

The sole responsibility for the content published on this document lies with the authors. It does not necessarily reflect the opinion of the Innovation and Networks Executive Agency (INEA) or the European Commission (EC). INEA or the EC are not responsible for any use that may be made of the information contained therein.

## WP4

### Predictive Management Tools for Transmission and Distribution Systems Operation

# Tool for State Estimation of Distribution Networks

D4.4



This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 864298.

**DOCUMENT CONTROL PAGE**

DOCUMENT	D4.4 – Tool for State Estimation of Distribution Networks
TYPE	Report
DISTRIBUTION LEVEL	Public
DUE DELIVERY DATE	30 / 08 / 2022
DATE OF DELIVERY	18 / 10 / 2022
VERSION	V0.5
DELIVERABLE RESPONSIBLE	KONČAR – DIGITAL
AUTHOR(S)	Hrvoje Keko, Donata Borić, Zlatan Sičanica, Jakov Krstulović Opara, Leila Luttenberger Marić
OFFICIAL REVIEWER(S)	Florin Capitanescu, Filipe Joel Soares

**DOCUMENT HISTORY**

VERSION	AUTHORS	DATE	CHANGES
0.1	Hrvoje Keko KONČAR-DIGITAL	10 / 06 / 2022	Initial ToC version
0.2	Hrvoje Keko KONČAR-DIGITAL	30 / 07 / 2022	Description of the context, functional specification
0.3	Donata Borić, Leila Luttenberger Marić, Zlatan Sičanica, Hrvoje Keko KONČAR-DIGITAL	30 / 08 / 2022	Full specification of CIM integration, topological processing and load estimation modules
0.4	Hrvoje Keko, Jakov Krstulović Opara, Zlatan Sičanica KONČAR-DIGITAL	16 / 09 / 2022	Integral version of the document
0.5	Hrvoje Keko, Zlatan Sičanica KONČAR-DIGITAL	03 / 10 / 2022	Corrected full version of the document
0.6	Florin Capitanescu LIST	13 / 10 / 2022	Review, structural and contextual improvements
0.7	Hrvoje Keko KONČAR-DIGITAL	16 / 10 / 2022	Final corrections and accepted reviewer suggestions
1.0	Filipe Joel Soares INESC TEC	16 / 10 / 2022	Submission of the deliverable

## Table of Contents

1. CONTEXT, CONCEPTUAL AND FUNCTIONAL DESCRIPTION .....	5
1.1. Context: State estimation in distribution networks.....	5
1.2. Topology Processing: Conversion of Node-breaker to Bus-branch model .....	6
1.2.1. Physical Equipment Modelling and Node-breaker Model.....	6
1.2.2. IEC CIM connector.....	6
1.2.3. Topological Processing and Creation of the Bus-branch Network Model.....	8
1.3. Important assumptions on topology processing context .....	13
1.4. Topology processor implementation.....	14
1.5. Load calibration implementation .....	18
1.6. Other implementation details and future work.....	20
2. REFERENCES.....	23

## List of Figures

Figure 1 – Illustration of the State Estimation problem .....	5
Figure 2 – Different business contexts of a typical DSO .....	7
Figure 3 – Data exchange component between the ATTEST toolbox and other systems .....	13
Figure 4 – Tool wrapper as an environment for the tool to run.....	13
Figure 5 – Data exchange components breakdown .....	14
Figure 6 – Exploratory analysis of the dataset for Koprivnica HEP ODS network .....	19
Figure 7 - Overview of all implemented state estimator modules .....	20
Figure 8 – An example of state estimator results visualization .....	21

## Abbreviations and Acronyms

ACRONYM / ABBREVIATION	Extensive form
AMR	Automated Meter Readout
CGMES	Common Grid Model Exchange System
CIM	Common Information Model
DMS	Distribution Management System
DSO	Distribution System Operator
ENTSO-E	European Network of Transmission System Operators for Electricity
ERP	Enterprise Resource Planning
GIS	Geographic Information Systems
ID	Identifier
IEC	International Electrotechnical Commission
JSON	JavaScript Object Notation
LV	Low voltage
MV	Medium voltage
SCADA	Supervisory Control and Data Acquisition
TSO	Transmission System Operator

---

## Summary

The deliverable titled “Tool for State Estimation of Distribution Networks” is the only deliverable of the task T4.3 of the project. This task aims to develop a state estimator for the MV grids of the future, which will feed the real-time OPF developed in T4.2. Unlike the transmission system, the crucial challenge of state estimation in MV grids is the scarcity of real-time monitoring across the network, so a key task is to generate the coherent set of pseudo-measurements. This deliverable first provides a wider context, and then focuses on the actual implementation within that context.

As several of the tools in the ATTEST project require the bus-branch network model in a format compatible with the MATPOWER tool, state estimation is split into two parts – topology processing from the node-breaker model typically used in SCADA systems and the load calibration.

The architecture of the tool allows deploying it in a tightly knit fashion with the MV SCADA systems in the DSO data centre. Furthermore, the architecture of the tool allows tailoring the pseudo-measurement estimation tool to the operational context. Where more fine-grained data is available, a more sophisticated and more precise model can be used, while in cases of very low data availability, robustness is the desired characteristic.

# 1. Context, conceptual and functional description

## 1.1. Context: State estimation in distribution networks

The classic state estimation problem in transmission networks is well-established and well-researched problem, for which there is significant body of works [1]. The problem of state estimation consists in computing the electric system state i.e. a set of coherent voltage magnitudes and angles. This problem can in turn be broken down into subproblems of network topology estimation, observability analyses, gross errors processing and elimination and parametric or structural errors processing.

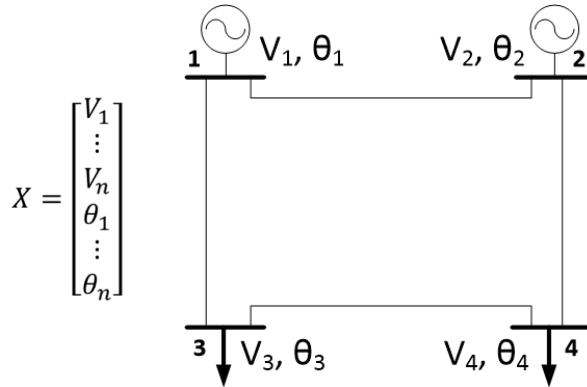


FIGURE 1 – ILLUSTRATION OF THE STATE ESTIMATION PROBLEM

In the figure 1 above, the problem of state estimation is illustrated. The system state is the vector  $X$  which consists of voltage magnitudes  $V_i$  and voltage angles  $\theta_i$ . A set of power flow equations ties the values at the nodes.

