An efficient method of extracting network information from CIM asset model

Conference Paper · September 2016

DOI: 10.1109/ISGT.2016.7781190

2 authors:

Srivats Shukla
Virginia Polytechnic Institute and State Unive…
6 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Mark G. Yao
IBM
17 PUBLICATIONS **235** CITATIONS
SEE PROFILE

An Efficient Method for Extracting Network Information from the CIM Asset Model

Srivats Shukla[1]
Bradley Department of Electrical and Computer Engg.
Virginia Tech.
Falls Church, VA, USA.
srivats@vt.edu

Mark G. Yao
Smarter Energy Research Institute
IBM T.J.Watson Research Center
Yorktown Heights, USA
markgyao@us.ibm.com

***Abstract*—The Common Information Model (CIM) - IEC 61970, a well-known industry standard data model, is used to describe networked physical assets in an electrical power grid. Whereas this data model can be used to represent most detailed description of grid assets, currently it has limited application in power system network modeling. This is due to a couple of reasons. Firstly, many applications only require a simplified, mathematically abstracted network model and secondly, the CIM represents only a snapshot of grid state and therefore cannot accommodate time-series data from SCADA/PMU measurements. Furthermore, the legacy software developed for applications like powerflow consume data in proprietary formats, none of which is essentially CIM. As a first step in developing a new interoperable power grid data model to satisfy grid-analytic requirements, we have investigated and used CIM as one of the data ingestion sources. In this paper, we present our work of data ingestion, abstraction and translation from CIM to Bus-Branch Model (BBM) as a part of an overall system design and implementation for grid analytics and data interoperability.**

***Index Terms*—Common Information Model, Bus-Branch Model, Model-driven Analytics, Interoperability.**

I. Introduction

As predictive analytics and big data technology become standard practices in smart-grid operation and planning, a comprehensive data model of the electrical power grid is required. Model-driven analytics impose many challenging requirements on the grid data model and the technology supporting such data-systems, including features like:

-Representation of physical properties of individual grid assets, such as: generators, loads, lines, and transformers,

-Representation of logical or cyber properties of each asset,

-Relationship and association of assets,

-Flexibility of accommodating various historical measurements or forecasted temporal and geo-spatial data.

-Interoperability and ability of abstraction.

Our approach to achieve these requirements is to build a data management system around a canonical grid data model, based on existing power industry data standards such as Common Information Model (CIM) [1], IEC 61850 [2], etc. This system provides features like: (1) a standard API to access the data at different levels of abstraction, driven by the need of the application; and (2) two-way data-interoperable adapters to translate from different data model standards. Fig. 1, gives an illustration of the proposed data management system. In the following text we describe a use-case wherein this architecture was applied.

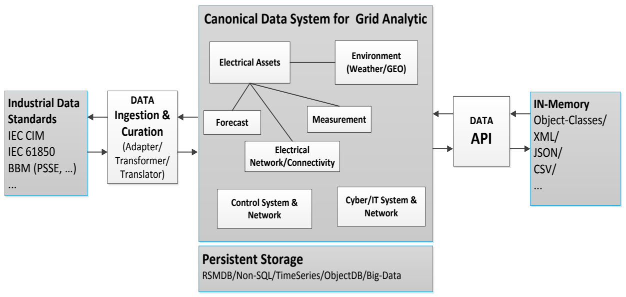
Canonical Data System for Grid Analytic
Electrical Assets
Environment (Weather/GEO)
Forecast
Measurement
Electrical Network/Connectivity
Control System & Network
Cyber/IT System & Network
Persistent Storage
RSMDB/Non-SQL/TimeSeries/ObjectDB/Big-Data
Industrial Data Standards
IEC CIM
IEC 61850
BBM (PSSE, ...)
...
DATA
Ingestion & Curation
(Adapter/ Transformer/ Translator)
DATA
API
IN-Memory
Object-Classes/
XML/
JSON/
CSV/
...

