

RESEARCH

Open Access



A model-based approach for converting CGMES power system models into operational networks

Ammar Memari^{1*} and Adam Aljamous¹

From The 12th DACH+ Conference on Energy Informatics 2023
Vienna, Austria. 4-6 October 2023. <https://www.energy-informatics2023.org/>

*Correspondence:
ammar.memari@jade-hs.de

¹ Department of Engineering
Sciences, Jade University
of Applied Sciences,
Friedrich-Paffrath-Str. 101,
26389 Wilhelmshaven, Germany

Abstract

This paper presents a model-based approach for converting Common Grid Model Exchange Specification (CGMES) power system models into operational networks. The approach utilizes the Model-Driven Architecture (MDA), borrowed from software engineering, to create a Platform-Independent Model (PIM) saved in a graph database, neo4j. The PIM is then transformed into Platform-Specific Model (PSM) using cypher queries. Finally, from the PSM, a Pandapower network is generated that is used later for running power flow simulations. The Common Information Model (CIM) ontology classes, which form the metamodel behind CGMES, are loaded into the same database, connecting them to the grid instances, allowing for automatic validation and inference. The proposed approach has been successfully applied to large-scale real-world grids that include renewable energy sources.

Keywords: CGMES, Neo4j, Pandapower, MDA, Ontology

Introduction

Modern power grid is facing new challenges due to the increasing integration of renewable energy sources such as wind and solar power, as well as energy efficiency measures. Intermittent renewable generation and energy efficiency measures have led to an increasing demand for a more secure, stable, and efficient power systems. One of the main challenges is the conversion of large-scale models of the power system, which includes various components and interconnections, into operational networks in order to analyze and simulate power flow in different situations.

European network operators are encouraged to use the CGMES format for intercommunicating the state of their networks. Being able to convert this format into a platform-independent representation in a graph database increases the accessibility and interpretability of the data, facilitating data-driven decisions, inspection, validation, and further conversion into other formats.

The CGMES format features multiple eXtensible Markup Language (XML)/ Resource Description Framework (RDF) files, which are cumbersome to handle without proper tools, especially for large real-world grids. Therefore, the proposed approach provides a more efficient and accessible solution. The platform-independent model is stored in a neo4j graph database in the form of nodes and arcs, which allows for an interactive visualization of the data, filtering using advanced graph queries, and the application of transformations in the form of cypher queries to derive platform-specific models.

Relevant technologies

The CGMES is a standard format for exchanging power system models. It is used to describe the elements of a power system, such as generators, transformers, transmission lines, and loads, and their interconnections. The format is based on the RDF and uses the Web Ontology Language (OWL) to define the ontological concepts. It is based on the CIM developed by the International Electrotechnical Commission (IEC) (Uslar et al. 2012), which provides a standardized way of representing power system data in a machine-readable format. The CGMES format extends the CIM by adding specific profiles and constraints relevant to the exchange of power system models.

The CGMES format provides a common language for power system models that can be shared between different software tools and applications. It enables interoperability between different power system models, facilitating the exchange of data between network operators, regulators, and other stakeholders.

Pandapower (Thurner et al. 2018) is an open-source Python library for power system analysis and optimization. It provides a set of tools for creating, modifying, and analyzing power system models. Pandapower can be used to model a wide range of power system components, including generators, transformers, loads, and transmission lines, and to perform various types of analyses, such as power flow analysis, short-circuit analysis, and optimal power flow.

Pandapower is built on top of the Pandas widespread data analysis library and provides a user-friendly interface for creating and manipulating power system models. It also supports integration with other Python libraries, such as Matplotlib for visualization and NumPy for numerical computations. Pandapower's modular architecture allows for easy extension and customization, making it a flexible and powerful tool for power system analysis and optimization.

A graph database is a type of database that stores data in a graph-like structure. It consists of nodes (also called vertices) that represent entities or objects and edges (also called relationships) that connect the nodes and represent the connections between them. Graph databases are used to model and store data that has complex relationships and can be difficult to represent in a traditional tabular database.

These databases offer several advantages over traditional ones, such as the ability to perform complex queries that involve traversing multiple relationships between nodes, making them well-suited for handling large and complex graphs.

Neo4j (2023) is a leading open-source graph database that is designed for storing and querying graph data. It is built on a native graph storage and processing engine that provides fast performance for complex queries. Neo4j allows users to model and store

complex data structures, including directed and undirected graphs, hierarchical structures, and networks.

Neo4j provides a flexible and expressive query language called Cypher, which allows users to query and manipulate data in a graph database using a simple syntax. It also supports a range of APIs and integrations, making it easy to use with a variety of programming languages and platforms.

Neo4j has become popular in a wide range of domains, including social networking, recommendation engines, fraud detection, and network analysis. Its ability to handle complex relationships between entities makes it well-suited for applications in the power system domain, where data is often highly interconnected and difficult to represent in traditional databases.

MDA (2006) is an approach to software development that emphasizes the use of models to guide the design, implementation, and testing of software systems. It is a methodology that borrows concepts from software engineering to create a standardized approach to software development.

MDA is based on the idea of separating the concerns of software development into different layers or levels of abstraction. The highest level of abstraction is the Computation-Independent Model (CoIM), which represents the system in terms of its requirements and functionality. The CoIM is then transformed into a PIM, which describes the system in terms of its architecture and behavior, but is independent of any specific platform or implementation language. Finally, the PIM is transformed into a PSM, which specifies the details of the implementation, such as the programming language and operating system.

MDA provides a standardized approach to software development that separates concerns and facilitates the reuse of software components. It allows for the creation of software systems that are more modular, maintainable, and adaptable to changing requirements. In our paper, we will borrow the concept of MDA from software engineering and apply it to the modeling of power networks. Thereby, we aim to create a standardized approach to the modeling of power networks that separates concerns and facilitates the reuse of models.

Related work

The paper titled “Modeling and Processing Big Data of Power Transmission Grid Substation Using Neo4j” (Perçuku et al. 2017) proposes a method for modeling and processing large volumes of data from power transmission grid substations using the Neo4j graph database. The paper highlights the challenges posed by the increasing volume and variety of data from various sources, and the inadequacy of traditional relational databases in handling these data sets.

