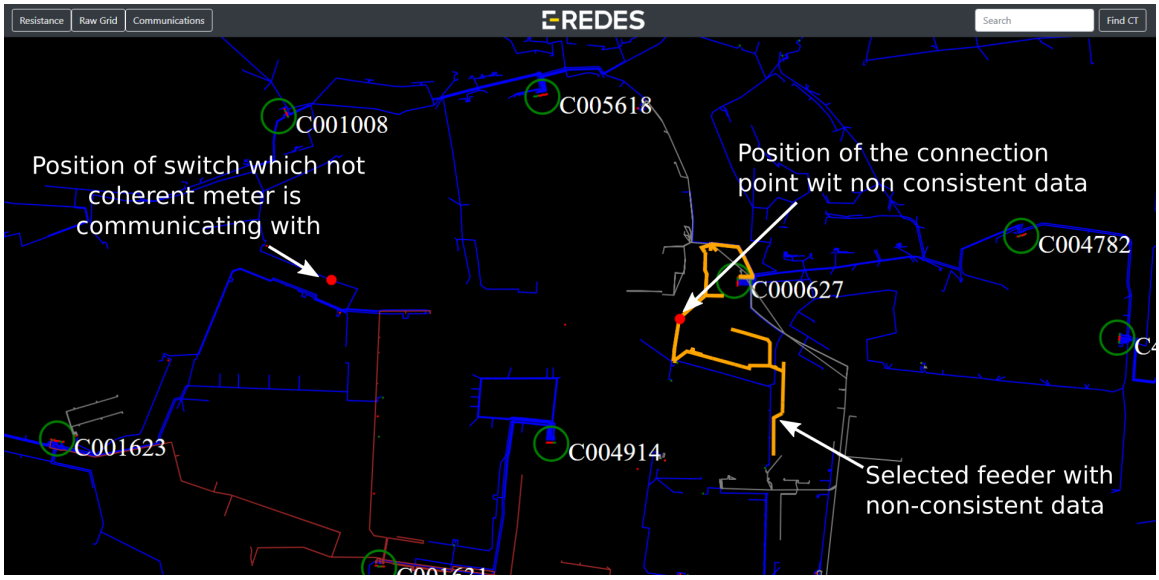


Figure 5-31: Visualization of a inconsistencies handler resolution.

In Figure 5-31 it is shown that a meter registered in a connection point connected

to C000627 is communicating with a switch in C001008. This is a clear example of an error in the registers of the meters. The detected connection point of C000627 is placed somewhere near to the indicated switch.

5.5 Performance

As a final test, it was performed a massive query over the grid. This query will show the performance of the system working with big ammounts of data. In Figure 5-32 is shown the performance analyzer result after running the topology analysis over the whole grid in the database.

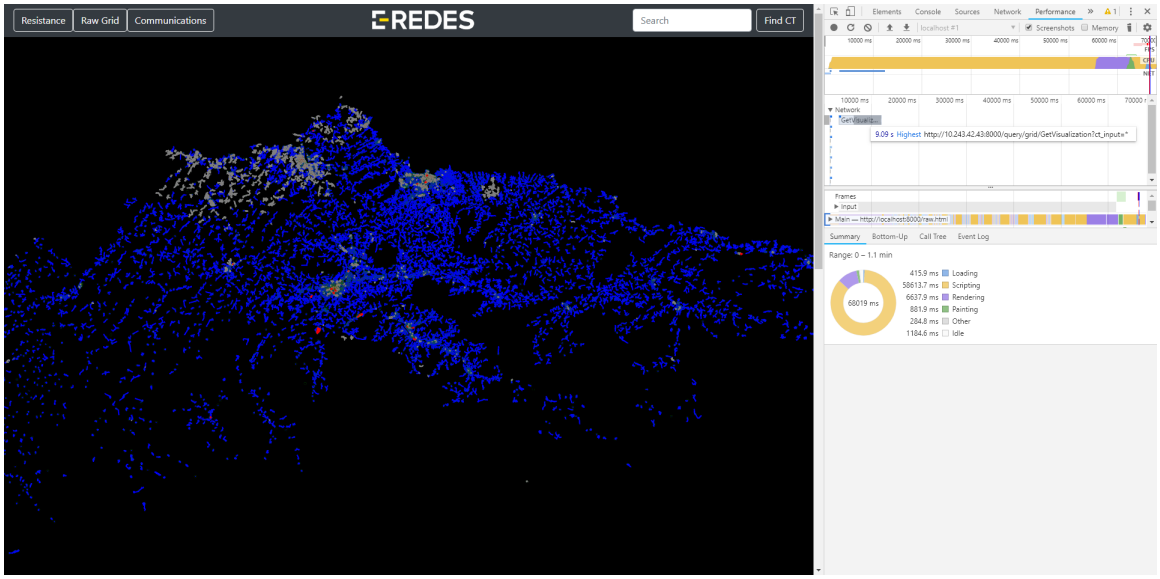


Figure 5-32: Performance of the application with a massive query over the whole grid of EDP in Asturias.

The visualization lasts a total of 68 seconds, with the processes execution times showed in Table 5.1. It can be concluded that the topology analysis is made in a couple of seconds, and that the downloading time from database machine is significant. Moreover, it is clear that the JSON processing and creation of SVG elements in the front-end takes the major part of the execution time of the application.

Table 5.1: Timing of processes execution during a massive query.

Job	**Time**
Loading Script	0.42 s
Algorithm execution	3.84 s
Downloading of result	5.25 s
Scripting	58.61 s
Rendering	6.64 s
Painting	0.88 s
Other	0.28 s
Idle	1.18 s

Chapter 6

Conclusions and Future Work

In this thesis it was developed a distribution low voltage network data management system based on graph databases. The conclusions extracted from the present work are the following:

- It was demonstrated the important advantages in terms of processing time that graph databases can reach. This was theoretically developed and lately demonstrated, achieving massive network processing times in the order of seconds.

- The CIM model of IEC 61970 can be adapted to a graph structure. To do son, it is needed to perform some modifications in order to adapt the structure proposed in the standard to the information managed by the EMS in the utility. This procedure will allow the development of easier interfaces with multi-agent applications.

- The PLC communication protocol was demonstrated to be useful at the time of supervising the topology information in the database. This was demonstrated for a practical case in Pola de Siero, and offers the operator two versions of the same information to compare and contrast in order to improve the available information.

- A simple visualization module was developed. The main purpose of this module was to easily check the correct performance of the topology analyzer. Never-

theless, it can be concluded that this module must be improved in order to achieve real-time operation. This module constitutes the current bottleneck of the developed system.

From the above mentioned conclusions, a sort of work lines can be defined in order to improve the achieved results:

- An improvement of the visualization module is needed. This module shoud be focused on allowing the user the concrete definition of the information required. This way, the returned information can be limited or compressed depending on the required visualization.

- CIM interfaces modules can be developed in order to allow this system to communicate with other applications in a standarized way. This interface should require minor adaptations of the incoming and outcoming documents due to the CIM oriented internal data structure. This way the utility will be able to import/export data with other agents in the system, which can be a useful feature depending on the future regulations framework evolution.

- The PLC communication protocol was demonstrated to be usefull in a simple case, but a deeper research on this relationship communications-topology can be done in this line. Some data as the quality of communication, communication time, correlations between consumptions... can be performed in order to extract a deeper understanding of the state of the grid.

- The creation of a system to manage the timestamped data, as well as the linking module with the system developed in this project. This would represent a complete data management system of a real time EMS. The combination of the developed fast topology processing and the fast request of timestamped variables can give deeper understanding of the state of the gird.

Appendix A

Tables

A.1 STG-DC reports list

Table A.1: Statistic reports defined by STG-DC.

Report	Content
G01	Hourly statistics of meters communication
G02	Daily statistics of meters communication
G03	Average curve of voltage, current and power
G04	Maximum curve of voltage, current and power
G05	Minimum curve of voltage, current and power
G06	Instantaneous curve of voltage, current and power
G07	Average curve of unbalances and harmonics
G08	Parameters of the meters
G09	Digital input and output parameters

Table A.2: Data straction reports defined by STG-DC.

Report	Content	Type
S01	Instantaneous readings of the meter	Synchronous
S02	Hourly incremental load curve	Asynchronous
S03	Absolute daily load curve	Asynchronous
S04	Monthly close	Asynchronous
S05	Daily close	Asynchronous
S06	Meter parameters	Asynchronous
S07	Voltage faults	Asynchronous
S08	Current faults	Asynchronous
S09	Events register of the meter	Asynchronous
S10	Intrusive equipment detected in the network	Asynchronous
S11	Node base information	Asynchronous
S12	Concentrator parameters	Asynchronous
S13	Espontaneous events of the concentrator	Asynchronous
S14	Hourly voltage and current profiles	Asynchronous
S15	Espontaneous events of the concentrator	Asynchronous
S17	Register of espontaneous events of the concentrator	Asynchronous
S18	Cut and reconnection confirmation	Asynchronous
S19	Firmware update confirmation	Asynchronous
S20	List of managed meters	Asynchronous
S21	Instantaneous measurements of the meter (advanced)	Synchronous
S22	Concentrator firmware update confirmation	Asynchronous
S23	Contract defined for the meter	Asynchronous
S24	Present meters in the network	Asynchronous
S26	Instantaneous values on demand	Asynchronous
S27	Monthly close on demand	Synchronous