Fig. 1.Data management system for grid analytics engine

Prior literature focuses around developing architectures for converting data from CIM to various proprietary formats [3]-[8]. Although several ideas have been proposed [3], [7]-[8] for extraction and storage of the data represented in CIM, yet a thorough discussion of step-by-step implementation of these architectures is lacking. Pradeep et. al. [7] propose pseudo-algorithms related to CIM-based substation and network topology processing, leveraging the proposed 'topological branch' class. It is not clear how this idea scales for different real utility systems. McMorran et. al. [4] discuss a CIM toolkit implemented in JAVA, but they rely on the availability of direct parsers which can ingest UML-schemas, and CIM XML files to create related JAVA objects. We found that most of the utility data is integrated in a simplified CIM-Resource Description Framework (RDF) [9] format which doesn't conform to the complex UML hierarchies of composition or aggregation, rendering commercial parsers inadequate.

Motivated by such considerations, in the present work, we summarize the design and implementation of a data management tool that converts the asset data of a real utility system represented in CIM-RDF into an internal canonical in-memory data model. This tool successfully captures the hierarchies of association and aggregation. Then it translates the captured information into an abstracted Bus-Branch Model (BBM). The tool also ingests contingency information in terms

[1] This work was done when the author was an intern at the IBM T. J. Watson Research Center

of breaker status and creates a lookup table listing the nodes disconnected from the network for every contingency. The in-memory canonical data model helps in fast network traversals. The information generated by this tool can be used to build applications like congestion analysis/OPF.

The remainder of this text is organized as follows: section II gives an overview of the architecture of the CIM-adapter tool; section III discusses the process of extracting information of hierarchical data objects and storing them; sections IV and V respectively discuss a traversal algorithm for deriving the network information from CIM model and its implementation on a test-bed. Section VI gives an outline for converting the derived CIM data into raw formats for simulations.

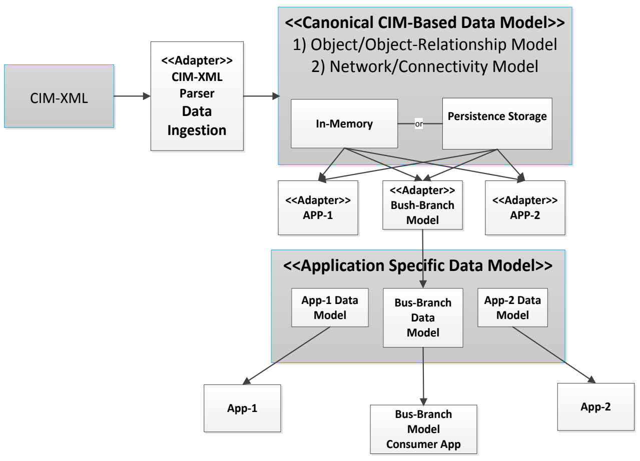

Fig. 2. Architecture of the CIM adapter

II. Architecture of Adapter Tool

The basic architecture of the developed CIM adapter is given in Fig.2. As discussed, if the CIM instantiation developed by the utility is in simplified CIM-RDF format, the data ingestion adapter needs to implement the XML parser, create hierarchical power system resource objects and also build an in-memory representation of the network connectivity. Once these objects are created, they need to be accessed in memory and/or be persistent in a database. The net-sum of the information about connectivity and the electrical asset properties (like conductivity) are stored in the internal data model. The information data model can then be retrieved by various adapters to translate the data to a format specific to the application. As an example, a service running Optimal Power Flow will use an abstracted Bus-Branch Model of the utility network. Hence, a specific adapter needs to be designed that extracts electrical attribute information from the internal canonical data model. Next, an implementation of this adapter is summarized.