The task of state estimation is finding the probability the state vector explains the data. With a certain error metric providing a measure of distance, the state estimation becomes a problem of constrained optimization. Classic techniques use the least squares method, Gauss-Newton methods and in more recent cases the methods based on information theoretic learning fundamentals [2].

However, for the medium voltage and for the low voltage distribution networks, the state estimation problem is quite different. In case of transmission networks, the nodes are typically all covered by measurements at different time scales and in most cases also with redundant information.

In distribution networks, the data set is much scarcer. Essentially, the problem is that the real time monitored values do not cover the MV network.

There are measurements in a partial set of MV substations, some of which are not being read out in real time. Network parameters such as line lengths and impedances are relatively static and can be collected from the business systems in the DSO. Switch statuses and the part of the data that is read from the network in real-time can be collected from the medium voltage SCADA system. An additional issue with such real-time metered data is that this type of metering at substation level is used in system management and operation and not in billing, so often this value is designed as only an indicative to the dispatcher in the control room, while the calibrated meters providing values to be used in billing systems are normally read out with automated meter readout (AMR) systems.

To complete the state vector in a distribution network, an estimation of correct loads in the network must be done so the MV network state estimation consists of two principal parts:

- ▷ Estimation of topology and network parameters
- ▷ Estimation of network loads

The state estimation tool is designed with these two main building blocks in mind, explained in the following chapters.

## 1.2. Topology Processing: Conversion of Node-breaker to Bus-branch model

---

### 1.2.1. Physical Equipment Modelling and Node-breaker Model

The need to explicitly model physical equipment and devices into the SCADA has resulted in them relying on a so-called node/breaker network model.

Namely, the SCADA systems typically operate on a level of commands and signals – and a basic SCADA concept does not have a notion of any entity aggregating these commands, nor is the data semantics modelled directly. In other words: a signal measuring temperature of transformer windings and a signal measuring current or voltage do not differ substantially. Both are numeric signals, and the basic concept of a SCADA system does not aggregate them semantically.

The principal benefit of a node-breaker model is that the signals directly correspond to the physical components of the industrial automation system being controlled.

In this context, most electrical companies in the world were using their own internal ways of describing network elements so it was important to come up with standardized model, so-called CIM (Common Information Model), which everyone can use to easily exchange information. The most successful use case of IEC CIM is the ENTSO-E level Common Grid Exchange Model (CGMES), utilized among the TSOs to exchange grid models in a compatible way [3]. In late 2021, the European level model has been successfully and automatically assembled from partial CGMES-based network models of European countries. The IEC CIM is described in the IEC 61968 [4] and IEC 61970 [5] series of standards. It is designed as an object oriented standard and is in practice an ontological standard defining the classes that representing node/breaker model in detail.

### 1.2.2. IEC CIM connector

Admittedly, at first, discussing CIM might sound unrelated to the problem of state estimation. This is not the case: the IEC CIM has become a de facto standard for information exchange in all levels of business operations in a typical modern DSO. The lack of input data is the most significant challenge for the subtask of load estimation within medium voltage state estimation, being compliant with this standard significantly lowers the barrier to including external data.

For example, modern SCADA systems are capable of building and exporting their internal network model in the format compatible with CIM. Similarly, tools often used in the DSO day-to-day operation often use CIM-compatible input data to interact with other systems. The reality of the business operations in a typical DSO is represented in the following figure.

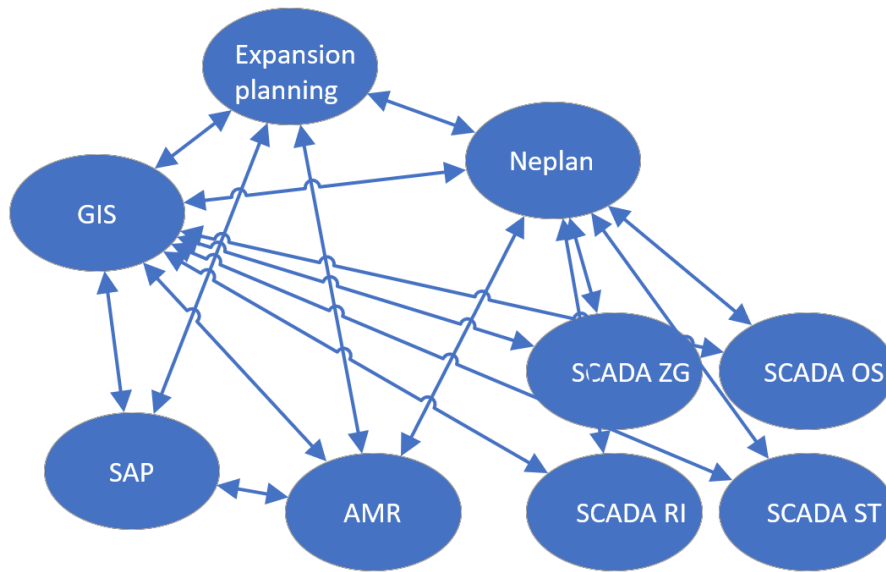


FIGURE 2 – DIFFERENT BUSINESS CONTEXTS OF A TYPICAL DSO

The business context is that in each business need of the DSO, e.g. automated meter readout, ERP or the SCADA system, a separate islanded software system has been started up to take care of that business needs. These systems often end up as partially integrated subsystems, and we run into the problem of the object not holding the same identifier in all the subsystems. Effectively, the same object has a part of its attributes defined in one system, and the rest of the attributes in other systems. For example, a simple electricity meter may have one ID in the ERP system, and a completely another one in the AMR system, with perhaps a third one in the asset management system. Effectively, there are multiple loosely coupled sources of truth for the same object. This is not surprising as the data required for expansion planning context is quite different from the SCADA operational context.

Yet, the system and the DSO is the same, and this discrepancy often introduces friction into the DSO business context. To address this, KONČAR has developed an integration solution, compliant with the IEC CIM that allows respecting the semantic description of data according to the IEC CIM standard. Additionally, this solution allows bitemporal versioning supporting time of awareness and time of validity timestamps for each of the entities and allows multi-branch tracking like source code control systems. Multi-branching allows each of the subsystems to “look” at the same system and seeing different and sometimes differing subsets of the information.

A derivative of the above solution is included in the deployment of ATTEST in the Croatian pilot and ATTEST tools benefit from the indirect compatibility with CIM and easier integration into the DSO systems. This is more relevant for the context of integration, so it is described in more detail in WP6 deliverables.

The important takeaway for the state estimation context is that the input network model will arrive in the node-breaker form, appropriate for tracking the status of network elements, but unfortunately not appropriate for the calculations and optimizations of the network as virtually none of the ATTEST tools uses this type of network model.