Appendix B

Source-code

B.1 Tigergraph Exploration Algorithm with Loop Detection

```
1  CREATE QUERY GetVisualization(STRING ct_input) FOR GRAPH grid {
2  /*University of Oviedo
3  LEMUR Research Group 2020
4  Author: Adrian Miranda Paz
5
6  Query used to get and synthetize the data of a portion of the grid.
       */
7  /*User-defined tuples declaration for selecting the needed
       attributes in the return of the query.*/
8  TYPEDEF TUPLE <UINT MSLINK, STRING X, STRING Y, FLOAT R>
       SEGMENTS_DATA_TYPE;
9  TYPEDEF TUPLE <STRING X, STRING Y> COORDS_DATA_TYPE;
10 TYPEDEF TUPLE <STRING CLAVE_CT, FLOAT X, FLOAT Y> CT_DATA_TYPE;
11 TYPEDEF TUPLE <UINT MSLINK, FLOAT X, FLOAT Y, STRING STATE>
       FUSE_DATA_TYPE;
12 TYPEDEF TUPLE <UINT CLAVE_CGP, FLOAT X, FLOAT Y, STRING CLAVE_CT>
       ACOMETIDA_DATA_TYPE;
13 /*Declaration of global and local accumulators:
14 Global Accumulators : @@
15 Local Accumulators: @*/
16 MapAccum<STRING, ListAccum<SEGMENTS_DATA_TYPE>> @@Segments;
17 MapAccum<UINT, ListAccum<STRING>> @@Meters_Nodes;
18 ListAccum<ACOMETIDA_DATA_TYPE> @@Acometidas;
19 MapAccum<UINT,COORDS_DATA_TYPE> @@errorSegments;
20 MapAccum<UINT,COORDS_DATA_TYPE> @@DisconnSegments;
21 ListAccum<CT_DATA_TYPE> @@CT;
22 ListAccum<FUSE_DATA_TYPE> @@Breaker;
23 ListAccum<VERTEX<FUSIBLE>> @@foundFuses;
24 ListAccum<UINT> @@CTS;
25
26 /* Variable usada "para no dar marcha atras".*/
27 OrAccum @notSeen=True;
28 /* Variable para comprobar loops */
29 OrAccum @loopChecked = False;
30 /* Nos dice a que CTs esta conectado un nudo*/
31 ListAccum<STRING> @CT;
32 /* Resistencia acumulada hasta el nudo*/
```

```
33 MaxAccum<FLOAT> @AccumulatedR=0;
34 ListAccum<STRING> @Meters;
35
36 /*Acumuladores usados para filtrar por coordenadas*/
37 FLOAT DX = 2500.000; /*Margen horizontal de la ventana*/
38 FLOAT DY = 2500.000;  /* Margen vertical de la ventana*/
39 MaxAccum<FLOAT> @@XMAX;
40 MinAccum<FLOAT> @@XMIN;
41 MaxAccum<FLOAT> @@YMAX;
42 MinAccum<FLOAT> @@YMIN;
43
44 /########## Seleccion del grupo de inicio #################
45 Al iniciar una query, el primer paso es definir en que parte del
     grafo
46 queremos empezar. En este caso hay 3 posibilidades:
47 1.- "*": el "grupo de inicio" son todos los CTs
48 2.- {NOMBRE_DE_MUNICIPIO}: el "grupo de inicio" son los CTs en ese
     municipio
49 3.- {CLAVE_DE_UN_CT}: el "grupo de inicio" es el CT seleccionado y
     los de los
50 alrededores (dentro de los margenes definidos al principio).
51 */
52 CTs = {CT.*};
53 IF ct_input!="*" THEN
54   CTs_MUNICIPIO = SELECT ct FROM CTs:ct
55                                 WHERE ct.MUNICIPIO==ct_input;
56   IF CTs_MUNICIPIO.size()==0 THEN
57     /* CT que se ha buscado */
58     MainCT=SELECT ct FROM CTs:ct
59                       WHERE ct.CLAVE_BDI==ct_input
60                       POST-ACCUM @@XMAX+=ct.X+DX/2,@@XMIN+=ct.X-DX/2,
                            @@YMAX+=ct.Y+DY/2,@@YMIN+=ct.Y-DY/2;
61     /* Transformer stations of the surroundings */
62     CTs = SELECT ct FROM CTs:ct
63                       WHERE (ct.X<@@XMAX AND ct.X>@@XMIN AND ct.Y<
                            @@YMAX AND ct.Y>@@YMIN);
64   ELSE
65     CTs = SELECT ct FROM CTs_MUNICIPIO:ct
66                       POST-ACCUM @@XMAX+=ct.X+DX/2,@@XMIN+=ct.X-DX/2,
                            @@YMAX+=ct.Y+DY/2,@@YMIN+=ct.Y-DY/2;
67   END;
68 END;
69
70 /* BEGGINING OF THE EXPLORATION ALGORITHM
71 The first step is to look for the fuse board of the transformer.
72 Take advantage to store the station characteristics in @@CT
     accumulator (for output)*/
73
74 Celda = SELECT c FROM CTs:ct-(WITH_CELDA)->CELDA:c
75                   WHERE ct.PROPIEDAD=="Hc Distribucion"
76                   ACCUM c.@CT+=ct.CLAVE_BDI, @@CT+=CT_DATA_TYPE(ct.
                        CLAVE_BDI,ct.X,ct.Y);
77 /* Request of the list of all lines' fuse fed by the low voltage
     board. Store them in @@Breakers*/
78 Lines = SELECT l FROM Celda:c-(CONNECTION)-LINEA_BT:l
79                   ACCUM l.@CT=c.@CT, @@Breaker+=FUSE_DATA_TYPE(l.
                        MSLINK, l.X, l.Y, l.ESTADO_REAL);
80 /* Select only the closed lines */
81 Lines = SELECT l FROM Lines:l WHERE l.ESTADO_REAL=="C";
82 /* Selection of the initial node of each feeder */
83 Nudo = SELECT n FROM Lines:l-(CONNECTION)-NUDO:n
84                  ACCUM n.@notSeen=False, n.@CT=l.@CT;
85
86 WHILE True LIMIT 3000 DO /* Infinite loop to stop with the break */
87   @@foundFuses.clear(); /* Reset the list of fuses */
88   WHILE Nudo.size()>0 DO /* While not all nodes were analyzed... */
89     /* Look for the CGP connected to the current node */
```

```
90      CGPlist = SELECT cgp FROM Nudo:n1-(CONNECTION)-CGP:cgp
91                           ACCUM @@Acometidas+=ACOMETIDA_DATA_TYPE(cgp
                                .CLAVE_BDI, n1.X, n1.Y, n1.@CT.get(0));
92      IF CGPlist.size()>0 THEN
93        CGPlist =SELECT cgp FROM CGPlist:cgp-(CONNECTION)-CONTADOR:m
94                             ACCUM cgp.@Meters+=m.REFERENCIA;
95        Nudos_costumer = SELECT n FROM CGPlist:cgp-(CONNECTION)-NUDO:n
96                                  ACCUM @@Meters_Nodes+=(n.ID->cgp.
                                      @Meters);
97      END;
98      /* Closed fuses connected to the current nodes set */
99      Fuses = SELECT fuse FROM Nudo:n1-(CONNECTION)-FUSIBLE:fuse
100                          WHERE fuse.ESTADO_REAL=="C" AND (fuse.
                                @notSeen==True OR fuse.@CT.get(0)!=n1.@CT
                                .get(0))
101                          ACCUM fuse.@notSeen=False, @@foundFuses+=
                                fuse, fuse.@CT+=n1.@CT, fuse.
                                @AccumulatedR += n1.@AccumulatedR;
102           /* Look for the nodes that were already seen but from
                another transformer station, this means that any loop is
                here. We mark them with @loopChecked*/
103     ErrorNodes = SELECT n FROM Nudo:n1-(SEGMENTO:s)-NUDO:n
104                           WHERE n.@notSeen==False AND n.@CT.get(0)!=
                                n1.@CT.get(0)
105                           ACCUM n.@loopChecked=True;
106     /* Pass no next node and store the edge in @@Segments */
107     Nudo = SELECT n FROM Nudo:n1-(SEGMENTO:s)-NUDO:n
108                     WHERE n.@notSeen==True
109                     ACCUM n.@notSeen=False, n.@CT+=n1.@CT, n.
                            @AccumulatedR+=(n1.@AccumulatedR+s.R*s.
                            LONGITUD),@@Segments+=(n1.@CT.get(0)->
                            SEGMENTS_DATA_TYPE(s.MSLINK,s.X,s.Y,n1.
                            @AccumulatedR+s.R*s.LONGITUD));
110   END;
111   Fuses_last = {@@foundFuses};/* List of found closed fuses */
112   IF Fuses_last.size()==0 THEN
113     BREAK; /* In no more fuses: STOP SEARCHING */
114   ELSE
115     /* Otherwise, start again from the following notSeen node */
116     Nudo = SELECT n FROM Fuses_last:f-(CONNECTION)-NUDO:n
117                     WHERE n.@notSeen==True
118                     ACCUM n.@notSeen=False, n.@CT+=f.@CT, n.
                            @AccumulatedR+=f.@AccumulatedR;
119   END;
120 END;
121
122 /* End of the searching process of the "blue lines" (lines connected
      to any station) */
123 AllNodes(NUDO) = {NUDO.*};
124 /* 1.- Look for notSeen nodes within the selected coordinates window
    ,
125 save them to represent the segments connected to them as "
    disconnected"*/
126 NotSeenNodes = SELECT n FROM AllNodes:n-(SEGMENTO:s)-NUDO
127                         WHERE (n.@notSeen==True AND n.X>@@XMIN AND n
                              .Y>@@YMIN AND n.X<@@XMAX AND n.Y<@@YMAX)
128                         ACCUM @@DisconnSegments+=(s.MSLINK->
                              COORDS_DATA_TYPE(s.X,s.Y));
129
130 /* 2.- Loop detection
131 Seleccionamos los nudos en los que habiamos identificado un loop */
132 ErrorNodes = SELECT n FROM AllNodes:n
133                       WHERE n.@loopChecked==True;
134 /* Same algorithm than in first case (BFS):
135 Ride all interconnected nodes by segments or closed fuses and store
    them in @@errorSegments */
136 WHILE True LIMIT 50 DO
```

```
137     @@foundFuses.clear();
138     WHILE ErrorNodes.size()>0 LIMIT 50 DO
139       /* This time @loopChecked is used instead of @notSeen */
140       ErrorFuses = SELECT f FROM ErrorNodes-(CONNECTION)-FUSIBLE:f
141                                WHERE f.@loopChecked==False
142                                ACCUM f.@loopChecked=True, @@foundFuses+=f
                                       ;
143
144       ErrorNodes = SELECT n FROM ErrorNodes-(SEGMENTO:s)-NUDO:n
145                                WHERE n.@loopChecked==False
146                                ACCUM n.@loopChecked=True, @@errorSegments
                                       +=(s.MSLINK->COORDS_DATA_TYPE(s.X,s.Y))
                                       ;
147     END;
148     FL = {@@foundFuses};
149     IF FL.size()==0 THEN
150       BREAK;
151     ELSE
152       ErrorNodes = SELECT n FROM FL:fuse-(CONNECTION)-NUDO:n
153                                WHERE n.@loopChecked==False;
154     END;
155   END;
156
157   /* 3.- Look for all the fuses with coordinates within the selected
          window */
158   AllFuses(FUSIBLE) = {FUSIBLE.*};
159   AllFuses = SELECT f FROM AllFuses:f
160                       WHERE f.X>@@XMIN AND f.Y>@@YMIN AND f.X<@@XMAX
                             AND f.Y<@@YMAX
161                       ACCUM @@Breaker+=FUSE_DATA_TYPE(f.MSLINK, f.X, f
                             .Y, f.ESTADO_REAL);
162
163   /* RETURN THE RESULTS */
164   PRINT @@Meters_Nodes, @@Acometidas, @@XMAX, @@XMIN, @@YMAX, @@YMIN,
          @@CT, @@Segments, @@errorSegments, @@Breaker, @@DisconnSegments;
165   }
```