The authors demonstrate how Neo4j’s graph database can be used to model and process the data of power transmission grid substations with two power transformers, and then add a new power transformer to simulate the evolving feature of Neo4j database according to the business needs. The paper emphasizes the benefits of using Neo4j, including faster response times and improved performance, and discusses the practical implications of this approach.

The paper's relevance to our work lies in the fact that the authors have found Neo4j to be useful for hosting a CGMES model, highlighting the performance benefits of this conversion. While our approach focuses on practicality and the ability to transform the data into further models and perform semantic inference and verification, the paper provides a useful perspective on the use of Neo4j for modeling and processing large volumes of data in the power system domain.

Another research (Dalćeković et al. 2021) also focused on representing (smart) power grids in graph databases. In contrast to our work, which focuses on converting CGMES power system models into operational networks using a model-based approach, their research primarily aims to optimize graph models for efficient retrieval of power network model change history. While both studies employ graph databases in the context of power grids, our approach utilizes the MDA and Neo4j to create a PIM and transform it into PSM. This enables automatic validation and inference by connecting CIM ontology classes to grid instances. The findings from the related work can complement our study by providing insights into designing optimal graph models that satisfy smart grid requirements, potentially enhancing the efficiency of our proposed approach when dealing with historical features in operational technologies.

The paper titled "A Novel Extended Graph Strategy to Model Microgrids" (Reyes et al. 2019) presents an extended graph model that goes beyond traditional impedance models to incorporate additional information about grid elements, such as saturation, current and voltage limits, and energy resource availability. This approach demonstrates the versatility and extensibility of graph models in representing not only the grid components and their relations, but also the social and environmental factors that influence microgrid systems.

In relation to our work, both studies employ graph databases to model power grid systems; however, our research focuses on converting into operational networks. The extended graph model presented in the related work can complement our study by showcasing the potential of graph models to store and manage additional information that may be valuable for design, evaluation, and operation of microgrid systems. Furthermore, the graph metrics proposed in the related work could be utilized to assess the performance and robustness of our approach when applied to microgrids. By integrating the insights from the extended graph model, our proposed method for converting CGMES models may be further enhanced, ultimately contributing to the development of more efficient and robust power grid systems.

The authors of "GridKG: Knowledge Graph Representation of Distribution Grid Data" (Kor et al. 2020) explore the application of knowledge graphs to represent information about power grids, specifically using Neo4j and the RDF format. The paper proposes GridKG, a knowledge graph model that captures various components of a power grid, their connections, and relationships. They also present an algorithm for identifying electrical paths within the grid and demonstrate the utility of the knowledge graph through several grid analysis examples related to primary switches.

In relation to our work, both studies employ graph databases, specifically Neo4j, to model power grid systems. However, our research focuses on model convertibility into operational networks using a model-based approach. The GridKG model presented in the related work can serve as a valuable reference for understanding the potential

of knowledge graphs in representing and analyzing power grid data. By incorporating insights from GridKG, our proposed method for converting CGMES models into operational networks may benefit from the efficient representation and querying capabilities of GridKG. Furthermore, the algorithm for identifying electrical paths and the grid analysis examples provided in the related work could inspire additional applications and enhancements to our approach, aiding in the advancement of power grid systems that are both more effective and resilient.

The paper titled “Topology Modeling and Analysis of a Power Grid Network Using a Graph Database” (Kan et al. 2017) proposes a novel method for storing, modeling, and analyzing power grid data using the Neo4j. The authors present an architecture for constructing power grid network models and design single- and multi-threading systems for initial energization analysis. They also develop shortest path search functions and conditional search functions based on Neo4j’s capabilities. By comparing the performance of their graph database approach with traditional relational databases, such as PostgreSQL, the study demonstrates the efficiency and effectiveness of using graph databases for power grid network modeling and analysis.

In relation to our work, both studies employ Neo4j to model power grid systems; however, our research focuses specifically on converting CGMES power system models after uploading to Neo4j further into operational networks employing MDA. The methodology presented in the related work can serve as a valuable reference for understanding the potential benefits of graph databases in handling complex data relations within power grid networks, and assessing their performance. By integrating insights from this study, our proposed method may benefit from improved performance in topology modeling and analysis tasks. Furthermore, the single- and multi-threading systems developed by the authors could inspire additional optimizations to our approach when dealing with large-scale real-world grids.

The paper titled “A Novel Graph-Based Energy Management System” (Dai et al. 2020) introduces a graph-computing based Energy Management System (EMS) aimed at significantly reducing the computational latency typically experienced in current EMSs. By leveraging nodal and hierarchical parallelism, the authors develop high-speed graph-based state estimation, power flow, and contingency analysis applications that can be completed within a Supervisory Control And Data Acquisition (SCADA) sampling cycle. The case study results demonstrate that their proposed graph-based approach is over 20 times faster than traditional commercial EMSs based on relational databases and serial computing.

In relation to our work, both studies employ graph databases for modeling power grid systems. The methodology presented in the related work highlights the potential of utilizing parallel computing facilitated by graph databases to achieve improved situational awareness in energy management systems. By incorporating insights from this study into our proposed method for converting CGMES models into operational networks, we may benefit from enhanced performance when executing tasks such as state estimation or contingency analysis. Furthermore, exploring parallel solution techniques inspired by this novel EMS could potentially extend its applicability beyond these specific tasks to other areas like dynamic security assessment or transient stability simulation within large-scale real-world grids.

Authors of the paper titled “Towards Graph Machine Learning for Smart Grid Knowledge Graphs in Industrial Scenarios” (Di Donato et al. 2021) explore the potential of combining Knowledge Graphs (KGs) and Graph Machine Learning (GML) techniques to manage massive power resources and provide intelligent applications within Smart Grid systems. The authors present a methodology for extracting various significant views of Smart Grid Knowledge Graphs (SGKGs), which are based on the IEC CIM standard, by iteratively applying a series of transformations. They implement this approach using a declarative method to ensure easier portability and deploy it as a stateless microservice, facilitating modular integration with existing Semantic Platforms.