### 1.2.3. Topological Processing and Creation of the Bus-branch Network Model

Virtually all optimizations and calculations in the transmission network require a correct network topology. The network topology can be determined from statuses of switches (breakers and disconnectors) and this creates an abstract model known as bus/branch network model.

A switch is not an element which is a part of calculations such as power flows – a closed switch is effectively invisible to such calculations, and an open switch is a break in the network. A closed switch between the two physical nodes effectively aggregates these two nodes into the same abstract bus. An open switch means the two physical nodes must correspondingly be represented as two buses in the bus-branch model with no branch between them.

The bus-branch model eliminates switches based on their open/closed status and groups the so-called connectivity nodes, connected with closed switches, into a single topological node (bus). The other elements of electrical network are modelled as topological branches and this process of conversion is called network topology processing.

The collected statuses of switches may not be true to the current situation so it is important to keep the representation of each element from the node/breaker model so the dynamic changes can be applied back to the original node-breaker model. In other words, the conversion should be possible from the node-breaker model to bus-branch model and vice versa.

Additionally, if the original representation of the node-breaker model is kept track of, then it is not necessary to rebuild the bus-branch model from scratch on each switching operation. Only the parts of the model affected by a switching operation compared to the previous instance must be altered.

The actual physical elements in the CIM network model used in topological processing are: Switch, BusBarSegment, ACLineSegment, EquivalentInjection and EnergyConsumer. The other models in CIM model are: ConnectivityNode and Terminal.

The classes in CIM have multiple attributes, but for the first step in processing only the attribute for mRID of element is needed. The mRID is a unique identifier of an object in CIM, immutable over the lifetime of the object. Put differently: knowing the mRID is enough to uniquely identify and find the original element in the bus-breaker model. For this reason, the topological processor relies on the mRID as unique identifiers of the corresponding equipment.

- ▷ **Switch:** has 2 associated terminals and subclasses Breaker and Disconnector
- ▷ **BusBarSegment:** has 1 associated terminal.
- ▷ **ACLineSegment:** has 2 associated terminals and parameters for resistance (r), inductance (x), shunt susceptance (bch), shunt conductance (gch) and conductor length; all physical values that are inputs to the impedance calculation
- ▷ **EquivalentInjection:** has 1 associated terminal.
- ▷ **EnergyConsumer:** has 1 associated terminal.
- ▷ **ConnectivityNode:** has one or more associated Terminals, it aggregates these Terminals
- ▷ **Terminal:** has a reference for belonging ConnectivityNode (the ConnectivityNode it belongs to) and ConductingEquipment, plus ordinal number of terminal for belonging physical element – basically specifies the incidence with the above.

With the context provided above, the topological processing of the node-breaker model into the bus-branch model is defined as follows:

A switch is in **closed** state if its status is equal to 1, otherwise it is open.

If there is a **closed** switch between two connectivity nodes  $cn1$ ,  $cn2$  and, e.g.,  $cn1 < cn2$  (where “<” is a comparison or ordering operator to determine the higher ranked element), then  $cn1$  **becomes**  $cn2$ . This process is called **merging** of the nodes (into the lexicographically higher one, or whichever is preferred and implemented via a comparison operator).

The comparison operator is used so the aggregation is predictable and the order is explicitly specified, e.g. so that it is clear the node A1 and node A2 get always get aggregated into the bus A1.

The required input variables for topological processing:

- ▷ **node set**: sorted mRIDs of all connectivity nodes,
- ▷ **asset map**: mRID of the element is key to identify the actual instance of that element and
- ▷ **switch map**: mRID of the switch is the key for the (externally retrieved) status of that switch.

These are the additional mappings:

- ▷ **terminal map**: the mRID of physical element is a key for the list of mRIDs of associated terminals. The initialization of this map is based on ConductingEquipment element referenced from the Terminal. From the asset map, all Terminal elements with the same mRID of reference for ConductingEquipment as key-mRID must be found. The mRIDs of these Terminal elements are put in a list. This lists the elements incidental to this Terminal.
- ▷ **connectivity map**: the mRID of connectivity node is a key for the list of mRIDs of the associated terminals. The initialization of this map is based on the ConnectivityNode element, referenced from the Terminal. From the asset map all Terminal elements which have same mRID of reference for ConnectivityNode as key-mRID need to be found. mRIDs of these Terminal elements are put in a list.
- ▷ **merge map**: the mRID of the connectivity node is a key for the mRID of connectivity node **into which it is merged**. The initialization of map is  $map[key]=key$ , i.e. at the start of topological processing the connectivity node only maps or merges into itself.

An additional simplification step is done prior to the actual topological processing: the ACLineSegments are merged (aggregated) into an ACLine and therefore the nodes between the line segments are dropped from the topological processing. These can't be aggregated as there is no switch and in the branch model the line segments combined together actually represent a branch.

**For every element in asset map** (iterate through keys, assuming they are sorted):

**If** it is ACLineSegment:

Initialize **ACLine**, mRID is set to mRID of current ACLineSegment

From **terminal map**, the first associated terminal of current ACLineSegment is the final first terminal of ACLine, and second terminal is trial second terminal

---

```

In first associated Terminal, change reference for
ConductingEquipment from current ACLineSegment to new ACLine
(ordinal number stays the same) number is set to 1

Attributes r,x,bch,gch and conductor length of ACLine are set to
the same attributes of ACLineSegment

From node set remove ConnectivityNode of second terminal (from
reference of Terminal element find associated ConnectivityNode)

For this element in asset map change its value to ACLine, and go to
next element While next element in asset map is ACLineSegment:

    From terminal map, second associated terminal of current
    ACLineSegment is trial second terminal of ACLine

    Attributes r,x,bch,gch and conductor length of ACLineSegment
    are summed up respectively with attributes of ACLine number is
    increased by 1

    From node set remove ConnectivityNode of both terminals (from
    reference of Terminal element find associated ConnectivityNode)

    From asset map delete key for this ACLineSegment and go to next
    element in map

    From terminal map delete key for this ACLineSegment

# Now we have the last line segment

From terminal map, second associated terminal of current
ACLineSegment is final second terminal of ACLine

Attributes r,x,bch,gch and conductor length of ACLineSegment are
summed up respectively with attributes of ACLine
number is increased by 1

From node set remove ConnectivityNode of first terminal (from
reference of Terminal element find associated ConnectivityNode)

From asset map delete key for this ACLineSegment

From terminal map delete key for this ACLineSegment

Using mRID of ACLine as key in asset map, replace value with
instance of this ACLine (from first segment of line to whole line)

Using mRID of ACLine as key in terminal map, replace value with new
list which consist of first and second terminal of this ACLine

For second associated Terminal of ACLine, change its reference from
ACLineSegment to ACLine (ordinal number stays the same)

# Whole line is now connected