B.2 Tigergraph Communication Analysis

```
1   CREATE QUERY GetAllCoordinates_com_3() FOR GRAPH grid {
2   /*University of Oviedo
3   LEMUR Research Group 2020
4   Author: Adrian Miranda Paz
5
6   Query used to get and synthetize the data ignoring the registry's
       data and only analyzing the communications information.*/
7   /* User-defined tuples definition */
8   TYPEDEF TUPLE <UINT MSLINK, STRING X, STRING Y> SEGMENTS_DATA_TYPE;
9   TYPEDEF TUPLE <STRING X, STRING Y> COORDS_DATA_TYPE;
10  TYPEDEF TUPLE <STRING CLAVE_CT, FLOAT X, FLOAT Y> CT_DATA_TYPE;
11  TYPEDEF TUPLE <UINT MSLINK, FLOAT X, FLOAT Y, STRING STATE>
       FUSE_DATA_TYPE;
12  TYPEDEF TUPLE <FLOAT X, FLOAT Y> NODE_DATA_TYPE;
13  /* Accumulator to store possible nodes containing inconsistent data
       */
14  MapAccum<UINT, ListAccum<NODE_DATA_TYPE>> @@ErrorNodes;
15  /* Accumulators to store the segments for the different
       communication situations detected */
16  MapAccum<STRING, ListAccum<SEGMENTS_DATA_TYPE>> @@Segments;
17  MapAccum<STRING, ListAccum<SEGMENTS_DATA_TYPE>> @@SegmentsPortatiles
       ;
18  MapAccum<STRING, ListAccum<SEGMENTS_DATA_TYPE>> @@SegmentsTmp;
19  MapAccum<UINT,ListAccum<SEGMENTS_DATA_TYPE>> @@errorSegments;
20  MapAccum<UINT,COORDS_DATA_TYPE> @@DisconnSegments;
21  /* Other used accumulators for returning data */
```

```
22 ListAccum<CT_DATA_TYPE> @@CT;
23 ListAccum<FUSE_DATA_TYPE> @@Breaker;
24 ListAccum<SEGMENTS_DATA_TYPE> @@CurrentSegmento;
25 ListAccum<SEGMENTS_DATA_TYPE> @@errorNodeSeg;
26 /* Accumulators to define the stations to analyze */
27 ListAccum<UINT> @@CTS;
28 /* Accumulator to store the lines outgoing from a station */
29 ListAccum<UINT> @@AllLines;
30 /* Variable to store the reference reported by the seen meters*/
31 STRING referencia;
32 /* Accumulator to store the ridden edges in ConnectivityNodes BFS
      before deciding if saving to result or not */
33 ListAccum<UINT> @@CurrentSegmentoEdges;
34 /* Variable to store the segments where unconsistent data was found
      */
35 ListAccum<EDGE<SEGMENTO>> @errorSegs;
36 /* Indicator to decide the communication status */
37 OrAccum @@anyConnected=False;
38 AndAccum @@stillConnected=True;
39 /* Indicator to avoid back flow */
40 OrAccum @Seen=False;
41 /* Accumulator to save found vertices in breakers dimension */
42 ListAccum<VERTEX<FUSIBLE>> @@FoundFuses;
43 ListAccum<UINT> @@LastFuses;
44 BOOL stillFuses=True;
45 /* Variable to store the not seen segments (disconnected) */
46 ListAccum<EDGE<SEGMENTO>> @@NotFoundSegs;
47 ListAccum<VERTEX<NUDO>> @@AnalyzedNodes;
48 /* Variable to store the reference of the main concentrator */
49 STRING referencia_concentrador;
50 /* Variable to store references of portable concentrators */
51 ListAccum<STRING> @@referencia_portatiles;
52 /* Variable to store the ID of the switch whith the meters are
      communicating with */
53 UINT switchID;
54 /* Variable to store the segment where the switch was found */
55 ListAccum<EDGE<SEGMENTO>> @@SegSwitch;
56 /* Local accumulator to store the segment id where the switch of the
       connection point was found */
57 ListAccum<EDGE<SEGMENTO>> @SegSwitch;
58
59 /*Accumulators for the return of the maximum and minimum X and Y
      coordinates (used by the visualization module)*/
60 MinAccum<FLOAT> @@XMIN=283581.3125;
61 MaxAccum<FLOAT> @@XMAX=285733.3125;
62 MinAccum<FLOAT> @@YMIN=4807315;
63 MaxAccum<FLOAT> @@YMAX=4808595;
64 /*List of MSLINK of Transformer Stations of Pola de Siero*/
65 @@CTS=[1830188,65043,65079,65040,65025,65082,65028,65037,
66        65068,65034,65046,65031,65052,65049,65019,65076,86315,
67        1076069,1076128,1076109,1132967,1136042,1136039,1136056,
68        1136065,1142831,1351982,1459343,2732692,65016];
69 /* Initialize sets of all meters, ConnectivityNodes and
      TransformerStations */
70 AllMeters = {CONTADOR.*};
71 AllNodes = {NUDO.*};
72 CTs = {CT.*};
73 /* Selection of the used Transformer stations */
74 CTs = SELECT c FROM CTs:c WHERE c.PROPIEDAD=="Hc Distribucion" AND
      @@CTS.contains(c.MSLINK) ACCUM @@CT+=CT_DATA_TYPE(c.CLAVE_BDI, c.
      X, c.Y);
75
76 FOREACH ctid IN @@CTS DO /* Loop over CTs*/
77         /* Clear variables from pasted iterations*/
78     @@AllLines.clear(); @@referencia_portatiles.clear();
79     /* Query of the current CT and Concentrators references and type
```

```
          of concentrators */
80      CurrentCT = SELECT ct FROM CTs:ct WHERE ct.MSLINK==ctid;
81      CurrentConcentrator = SELECT conc FROM CurrentCT-(
            WITH_CONCENTRADOR)-CONCENTRADOR:conc WHERE conc.
            DES_ESTADO_COM=="Activo" AND conc.TIPO=="PRINCIPAL" POST-
            ACCUM referencia_concentrador=conc.REFERENCE_CONC;
82      CurrentConcentrator = SELECT conc FROM CurrentCT-(
            WITH_CONCENTRADOR)-CONCENTRADOR:conc WHERE conc.
            DES_ESTADO_COM=="Activo" AND conc.TIPO=="PRINCIPAL" POST-
            ACCUM @@referencia_portatiles+=conc.REFERENCE_CONC;
83      /* Request the Celda of current Station */
84      CurrentCelda = SELECT cel FROM CurrentCT-(WITH_CELDA)->CELDA:cel
            ;
85      /* Request list of lines outgoing from the current station */
86      AllLines = SELECT lin FROM CurrentCelda-(CONNECTION)-LINEA_BT:
            lin ACCUM @@AllLines+=lin.MSLINK;
87      FOREACH lid IN @@AllLines DO /* Loop over lines */
88          stillFuses=True;
89          CurrentLine = SELECT lin FROM AllLines:lin WHERE lin.MSLINK
                ==lid;
90          /* Initially it is assumed that the segment is connected to
                the station */
91          @@stillConnected=True;
92          /* Initialize the frontier as the initial node of the feeder
                */
93          CurrentNode = SELECT n FROM CurrentLine-(CONNECTION)-NUDO:n
                ACCUM n.@Seen=True;
94          /* Starting of the BFS algorithm (Breakers dimension) */
95          WHILE(stillFuses) DO
96              @@anyConnected=False;
97              @@CurrentSegmento.clear();
98              @@CurrentSegmentoEdges.clear();
99              /* Starting of BFS on Nodes dimension */
100             WHILE (CurrentNode.size()>0 AND @@stillConnected) DO
101                 CGPs = SELECT cgp FROM CurrentNode-(CONNECTION)-CGP:
                        cgp;
102                 IF (CGPs.size()>0) THEN
103                     METERS = SELECT m FROM CGPs-(CONNECTION)-
                            CONTADOR:m WHERE m.STATE=="Terminal" POST-
                            ACCUM switchID=m.SID, referencia=m.
                            CONCENTRADOR LIMIT 1;
104                     IF METERS.size()>0 THEN
105                         @@stillConnected+=(referencia_concentrador ==
                                referencia) OR (@@referencia_portatiles.
                               contains(referencia));
106                         @@anyConnected+=@@stillConnected;
107                     END;
108                 END;
109                 FUSEs = SELECT f FROM CurrentNode-(CONNECTION)-
                        FUSIBLE:f ACCUM @@FoundFuses+=f;
110                 CurrentNode = SELECT n FROM CurrentNode-(SEGMENTO:s)
                        -NUDO:n WHERE n.@Seen==False ACCUM
                        @@CurrentSegmento+=SEGMENTS_DATA_TYPE(s.MSLINK, s
                        .X, s.Y), n.@Seen=True,
111                 @@CurrentSegmentoEdges+=s.MSLINK;
112             END;
113 /* Three possible cases: a) there are breakers found, b) the
        algorithm has been "disconnected" -> it was found meters and they
         are not connected with the current station, c) the line is
        finished.
114
115             /* CASE C */
116             IF (@@stillConnected) THEN
117               IF (referencia_concentrador==referencia) THEN
118                 CurrentCT = SELECT ct FROM CurrentCT:ct POST-ACCUM
                        @@Segments+=(ct.CLAVE_BDI->@@CurrentSegmento);
119               ELSE
120                 CurrentCT = SELECT ct FROM CurrentCT:ct POST-ACCUM
```

```
                    @@SegmentsPortatiles+=(ct.CLAVE_BDI->
                    @@CurrentSegmento);
121               END;
122               /* CASE A */
123               FUSEs = {@@FoundFuses};
124               FUSEs = SELECT f FROM FUSEs:f WHERE f.@Seen==False;
125               stillFuses=FUSEs.size()>0;
126               IF (stillFuses) THEN
127                  FUSE = SELECT f FROM FUSEs:f WHERE f.@Seen==False
                        POST-ACCUM f.@Seen=True LIMIT 1;
128                  CurrentNode = SELECT n FROM FUSE-(CONNECTION)-NUDO:
                        n WHERE n.@Seen==False ACCUM n.@Seen=True;
129               END;
130             ELSE
131               IF @@anyConnected THEN
132                 NudosErroneos = SELECT n FROM AllNodes:n-(SEGMENTO:s
                       )-NUDO:n2 WHERE @@CurrentSegmentoEdges.contains(s
                       .MSLINK) ACCUM
133                 @@errorSegments+=(ctid->SEGMENTS_DATA_TYPE(s.MSLINK,
                       s.X,s.Y));
134                 ErrorNode = SELECT n FROM CurrentNode:n POST-ACCUM
                       @@ErrorNodes+=(ctid->NODE_DATA_TYPE(n.X,n.Y));
135                 CurrentCT = Select ct FROM CurrentCT:ct POST-ACCUM
                       @@errorNodeSeg+=@@CurrentSegmento.get(
                       @@CurrentSegmento.size()-1);
136                 WHILE CurrentNode.size()>0 DO
137                    CurrentNode = SELECT n FROM CurrentNode-(SEGMENTO
                          :s)-NUDO:n WHERE n.@Seen==False ACCUM
                          @@CurrentSegmento+=SEGMENTS_DATA_TYPE(s.MSLINK
                          , s.X, s.Y),
138                    @@errorSegments+=(ctid->SEGMENTS_DATA_TYPE(s.
                          MSLINK,s.X, s.Y)) ,n.@Seen=True;
139                 END;
140                 /* Look for the slack of the unconsistent meter */
141                 Switch = SELECT m FROM AllMeters:m WHERE m.SSID==
                       switchID AND m.STATE=="Switch" LIMIT 1;
142                 CGPSw = SELECT cgp FROM Switch-(CONNECTION)-CGP:cgp;
143                 NodeSw = SELECT n FROM CGPSw-(CONNECTION)-NUDO:n
                       ACCUM @@ErrorNodes+=(ctid->NODE_DATA_TYPE(n.X,n.Y
                       ));
144                 SegSw = SELECT n FROM NodeSw:n-(SEGMENTO:s)-NUDO
                       ACCUM @@SegSwitch+=s LIMIT 1;
145                 CurrentCT = Select ct FROM CurrentCT:ct POST-ACCUM
                       ct.@SegSwitch+=@@SegSwitch;
146                 @@SegSwitch.clear();
147               ELSE
148                 WHILE CurrentNode.size()>0 DO
149                   CurrentNode = SELECT n FROM CurrentNode-(SEGMENTO)
                         -NUDO:n WHERE n.@Seen==True ACCUM n.@Seen=False
                         ;
150                 END;
151               END;
152               stillFuses=False; /* Pass to next line */
153             END;
154 /* CASE B - NOTHING TO DO: PASS TO THE NEXT LOOP AND LET
       ACCUMULATORS TO BE OVERWRITTEN */
155         END;
156     END;
157 END;
158
159 NotSeenNodes = SELECT n FROM AllNodes:n-(SEGMENTO:s)-NUDO WHERE n.
       @Seen==False AND n.X>@@XMIN AND n.Y>@@YMIN AND n.X<@@XMAX AND n.Y
       <@@YMAX ACCUM @@DisconnSegments+=(s.MSLINK->COORDS_DATA_TYPE(s.X,
        s.Y));
160
161 AllFuses(FUSIBLE) = {FUSIBLE.*};
162 AllFuses = SELECT f FROM AllFuses:f WHERE f.X>@@XMIN AND f.Y>@@YMIN
```

```
    AND f.X<@@XMAX AND f.Y<@@YMAX ACCUM @@Breaker+=FUSE_DATA_TYPE(f.
    MSLINK, f.X, f.Y, f.ESTADO_REAL);
#Fuses = SELECT f FROM AllFuses:f POST-ACCUM @@Breaker+=
    FUSE_DATA_TYPE(f.MSLINK, f.X, f.Y, f.ESTADO_REAL);
TotalLines = {LINEA_BT.*};
Lines = SELECT f FROM TotalLines:f WHERE f.X>@@XMIN AND f.Y>@@YMIN
    AND f.X<@@XMAX AND f.Y<@@YMAX ACCUM @@Breaker+=FUSE_DATA_TYPE(f.
    MSLINK, f.X, f.Y, f.ESTADO_REAL);
#Lines = SELECT f FROM TotalLines:f POST-ACCUM @@Breaker+=
    FUSE_DATA_TYPE(f.MSLINK, f.X, f.Y, f.ESTADO_REAL);

PRINT @@SegmentsPortatiles, @@CT, @@errorSegments, @@DisconnSegments
    , @@Breaker, @@Segments, @@ErrorNodes;
}
```