III. Building Hierarchical Power System Objects

As mentioned previously, most utilities capture their data in a simplified CIM-RDF XML format [9]. In this format, the XML's function of modeling hierarchical data type is not exploited. As shown in Fig. 3, a class (e.g., a synchronous machine) composing another class (e.g., a generating unit) is not expressed as one of the of the parent class. Rather, both classes are direct children of the root node. A method used to express the *compositional* hierarchy of UML is via the concept of containment. CIM has an '*EquipmentContainer*' class to define instances where a power system resource is contained physically and/or electrically within another resource. In the simplified representation, RDF identifiers are used as an attribute to map a contained object to its container. Furthermore, the RDF is also used to capture *aggregation type* hierarchies. A good example of this hierarchy is the relationship between transformer windings and taps. Our data extraction algorithm, described next, includes a procedure to extract these hierarchies.

A. Identification of Canonical Power system resources

It should be noted that CIM standard has rapidly evolved, with later versions adding more precise definitions of various object classes. Different energy utilities have used different CIM versions to capture their asset information, and many utilities are yet to update to the latest versions. If adapters have to be developed for models captured in different CIM versions, canonical object classes have to be identified for each CIM version. As an example, a 2001 CIM version instantiation may use an object class '*EnergyConsumer*' to define a consumer load while a 2008 version might use an object class '*NonConformLoad*' for the same resource. Hence, if a custom parser is written, then there is a need to identify the canonical objects for each instantiation conforming to a particular CIM version. A simplistic algorithm to extract canonical data objects from a CIM-RDF instantiation is as follows: (1) find the set of all the children objects of the root node, (2) remove all the redundant object types to find a set of canonical object types. It should be noted that the XML and RDF namespaces are required to parse each object from the model.

B. Extraction of Object attributes

First, we define classes for each canonical resource type found in the CIM version. Then, we read in all the objects present in the asset model and sort each of those objects into their respective classes. While creating an object, the values of attributes specific (e.g., R, X) to that class (e.g. ,AC Line) have to be extracted. Example data objects are illustrated in fig. 4.

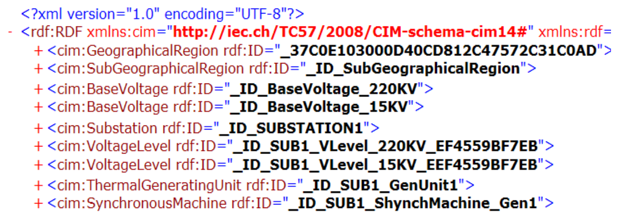
```
  <?xml version="1.0" encoding="UTF-8"?>
- <rdf:RDF xmlns:cim="http://iec.ch/TC57/2008/CIM-schema-cim14#" xmlns:rdf=
   + <cim:GeographicalRegion rdf:ID="_37C0E103000D40CD812C47572C31C0AD">
   + <cim:SubGeographicalRegion rdf:ID="_ID_SubGeographicalRegion">
   + <cim:BaseVoltage rdf:ID="_ID_BaseVoltage_220KV">
   + <cim:BaseVoltage rdf:ID="_ID_BaseVoltage_15KV">
   + <cim:Substation rdf:ID="_ID_SUBSTATION1">
   + <cim:VoltageLevel rdf:ID="_ID_SUB1_VLevel_220KV_EF4559BF7EB">
   + <cim:VoltageLevel rdf:ID="_ID_SUB1_VLevel_15KV_EEF4559BF7EB">
   + <cim:ThermalGeneratingUnit rdf:ID="_ID_SUB1_GenUnit1">
   + <cim:SynchronousMachine rdf:ID="_ID_SUB1_ShynchMachine_Gen1">
```

Fig. 3. A snapshot of the CIM-RDF XML file from OpenCIM website [10]

PowerSystemResource
ResrourceType X{
Attribute XA1;
Attibute XAn;
} X1, X2,…Xn;

PowerSystemResource
ResrourceType Y{
Attribute YA1;
Attibute YAn;
} Y1, Y2,…Yn;