```

With the above definitions and AC line simplification, the topological processing algorithm specification follows. The *cursive* text in brackets specifies the operations above.

```

For every node in node set
  Get the list of terminals from the connectivity map - the key is the
  mRID of current node
  For every terminal in list
    If it has a connected closed switch (from asset map get instance of
    Terminal element and check if its reference for ConductingEquipment
    is switch; if it is, use mRID of that switch as key in switch map
    and check if its status is equal to 1)
      Get the other terminal of that switch (from the terminal map
      get the two terminals and choose the other one)

    If there were closed switches, find max of saved mRIDs
      For current and saved nodes, except one with max mRID, in merge
      map rewrite value with max mRID

      Transfer all terminals from nodes with smaller mRIDs to the
      node with max mRID
      (in connectivity map for saved nodes with smaller mRID move
      values to value of max key-mRID)

    # Because some nodes that represented maximum in one iteration
    # could represent minimum in some of next iterations, it is
    # necessary to iterate through node set in reversible way and make
    # adjustments to merge map. The connectivity map is updated.

  For every node in node set (in reversible way)
    Merge node to associated maximum node
    (merge map[mRID] = merge map[merge map [mRID]])

    # If for a node is valid: merge map[mRID] = mRID, it is called
    # final node. TopologicalNode is a set of connectivity nodes which
    # are merged into same ConnectivityNode.

  For every key-node in merge map
    Put it in the set of associated TopologicalNode
    If a TopologicalNode doesn't exist yet, make a new one
  
```

The final output of the topological processor is a **topological map**, where the keys are the **mRIDs of final nodes** for the associated **TopologicalNodes**.

Finally, the analogous algorithm is also utilized to calculate the incidence and the elements of nodal admittance matrix for the power system: for each node in the topological map, the set of connectivity nodes for these nodes is processed and the admittance is calculated. Essentially, this topological processing model aggregates the original nodes into buses of the modeled power system and calculates the impedances between the nodes.

```

Set dimension N of matrix to zero
For every key in merge map
  If node is final node
    mRID of node is key in final map for value N
    N is increased by one

# Now we know number of buses in the model

Matrix with dimension N is set to zero
  
```

---

```
For every node in merge map:
  If node is final node:
    Go through connectivity list of that node:
      If reference for ConductingEquipment of Terminal is ACLine:
        From terminal map get the other Terminal of ACLine
        (where key is mRID of ACLine)

        From other Terminal take reference for ConnectivityNode

        For found ConnectivityNode, from merge map find
        TopologicalNode to which it belongs (key is mRID of
        found ConnectivityNode)

        Calculate admittance (Using attributes r, x from
        ACLine: admittance =  $r/(r^2+x^2) - j*[x/(r^2+x^2)]$ )

        On place (final map[mRID of first node], final
        map[mRID of second node]) in matrix set value to:
        admittance*(-1)

        On place (final map[mRID of first node],
        final map[mRID of first node]) in matrix add up value
        of admittance

    On place (final map[mRID of first node],
    final map[mRID of first node]) in matrix add up shunt value
    (Using attributes gch, bch from ACLine:
    shunt =  $gch/2 + j * (bch/2)$ )
```

## 1.3. Important assumptions on topology processing context

Normally, the topological processing is seen as an integral part of the state estimation context. Given the context of ATTEST, and as the relevant input data coming into ATTEST tools is essentially in node-breaker format (be it from GIS or from SCADA system), we have decided to split the topological processor into two halves and deliver the initial topological processing as a service for all the other tools.

This way all the ATTEST tools effectively benefit from the topological processing and the conversion of input data based on the node-breaker model into the bus-branch model, and the ATTEST toolbox can be integrated with the rest of the software systems in the TSO or DSO.

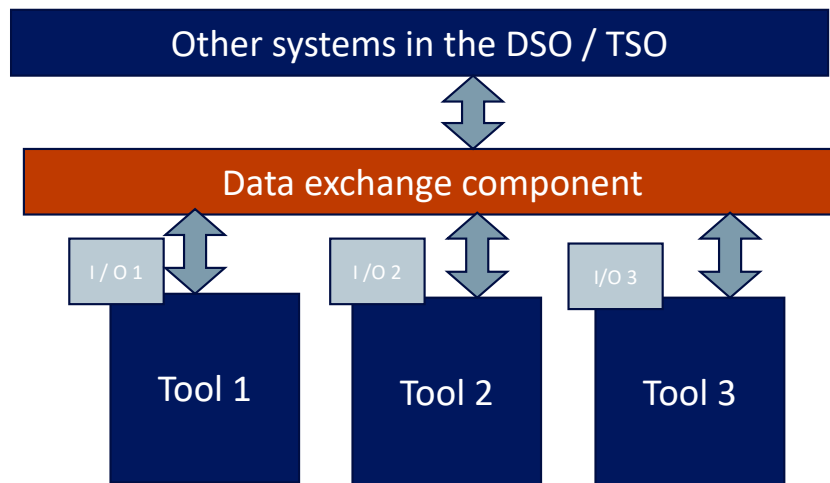


FIGURE 3 – DATA EXCHANGE COMPONENT BETWEEN THE ATTEST TOOLBOX AND OTHER SYSTEMS

The ATTEST toolbox consists of several tools, each with their own requirements, inputs, and outputs. While it is the task of the ATTEST WP6 to provide the integration details, a brief overview of the context is provided here.

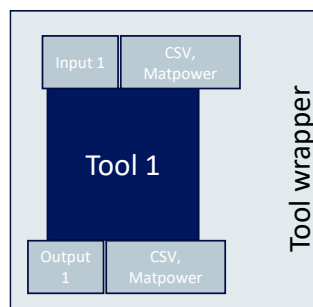


FIGURE 4 – TOOL WRAPPER AS AN ENVIRONMENT FOR THE TOOL TO RUN

In the integration context, each of the tools is wrapped into a wrapper that supplies the inputs and takes care of the outputs delivered by the tool. In essence, each of the tools runs in an environment closely resembling the one where it has been developed. This includes the formats of input and output data, and consequently, also the network models.

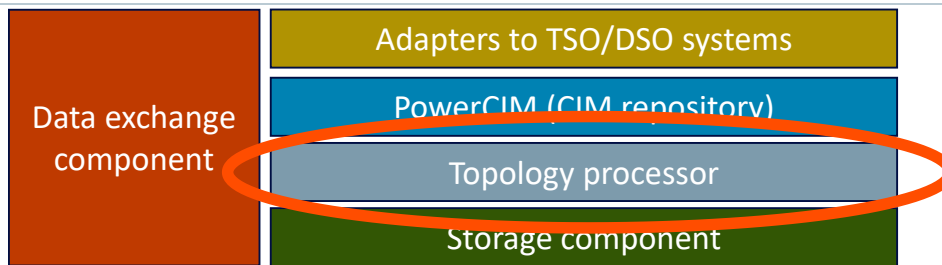


FIGURE 5 – DATA EXCHANGE COMPONENTS BREAKDOWN

From the Figure 3, a wider context of the data exchange component is visible. It is tasked with interfacing with other systems in the TSO and DSO and providing the tool wrappers with appropriate inputs and outputs. The Figure 5 specifies the subcomponents of the data exchange component, one of which must be the topology processor specified in Chapter 1.2.3.