B.3 Creation of Graph Schema

```
/* CREATE STRUCTURE OF THE DATABASE: */
/* VERTEX DEFFINITION: */
CREATE VERTEX CT(PRIMARY_ID MSLINK UINT, MSLINK UINT, CLAVE_BDI
    STRING, DESCRIPCION STRING, PROVINCIA STRING, MUNICIPIO STRING,
    PROPIEDAD STRING, ESTADO STRING, TIPO STRING, X FLOAT, Y FLOAT)
CREATE VERTEX CELDA(PRIMARY_ID MSLINK UINT, MSLINK UINT, CLAVE_BDI
    STRING, ESTADO STRING, N_O UINT, N_E UINT)
CREATE VERTEX TRAFO(PRIMARY_ID CLAVE_TRAFO STRING, CLAVE_TRAFO
    STRING, POTENCIA UINT)
CREATE VERTEX CONCENTRADOR(PRIMARY_ID MSLINK INT, CODIGO_CT STRING,
    DES_ESTADO STRING, DT_STATE_CONC DATETIME, NUM_CONTADORES INT,
    CIERRE_7 INT, CURVA_5 INT, PTO_TRABAJO STRING, STATUS STRING,
    FECHA_AVISO DATETIME, DESC_AVISO STRING, SINTOMAS STRING,
    FECHA_CREACION DATETIME, FECHA_SUPERVISION DATETIME, IP STRING,
    MODEL_CONC STRING, SW_VERSION STRING, CONFIG_DATE DATETIME,
    MARK_CONC STRING, REFERENCIA_CONC STRING, TIPO_CONCENTRADOR
    STRING, REFERENCIA_SUP STRING, RTI_CBT_1 STRING, DES_ESTADO_COM
    STRING, DATE_COMM_STATE_CONC STRING, TIPO_COMUNICACION STRING)
CREATE VERTEX ROUTER(PRIMARY_ID ROUTER_ID UINT, ROUTER_ID UINT,
    REFERENCIA_CCT STRING, CODIGO_CT STRING, FECHA_INICIO DATETIME,
    NUM_SERIE STRING, FABRICANTE STRING, MODELO STRING, ESTADO STRING
    , IMEI UINT, IP STRING, IP_LAN STRING, COBERTURA INT,
    FECHA_COBERTURA DATETIME, VERSION_FIRMWARE STRING,
    FECHA_VFIRMWARE DATETIME, RED STRING, TIPO_CONEXION STRING,
    PERFIL_CONFIGURACION STRING, USUARIO STRING, CONTRASENA STRING,
    PROVEEDOR_SIM STRING, ICCID STRING, NUM_LINEA_SIM UINT, PIN_SIM
    UINT)
CREATE VERTEX LINEA_BT(PRIMARY_ID MSLINK UINT, MSLINK UINT,
    MSLINK_CTM UINT, CTM STRING, NUMERO_LINEA UINT, DESCRIPCION
    STRING, PROPIEDAD STRING, ESTADO STRING, TENSION STRING,
    ESTADO_NORMAL STRING, ESTADO_REAL STRING, N_O UINT, N_E UINT, X
    FLOAT, Y FLOAT)
CREATE VERTEX INTERRUPTOR_BT(PRIMARY_ID MSLINK_EM UINT, MSLINK_EM
    UINT, CLAVE_CT STRING, DESCRIPCION STRING, N_O UINT, N_E UINT,
    EST_NORMAL STRING, EST_OPERAC STRING)
CREATE VERTEX NUDO(PRIMARY_ID ID UINT, ID UINT)
CREATE VERTEX FUSIBLE(PRIMARY_ID MSLINK UINT, MSLINK UINT, PROPIEDAD
     STRING, ESTADO STRING, TIPO STRING, ESTADO_NORMAL STRING,
    ESTADO_REAL STRING, X FLOAT, Y FLOAT)
CREATE VERTEX CGP(PRIMARY_ID MSLINK UINT, MSLINK UINT, CLAVE_BDI
    UINT, PROPIEDAD STRING, ESTADO STRING, TENSION STRING, TIPO
    STRING)
CREATE VERTEX CONTADOR(PRIMARY_ID CUPS STRING, REFERENCIA STRING,
    CLASE_MATERIAL STRING, CUPS STRING, EQUIPO UINT, NUMERO_SERIE
    STRING, FABRICANTE STRING, FECHA_MONTAJE DATETIME, TIPO_CONTADOR
    UINT, POT_1 FLOAT, POT_2 FLOAT, POT_3 FLOAT, TARIFA STRING,
    FECHA_ULTIMO_CAMBIO_POTENCIA DATETIME, MUNICIPIO STRING,
    COD_ENTIDAD_COLECTIVA STRING, COD_ENTIDAD_SINGULAR STRING,
    ENTIDAD_SINGULAR STRING, ACOMETIDA UINT)
```

```
14 CREATE VERTEX EVENTO(PRIMARY_ID MSLINK_EV STRING, MSLINK_EV STRING,
       CODIGO_ELEMENTO STRING, TIPO_OBJETO UINT, DESCRIPCION_OBJETO
       STRING, FECHA DATETIME, VALOR_INICIAL STRING, VALOR_FINAL STRING,
        TABLA_CAMPO STRING, USUARIO STRING)
15 /* EDGES DEFINITION: */
16 CREATE UNDIRECTED EDGE CONNECTION(FROM *, TO *)
17 /* RELATIONSHIPS WITHIN THE CT CONTEXT */
18 CREATE DIRECTED EDGE OF_CT(FROM *, TO CT)
19 CREATE DIRECTED EDGE WITH_TRAFO(FROM CT, TO TRAFO)
20 CREATE DIRECTED EDGE WITH_INTERRUPTOR_BT(FROM CT, TO INTERRUPTOR_BT)
21 CREATE DIRECTED EDGE WITH_CELDA(FROM CT, TO CELDA)
22 CREATE UNDIRECTED EDGE SEGMENTO(FROM NUDO, TO NUDO, LONGITUD FLOAT,
       MSLINK UINT, X STRING, Y STRING, COUNTER FLOAT, UNE STRING)
23 CREATE DIRECTED EDGE WITH_CONCENTRADOR(FROM CT, TO CONCENTRADOR)
24 CREATE DIRECTED EDGE WITH_ROUTER(FROM CT, TO ROUTER)
25 CREATE DIRECTED EDGE WITH_EVENTO(FROM *, TO EVENTO)
26 /* CREATE THE GRAPH STRUCTURE WITH ALL THE PREVIOUSLY CREATED
       VERTICES AND EDGES */
27 CREATE GRAPH grid(*)
```