Fig. 4. Internal representation of identified objects from CIM

C. Representation of Objects with Hierarchical Relationships

One of the interesting aspects of CIM–RDF models is that each of the canonical classes has sets of attributes that can either represent a specific property (e.g., electrical/ magnetic) or a *containment* in another class. Several applications need a hierarchical object to be represented as one single composite of all the contained objects. As an example, whereas the CIM representation captures transformer (core) properties and winding properties in two different classes, a powerflow calculation application requires an abstracted representation of a power transformer as a net sum of the core and the windings having specific attributes taken from both of these classes. Similarly, a Bus-Branch Model requires both: (1) the connectivity information captured in the attributes of a conducting equipment (e.g., Synchronous Machine), and (2) properties of the container of that equipment (e.g., name/ID of the Generating Unit).

```
ResourceX1
= { attributeXA1: val ,
  attributeXA2: val,
  Childlist:          [resourceY1,
resourceY2],
  childattrY1  :  {  attributeYA1:
val,
                  attributeYA2: val,
                  }
childattrY2 : { attributeYA1: val,
                  attributeYA2: val,
                  }
```

```
Transformer [X1]
= { 'ID': 'Trans_1' ,
  'name': 'xyz',
  'Windlist':            ['Xwind1',
'Xwind2'],
  'childattrXwind1' : { 'r': 0.001,
                        'x': 0.001,
                        }
'childattrXwind2': { 'r': 0.001,
                      'x': 0.001,
                      }
}
```

Fig. 5. Nested dictionary (in-memory) representation of hierarchical classes

In such a scenario, relational objects have to be created which are needed to be both stored in memory and persistent in a database. Our CIM ingestion adapter captures relationship in-memory in the form of dictionaries, with properties represented in key-value pairs. This lets us represent the hierarchical classes in form of nested dictionaries. The proposed nested dictionary representation is shown in the Fig. 5, along with an example of a transformer object in same format. As far as persisting the data objects in a database is concerned, it is understood that it is rather complex to persist hierarchical objects in databases. Furthermore, speed of retrieval of such objects from databases is slow, leading to inadequacy for application in real-time services. There are a couple of options for storing such hierarchical objects:

- Development of CIM oriented databases using Object Relational Mapping (ORM) as discussed in [11].
- Use of some ad-hoc data structures where the data objects related to each other are merged into one relational object, without using nesting, and just maintaining one level depth of hierarchy. One such mechanism is discussed below.

Another technique to denote such hierarchical data structures is to use dictionaries with a key-value pair representation, and instead of nesting, the text string used for naming the keys can represent hierarchies. For example, when representing the attributes of a child object, a keyword 'childattr' can be appended with every key. These ad-hoc data objects can be directly stored in the databases. A schema needs to be developed to correctly develop data queries under such a representation, but it is less complex than storing hierarchical objects and the speed of information retrieval is faster. The proposed representation is illustrated in Fig.6, accompanied with an example of a transformer object. Once the identified objects are persisted in a database, we build a network graph from the CIM information.

```
ResourceX1
= { attributeXA1: val ,
  attributeXA2: val,
  numchild : val,
  child1: resourceY1,
 child2: resourceY2,
 childattrY1A1:val,
 childattrY1A2: val,
 childattrY2A1:val,
 childattrY2A2: val,
  }
```

```
Transformer [X1]
= { 'ID': 'Trans_1' ,
  'name': 'xyz',
   'numWind': 2
  'XWind1': 'Windname1'
  'XWind2': 'Windname2'
  'childattrXwind1_r' : 0.001,
  'childattrXwind1_x' :  0.001,
  'childattrXwind2_r' : 0.001,
  'childattrXwind2_x' :  0.001,
}
```

Fig. 6. Dictionary based data structures with key names denoting hierarchies

IV. Network Representation

In this section, we give the details of the developed traversal algorithm which builds a network of identified power system resources by interpreting their connectivity information from the CIM file. Before discussing the proposed algorithm for tree traversal, the following points should be noted:

- The connectivity information in CIM is captured using the wires package;
- Connectivity representation is done via- conducting equipment, terminals, and connectivity nodes;
- Two pieces of conducting equipment having a direct electrical connection will be depicted in CIM as having one terminal each pointing to the same connectivity node, as shown in Fig. 7.
- The concept of (electrical) connectivity, obviously doesn't apply to two windings within a transformer, because they are magnetically connected. Hence, care has to be taken while processing a transformer node during traversal.