This means that the topological processing is a service offered to virtually all the other tools. Here, a pragmatic assumption is made that the obtained switch statuses are assumed to be generally correct, so the resultant topology from the processing is immediately utilized as part of the input for all the other ATTEST tools through the data exchange component and by placing the derived inputs into the tool wrappers. No difference is made between the estimated switch statuses and the ones directly inferred from the SCADA system. A caveat here is related to the parts of today's MV networks, where some MV/LV substations might not be covered by any type of measurements at all. By 2030, however, we expect at least inferred switch statuses to be available. State estimation could support external higher frequency measured values, that gather their measurements from the downstream meters at the low voltage side. These measurements could generate additional switch statuses inferred from the measurements and thus benefit the substations where no measurement equipment is installed.

An interesting future step of the development would be integrating the estimator in a tighter fashion with the SCADA software, as a part of the DMS functionalities. This way, the topology validity check could be performed iteratively and automatically. The overall architecture of the estimator supports this approach fully.

#### 1.4. Topology processor implementation

Topology processor is implemented as a library that provides functions and classes that allow users to connect indirectly to a CIM record database, implemented in the KONČAR PowerCIM solution.

The PowerCIM solution is KONČAR's product that allows bitemporal versioning of CIM network models, and for purposes of ATTEST project is also utilized as integrator towards CIM-aware systems in the TSO or DSO.

The topology processor library queries the PowerCIM and collects the CIM elements generating the bus-branch model and calculating admittance matrix based on connectivity nodes of the system.

This implementation follows the pseudocodes proposed in 1.2.3, with a minor difference in the way the merge map handling vs. the connectivity map. For the admittance matrix the connectivity map is used primarily to improve the performance as its structure is more condensed. Functionally, there are no difference regarding the final calculated admittance values.

Topological processor is implemented as a Python package. It assumes there is a live PostgreSQL database which holds instance of PowerCIM and there are appropriate access rights. It also assumes that the database contains an internal PowerCIM function:

```
recordat(lastbranchid integer,
         lastcommitid integer,
         lastvalidtime timestamp without time zone,
         cimclassids int4[],
         hmrids uuid[],
         mrids uuid[],
         pmrids uuid[],
         rnamelikes text[])
returns table(mrid uuid,
             mridtext text,
             rname text,
             cimclassid integer,
             cimclass text,
             pmrid uuid,
             fullobject jsonb)
```

As implemented, the topology package contains several modules that match the steps in topology processing. These modules are:

- ▷ **db** – Python module handling database communication, provides a connection object with the methods for connecting and executing common queries
- ▷ **unprocessed** - initializes data structures and different maps needed for the processing algorithm
- ▷ **node\_breaker** - implementation of the first topology processor algorithm segment, which detects connectivity nodes by analyzing the switch positions
- ▷ **node\_branch** - implementation of the second topology processor algorithm segment, detects line admittances and generates an admittance matrix between the connectivity nodes

Additionally, for the purpose of integrating with other subcomponents of the data exchange component, illustrated in Figure 5, there is a server API provided that offers topological processor functions as a web service.

The /api endpoint supports **GET** requests with the branch\_id and commit\_id parameters that are passed to the PowerCIM to select the correct CIM network model.

The result is a JSON representation of the admittance matrix, whose structure respects the following JSON schema:

```
---
type: object
required:
  - branch_id
  - commit_id
  - datetime
  - admittance_sparse_matrix
  - topological_nodes
properties:
  branch_id:
    type: int
    description: CIM repo branch id
  commit_id:
    type: int
```



```

    description: CIM repo commit id
  datetime:
    type: int
    description: UNIX timestamp of the time when the snapshot was taken
  admittance_sparse_matrix:
    type: array
    items:
      type: object
      description: |
        Sparse matrix, represents connection between the
        topological
        node at row and col, with the admittance value. Rows and
        cols are indices of topological nodes under the
opological_nodes
    property.
    required:
      - row
      - col
      - value
    properties:
      row:
        type: int
      col:
        type: int
      value:
        type: array
        maxContains: 2
        description: |
          complex number, first value is the real and second
is
          the imaginary segment
      items:
        type: float
  topological_nodes:
    type: array
    items:
      type: array
      maxContains: 2
      description: |
        pair of topological node UUID (first UUID in the group by
        lexicographic order) and all element UUIDs belonging to
that
        node
    prefixItems:
      - type: string
        description: UUID, topological node id
      - type: array
        description: |
          ids of all elements belonging to the topological node
        items:
          type: string
          description: UUID
...

```

Example request:

```
GET /api/branch_id=4&commit_id=1
```

Response:

```
{
  "branch_id": 4,
  "commit_id": 1,
  "timestamp": 1664181352,
  "admittance_sparse_matrix": [
    {"row": 0, "col": 0, "value": [1.1, 2.2]},
    {"row": 0, "col": 1, "value": [-1.1, -2.2]},
    {"row": 1, "col": 0, "value": [-1.1, -2.2]},
    {"row": 1, "col": 1, "value": [1.1, 2.2]},
    {"row": 3, "col": 3, "value": [5.3, 7.1]},
    {"row": 3, "col": 4, "value": [-5.3, -7.1]},
    {"row": 4, "col": 3, "value": [-5.3, -7.1]},
    {"row": 4, "col": 4, "value": [5.3, 7.1]},
  ],
  "topological_nodes": [
    ["c134850a-3d76-11ed-b16f-201e88d11df2", [
      "c134850a-3d76-11ed-b16f-201e88d11df2",
      "d05c1944-3d76-11ed-b16f-201e88d11df2"]],
    ["f4d6681a-3d76-11ed-b16f-201e88d11df2", [
      "f4d6681a-3d76-11ed-b16f-201e88d11df2"]],
    ["0bf53f62-3d77-11ed-b16f-201e88d11df2", [
      "0bf53f62-3d77-11ed-b16f-201e88d11df2"]],
    ["16094d7c-3d77-11ed-b16f-201e88d11df2", [
      "16094d7c-3d77-11ed-b16f-201e88d11df2"]],
    ["234b67ea-3d77-11ed-b16f-201e88d11df2", [
      "234b67ea-3d77-11ed-b16f-201e88d11df2"]]
  ]
}
```

This could be interpreted as the following admittance matrix:

$1.1 + 2.2j$	$-1.1 - 2.2j$	0	0	0
$-1.1 - 2.2j$	$1.1 + 2.2j$	0	0	0
0	0	0	0	0
0	0	0	$5.3 + 7.1j$	$-5.3 - 7.1j$
0	0	0	$-5.3 - 7.1j$	$5.3 + 7.1j$

Furthermore, adding the topological nodes interpretation we can see that the node `c134850a-3d76-11ed-b16f-201e88d11df2` and `f4d6681a-3d76-11ed-b16f-201e88d11df2` are connected, with admittance of  $1.1 + 2.2j$ , and `16094d7c-3d77-11ed-b16f-201e88d11df2` is connected with `234b67ea-3d77-11ed-b16f-201e88d11df2` with admittance of  $5.3 + 2.2j$ . Lastly, the values on the diagonal the represent shunt values of the individual nodes.