B.4 Creation of Loading Job to Upload Data to TigerGraph

```
1  CREATE LOADING JOB load_gijon FOR GRAPH grid{
2  /* Define the files to load to the database */
3  DEFINE FILENAME file_cts = "../TigerInput/ct.csv";
4  DEFINE FILENAME file_lines = "../TigerInput/linea_bt_output.csv";
5  DEFINE FILENAME file_intBT = "../TigerInput/maniobra_output.csv";
6  DEFINE FILENAME file_celda = "../TigerInput/celdas_output.csv";
7  DEFINE FILENAME rels_cel_line = "../TigerInput/cel_line.csv";
8  DEFINE FILENAME file_segmentos = "../TigerInput/segments_coord.csv";
9  DEFINE FILENAME file_nudos = "../TigerInput/nudos.csv";
10 DEFINE FILENAME file_fusibles = "../TigerInput/fusible.csv";
11 DEFINE FILENAME file_cgp = "../TigerInput/cgp_output.csv";
12 DEFINE FILENAME file_meters = "../TigerInput/meters.csv";
13 DEFINE FILENAME file_meters = "../../TodosContadores.csv";
14 DEFINE FILENAME file_concent = "../TigerInput/concentrators.csv";
15 DEFINE FILENAME file_rels_conc_ct = "../TigerInput/conc_ct.csv";
16 /* Map columns to the vertices' attributes */
17 LOAD file_cts
18     TO VERTEX CT VALUES($"MSLINK",$"MSLINK",$"CLAVE_BDI",$"
       DESCRIPCION",$"PROVINCIA",$"MUNICIPIO",$"PROPIEDAD",$"ESTADO"
       ,$"TIPO",$"X",$"Y") USING header="true", separator=",";
19 LOAD file_intBT
20     TO VERTEX INTERRUPTOR_BT VALUES($"MSLINK_EM",$"MSLINK_EM",$"
       CLAVE_CT",$"DESCRIPCION_MAN",$"NO",$"NE",$"EST_NORMAL",$"
       EST_OPERAC"),
21     TO EDGE WITH_INTERRUPTOR_BT VALUES($"MSLINK" CT,$"MSLINK_EM"
       INTERRUPTOR_BT),
22     TO EDGE OF_CT VALUES($"MSLINK_EM" INTERRUPTOR_BT, $"MSLINK" CT)
       USING header="true",separator=",";
23 LOAD file_nudos TO VERTEX NUDO VALUES($"N",$"N",$"X",$"Y") USING
   header="true",separator=",";
24 LOAD file_celda
25     TO VERTEX CELDA VALUES($"MSLINK_CELDA",$"MSLINK_CELDA",$"
       CLAVE_BDI_CELDA", $"ESTADO_CELDA", $"NO", $"NE"),
26     TO VERTEX TRAFO VALUES($"TRAFO", $"TRAFO", $"POTENCIA"),
27     TO EDGE WITH_CELDA VALUES($"MSLINK_CT" CT,$"MSLINK_CELDA" CELDA)
       ,
28     TO EDGE WITH_TRAFO VALUES($"MSLINK_CT" CT,$"TRAFO" TRAFO),
29     TO EDGE OF_CT VALUES($"MSLINK_CELDA" CELDA,$"MSLINK_CT" CT),
```

```
30      TO EDGE OF_CT VALUES($"TRAFO" TRAFO,$"MSLINK_CT" CT) USING
            header="true", separator=",";
31      TO EDGE CONNECTION VALUES($"TRAFO" TRAFO,$"MSLINK_EM"
            INTERRUPTOR_BT),
32      TO EDGE CONNECTION VALUES($"MSLINK_EM" INTERRUPTOR_BT, $"
            MSLINK_CELDA" CELDA)
33  LOAD file_lines
34      TO VERTEX LINEA_BT VALUES($"MSLINK",$"MSLINK",$"MSLINK_CTM",$"
            CTM",$"NUMERO_LINEA",$"DESCRIPCION",$"PROPIEDAD",$"ESTADO",$"
            TENSION",$"ESTADO_NORMAL",$"ESTADO_REAL",$"NO",$"NE",$"X",$"Y
            "),
35      TO EDGE CONNECTION VALUES($"MSLINK" LINEA_BT,$"NE" NUDO) USING
            header="true", separator=",";
36  LOAD rels_cel_line TO EDGE CONNECTION VALUES($"MSLINK_CELDA" CELDA,
        $"MSLINK" LINEA_BT) USING header="true", separator=",";
37  LOAD file_segmentos TO EDGE SEGMENTO VALUES($"NO" NUDO, $"NE" NUDO,
        $"LONGITUD", $"MSLINK", $"X", $"Y", $"count", $"UNE",$"TIPO",$"
        CODE",$"R", $"X(Ohm)", $"SECTION", $"Imax") USING header="true",
        separator=",";
38  LOAD file_fusibles
39      TO VERTEX FUSIBLE VALUES($"MSLINK",$"MSLINK",$"Estado Normal",$"
            Estado Real",$"ZONA",$"Caja",$"Tipo",$"X",$"Y"),
40      TO EDGE CONNECTION VALUES($"NO" NUDO,$"MSLINK" FUSIBLE),
41      TO EDGE CONNECTION VALUES($"NE" NUDO,$"MSLINK" FUSIBLE) USING
            header="true",separator=",";
42  LOAD file_cgp
43      TO VERTEX CGP VALUES($"MSLINK", $"MSLINK", $"CLAVE_BDI", $"
            PROPIEDAD", $"ESTADO", $"TENSION", $"TIPO"),
44      TO EDGE CONNECTION VALUES($"MSLINK" CGP, $"NO" NUDO) USING
            header="true", separator=",";
45  LOAD file_meters
46      TO VERTEX CONTADOR VALUES($"REFERENCIA",$"REFERENCIA",$"
            LEVEL_COM", $"CODIGO_ACOMETIDA",$"CCT",$"SID",$"STATE",$"SSID
            ",$"CONNECTION_TIME",$"DISCONNECTION_TIME",$"ACTIVE_TIME",$"
            PHASE",$"P",$"Q"),
47      TO EDGE CONNECTION VALUES($"MSLINK" CGP, $"REFERENCIA" CONTADOR)
             USING header="true", separator=",";
48  LOAD file_concent
49      TO VERTEX CONCENTRADOR VALUES($"MSLINK", $"CODIGO_CT", $"
            DES_ESTADO",$"DT_STATE_CONC", $"NUM_CONTADORES", $"CIERRE_7",
             $"CURVA_5", $"PTO_TRABAJO", $"STATUS", $"FECHA_AVISO", $"
            DESC_AVISO", $"SINTOMAS", $"FECHA_CREACION", $"
            FECHA_SUPERVISION", $"IP", $"MODEL_CONC", $"SW_VERSION", $"
            CONFIG_DATE", $"MARK_CONC", $"REFERENCE_CONC", $"
            TIPO_CONCENTRADOR", $"REF_SUPERV", $"RTI_CBT_1", $"
            DES_ESTADO_COM", "DATE_COMM_STATE_CONC", $"TIPO_COMUNICACION"
            ) USING header="true", separator=",";
50  LOAD file_rels_conc_ct
51      TO EDGE WITH_CONCENTRADOR VALUES($"MSLINK_CT" CT,$"MSLINK_CONC"
            CONCENTRADOR),
52      TO EDGE OF_CT VALUES($"MSLINK_CONC" CONCENTRADOR,$"MSLINK_CT" CT
            ) USING header="true", separator=",";
53  LOAD file_meters
54      TO VERTEX CONTADOR VALUES($"Serial",$"Serial",$"Prof.", $"
            Acometida",$"Referencia cct_claveCT", $"SID", $"State", $"
            SSID", $"Conn. Time", $"Disc. Time", $"L.Active time",$"Fase"
            ,$"P",$"Q"),
55      TO EDGE CONNECTION VALUES($"MSLINK" CGP, $"Serial" CONTADOR)
            USING header="true", separator=",";
56  }
57  RUN LOADING JOB load_segments
```

B.5 HTML Code

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
     scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
  <link href="https://fonts.googleapis.com/css?family=Saira
     :100,200,300,400,500,600,700,800,900" rel="stylesheet">
  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com
     /bootstrap/4.4.1/css/bootstrap.min.css" integrity="sha384-
     Vkoo8x4CGsO3+Hhxv8T/
     Q5PaXtkKtu6ug5TOeNV6gBiFeWPGFN9MuhOf23Q9Ifjh" crossorigin="
     anonymous">
  <title>Visualization</title>
</head>
<body bgcolor="#0000">
  <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"
     integrity="sha384-J6qa4849blE2+
     poT4WnyKhv5vZF5SrPo0iEjwBvKU7imGFAV0wwj1yYfoRSJoZ+n"
     crossorigin="anonymous"></script>
  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist
     /umd/popper.min.js" integrity="sha384-Q6E9RHvbIyZFJoft+2
     mJbHaEWldlvI9IOYy5n3zV9zzTtmI3UksdQRVvoxMfooAo" crossorigin=
     "anonymous"></script>
  <script src="https://stackpath.bootstrapcdn.com/bootstrap
     /4.4.1/js/bootstrap.min.js" integrity="sha384-
     wfSDF2E50Y2D1uUdj0O3uMBJnjuUD4Ih7YwaYd1iqfktj0Uod8GCExl3Og8ifwB6
     " crossorigin="anonymous"></script>
  <nav class="navbar navbar-light bg-dark">
    <div align="right">
      <form class="form-inline">
        <a class="btn btn-outline-light" href="resistances.html"
            role="button">Resistance</a>
        <a class="btn btn-outline-light" href="#" role="button">
           Raw Grid</a>
      </form>
    </div>
    <div align="right">
      <img
        src="img/eredes_logo.png"
        class="d-flex justify-content-center mx-auto"
        id="icon_eredes"
        width="200px"
      />
    </div>
  </nav>
  <svg width="1000px" height="900px" id="canvas" style="
     background-color:black"></svg>
  <script src="js/d3.v5.min.js"></script>
  <script src="js/raw.js"></script>
</body>
</html>
```

B.6 JavaScript code

```
function httpRequest(address) {
/*Function to make "GET" requests to the database.
Synchronous request: till the results are not returned, WAIT*/
var req = window.XMLHttpRequest ? new XMLHttpRequest() : new
   ActiveXObject("Microsoft.XMLHTTP");
//req.timeout = 4000;  // Reduce default 2mn-like timeout to 4 s
   if synchronous
```

```
req.open("GET",address,false);
req.send();
return req.responseText;
}
/* Creation of the d3 variable to store the drawing */
/* The svgContainer will be a D3 group of elements that will form the drawing of the grid */
var svgContainer = d3.select("#canvas")
                     .attr("width", "2000")
                     .attr("height", "900")
                     .call(d3.zoom().on("zoom", function(){
                        svgContainer.attr("transform", d3.event.transform)
                     }))
                     .append("g");
// Function to create path objects in D3
var lineFunction = d3.line()
                     .x((obj)=>obj.x)
                     .y((obj)=>obj.y);
// Function to create circles to indicate the position of the Transformer Stations
function plotCircle(Coords,id){
// Arguments: coords=[X,Y] and id with the identifier of the ct
    svgContainer.append("circle")
                .attr("cx", Coords[0])
                .attr("cy", Coords[1])
                .attr("r", 10*rel)
                .attr("fill", "none")
                .attr("stroke-width",4)
                .attr("stroke","green")
                .attr("vector-effect", "non-scaling-stroke")
                .attr("id",id);
};
// Function to scale the coordinates from the latitute longitude format to pixels in the drawing
function scaleCoords(Coords){
  // It will be scaled according to variable rel. This variable is calculated from the maximum and minimum value of X and Y coordinates.
  // The sign of the Y coordinate is changed since the Y coordinate in the canvas goes up to down.
    return([(Coords[0]-Xmin)*rel,1000-(Coords[1]-Ymin)*rel]);
};
// Main function of the script
function plotGrid(){
  // Run query from RESTPP API
    response = JSON.parse(httpRequest("http://localhost:9000/query/grid/GetVisualization"))["results"][0];
    // Get the different elements in the returned object
    Segments_data=response["@@Segments"];
    CTs_data=response["@@CT"];
    NotConnSegments_data=response["@@DisconnSegments"];
    Breakers_data=response["@@Breaker"];
    Loops_data = response["@@errorSegments"];
  // Expected maximum and minimum values of the X and Y coordinates (for the case of the whole grid)
    Xmin = 204490.43;
    Xmax = 327349.67;
    Ymin = 4778550.35;
    Ymax = 4780505.4;
  // Differences between maximums and minimums
    diffX = Xmax-Xmin;
    diffY = Ymax-Ymin;
    // Get the maximum diference as reference
    diff = Math.max(diffX,diffX);
```