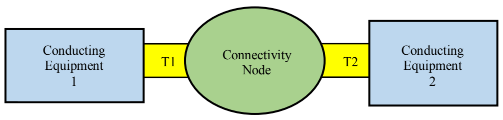

Fig. 7. Depiction of electrical connection in CIM

A. Traversal nodes

The developed algorithm relies on the fact that only three different types of nodes will be encountered during traversal of the network: a terminal (Te), a conducting equipment (CE) and a connectivity node (CN). Hence, a class is developed to denote a node object during traversal that can represent any of these three distinct types. An illustration of the class that represents a traversed node object is given in Fig. 8. If the node is a terminal, then it has a node ID and a type. If the node is a CE or a CN, it has a list of terminals and their number as an attribute. It is easy to see that the traversal process entails: (a) moving from one node to the other and (b) traversing through all the inter-connected nodes in the network. Next, we describe some of the functions which were used in for traversal.

```
class node_trav:
    ID = 'null'
    node_type = 'null'
    CE_type = 'null'
    Terminal_List = []
    num_attchTerms =0
```

Attribute Name	Attribute Description
Node_Type	Type of node: CE, TE or CN
CE_type	Type of Equipment: Generator, Load, Lines, Breaker, Bus, Compensator, etc.
Terminal_List	The name of terminals (only if the current node is a CE or a CN)
Num_attachTerms	Number of terminals, if the node is a CE or a CN

Fig. 8. Traversed node class and its description

B. Finding the next node to be traversed

The function (*find_next_node*) for movement from one node to the other is implemented via three traversal node objects, representing the previous (prev_node), current (curr_node) and next node (next_node) to be traversed respectively. Because of the way in which the network is characterized in CIM, it can be easily seen that during traversal, when the current traversed node (curr_node) is conducting equipment (CE) or a connectivity node (CN), the next node will always be a terminal (Te). When the current traversed node is a terminal, then the node to be traversed next would be a CE, if the previous node was a CN and vice-versa.

C. Terminal traversal flag

As mentioned before, each CE and CN in the network will have one or multiple terminals attached to it. While traversing a CE or CN, we need to keep track of all the attached terminals that haven't been encountered (traversed) before. This is realized by adding an attribute called a 'traversal flag' to every terminal in the system with the flag value initialized at zero. The traversal flag is set to one once the terminal has been traversed. For the purpose of the algorithm, a single terminal conducting equipment will be called an 'end-device'. A pseudo-algorithm for traversal is presented in the next column.

V. Network Traversal of an Example System

The proposed data extraction and traversal algorithm is tested on an example 3-bus system as shown in Fig. 9. Fig. 10 shows a snapshot of the AC Line (branches) database for the 3-bus system. Fig. 11 illustrates the result of the proposed traversal algorithm. Each step (P1-P6) has the IDs of equipment connected between two buses, or a bus and an end-device.