## 1.5. Load calibration implementation

---

The task of load calibration is to take the best possible effort to estimate the loads at the nodes where live measurements do not exist at all. Virtually the only measurements that can be had at these network nodes are the cumulative measurements of all customers downstream, obtained from the billing systems. These are typically not available at all in real time and normally these measurements don't carry the topological information. These values are typically only available as historical data, and even then the granularity of the data may not be the desired one as not all users have smart meters.

While it is considerably realistic to assume that by 2030, the year the ATTEST developments target, most of the end users will be covered with smart meters capable of producing 15-minute readouts, to handle the transition period the load calibrator relies on the typical load curves, where 15 minutes values are not available.

This approach uses the typical load curve inferred from the data, and, essentially, scales the values to predict the value in the network nodes. Implicitly, the load calibrator classifies each of the network nodes, and then estimates the load value based on the predictor input parameters, such as the season or the time of day. The predictor input values can vary significantly. An example, not utilized here, can be the local temperature or the level of deployment of electrical vehicles. It is reasonable to expect the increase of external data that impacts the operation of the distribution network.

For training, the dataset provided by HEP DSO, a set of 15-minute values for load curves for four seasons, split into business day and weekend (holiday) days are available, for the Koprivnica network.

The current version of the load calibrator is a relatively simple but robust estimator that uses the hybrid of a simple lasso regression and a neural network-based estimator. In the training phase the estimation error, compared to the measured values, is in worst cases around 17% of the peak load for a given node. This is, however, the value that only uses the absolute value of load at the load points, with no consideration on, e.g. the number of households at that connection point. A more fine-grained data preparation method would result in a finer training dataset and better estimation performance.

The input dataset for this network is illustrated depicted in the figure below. The load calibration then deduces the current season and the day type from the time when it is invoked, and passes these inferred parameters to the previously trained estimator.

As with any forecasting and estimation task, the load calibration predictor is not generalized. It is highly specific to the network and to the input dataset. Significant changes to the observed network would require retraining and a new input data set.

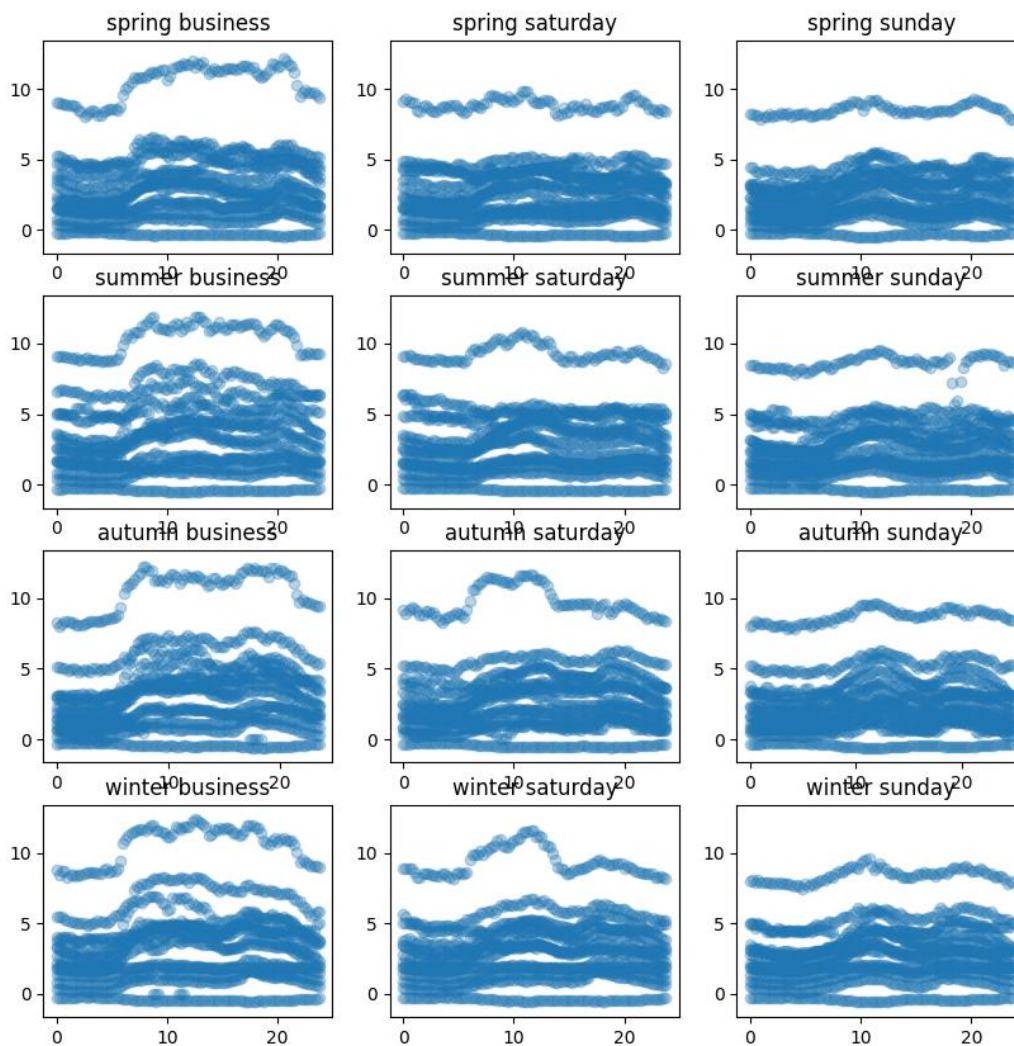


FIGURE 6 – EXPLORATORY ANALYSIS OF THE DATASET FOR KOPRIVNICA HEP ODS NETWORK

The final step of the state estimation procedure takes the estimated values of the loads and performs the load flow, so the set of values is coherent. The power flow result compared with the measured values, especially the ones at feeder level, provides a degree of quality evaluation for the estimated state, i.e. a degree of coherence with the measured values. The criterion used is the maximum deviation from the actual measured values,  $\max |V_{est} - V_{measured}|$ .