```
62 |     // Scale relation according to the set number of pixels in
   |        the canvas
63 |     rel = 2000/diff;
64 |     // Get the list of stations keys
65 |     CTs = Object.keys(Segments_data);
66 |     // Loop over stations keys in order to access the groups of
   |        segments
67 |     for (var c=0;c<CTs.length;c++){
68 |         // Get the station's key
69 |         ct = CTs[c];
70 |         // Get the group of segments corrensponding to the
   |            current station
71 |         segments = Segments_data[ct];
72 |         // Loop over all segments
73 |         for (var s=0;s<segments.length;s++){
74 |              var lineData=[]; // Initialize the array of
   |                 coordinates of the segment
75 |              // Get the data, and parse the string to an array of
   |                  numbers
76 |              segment_data = segments[s];
77 |              X = segment_data.X.split("|").map(parseFloat);
78 |              Y = segment_data.Y.split("|").map(parseFloat);
79 |              // Get the resistance corresponding to the segment
80 |              resistance = segment_data.R;
81 |              // Loop over the vertices of the segment
82 |              for (subseg=0;subseg<X.length;subseg++){
83 |                  if (Y[subseg]>48000){ // Filter some outlayers
84 |                       // Scale the coordinates to pixels
85 |                       CoordsSeg = scaleCoords([X[subseg],Y[subseg
   |                          ]]);
86 |                       // Convert the coordinates to D3 format
87 |                       lineData.push({"x":CoordsSeg[0], "y":
   |                          CoordsSeg[1]});
88 |                  };
89 |              };
90 |              // Create a "path" object with the data collected in
   |                  the previous loop
91 |              svgContainer.append("path")
92 |                       .attr("d", lineFunction(lineData))
93 |                       .attr("stroke", "blue")
94 |                       .attr("stroke-width", 2)
95 |                       .attr("vector-effect", "non-scaling-stroke")
96 |                       .attr("fill", "none")
97 |                       .attr("class",()=> "ts_" + ct)
98 |                       .attr("id","path_"+segment_data.MSLINK)
99 |                       .on("mouseover",function(){
100|              // Definition of the action on mouseover (
   |                 highlighting of the transformer station
   |                 subnetwork)
101|              // Get the corresponding station key
102|              ct = this.getAttribute("class");
103|              ct = ct.split("_")[1];
104|              // Highlight the circle of the station
105|              svgContainer.selectAll("#" + ct)
106|                       .style("stroke","orange")
107|                       .style("stroke-width",6);
108|              // Modify ALL segments connected to the station
109|              svgContainer.selectAll("." + this.getAttribute("
   |                 class"))
110|                    .style("stroke","cyan")
111|                    .style("stroke-width",4);})
112|                    .on("mouseout",function(){
113|                    // Function to reset initial style
   |                       configuration on mouseout
114|                      ct = this.getAttribute("class");
```

```
115 |                     ct = ct.split("_")[1];
116 |                     segments = Segments_data[ct];
117 |                     svgContainer.selectAll("#" + ct)
118 |                             .style("stroke","green")
119 |                             .style("stroke-width",4);
120 |                     for (var s=0; s<segments.length;s++){
121 |                         resistance = segments[s].R;
122 |                         svgContainer.selectAll("#path_"+segments
    |                             [s].MSLINK)
123 |                                 .style("stroke", "blue")
124 |                                 .style("stroke-width",2);
125 |                     };
126 |                 });
127 |         };
128 |     };
129 |     // Plot the circles that indicate the positions of the
    |         stations
130 |     for (var c=0;c<CTs_data.length;c++){
131 |         ct_data =CTs_data[c]; // Get station's data
132 |         ct = ct_data.CLAVE_CT; // Get station's key
133 |         // Scale coordinates
134 |         CoordsCT = scaleCoords([ct_data.X,ct_data.Y]);
135 |         // Plot the position of the CT
136 |         plotCircle(CoordsCT,ct);
137 |         // Write the name of the CT next to the circle
138 |         svgContainer.append("text")
139 |                     .attr("x", CoordsCT[0]+10*rel)
140 |                     .attr("y", CoordsCT[1]+10*rel)
141 |                     .attr("font-family", "Saira")
142 |                     .attr("font-size",15*rel)
143 |                     .attr("fill","white")
144 |                     .text(ct);
145 |     };
146 |     // PLOT DISCONNECTED SEGMENTS (similar to the normal segments
    |         plot without the highlightin feature)
147 |     NotConnSegments = Object.keys(NotConnSegments_data);
148 |     for (var s=0; s<NotConnSegments.length;s++){
149 |         lineData=[];
150 |         seg = NotConnSegments[s]
151 |         segment_data = NotConnSegments_data[seg];
152 |         X = segment_data.X.split("|").map(parseFloat);
153 |         Y = segment_data.Y.split("|").map(parseFloat);
154 |         for (var subseg=0;subseg<X.length;subseg++){
155 |             CoordsSeg = scaleCoords([X[subseg],Y[subseg]]);
156 |             lineData.push({"x":CoordsSeg[0], "y":CoordsSeg[1]})
157 |         };
158 |         svgContainer.append("path")
159 |                     .attr("d", lineFunction(lineData))
160 |                     .attr("stroke", "grey")
161 |                     .attr("stroke-width", 2)
162 |                     .attr("vector-effect", "non-scaling-stroke")
163 |                     .attr("fill", "none")
164 |                     .attr("id","path_"+segment_data.MSLINK);
165 |     };
166 |     // PLOT LOOP SEGMENTS (same method as the disconnected
    |         segments)
167 |     Loops = Object.keys(Loops_data);
168 |     for (var s=0; s<Loops.length;s++){
169 |         lineData=[];
170 |         seg = Loops[s]
171 |         segment_data = Loops_data[seg];
172 |         X = segment_data.X.split("|").map(parseFloat);
173 |         Y = segment_data.Y.split("|").map(parseFloat);
174 |         for (var subseg=0;subseg<X.length;subseg++){
```

```
            CoordsSeg = scaleCoords([X[subseg],Y[subseg]]);
            lineData.push({"x":CoordsSeg[0], "y":CoordsSeg[1]})
        };
        svgContainer.append("path")
                    .attr("d", lineFunction(lineData))
                    .attr("stroke", "red")
                    .attr("stroke-width", 2)
                    .attr("vector-effect", "non-scaling-stroke")
                    .attr("fill", "none");
    };
    // PLOT BREAKERS AND Lines
    for (var line=0;line<Breakers_data.length;line++){
      // Get the breaker information
        data = Breakers_data[line];
        // Get the breaker status
        state = data.STATE;
        // Select the color to plot in terms of its status
        if (state=="C"){
            colour="red";
        }else{
            colour="green";
        };
        // Scale the coordinates
        Coords = scaleCoords([data.X,data.Y]);
        // Plot a small rectangle in the specified coordinates
        svgContainer.append("rect")
        .attr("x", Coords[0])
        .attr("y", Coords[1])
        .attr("width", 0.8*rel)
        .attr("height",0.8*rel)
        .attr("fill", colour)
        .attr("stroke",colour)
        .attr("vector-effect", "non-scaling-stroke");
    };
};
// MAIN
plotGrid();
```

B.7 Code used to create the input data files to TigerGraph from the exported data

```
"""
File for wrangling of the input data.
This script is used to adapt the structure of the files to the one required by TigerGraph loading job.
"""
# Import of libraries
import pandas as pd
from ast import literal_eval
# Define path of the folder with the files containing the data of the grid
folder = "../../Input_Data/"
# Read the data from csv files (input files)
ct_df = pd.read_csv(folder+"ct.csv",encoding="utf-8-sig") # Transformer stations
cel_df = pd.read_csv(folder+"celda_trafo.csv") # Low voltage board files
lin_df = pd.read_csv(folder+"linea_bt.csv") # Feeders files
maniobra_df = pd.read_csv(folder+"ManiobraBT.csv") # Files about the general breaker
segmentos_df = pd.read_csv(folder+"segmento_bt.csv", decimal=",") # Segments files
```

```
fusible_df = pd.read_csv(folder+"fusible.csv") # Information of fuses and breakers
cgp_df = pd.read_csv(folder+"cgp.csv",delimiter=";") # Connection points information
########################## WRANGLING OF SEGMENTS DATA ############################
# The structure of segments file is maintained
# Resave the segments file just to replace "," by "." as decimal delimiter
segmentos_df.to_csv("segmentos_output.csv",encoding="utf-8-sig")
###################### WRANGLING OF TRANSFORMER STATIONS DATA ############################
# Replace not recognized characters in utf-8 encoding
ct_df["MUNICIPIO"] = ct_df["MUNICIPIO"].replace("?","O",regex=True)
ct_df["PROPIEDAD"] = ct_df["PROPIEDAD"].replace("?","o",regex=True)
ct_df["DESCRIPCION"] = ct_df["DESCRIPCION"].replace("?","o",regex=True)
# Read the file that contains the coordinates of the transformer stations
ct_coords = pd.read_excel(folder + "Marte_Oviedo.xlsx",sheet_name="Centro Transformacion")
# Merge the transformers dataset with the one that contains the coordinates, using the MSLINK
ct_coords = pd.merge(ct_coords[["Mslink","X","Y"]],ct_df, left_on="Mslink", right_on="MSLINK")
# Remove repeated MSLINK column
ct_coords = ct_coords.drop(columns = ["Mslink"])
# Use "." to separate decimals
ct_coords["X"] = ct_coords["X"].replace(",",".",regex=True)
ct_coords["Y"] = ct_coords["Y"].replace(",",".",regex=True)
# Export data to csv file
ct_coords.to_csv("ct.csv",encoding="utf-8-sig")
########################## WRANGLING OF THE LINES DATA ###################################
# Filter the datasets of the lines: some error in the exportation caused the shift of some columns in some of the rows.
# This is identified when some string or empty cell is placed at the MSLINK column
line_info = lin_df[lin_df.MSLINK.apply(lambda x:x.isnumeric())]
# Select only the desired columns
line_info = line_info[["MSLINK","MSLINK_CTM","CTM","NUMERO_LINEA","DESCRIPCION","PROPIEDAD","ESTADO","TENSION","ESTADO_NORMAL","ESTADO_REAL","EST_NORMAL","EST_OPERAC","NO","NE"]]
# Remove commas from strings: they cause problems during loading jobs in TigerGraph
line_info["DESCRIPCION"]=line_info["DESCRIPCION"].replace(",",":",regex=True)
# Read the file with the coordinate of the lines
line_coords_df = pd.read_excel(folder + "Marte_Oviedo.xlsx", sheet_name="Salida BT")
# Ensure that MSLINK is an integer
line_info["MSLINK"] = line_info["MSLINK"].apply(int)
# Add coordinates to lines dataset: using MSLINK as the common column
line_w_coords = pd.merge(line_coords_df[["Mslink","X","Y"]], line_info, left_on="Mslink", right_on="MSLINK")
# Remove repeated column from merging
line_w_coords = line_w_coords.drop(columns=["Mslink"])
# Set . as decimal separator
line_w_coords["X"] = line_w_coords["X"].replace(",",".",regex=True)
line_w_coords["Y"] = line_w_coords["Y"].replace(",",".",regex=True)
# Avoid commans in strings
line_w_coords["CTM"] = line_w_coords["CTM"].replace(",",".",regex=True)
# Replace not recognized characters
line_w_coords["PROPIEDAD"] = line_w_coords["PROPIEDAD"].replace("?","o",regex=True)
line_w_coords["TENSION"] = line_w_coords["TENSION"].replace("?","o",regex=True)
# Export resulting dataset to csv file
```