The discussed algorithm takes around 45 seconds to parse the data from a thirty-thousand node customer CIM data on an Intel® core™ i7 processor with 16 GB RAM. While the complexity of current algorithm is *$O(n^2)$*, efforts are being made for reducing it to *$O(nlogn)$* by using hash maps. At the time of this writing, due to lack of data on computational time and efficiency of other existing CIM-BBM translation algorithms, we could not compare the computational efficiency of our method with other existing ones.

```
Psuedo-code for network traversal
Step0: Initialize:
    - CN_stack : to push a CN as soon as it is visited, pop when all
      terminals attached to this node are traversed
    - CE_stack: to push a CE, as and when encountered
    - everything_stack: to push all the visited nodes (CE, CN, Te)
Step1: For traversal, select the starting node as an end device; initialize the
current node as the starting node
Step2: Get the next node to be traversed from the function: next_node =
find_next_node(prev_node, curr_node)
Step3: If the current node
    - is a Te:
        - if next_node is CN:
          Option1. if the CN is not attached to a busbar, then update
          the current node as nextnode and go to step 2
          Option2. if the CN is attached to a busbar, then terminate
          traversal at the busbar, publish the CE_stack,
          everything_stack
        - if next_node is CE: update the current node as nextnode
          and go to step 2
    - is a CN: find an untraversed terminal from the list of attached
      terminals.
        Option1. If there is an untraversed terminal then update the
        current node as nextnode and go to step 2
        Option2. If there is no untraversed terminal remaining, then
        terminate traversal, publish the CE_stack, everything_stack,
        pop the current CN off the CN stack, mark the next node as
        the CN on top of CN stack, go to step 2.
    - is a CE: find an untraversed terminal from the list of
      terminals.
        Option1. If there is an untraversed terminal and current CE is
        not an open breaker, then update the current node as nextnode
        and go to step 2
        Option2. If there is no untraversed terminal remaining (this
        happens at an end-device) or current CE is an open breaker,
        then terminate traversal, publish the CE_stack,
        everything_stack, mark the next node as the CN on top of CN
        stack, go to step2.
```

VI. Creating Raw Data Format & Contingency Cases

The proposed scheme involves implementation of a post-processing method where (a) virtual buses are added between any two pieces of conducting equipment which are not connected via a bus-bar section; (b) circuit equivalents are derived if required, for example, a transformer might be represented as an AC branch; (c) any series closed switches are represented as short circuits; (d) data tables for each conducting equipment is created as discussed below.

The result of the discussed traversal algorithm will be a connected graph of conducting equipment nodes that represents the physical network connections between the power system resources. To produce the data in a raw format which can be used, for example, to run powerflow simulations, the network information needs to be grouped with the specific attribute information of each node, and this net information is then represented in tables. Each table represents information of a specific type of conducting equipment like a bus, AC line, load, generator etc. Table I shows a row from 'Branch-data table' for the discussed 3-bus system. The 'from-bus' and 'to-bus' information is derived from network traversal results, and the line data from the attribute database (Fig. 10).

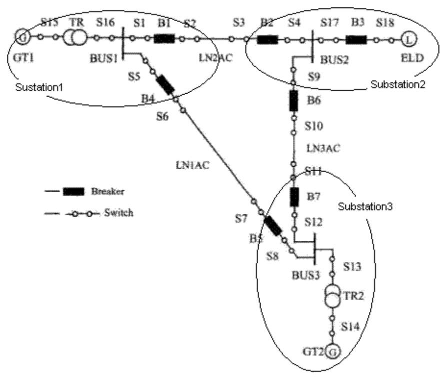

Fig. 9. A 3-bus system example taken from OpenCIM website [10]

LineData[0] = {'Name':' LN1AC', 'r': 0.231, 'x' : 130.25, …}
LineData[1] = {'Name':' LN2AC', 'r': 0.215, 'x' : 20.45, …}
LineData[2] = {'Name':' LN3AC', 'r': 5.656, 'x' : 50.33, …}

Fig. 10. Dictionary storage of AC Line data from 3- bus system

```
    P1: ['#_ID_SUB1_ShynchMachine_Gen1', '#_ID_TR1_W1',
'#_ID_TR1_W2',
'#_ID_SUB1_BUS1_2F6957E4EF4559BF7EB']
    P2: ['_ID_SUB1_BUS1_2F6957E4EF4559BF7EB',
'#_ID_LN2AC_ACLineSeg',
#_ID_SUB2_BUS2_2F6957E4EF4559BF7EB']
    P3: ['_ID_SUB2_BUS2_2F6957E4EF4559BF7EB',
'#_ID_LN3AC_ACLineSeg',
'#_ID_SUB3_BUS3_2F6957E4EF4559BF7EB']
    P4: ['_ID_SUB3_BUS3_2F6957E4EF4559BF7EB',
'#_ID_LN1AC_ACLineSeg',
'_ID_SUB1_BUS1_2F6957E4EF4559BF7EB']
    P5: ['_ID_SUB3_BUS3_2F6957E4EF4559BF7EB',
'#_ID_TR2_W2', '#_ID_TR2_W1',
'#_ID_SUB2_ShynchMachine_Gen2']
    P6: ['_ID_SUB2_BUS2_2F6957E4EF4559BF7EB',
'#_ID_SUB2_LOAD1']
```

Fig. 11. Result of traversal algorithm on 3-bus network

Line Name	From Bus	To Bus	Line r	Line x
LN1AC	Bus1	Bus3	0.231	130.25

Table I. Line Data in raw format for powerflow simulation.