## 1.6. Other implementation details and future work

The state estimation tool is, essentially, a network model source for all tools that require the complete model of the medium voltage network. As discussed earlier, this wider context resulted into a split into two components, the topological processor and the load calibration or the pseudo measurement estimation module. The core state estimation code is implemented in Python in a modular fashion, relying on KONČAR open-source event-driven platform, as illustrated in Figure 7 below.

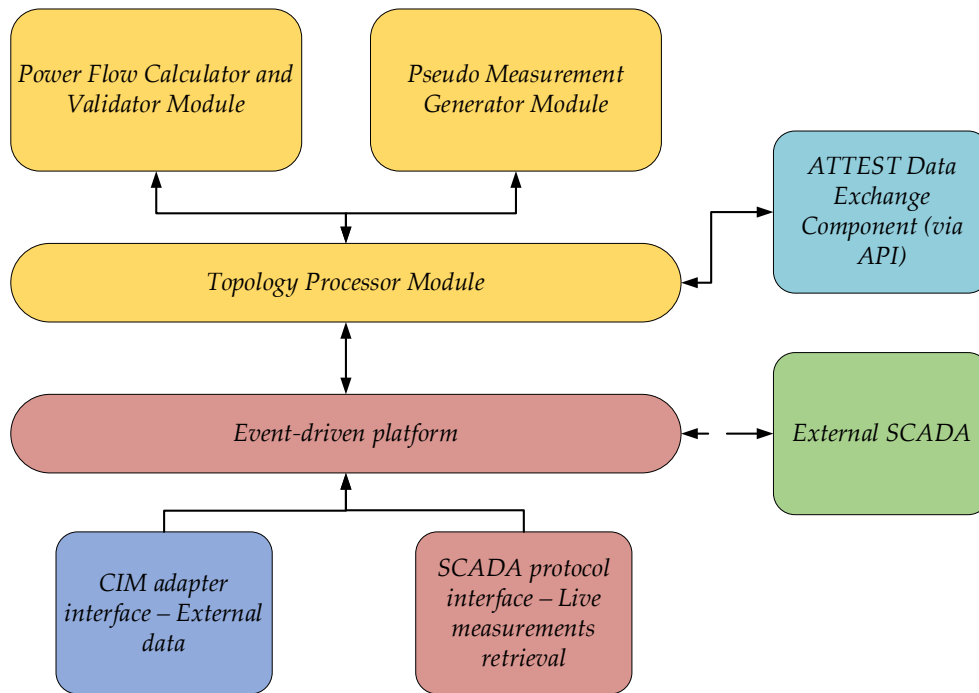


FIGURE 7 - OVERVIEW OF ALL IMPLEMENTED STATE ESTIMATOR MODULES

Besides interfacing through the CIM connector, using the event-driven platform that natively supports SCADA protocols allows the usage of these protocols to couple the estimator more tightly with the MV SCADA system to gather the live input data. Besides being able to interface with live data *inputs*, this also allows eventual integration of the estimation *results*, using the same SCADA protocols. This could allow the external SCADA GUI to visualize the state estimation results within the usual user interface. An example of possible visualization of state estimator results is depicted below.

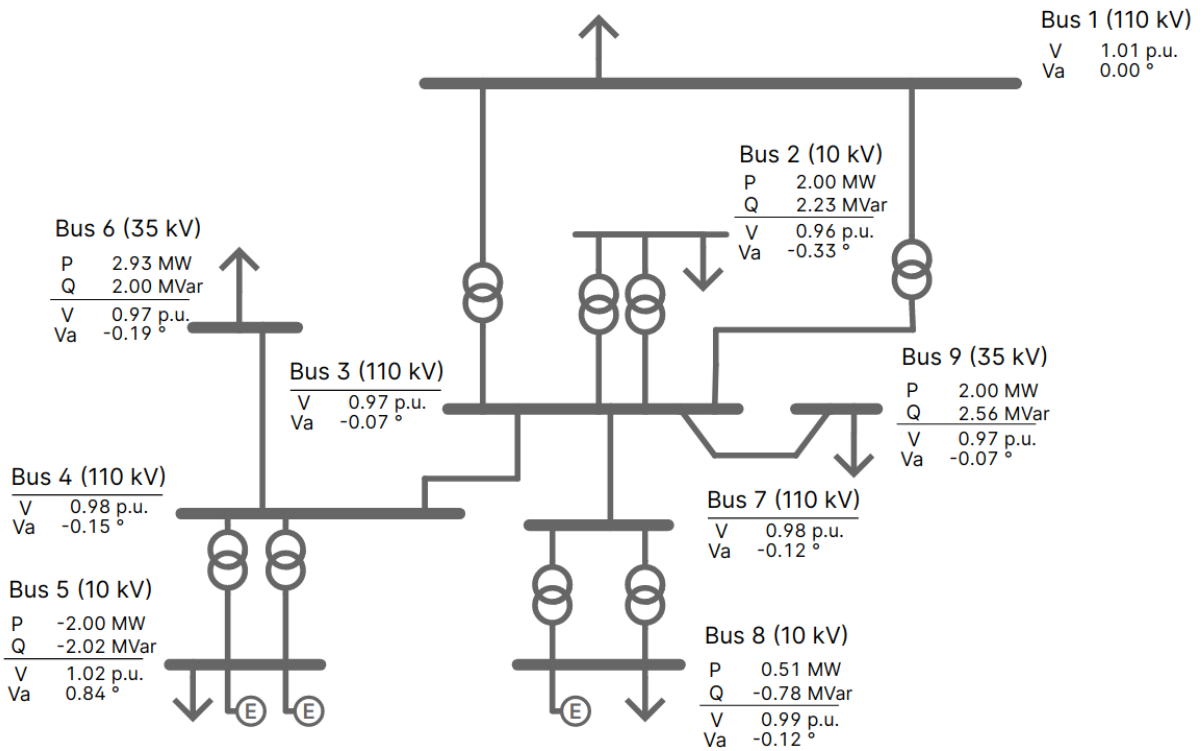


FIGURE 8 – AN EXAMPLE OF STATE ESTIMATOR RESULTS VISUALIZATION

The above visualization has been implemented in the state estimation tool development and uses the widgets from the KONČAR PROZA Station SCADA platform to visualize the simulated measured values together with the estimated ones.

Integrating with the principal SCADA system in the DSO might allow an additional manner of launching the ATTEST tools, as well – these could be used as part of the distribution network management system. This functionality exists but has only been tested and used during the development at the time of writing this document. This is the reason for the dashed line towards the external SCADA system in Figure 7.