```
line_w_coords.to_csv("linea_bt_output.csv", encoding="utf-8-sig")
###################### WRANGLING OF CELDAS ##############################3
# Get the "clave_bdi" from the field "CTM", remove the description of the ct
cel_df['CTM'] = cel_df['CTM'].str.split('-',expand=True)
# Merge the csvs
celdas_output = pd.merge(ct_df,cel_df, left_on='CLAVE_BDI', right_on="CTM", suffixes=("_CT","_CELDA"))
# Add also the mslink of the corresponding breaker of the transformer(directly connected)
celdas_output = pd.merge(celdas_output,maniobra_df[['MSLINK_EM','CLAVE_CT']], left_on='CTM', right_on='CLAVE_CT', suffixes=("_CEL","_MAN"))
# Remove some useless columns
celdas_output.drop(["ESTADO_CT","CLAVE_BDI_CT","TIPO","CTM"],axis=1)
# Ensure that there is no commas in the field "DESCRIPCION"
celdas_output["DESCRIPCION"] = celdas_output["DESCRIPCION"].replace(",",":",regex=True);
# Replaced the not recognized spanish characters
celdas_output["MUNICIPIO"] = celdas_output["MUNICIPIO"].replace("?","O",regex=True);
celdas_output["PROPIEDAD"] = celdas_output["PROPIEDAD"].replace("?","o",regex=True);
# Export dataframe to a csv
celdas_output.to_csv("celdas_output.csv", encoding="utf-8-sig")

############## WRANGLING OF TRANSFORMER BREAKERS ########################3
# Merge the files by the "clave bdi" of the transformation center
man_out = pd.merge(ct_df, maniobra_df, left_on='CLAVE_BDI', right_on='CLAVE_CT', suffixes=('_CT', '_MAN'))
# Remove useless columns of the dataframe
man_out = man_out.drop(['CLAVE_BDI','DESCRIPCION_CT', 'MUNICIPIO', 'ESTADO', 'TIPO', 'PROPIEDAD', 'PROVINCIA'], axis=1)
# Export dataframe to csv file
man_out.to_csv('maniobra_output.csv', encoding='utf-8-sig')

####  Wrangling of segments and fuses file to obtain the nodes and the connections betweeen them ########
segmentos_df = segmentos_df[["MSLINK","TENSION","LONGITUD","UNE","NO","NE"]]
fusible_df = fusible_df[["MSLINK","ESTADO","PROPIEDAD","TIPO","ESTADO_NORMAL","ESTADO_REAL","NO","NE"]]
fusible_df.to_csv("fusible.csv", encoding="utf-8-sig")

# Arrange the file of segmentos to be able to go into TigerGraph

# Create the relationships file for celda-salida linea_bt
line_info = lin_df[['MSLINK','MSLINK_CTM']]
# There are some nonsense rows with non numeric values for the fields mslink and mslink_ctm
line_info = line_info[line_info.MSLINK.apply(lambda x: x.isnumeric())]
# Cast columns to integer type
line_info['MSLINK'] = line_info['MSLINK'].astype(int)
line_info['MSLINK_CTM'] = line_info['MSLINK_CTM'].astype(int)

cel_info = celdas_output[['MSLINK_CT','MSLINK_CELDA']]
relationships = cel_info.merge(line_info,how='left',left_on='MSLINK_CT',right_on='MSLINK_CTM')
relationships = relationships[['MSLINK_CELDA','MSLINK']]
relationships.to_csv('cel_line.csv')

################ WRANGLING OF THE SEGMENTS DATA ##################################3
# Load data of the coordinates of the segments
segmentos_coords_df = pd.read_csv(folder+"segmento_coords.csv", decimal=",")
# Remove the pre calculated "electrical direction"
segmentos_coords_df = segmentos_coords_df.drop(columns=["DIRECCION_ELECTRICA"])
```

```
114 # Replace commas in the names: they are used for numbers most of
    the times
115 segmentos_coords_df["UNE"] = segmentos_coords_df["UNE"].replace("
    ,",".",regex=True)
116 # Load data with the connectivity of the segments
117 segmentos_df = pd.read_csv(folder+"segmento_bt.csv", decimal=",")
118 segmentos_df = segmentos_df.drop(columns=["'SEGMENTOBT'","
    PROPIEDAD","ESTADO","DIRECCION_ELECTRICA","TENSION"])
119 segments_type_df = pd.read_csv(folder+"segmentosBT_type.txt",
    delimiter=";",header=None)
120 segmentos_df = pd.merge(segmentos_df,segments_type_df,left_on="
    MSLINK",right_on=0,how="left")
121 segmentos_df = segmentos_df.drop(columns=[0])
122 segmentos_df = segmentos_df.rename(columns={1:"TIPO"})
123 segmentos_df["TIPO"] = segmentos_df["TIPO"].fillna("U")
124 # Get all the X and Y coordinates from all the vertices and store
     them within an array
125 resultX = segmentos_coords_df.groupby('MSLINK')['X'].apply(list).
    reset_index()
126 resultY = segmentos_coords_df.groupby('MSLINK')['Y'].apply(list).
    to_frame().reset_index()
127 # Create a dataset with the resulting arrays
128 result = pd.merge(resultX,resultY, left_on="MSLINK", right_on="
    MSLINK",suffixes=("_X","_Y"))
129 result = result.drop(columns=["LONGITUD"])
130 final_df = pd.merge(result, segmentos_df, left_on="MSLINK",
    right_on="MSLINK")
131 # Add a column "count" with the number of coordinates assignated
    to the segment
132 final_df["count"] = final_df["X"].apply(len)
133 segment_w_list = final_df
134
135 # Rearrange "from" and "to" buses of each segment
136 counter = 0
137 for nudo in final_df["NO"].to_list():
138     counter = counter+1
139     result1 = final_df.loc[final_df["NO"]==nudo]
140     index1 = final_df.index[final_df["NO"]==nudo].to_list()
141     result1["index"] = index1
142     result2 = final_df.loc[final_df["NE"]==nudo]
143     index2 = final_df.index[final_df["NE"]==nudo].to_list()
144     result2["index"] = index2
145     connected=pd.concat([result1,result2])
146     number_connected = connected.shape[0]
147     if (number_connected>1):
148         # Get data of two of the resulting segments
149         seg1 = connected.iloc[0]
150         seg2 = connected.iloc[1]
151
152         idx1 = seg1["index"]
153         idx2 = seg2["index"]
154         # Get first and last coordinates of both segments
155         # First segment
156         from1X = seg1.X[0]
157         from1Y = seg1.Y[0]
158         to1X = seg1.X[-1]
159         to1Y = seg1.Y[-1]
160         # Second segment
161         from2X = seg2.X[0]
162         from2Y = seg2.Y[0]
163         to2X = seg2.X[-1]
164         to2Y = seg2.Y[-1]
165
166         # Four possible cases
167         if (from1X==from2X): # bus on the from of both segments
168             if (seg1.NE==nudo):
169                 seg1.NE=seg1.NO
170                 seg1.NO=nudo
171             if (seg2.NE==nudo):
172                 seg2.NE=seg2.NO
173                 seg2.NO=nudo
174         elif (from1X==to2X): # bus on the from of seg1 and in the
                to of seg2
```

```
175                 if (seg1.NE==nudo):
176                     seg1.NE=seg1.NO
177                     seg1.NO=nudo
178                 if (seg2.NO==nudo):
179                     seg2.NO=seg2.NE
180                     seg2.NE=nudo
181             elif (to1X==from2X): # bus on the to of seg1 and in the
                    from of seg2
182                 if (seg1.NO==nudo):
183                     seg1.NO=seg1.NE
184                     seg1.NE=nudo
185                 if (seg2.NE==nudo):
186                     seg2.NE=seg2.NO
187                     seg2.NO=nudo
188             elif (to1X==to2X): # bus on the to of both buses
189                 if (seg1.NO==nudo):
190                     seg1.NO=seg1.NE
191                     seg1.NE=nudo
192                 if (seg2.NO==nudo):
193                     seg2.NO=seg2.NE
194                     seg2.NE=nudo
195
196             final_df.loc[idx1,"NO"] = seg1["NO"]
197             final_df.loc[idx1,"NE"] = seg1["NE"]
198             final_df.loc[idx2,"NO"] = seg2["NO"]
199             final_df.loc[idx2,"NE"] = seg2["NE"]
200
201     # Find the unique elements of "NO" AND "NE" -> NODES OF THE
            SYSTEM
202     df = pd.concat([fusible_df[["NO","NE"]],segmentos_df[["NO","NE"
            ]],lin_df[["NO","NE"]]])
203     nodes = pd.unique(df[["NO","NE"]].values.ravel("K"))
204     nodes = pd.DataFrame(nodes)
205     # Get the coordinates of nodes from the pre calculated dataset of
            segments
206     nodes_coordNO = final_df[["NO","X","Y"]]
207     nodes_coordNO["N"] = nodes_coordNO["NO"]
208     nodes_coordNO = nodes_coordNO.drop(columns=["NO"])
209     nodes_coordNE = final_df[["NE","X","Y"]]
210     nodes_coordNE["N"] = nodes_coordNE["NE"]
211     nodes_coordNE = nodes_coordNE.drop(columns=["NE"])
212     nodes_coordNO["X"] = nodes_coordNO["X"].apply(lambda x:x[0])
213     nodes_coordNO["Y"] = nodes_coordNO["Y"].apply(lambda x:x[0])
214     nodes_coordNE["X"] = nodes_coordNE["X"].apply(lambda x:x[-1])
215     nodes_coordNE["Y"] = nodes_coordNE["Y"].apply(lambda x:x[-1])
216     nodes_coords = pd.concat([nodes_coordNO,nodes_coordNE])
217     nodes_coords = nodes_coords.drop_duplicates()
218     nodes_coords.to_csv("nudos.csv",encoding="utf-8-sig")
219
220     # Ensure that the field "UNE" doesn't contain commas
221     final_df["UNE"] = final_df["UNE"].replace(",",".",regex=True)
222     # Replace the commas in the lists and change to a list in format
            string separated by "|"
223     final_df["X"]=final_df["X"].apply(lambda l: "|".join(map(str,l)))
224     final_df["Y"]=final_df["Y"].apply(lambda l: "|".join(map(str,l)))
225     # Export to csv
226     final_df.to_csv("segments_coord.csv",encoding="utf-8-sig",index=
            False)
227
228     ######################## WRANGLING OF FUSES DATA
            ###################################
229     # Read the file that contains the coordinates of the fuses/
            breakers
230     fusible_coords = pd.read_excel(folder + "Marte_Oviedo.xlsx",
            sheet_name="Fusible")
231     # Merge the fuses dataset with their coordinates
232     fusible_w_coords = pd.merge(fusible_coords[["Mslink","X","Y"]],
            fusible_df, left_on="Mslink", right_on = "MSLINK")
233     # Drop repeated column
234     fusible_w_coords = fusible_w_coords.drop(columns = ["Mslink"])
235     # Set . as decimal separator
236     fusible_w_coords["X"] = fusible_w_coords["X"].replace(",",".",
            regex=True)
```