In relation to our work, both studies employ graph databases for modeling power grid systems and utilize semantics based on the CIM ontology. The methodology presented in the related work highlights the importance of leveraging GML techniques alongside SGKG representations to enable diverse applications within industrial scenarios. By incorporating insights from this study into our proposed method into operational networks, we may benefit from enhanced performance when managing large-scale real-world grids through improved knowledge extraction capabilities offered by GML methods applied over multiple views of grid data. Furthermore, exploring these iterative transformation approaches could potentially extend their applicability beyond specific tasks or objectives while ensuring easy portability and modular integration with other components within smart grid ecosystems.

In our previous work, titled “A Data Center Simulation Framework Based on an Ontological Foundation,” (Memari et al. 2016) we developed a data center simulation framework as part of the IT-for-Green project. The primary goal was to support the analysis and design of energy-efficient data centers by simulating alternative architectures within a Corporate Environmental Management Information System (CEMIS) platform. Our approach focused on flexibility, high interoperability, and open standards by building the framework upon a foundational data center ontology.

This earlier research shares similarities with our current study in terms of utilizing ontologies for modeling complex systems, and using an MDA approach. It differs, however, in several aspects. In contrast to focusing on data centers as the application domain, our present work aims at converting power grid models into operational networks. Our current study uses a graph database for a better performance in real-world applications.

By applying lessons learned from developing an ontological foundation for simulating energy-efficient designs in previous works, we can further refine and optimize our current model-based approach for transforming into operational networks that include renewable energy sources while maintaining adaptability to future requirements.

Methodical approach

In the project discussed in this paper, the original CGMES format corresponds to the PIM layer. Using Neo4j and Python, CGMES RDF/XML is first converted into PIM-Graph and loaded into a Neo4j database. Then the PIM-Graph is transformed into the PSM graph inside Neo4j using cypher queries and keeping relations to original nodes for later debugging. The operational Pandapower network is created later based on the new nodes, again utilizing cypher queries. Figure 2 shows the

subsequent steps of the conversion process for this work, and how they correspond to the MDA approach, whereas Fig. 1 describes the whole method including steps that will be discussed later in the future work section, but still contribute to understanding the advantages of an MDA-based approach.

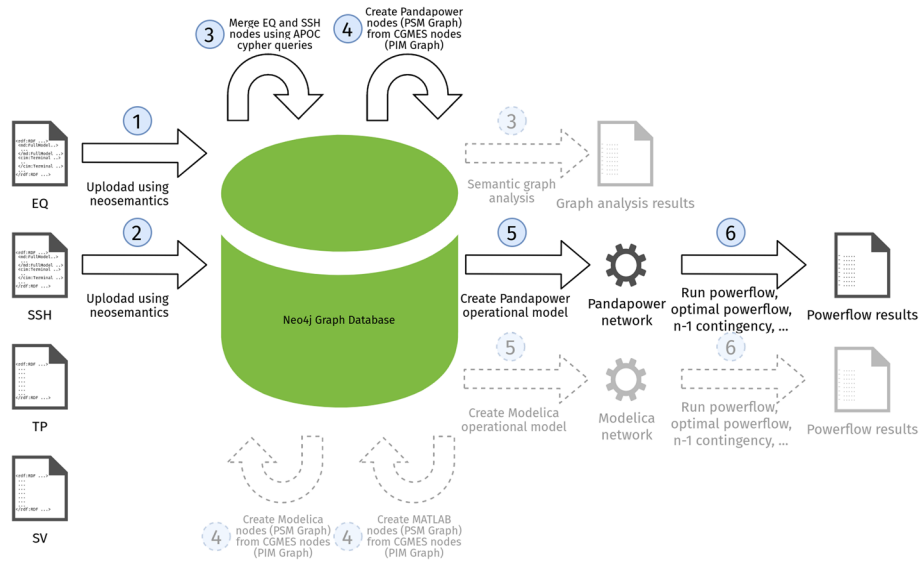


Fig. 1 Overview of the methodology used, including parts that were not directly parts of the approach (grayed out)

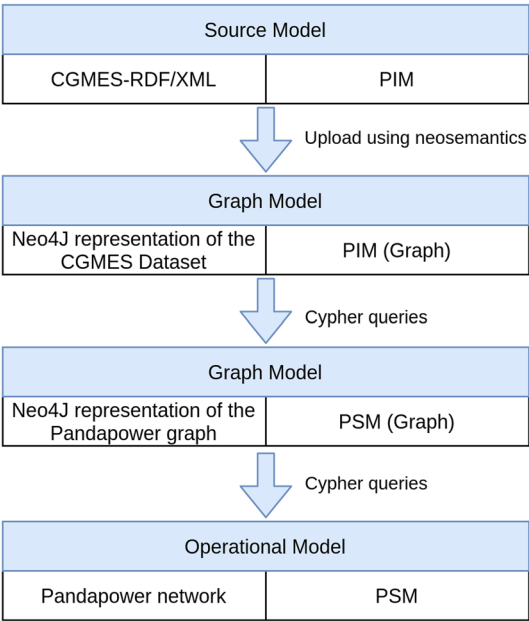


Fig. 2 Conversion steps from CGMES-XML/RDF to the Pandapower model

Importing the Data into Neo4j

A CGMES dataset comprises multiple RDF files, which typically include the Equipment (EQ), Substation Hierarchy (SSH), Topology (TP), and State Variables (SV) files, which provide information about different aspects of the power system model. Collectively, these files contain a comprehensive representation of the power system. We focus for loading our data on the EQ and SSH files.

To load RDF files into Neo4j, a plugin called “neosemantics” ([n10s](#)) is required. This plugin provides support for importing RDF data and querying it using the Cypher query language. Installing the plugin is a straightforward process. It involves downloading the plugin package, and copying it to the “plugins” directory of the Neo4j installation. Once the plugin is installed, it can be used to import RDF data into Neo4j.