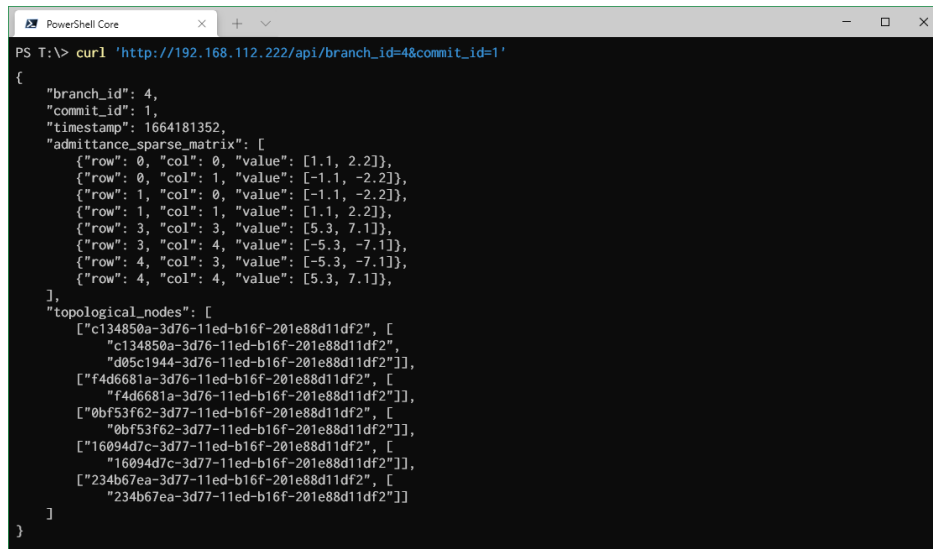
With regards to the calculation modules, the topology processor serves as a gatekeeper towards the ATTEST data exchange component, as described in previous chapters, and towards the comparatively smaller power flow calculator module and pseudo measurement generator module.

All modules of the state estimator are implemented in Python programming language, utilizing several libraries of which the most important are Pandapower used to calculate the power flows in the final state estimation calculation step in the power flow calculator module. The key benefit of the modular organization illustrated above is that the future development in any of the modules generally does not affect the other modules. For instance, replacing or upgrading the pseudo measurement generator module shouldn't affect the topology processor module. Similarly, modifying another power flow calculator does not affect the other modules.

## 1.7. How to run the tools

As discussed in this document, the tool has been split into two parts: topology processor and pseudo measurement generator.

The topology processor is designed to be run as part of the data exchange component, so it is not intended to be run separately. However, it is possible to run it via a tool that allows running the GET requests such as Insomnia, Postman or a simple curl command.



```
PS T:\> curl 'http://192.168.112.222/api/branch_id=4&commit_id=1'
{
  "branch_id": 4,
  "commit_id": 1,
  "timestamp": 1664181352,
  "admittance_sparse_matrix": [
    {"row": 0, "col": 0, "value": [1.1, 2.2]},
    {"row": 0, "col": 1, "value": [-1.1, -2.2]},
    {"row": 1, "col": 0, "value": [-1.1, -2.2]},
    {"row": 1, "col": 1, "value": [1.1, 2.2]},
    {"row": 3, "col": 3, "value": [5.3, 7.1]},
    {"row": 3, "col": 4, "value": [-5.3, -7.1]},
    {"row": 4, "col": 3, "value": [-5.3, -7.1]},
    {"row": 4, "col": 4, "value": [5.3, 7.1]},
  ],
  "topological_nodes": [
    ["c134850a-3d76-11ed-b16f-201e88d11df2", [
      "c134850a-3d76-11ed-b16f-201e88d11df2",
      "d05c1944-3d76-11ed-b16f-201e88d11df2"]],
    ["f4d6681a-3d76-11ed-b16f-201e88d11df2", [
      "f4d6681a-3d76-11ed-b16f-201e88d11df2"]],
    ["0bf53f62-3d77-11ed-b16f-201e88d11df2", [
      "0bf53f62-3d77-11ed-b16f-201e88d11df2"]],
    ["16094d7c-3d77-11ed-b16f-201e88d11df2", [
      "16094d7c-3d77-11ed-b16f-201e88d11df2"]],
    ["234b67ea-3d77-11ed-b16f-201e88d11df2", [
      "234b67ea-3d77-11ed-b16f-201e88d11df2"]]]
  ]
}
```

FIGURE 9 - TOPOLOGICAL PROCESSOR OUTPUT

The IP address in the request should point to the server where the topology processor is installed. The parameters in the request are explained in the chapter 1.4 above and represent the selected CIM network commit in the linked PowerCIM database. The user interface will allow user to select the network while the identifiers will be passed in the backend. Similarly, the interpretation of the API output is supposed to be handled by the subcomponent of data exchange component that creates the MATPOWER file from it as most of the ATTEST tools require this format.

The pseudo load estimator is also not designed to be run as a separate module, however it can be run from the command-line as well. It is a python module and has two principal modes of work: training and running the estimator. In both modes the first parameter is the MATPOWER format of the network. For training mode, a second parameter points to a CSV file with additional historical data for training, by default historical load curves for the nodes that do not have direct measurements in SCADA. The training mode runs the estimator. The training mode output is a pickle file where the configuration of the estimator is saved.

In the estimation mode, the second parameter is that pickle file holding the pseudo load estimator internal configuration, and the third parameter is the desired estimation timestamp, from which the day of the week and the hour are inferred, and the estimation run. The results are returned in Pandapower compatible JSON format. When it is run within the SCADA platform including the SCADA platform HTML5 based user interface, then the screenshot of the main outputs of the state estimator is depicted in the Figure 8.

## 2. References

- [1] A. Abur and A. G. Expósito, *Power System State Estimation: Theory and Implementation*, 1st edition. New York, NY: CRC Press, 2004.
- [2] V. Miranda, J. Krstulovic, H. Keko, C. Moreira, and J. Pereira, “Reconstructing Missing Data in State Estimation With Autoencoders,” *IEEE Trans. Power Syst.*, vol. 27, no. 2, pp. 604–611, 2012, doi: 10.1109/TPWRS.2011.2174810.
- [3] “IEC 61970-CGMES:2018 | IEC Webstore | automation, cyber security, smart city, smart energy, smart grid, CGMES.” <https://webstore.iec.ch/publication/61124> (accessed Jun. 01, 2018).
- [4] “IEC 61968-11:2013 | IEC Webstore.” <https://webstore.iec.ch/publication/6199> (accessed Jun. 01, 2018).
- [5] “IEC 61970-1:2005 | IEC Webstore | automation, cyber security, smart city, smart energy, smart grid.” <https://webstore.iec.ch/publication/6208> (accessed Jun. 01, 2018).