```
fusible_w_coords["Y"] = fusible_w_coords["Y"].replace(",",".",
    regex=True)
# Drop useless column from export of data
fusible_w_coords = fusible_w_coords.drop(columns = ["'FUSIBLE'"])
# Replace not recognized Spanish characters
fusible_w_coords["PROPIEDAD"] = fusible_w_coords["PROPIEDAD"].
    replace("?","o",regex=True)
# Save to csv file
fusible_w_coords.to_csv("fusible.csv")

##################### WRANGLING OF EVENTS DATA
    ######################
# Read the events Excel file
events_df = pd.read_excel(folder + "maniobras registradas desde 1
    enero 2018.xlsx")
# Get the CT dataset to relate the "CLAVE_BDI" with the "MSLINK"
ct_df = ct_df.drop(columns = ["DESCRIPCION","PROVINCIA","
    MUNICIPIO","PROPIEDAD","ESTADO","TIPO"])
# Merge both datasets by means of the "clave_bdi"
events = pd.merge(ct_df,events_df,right_on='CLAVE_CT',left_on='
    CLAVE_BDI', suffixes=("_CT","_EV"))
# Drop one of the "clave"
events = events.drop(columns=["CLAVE_BDI"])
# Create one index to uniquely identify the node in the database
events["MSLINK"] = "EV"+events.index.astype(str);
# Split the events in two different datasets: events of feeders
    and events of breakers/fuses
ev_fuse = events[events["DESCRIPCION_OBJETO"]=="Fusible CM"]
ev_line = events[events["DESCRIPCION_OBJETO"]=="Salida BT"]
ev_maniobra = events[events["DESCRIPCION_OBJETO"]=="Maniobra BT
    en Trafo"]
# Export the three datasets to csv files
ev_fuse.to_csv("events_fuse.csv",encoding="utf-8-sig")
ev_maniobra.to_csv("events_maniobra.csv",encoding="utf-8-sig")
ev_line.to_csv("events_line.csv",encoding="utf-8-sig")

##### WRANGLING OF CONNECTION POINTS DATA ############3
# Change the cgp format from ";" separator and set comma for
    separation of columns
cgp_df.to_csv("cgp_output.csv",encoding="utf-8-sig")

######## WRANGLING OF METERS DATA ############
# Load the data of the meters
meters_df = pd.read_csv(folder + 'Instalacion_Contadores.csv',
    decimal=',' ,encoding='latin-1', low_memory=False)
# Read the file with the information about the phase of each
    meter
phase_df = pd.read_csv(folder + "phase meters.csv",low_memory=
    False,delimiter=";")
# Read file with the S11 report of Pola de Siero
meters_coms_df = pd.read_csv(folder + '
    Topologia_Prime_CTsPoladeSiero.csv',decimal=',' ,encoding='
    latin-1', low_memory=False, delimiter=";")
meters_coms_df = meters_coms_df[["Serial","SID","SSID","State","
    Referencia cct_claveCT"]]
# Add the phase where the meter is connected to
meters_df = pd.merge(meters_df,phase_df,left_on="Referencia",
    right_on="Reference",how="left")
# Select the useful columns
meters_df = meters_df[['Referencia','Clase Material', 'Cups', '
    Equipo', 'Numero Serie', 'Fabricante', 'Fecha Montaje', 'Tipo
    Contador', 'Pot Contratada P1', 'Pot Contratada P2', 'Pot
    Contratada P3', 'Tarifa Acceso', 'Fecha Utl Cambio Potcon', '
    Municipio', 'Cod Entidad Colectiva', 'Cod Entidad Singular', '
    Entidad Singular','Acometida','Phase']]
# Format datetime values
meters_df['Fecha Montaje'] = pd.to_datetime(meters_df['Fecha
    Montaje'])
meters_df['Fecha Utl Cambio Potcon'] = pd.to_datetime(meters_df['
    Fecha Utl Cambio Potcon'])
meters_df["Phase"]=meters_df["Phase"].fillna("Unknown")
# Merge the information of the meters with S11
meters_df = pd.merge(meters_df,meters_coms_df, left_on="
    Referencia", right_on="Serial", how="left")
```

```
meters_df = meters_df.drop(columns=["Serial"])
meters_df[meters_df["State"]=="Switch"]
# Match the meters with the CGPs they are connected to
cgp_df = cgp_df[["MSLINK", "CLAVE_BDI"]]
meters_df = pd.merge(meters_df, cgp_df, right_on="CLAVE_BDI",
    left_on="Acometida")
# Export resulting dataset to csv
meters_df.to_csv("meters.csv", encoding="utf-8-sig")

########### WRANGLING OF CONCENTRATORS ##############
# Read the data of concentrators
concent_df = pd.read_csv(folder + "Inventario_concentradores.csv"
    , decimal=",", encoding="latin-1",low_memory=False)
# Format all the datetime values
concent_df["DT_STATE_CONC"] = pd.to_datetime(concent_df["
    DT_STATE_CONC"])
concent_df["CONFIG_DATE"] = pd.to_datetime(concent_df["
    CONFIG_DATE"])
concent_df["FECHA_AVISO"] = pd.to_datetime(concent_df["
    FECHA_AVISO"])
concent_df["FECHA_CREACION"] = pd.to_datetime(concent_df["
    FECHA_CREACION"])
concent_df["FECHA_SUPERVISION"] = pd.to_datetime(concent_df["
    FECHA_SUPERVISION"])
concent_df["DATE_COMM_STATE_CONC"] = pd.to_datetime(concent_df["
    DATE_COMM_STATE_CONC"])
concent_df["MSLINK"] = concent_df.index+1
# Avoid ">" and "<" symbols that cause probles in the loading of
    the data to TigerGraph
concent_df = concent_df.drop(columns=["NOMBRE_CT","PROVINCIA_CT",
    "MUNICIPIO_CT","ENT_COLECTIVA_CT","INTERVALO"])
concent_df = concent_df.rename(columns={"CIERRE>7":"CIERRE_7","
    CURVA>5":"CURVA_5"})
# Reload CT data
ct_df = pd.read_csv(folder + "ct.csv")
# Match every concentrator with the transformer
conc_ts = pd.merge(concent_df, ct_df, right_on='CLAVE_BDI',
    left_on="CODIGO_CT", suffixes=("_CONC","_CT"))
# Get only the relationship (MSLINK_CT, MSLINK_CONCENTRATOR)
conc_ts = conc_ts[["MSLINK_CONC", "MSLINK_CT"]]
# Add the MSLINK of the CT to an additional column in the dataset
     of the concentrators
concent_df = pd.merge(concent_df,conc_ts,right_on="MSLINK_CONC",
    left_on="MSLINK")
# Export the resulting dataset to a csv
concent_df.to_csv("concentrators.csv",encoding="utf-8-sig")

############ WRANGLING OF ROUTERS DATA #############3
# Load the data
routers_df = pd.read_csv(folder + "Inventario_routers.csv")
# Format the datetime values
routers_df["FECHA_INICIO"] = pd.to_datetime(routers_df["
    FECHA_INICIO"])
routers_df["FECHA_COBERTURA"] = pd.to_datetime(routers_df["
    FECHA_COBERTURA"])
routers_df["FECHA_VFIRMWARE"] = pd.to_datetime(routers_df["
    FECHA_VFIRMWARE"])
# Export data of the routers to a CSV (nodes file)
routers_df.to_csv("routers.csv", encoding="utf-8-sig")
# Relate the routers with the corresponding CT
router_ts = pd.merge(routers_df, ct_df, right_on='CLAVE_BDI',
    left_on="CODIGO_CT", suffixes=("_ROUTER","_CT"))
# Get only the relationship router-CT
router_ts = router_ts[["MSLINK", "ROUTER_ID"]]
# Export relationships csv file
router_ts.to_csv("router_ct.csv", encoding="utf-8-sig")
```

Bibliography

[1] Deloitte 2020 power and utilities industry outlook.

[2] ZIV 4CCT concentrator data sheet.

[3] TigerGraph: The First Native Parallel Graph. Documentation. [Online]. Available: docs.tigergraph.com/, 2019.

[4] D3.js - Data-Driven Documents. [Online]. Available: https://www.d3js.org/, 2020.

[5] Electric Vehicle Outlook 2020. BloombergNEF. 2020.

[6] Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base. International standard, International Electrotechnical Commission, June 2020.

[7] Neo4j Graph Platform - The Leader in Graph Databases. [Online]. Available: https://www.neo4j.com/, 2020.

[8] Common Information Model Primer: Third Edition. Standard, Electric Power Research Institute, Palo Alto, CA 94304-1338, June 2015.

[9] G.M. Adelson-Velskii and E.M. Landis. An algorithm for the organization of information. *Proceedings of the USSR Academy os Sciences*, 1962.

[10] D Bowermaster, M Alexander, and M Duvall. The Need for Charging: Evaluating utility infrastructures for electric vehicles while providing customer support. *IEEE Electrification Magazine*, 5:56–67, 2017.

[11] R. Dai, G. Liu, Z. Wang, B. Kan, and C. Yuan. A novel graph-based energy management system. *IEEE Transactions on Smart Grid*, 2019.

[12] Santiago Grijalva. Topology processing and real-time applications. *PowerWorld Client Conference, Operations and Planning Track*, 2007.

[13] Jose Manuel Carou Álvarez. Estimación de incidencias y patrones de uso en redes de distribución eléctrica utilizando concentradores de smartmeters. 2018.

[14] Gustav Kirchhoff. Über der Auflösung der Gleichungen, auf welche man bei der Untersuchung der linearen Verteilung galvanischer Ströme geführt wird. *Ann. Phys. Chem. 72*, pages 497–508, 1847.

[15] K. S. Larsen. AVL trees with relaxed balance. In *Proceedings of 8th International Parallel Processing Symposium*, pages 888–893, 1994.

[16] G. Ravikumar and S. A. Khaparde. CIM Oriented Graph Database for Network Topology Processing and Applications Integration. In *2015 50th International Universities Power Engineering Conference (UPEC)*, pages 1–7, Sep. 2015.

[17] G. Ravikumar and S. A. Khaparde. A Common Information Model Oriented Graph Database Framework for Power Systems. *IEEE Transactions on Power Systems*, 32(4):2560–2569, 2017.

[18] Alvin Razon, Tony Thomas, and Venkat Banunarayanan. Advanced distribution management systems. *IEEE power and energy magazine*, January 2020.

[19] Ian Robinson, Jim Webber, and Emil Eifrem. *Graph Databases*. O'Reilly, Beijing, 2 edition, 2015.

[20] ITU-T standard G.9904. Narrowband orthogonal frequency division multiplexing power line communication transceivers for prime networks. 2012.

[21] Koji Ueno, Toyotaro Suzumara, Naoya Maruyama, Katsuki Fujisawa, and Satoshi Matsuoka. Efficient breadth-first search on massively parallel and distributed-memory machines. *Parallel and Distributed-Memory Machines.*, 2:22–35, 2017.

[22] Zhangxin Zhou, Chen Yuan, Ziyan Yao, Jiangpeng Dai, Guangyi Liu, Renchang Dai, Zhiwei Wang, and Garng M. Huang. CIM/E Oriented Graph Database Model Architecture and Parallel Network Topology Processing. In *2018 IEEE Power Energy Society General Meeting (PESGM)*, pages 1–5, Aug 2018.