Using the plugin, the EQ and SSH files can be imported with simple procedure calls. Since the files have been saved on the drive, they should be loaded as follows:

```
CALL n10s.rdf.import.fetch("file:///C:\\EQ.rdf", "RDF/XML");
```

When we import the EQ and SSH files into Neo4j, we end up with two separate sets of nodes, one for each file. In order to create a unified representation of the power system model, it is necessary to merge these two sets. This can be achieved by creating a new set of nodes that incorporates the properties from both sets. Specifically, each node in the new set will contain a merged set of properties that combines the relevant information from the EQ and SSH files. This merge-process depends on the unique IDs of the nodes, a property found as the last part of the attribute `uri` of each imported node. Before we can efficiently use these IDs, we must isolate them into another attribute, as follows:

```
MATCH(n) SET n.id = split(n.uri, "#")[1];
```

Executing such cypher queries does not have to be done manually, it can rather be scripted using python. Simply creating a driver and using it to execute cypher queries over a connection to the Neo4j database. Once the driver is created, a session is started using `driver.session()`:

```
driver = GraphDatabase.driver(uri="bolt://localhost:7687",
                             auth=("neo4j", "neo4j91"))
session = driver.session()
session.run("MATCH(n) SET n.id = split(n.uri, '#')[1];")
```

After creating the `id` attribute for each node, it is advisable to create an index on it, to speedup further queries that use it:

```
CREATE INDEX node_id_index FOR (n:ns1__Resource) on (n.id);
```

Merging now based on the `id` attribute is much faster, and can be performed as follows:


```
MATCH (n) WITH n.id AS id, collect(n) as nodes
WHERE size(nodes) > 1
CALL apoc.refactor.mergeNodes(nodes) YIELD node RETURN count(*)
```

Notice that for running this query, the APOC (2023) plugin is required, as mentioned before, installing plugins is a straightforward process in Neo4j.

Creating the Pandapower model

To create Pandapower model nodes in Neo4j, the relevant information must be extracted from the nodes of the CGMES model. This is done after the data have been loaded and prepared as seen above, and involves identifying and selecting the necessary data fields that represent the various components of the power system. These extracted data are then used to create the new nodes in the Neo4j graph database.

In CGMES, the `ConnectivityNode` corresponds to a bus in Pandapower. However, to create a bus in Pandapower, the type must be specified as either “b” or “n”. This information is not available directly in the `ConnectivityNode`, but can be found in the connected `BusbarSection`. Additionally, the voltage in kv of the Pandapower bus is not specified in the `ConnectivityNode`, and must therefore be obtained from the indirectly connected `BaseVoltage` node. To be able to collect these information from the various aforementioned nodes, the `MATCH` statement of the query must look like this:

```
MATCH (t1:ns1__Terminal)-[]->(cn1:ns1__ConnectivityNode)-[]->
(t2:ns1__Terminal)-[]->(bs:ns1__BusbarSection)-[]->
(bv:ns1__BaseVoltage)
```

For Buses of type “n” we need to obtain the voltage level through a different path:

```
MATCH (cn2:ns1__ConnectivityNode)-[]->(bay:ns1__Bay)-[]->
(vl:ns1__VoltageLevel)-[]->(bv:ns1__BaseVoltage)
```

In the query, we not only extract relevant data from the CGMES nodes, but we also create a corresponding `Bus` node within the same query. The `Bus` node is given an `id`, `name`, `type`, and `in_service` attributes. To maintain a traceable and debuggable link to the original CGMES `ConnectivityNode` node, we establish a relationship between the `Bus` node and the `ConnectivityNode` node called `pp_hasInfoFrom` as seen in Fig. 3.

Once this query is executed, the basic framework of the grid is established with the created `Bus` nodes. However, the components connected to these buses are yet to be specified.

Furthermore, we use aliases to return the created attributes with column names that match those of the Pandapower bus dataframe. This allows us to create the bus dataframe of the network directly without using the Pandapower API, thus reducing

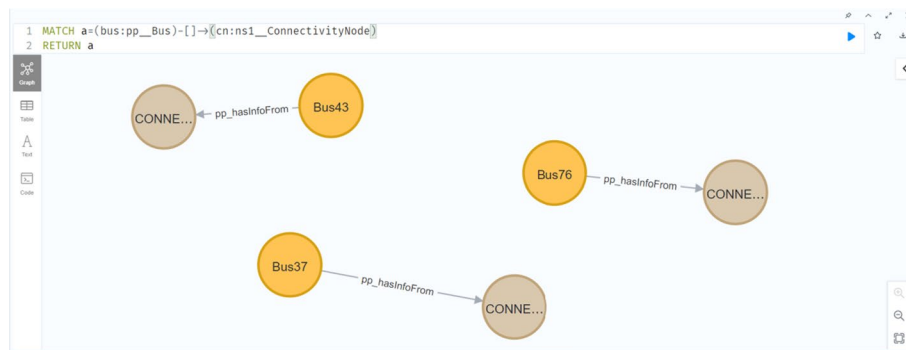


Fig. 3 The created Bus nodes and their corresponding ConnectivityNodes

overhead. All these dataframes will be later collected into a single Pandapower network. The RETURN statement of the query looks like this:

```
RETURN DISTINCT
    id(bus) AS id,
    bus.name AS name,
    bus.vn_kv AS vn_kv,
    bus.type AS type,
    bus.in_service AS in_service
ORDER BY bus.name
```

And the dataframe is created in such a way:

```
bus_results = session.run(bus_query)
bus_df = pd.DataFrame(bus_results, columns=bus_results.keys())
# create an empty pandapower network and set the dataframe directly
net = pp.create_empty_network()
net.bus = bus_df
```

Once the basic structure of the grid has been created, the next step is to map the other components, such as lines and transformers, to their equivalent Pandapower counterparts. This involves creating new nodes and relationships in the neo4j database and connecting them to the appropriate buses that we had created earlier. We made several mappings between the CGMES and Pandapower models to translate the data from one format to the other. These mappings are described in the following sections, although it is not the exhaustive list.