Several applications like online stability analysis require running N-1 contingency analysis on the network. As it can be easily seen, switch positions control the state of a power network, and such analysis can be conducted by simulating network status under different configurations of switches. The power network derived from the CIM data reveals critical information for contingency simulations. As a part of the traversal algorithm, all the breakers and the devices that they can isolate from the system are identified. Each pass of the traversal algorithm lists the resources connected in series between two buses, or a device directly connected to a bus. This information is used to create a table which lists the equipment that can be isolated by each switch.

VII. Conclusions

As smart grid technology becomes mature, the grid network topology will be modified and updated more frequently. These dynamic network changes must be reflected in CIM and requires an efficient, real-time CIM-BBM translation algorithm. The proposed method is useful for system architectural design and application implementation. For system architecture design, the CIM-BBM can function as one of key integration components between long-term grid asset monitoring, and dynamic grid operation system. This will help in developing a unified framework for analytic applications.

References

[1] D. Becker, and T.L. Saxton, "CIM standard for model exchange between planning and operations," *Power and Energy Society General Meeting 2008*, pp.1-5, July 2008.

[2] R.E. Mackiewicz, "Overview of IEC 61850 and Benefits," *Power Syst. Conf. and Expo., PSCE '06.*, pp.623-630, Oct-Nov. 2006

[3] A. McMorran, G. Ault, I. Elders, C.Foote, G. Burt, and J. McDonald, "Translating CIM XML power system data to a proprietary format for system simulation," *IEEE Trans. Power Syst*, vol.19, no.1, pp.229-235, Feb. 2004.

[4] A. McMorran, G. Ault, C. Morgan, I. Elders, and J. McDonald, "A common information model (CIM) toolkit framework implemented in Java," *IEEE Trans. Power Syst.*, vol. 21, no. 1, pp. 194 –201, Feb. 2006.

[5] L.E. Arnold, and J. Hajagos, "LIPA implementation of real-time stability monitoring in a CIM compliant environment,*" in Power Syst. Conf. and Expo., PSCE '09*. pp.1-6, March 2009.

[6] F.Milano, M.Zhou, and H. Guanji, "Open model for exchanging power system data," *Power & Energy Society General Meeting, 2009,* pp.1-7, July 2009.

[7] Y. Pradeep, P. Seshuraju, S.A. Khaparde, and R.K. Joshi, "CIM-Based Connectivity Model for Bus-Branch Topology Extraction and Exchange," *IEEE Trans. Smart Grid*, vol.2, no.2, pp.244-253, June 2011.

[8] J.D.Moseley, and N.V. Mago, "Methods of converting CIM power system models into bus-branch formats utilizing topology processing algorithms and minimal schema modifications to IEC 61968/70," *Power and Energy Society General Meeting 2013*, pp.1-5, July 2013.

[9] M. Uslar, T. Schmedes, A. Lucks, T. Luhmann, L. Winkels, and Hans-J. Appelrath. "Interaction of EMS related systems by using the CIM standard." *Second International Symposium on Information Technologies in Environmental Engineering (ITEE)*, pp. 596-610,Sep. 2005.

[10] InterPSS OpenCIM,, available at: https://sites.google.com/a/interpss.com/opencim/Home/interpss-opencim

[11] G. Ravikumar, S.A.Khaparde, and Y. Pradeep, "CIM oriented database for topology processing and integration of power system applications," *Power and Energy Society General Meeting 2013*, pp.1-5, July 2013.