Line

The Pandapower Line element is modeled based on the `ACLineSegment` object in the CGMES format, and most of its attributes are obtained from it, such as id, name, and length. However, some additional attributes must be retrieved from the connected `CurrentLimit` and `SvStatus` objects in the CGMES model as seen in Table 1. Some attributes need to be calculated, such as `r_ohm_per_km`, a step that can directly be accomplished in the cypher query:

```
...
r_ohm_per_km: toFloat(l.`ACLineSegment.r`)/toFloat(l.`Conductor.length`),
...
```

To ensure proper organization, we assign the newly created elements their corresponding classes (also known as labels in Neo4j), namely `pp__Line` and `pp__Element`. Additionally, we connect the newly created line elements to their corresponding buses that we previously created, ensuring that they are placed correctly within the network topology. This is achieved using the following parts of the same cypher query:

```
...
MERGE (line:pp__Line {
...
})-[:pp_hasInfoFrom]->(acl)

SET line:pp__Element
MERGE (bus1)-[:pp_from_bus]->(line)-[:pp_to_bus]->(bus2)
```

The last line creates a directed relationship between two previously created nodes, `bus1` and `bus2`, and a newly created node `line`. The direction of the relationship indicates the flow of power. The `MERGE` keyword is used instead of `CREATE` to ensure that the classes and relationships are only created if they don't already exist. Figure 4 shows the new nodes and their connection to the original `ACLineSegments`.

In pandapower, the `std_typ` attribute for a line refers to the standardized line type. It is a string that identifies the line type based on its characteristics such as the conductor type, insulation, and number of phases. Pandapower has a pre-defined set of standardized line types that users can choose from, or they can define their own based on their specific network requirements. The `std_typ` attribute is used

Table 1 Mapping of the CGMES model attributes of various classes to the attributes of a Pandapower line element

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
ACLineSegment	id	id	Line
	IdentifiedObject.name	name	
	Conductor.length	length_km	
	ACLineSegment.r / Conductor.length	r_ohm_per_km	
	ACLineSegment.x / Conductor.length	x_ohm_per_km	
	(ACLineSegment.bch * 10 ⁹) / Conductor.length	c_nf_per_km	
CurrentLimit	(norm.CurrentLimit.value) / 1000	max_i_ka	
	(norm.CurrentLimit.value) / (emrg. CurrentLimit.value)	df	
SvStatus	SvStatus.inService	in_service	

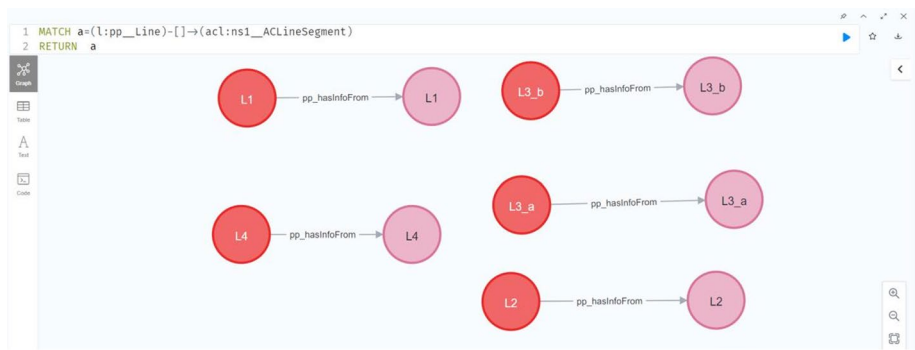


Fig. 4 The created Line nodes and their corresponding ACLineSegments after running the query

to determine the electrical parameters of the line, such as resistance and reactance, based on the line type. In our test network and in practice, we were faced with the situation of having to add our own types. This can be done by creating our own types using `create_std_types` and then setting the column `std_type` in the line data-frame using the method `find_std_type_by_parameter`.

Switch

The Pandapower Switch element corresponds to the Breaker or the Disconnector CGMES classes, which sets its type attribute. Whether the switch is closed or open is determined by its `SvStatus` attribute in the CGMES model, which indicates whether the switch is connected or disconnected, respectively.

The query to create the switches and connect them to the appropriate buses is similar to the aforementioned one for the lines. It is worth mentioning, that the closed attribute must be cast into a boolean as follows:

```
...
closed: toBoolean(sv.`ns1__SvStatus.inService`)
...
```

Table 2 shows a summary of the mapping, and Fig. 5 depicts the created nodes.

Generator

A Pandapower Generator (`gen`), such as a wind turbine, maps to a CGMES SynchronousMachine, which represents an electromechanical device that works with

Table 2 Mapping of the CGMES model attributes of various classes to the attributes of a Pandapower switch element

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
Disconnector or Breaker	id	id	Switch
	IdentifiedObject.name	name	
	DS or CB	type	
SvStatus	SvStatus.inService	closed	

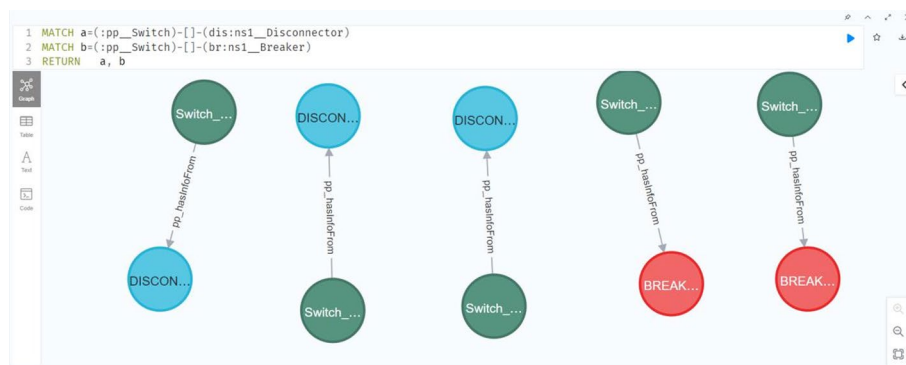


Fig. 5 The created Switch nodes and their corresponding Breakers and Disconnectors

Table 3 Mapping of the CGMES model attributes of various classes to the attributes of a Pandapower Generator element

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
SynchronousMachine	id	id	Generator
	IdentifiedObject.name	name	
	RotatingMachine.p	p_mw	
	RotatingMachine.ratedS	sn_mva	
	RotatingMachine.ratedU	vn_kv	
	SynchronousMachine.maxQ	max_q_mvar	
	SynchronousMachine.minQ	min_q_mvar	
	RotatingMachine.ratedPowerFactor	cos_phi	
	SynchronousMachine.satDirectSubtransX	xdss_pu	
	SynchronousMachine.r	rdss_ohm	
	RotatingMachine.ratedU / bus.vn_kv	vm_pu	
Substation	id	power_station_trafo	
SvStatus	SvStatus.inService	in_service	

a shaft, that rotates synchronously with the network. Table 3 describes the mapping between Generator and SynchronousMachine.

The query for creating these components contains a MATCH statement for collecting the involved classes along a path:

```
MATCH (st:ns1__Substation)-[:]- (tgu:ns1__ThermalGeneratingUnit)-[:]-
(sm:ns1__SynchronousMachine)-[:]- (t:ns1__Terminal)-[:]->
(cn:ns1__ConnectivityNode)-[:]- (bus:pp__Bus)
OPTIONAL MATCH (sm)<--(sv:ns1__SvStatus)
```

After running the query, the Generator classes are created and connected to their corresponding Buses, and to their original SynchronousMachines by the pp_hasInfoFrom relationship as seen before.

Transformer

A `PowerTransformer` according to CIM is “an electrical device that consists of two or more coupled windings with or without magnetic cores to introduce mutual coupling between circuits. Transformers can be used to control voltage and phase shift (active power flow). A power transformer may consist of separate transformer vessels, which need not be identical.” (Uslar et al. 2012).

The `PowerTransformer` class represents both the two-winding transformer and the three-winding transformer. However, in Pandapower, these transformers are distinguished by the `trafo` and `trafo3w` classes. The main difference between these two types is the number of windings, with the `trafo3w` having three and the `trafo` having two. For a successful mapping into the two Pandapower classes, the number of connections (`PowerTransformerEnds`) of the `PowerTransformer` must be counted.

We accommodate this distinction in our workflow by using separate queries for the two classes. A long `MATCH` statement is required for this query, since many classes on the CGMES side should contribute to the attributes of the `trafo` and `trafo3w` classes as detailed in Table 4. Using the following `WHERE` statement, we keep only the two winding `PowerTransformers`.

```
WHERE NOT (:pp__Bus)-[*2]-
(:ns1__Terminal{`ns1__ACDCTerminal.sequenceNumber`: "3"})-[]->
(pt:ns1__PowerTransformer) AND bus1.vn_kv < bus2.vn_kv
```

Table 4 Mapping of the CGMES model attributes of various classes to the attributes of a Pandapower 2 winding Trafo element

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
PowerTransformer	id	id	Trafo
	IdentifiedObject.name	name	
	PowerTransformer.isPartOfGeneratorUnit	power_station_unit	
	PowerTransformerEnd.phaseAngleClock	shift_degree	
PowerTransformerEnd	PowerTransformerEnd.ratedS	sn_mva	
	PowerTransformerEnd.ratedU	vn_hv_kv	
		vn_lv_kv	
CurrentLimit	(norm.CurrentLimit.value) / (emerg.CurrentLimit.value)	df	
RatioTapChanger	TapChanger.lowStep	tap_min	
	TapChanger.highStep	tap_max	
	TapChanger.neutralStep	tap_neutral	
	RatioTapChanger.stepVoltageIncrement	tap_step_percent	
	TapChanger.ltcFlag	oltc	
SvTapStep	SvTapStep.position	tap_pos	
SvStatus	SvStatus.inService	in_service	

This statement also determines which `PowerTransformerEnd` is the low voltage one and which one is the high voltage by comparing the rated voltages of their buses.

For a three-winding transformer, there are three winding ends, and we need to determine which one is the high voltage end, which one is the medium voltage end, and which one is the low voltage end. This is done based on the winding voltage levels. Using the following statement, Neo4j assigns the names `bus1`, `bus2`, and `bus3` accordingly.

```
...
WHERE bus1.vn_kv < bus2.vn_kv < bus3.vn_kv
...
```

Their connected `PowerTransformerEnds` get automatically then the names `tre1`, `tre2`, and `tre3` from which we can then read the `ratedS` and `ratedU` attributes and assign to `sn_hv_mva`, `sn_mv_mva`, `sn_lv_mva`, `vn_hv_kv`, `vn_mv_kv`, and `vn_lv_kv` according to Table 4. Table 5 shows the differences in mapping the attributes that originate from the `PowerTransformerEnd` objects.

For transformers, similar to the lines, standard types are provided by Pandapower, and if nothing matches our own, then we need to add them as custom types by calling `create_std_types` as seen before. For the `trafo3w` it can be done like this:

```
for i in range (len(df_trafo3w)):
    a1= df_trafo3w.loc[i,"sn_hv_mva"]
    a2= df_trafo3w.loc[i,"sn_mv_mva"]
    a3= df_trafo3w.loc[i,"sn_lv_mva"]
    b1= df_trafo3w.loc[i,'vn_hv_kv']
    b2= df_trafo3w.loc[i,'vn_mv_kv']
    b3= df_trafo3w.loc[i,'vn_lv_kv']
    df_trafo3w.loc[i,'std_type'] = ( ' ').join(pp.find_std_type_by_parameter(net,
    data={"sn_hv_mva": a1, "sn_mv_mva": a2, "sn_lv_mva": a3,
    "vn_hv_kv": b1, "vn_mv_kv": b2, "vn_lv_kv": b3},
    element = 'trafo3w'))
```

Table 5 Difference in mapping attributes of a three-winding trafo to the mapping of a two-winding one

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
PowerTransformerEnd	PowerTransformerEnd.phaseAngleClock	shift_mv_degree	Trafo3w
		shift_lv_degree	
	PowerTransformerEnd.ratedS	sn_hv_mva	
		sn_mv_mva	
		sn_lv_mva	
	PowerTransformerEnd.ratedU	vn_hv_kv	
		vn_mv_kv	
		vn_lv_kv	

External grid

An `ExternalNetworkInjection` is an entity that represents an external connection point to a power network. It is typically used to represent power system interconnections between different utility companies, or between a utility and an independent power producer. It can also represent connections to other types of networks, such as gas or water networks, that are related to power system operation. `ExternalNetworkInjection` objects can have attributes such as voltage, power, and phase angle, which describe the characteristics of the connection point.

The Pandapower equivalent class is the `External Grid (ext_grid)`. Mapping the various attributes is shown in Table 6.

In this section, we demonstrated some of the mappings that were used to convert into a Pandapower network. These mappings covered various types of network elements such as buses, lines, transformers, and external network injections. While not a comprehensive list, the presented examples provided insight into how we translated the CGMES into a Pandapower network using a combination of Cypher queries and python functions. At this point, we had successfully created a skeleton of the network topology and mapped some of its components to their Pandapower counterparts. The rest of the classes are converted in similar fashion, keeping a backlink for debugging.

The next step would be to attach the resulting dataframes to the created empty Pandapower net, avoiding thereby the overhead of adding the single elements one by one using the API. After that, and to be sure that the resulting network is coherent and logically consistent, we run the diagnostic.

Table 6 Mapping of the CGMES model attributes of the `ExternalNetworkInjection` to the attributes of a Pandapower External Grid element

CGMES model		Pandapower model	
Classes	Properties	Parameters	Element
External-Network-Injection	id	id	External Grid
	IdentifiedObject.name	name	
	ExternalNetworkInjection.p	p_mw	
	ExternalNetworkInjection.minQ	min_q_mvar	
	ExternalNetworkInjection.maxQ	max_q_mvar	
	ExternalNetworkInjection.minP	min_p_mw	
	ExternalNetworkInjection.maxP	max_p_mw	
	ExternalNetworkInjection.maxInitialSymShCCurrent	s_sc_max_mva	
	ExternalNetworkInjection.minInitialSymShCCurrent	s_sc_min_mva	
	ExternalNetworkInjection.minR1ToX1Ratio	rx_min	
	ExternalNetworkInjection.maxR1ToX1Ratio	rx_max	
	ExternalNetworkInjection.maxZ0ToZ1Ratio	x0x_max	
	ExternalNetworkInjection.maxR0ToX0Ratio	r0x0_max	
SvStatus	SvStatus.inService	in_service	

```

# assign the dataframes directly
net.bus = bus_df
net.gen = gen_df
net.trafo3w = trafo3w_df
net.motor = motor_df
...
pp.diagnostic(net)

```

If the diagnostic runs without errors, then we can attempt running a power flow on the network:

```
pp.runpp(net)
```

Results are saved into the corresponding dataframes `res_bus`, `res_gen`,... Writing these results back to the CGMES model would be the next step, that we did not include in this work. It would be useful for the network operator to have these results in CGMES format, and for further analysis in Neo4j. We leave this for future work.

Implementation in a real life scenario

Applying our method to a real-life scenario posed a few challenges that we had to address, mainly due to the larger size and complexity of the networks. The company had many network operators as clients and received several such networks in CGMES format every 15 min. Each network must be fully processed, including all the steps mentioned above, within a window of 10 min to keep some buffer for pre- and post-processing operations. Failing to process within the window would have several implications: the network analysis would be delayed. The company would not have the latest and most up-to-date information about the network's state and performance. This delay could hinder their and their clients' ability to make informed decisions or take timely actions based on the analysis results. Since the company receives networks every 15 min, missing the processing window would lead to the accumulation of unprocessed data. This accumulation would continue with each missed window, creating a backlog of networks waiting to be processed. To meet the processing deadline, there might be a temptation to skip or reduce the intensity of certain analysis steps. This could compromise the quality and reliability of the results obtained, and would disrupt the overall workflow of the company and its client network operators.

For these large networks, and before we start building the Pandapower network, we perform syntactic and semantic checks. Having uploaded all the files successfully means that we can be certain, that the syntax of the CGMES dataset is error free. We upload then the CIM ontology into the same Neo4j database, and connect the classes with their instances (objects and relationships). The ontology contains some constraints on the objects and relationships that must be held in order for the dataset to be a valid one. A reasoner is then used to assert these constraints, and notify us of any inconsistencies.

The feedback loop in our method involves providing the syntax and semantic analysis results to the client. The purpose of this is to allow the client to incorporate the results into their process for creating a CGMES dataset from their internal format. Typically, the conversion process is outsourced and correcting any errors can be time-consuming. Therefore, we collaborate with the client to find workarounds until they can rectify their internal processes.

After building an operational Pandapower network, further processing is required, such as performing “n-1 contingency” and “optimal power flow” analysis. These analyses in addition to the original power flow aim to detect potential bottlenecks in the network based on predictions of renewable energy production and consumption of the various consumers. This becomes particularly important in larger and more complex networks. The found bottlenecks are then communicated with the client, and further work is done on both sides for avoiding them, if possible. The whole workflow starting by receiving the CGMES files and ending with the results of the n-1 contingency and optimal power-flow had successfully been run within the window of 10 min. However, instead of a full n-1 contingency, a partial one involving only the top k elements had been run.

All is done within the frame of Redispatch 2.0 so that the client has enough time to take measures for avoiding a future bottleneck before having to resort to curtailment of some wind turbines, thus reducing the downtime of these and saving on resources. Comping up with suggestions for such measures for mitigating predicted bottlenecks had to be also limited to stay within the 10-minutes deadline by reducing the degrees of freedom and the controllable components involved.

Conclusion and future work

In this work, we have demonstrated the feasibility of using Pandapower and Neo4j for representing and analyzing electrical power systems. We have shown how the CIM ontology can be used to map the CGMES dataset to Pandapower and how this can be used to perform various analyses on the power system.

Our approach has several advantages, including the ability to handle large datasets, the ability to perform various types of analyses, and the ability to provide feedback to the client on the quality of their CGMES dataset. Additionally, by representing the power system in a graph database, we can perform graph-based analysis on the network topology, which can be useful for predicting and preventing potential issues. This comes bundled with some overhead, of course, contrary to using other workflows such as importing CGMES directly into PowSyBl (2021) and performing further analysis from there.

However, using our approach opens the possibilities to work on the uploaded PIM. For example, PSMs can be easily generated from the PIM covering other aspects of the system, such as a graphical representation on various abstraction levels, or targeting an entirely different platform such as Modelica (Elsheikh and Palensky 2021), Matlab The Mathworks (2021), or even PowSyBl (2021) itself.

While our approach is promising and has proven to be working in commercial applications, there are still some challenges that need to be addressed. For example, further work needs to be done on developing more sophisticated algorithms for identifying and mitigating bottlenecks in the power system. Additionally, there is a need for more work

on integrating real-time data into the system to allow for more accurate predictions of power flows.

Abbreviations

EQ	Equipment
SSH	Substation Hierarchy
TP	Topology
SV	State Variables
CEMIS	Corporate Environmental Management Information System
CGMES	Common Grid Model Exchange Specification
CIM	Common Information Model
CoIM	Computation-Independent Model
EMS	Energy Management System
KG	Knowledge Graph
GML	Graph Machine Learning
MDA	Model-Driven Architecture
OWL	Web Ontology Language
PIM	Platform-Independent Model
PSM	Platform-Specific Model
RDF	Resource Description Framework
SCADA	Supervisory Control And Data Acquisition
SGKG	Smart Grid Knowledge Graph
UML	Unified Modelling Language
XML	EXtensible Markup Language

About this supplement

This article has been published as part of Energy Informatics Volume 6 Supplement 1, 2023: Proceedings of the 12th DACH+ Conference on Energy Informatics 2023. The full contents of the supplement are available online at <https://energyinformatics.springeropen.com/articles/supplements/volume-6-supplement-1>.

Availability of data and materials

Sample network data and a full processing pipeline are available upon request.

Declarations

Conflict of interest

The authors declare that they have no competing interests.

Published: 19 October 2023

References

- APOC User Guide 4.3—APOC Extended Documentation. <https://neo4j.com/apoc/4.3/>. Accessed 25 Apr 2023
- Dai R, Liu G, Wang Zhiwei, Wang Z, Wang Z, Kan B, Yuan C (2020) A novel graph-based energy management system. IEEE Trans Smart Grid 11(3):1845–1853. <https://doi.org/10.1109/tsg.2019.2943815>
- Dalčević I, Erdeljan A, Dalčević N, Marjanović J (2021) Graph modeling for efficient retrieval of power network model change history. Energies 14(24):8351. <https://doi.org/10.3390/en14248351>
- Di Donato GW, Damiani A, Parravicini A, Bionda E, Soldan F, Tornelli C, Santambrogio MD (2021) Towards graph machine learning for smart grid knowledge graphs in industrial scenarios. <https://doi.org/10.1109/rtsi50628.2021.9597339>
- Elsheikh AMG, Palensky P (2021) Modelica by Application: Power Systems. Mathmodica.com. original-date: 2021-04-01T11:47:52Z. <https://github.com/Mathmodica/ModelicaPowerSystemBook>. Accessed 01 May 2023
- Kan B, Zhu W, Liu G, Guangyi Liu, Liu G, Chen X, Xi Chen, Chen X, Di Shi, Shi D, Shi D, Yu W (2017) Topology modeling and analysis of a power grid network using a graph database. Int J Comput Intell Syst 10(1):1355–1363. <https://doi.org/10.2991/ijcis.10.1.96>
- Kor Y, Tan L, Reformat M, Musilek P (2020) GridKG: knowledge graph representation of distribution grid data. <https://doi.org/10.1109/epec48502.2020.9320066>
- Memari A, Vornberger J, Gómez JM, Nebel W (2016) A data center simulation framework based on an ontological foundation. international conference on informatics for environmental protection, 39–57. https://doi.org/10.1007/978-3-319-23455-7_3
- Neo4j Graph Data Platform—the leader in graph databases. <https://neo4j.com/>. Accessed 02 Apr 2023
- Neosemantics (n10s) Neo4j RDF & Semantics toolkit—Neo4j Labs. <https://neo4j.com/labs/neosemantics/>. Accessed 2 Apr 2023
- Perçuku A, Minkovska D, Stoyanova L (2017) Modeling and processing big data of power transmission grid substation using Neo4j. Proc Comput Sci 113:9–16. <https://doi.org/10.1016/j.procs.2017.08.276>
- Power System Blocks. <https://www.powsybl.org/pages/documentation/grid/model/>. Accessed 1 Apr 2021

- Reyes AK, Reyes Angie K, Reyes Angie K, Hernandez AI, Hernandez Andres I, Gutiérrez RM, Gutierrez RM, Gutierrez R, Gutierrez Rafael M, Bolívar N, Jimenez DA, Jimenez Diego A, Bastidas JD, Solano J (2019) A novel extended graph strategy to model microgrids, 8849117. <https://doi.org/10.1109/sest.2019.8849117>
- The Mathworks, Inc (2021) MATLAB Version 9.10.0.1613233 (R2021a). Natick, Massachusetts. The Mathworks, Inc
- The Model Driven Architecture (MDA) (2006) In: Gašević D, Djurić D, Devedžić V (eds) Model driven architecture and ontology development, pp. 109–126. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-32182-9_4. Accessed 2 Apr 2023
- Thurner L, Scheidler A, Schäfer F, Menke J-H, Dollichon J, Meier F, Meinecke S, Braun M (2018) Pandapower—an open-source python tool for convenient modeling, analysis, and optimization of electric power systems. *IEEE Trans Power Syst* 33(6):6510–6521. <https://doi.org/10.1109/TPWRS.2018.2829021>
- Uslar M, Specht M, Rohjans S, Trefke J, Vasquez Gonzalez JM (2012) The common information model CIM: IEC 61968/61970 and 62325—a practical introduction to the CIM. *Power Systems*, vol. 66. Springer, Berlin, Heidelberg. <https://doi.org/10.1007/978-3-642-25215-0>. Accessed 2 Apr 2023

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen[®] journal and benefit from:

- Convenient online submission
- Rigorous peer review
- Open access: articles freely available online
- High visibility within the field
- Retaining the copyright to your article

Submit your next manuscript at ► [springeropen.com](https://www.springeropen.com)
