SoftwareX 9 (2019) 161–167

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

CIM-2-mod: A CIM to modelica mapping and model-2-model transformation engine

Francisco J. Gómez [a,*], Luigi Vanfretti [b], Miguel Aguilera [c], Svein H. Olsen [d]

[a] Electric Power and Energy Systems, Royal Institute of Technology, Sweden
[b] Electric, Computer and Systems Engineering, Rensselear Polytechnique Institute, Troy, NY, USA
[c] Instituto Costaricense de Electricidad (ICE), San José, Costa Rica
[d] Statnett SF, Oslo, Norway

ARTICLE INFO

Article history:
Received 15 April 2018
Received in revised form 3 January 2019
Accepted 17 January 2019

Keywords:
Common information model
CIM
Model transformation
Information modeling
Modelica
OpenIPSL
Electrical power grid
Power system modeling
Power system dynamics
Power system simulation

ABSTRACT

New requirement on power systems analysis tools consider information exchange for both steady-state and system dynamics information. New European regulations on information exchange power system dynamic simulations now require coordinating TSOs operations under different scenarios, some of which require to assess the dynamic behavior of power systems under a vast array of contingencies. As a mean to comply with these regulations and to advance the state-of-the-art, this work describes the software architecture of a Model-To-Model (M2M) transformation tool to create power system dynamic models using Modelica components by linking it to data from the Common Information Model (CIM). This software architecture is conceived to combine the CIM standard language with the Modelica standardized language, and to provide a Free/*Libre* Open Source Software (FLOSS) CIM-compliant unambiguous power system modeling solution considering both steady-state and dynamic model representations of the electrical grid.

© 2019 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

Code metadata

Current Code version	*Code v118*
Permanent link to code/repository used of this code version	https://github.com/ElsevierSoftwareX/SOFTX_2018_36
Legal Code License	*GPLv3*
Code Versioning system used	*Github*
Software Code Language used	*JAVA*
Compilation requirements, Operating environments & dependencies	*Eclipse Project, library dependence: Apache JENA, JAXB (they are provided as User libraries within the eclipse project)*
If available Link to developer documentation/manual	https://github.com/ALSETLab/cim2modelica/blob/master/docs/
Support email for questions	fragom@kth.se

Software metadata

Current software version	*1.0*
Permanent link to executables of this version	https://github.com/ALSETLab/cim2modelica
Legal Software License	*GPLv3*
Computing platform / Operating System	*OS X, Microsoft Windows, GNU/Linux*
Installation requirements & dependencies	*JAVA 1.8 (User libraries, with Apache JENA and JAXB, are included within the .jar file)*
If available Link to user manual-if formally published include a reference to the publication in the reference list	https://github.com/ALSETLab/cim2modelica/blob/master/docs/
Support email for questions	fragom@kth.se

* Corresponding author.
E-mail addresses: fragom@kth.se (F.J. Gómez), luigi.vanfretti@gmail.com (L. Vanfretti), maguilerach@ice.go.cr (M. Aguilera), svein.harald.olsen@statnett.sf (S.H. Olsen).

https://doi.org/10.1016/j.softx.2019.01.013
2352-7110/© 2019 Published by Elsevier B.V. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/).

1. Introduction

The European Network of Transmission System Operators for Electricity (ENTSO-E) adopted the regulation (EC) 714/2009 [1], which considers the need of coordination between Transmission System Operators (TSOs) with the "use of a common transmission model dealing efficiently with interdependent physical loopflows and having regard to discrepancies between physical and commercial flows". This common model should be used for "common network operation tools to ensure coordination of network operation in normal and emergency conditions". The Common Information Model (CIM) is the result of the International Electro-Technical Committee 57 (TC57) Working Group 13 and the Electrical Power Research Institute (EPRI) to provide a standard modeling semantic representation to comply with these regulations. Among the different packages that conform the CIM [2], all the basic electrical information from components and topology of the power network, with its steady-state behavior, is defined within the IEC 61970-301 CIM Core for Steady-State Information and IEC 61970-(452,453,456) CIM Core Specification Profiles. New developments on the CIM have resulted on the approval and implementation of the IEC 61970-302 CIM for Dynamics Specification and the IEC 61970-457 for Dynamics Profile. Although those packages provide a solid basis for information exchange, they do not provide needs to guarantee a consistent representation of the physical dynamic behavior that the components would expose in simulations carried out using different software tools.

1.1. Motivation

The CIM and the CGMES standards provide parameters and pictorial diagrams of power system steady-state for information exchange. Current proprietary analysis tools use these values within their internal data structures and internal solvers. These proprietary tools also support information exchange for CIM and CGMES including power systems dynamics information, that comply with the IEC 61970-302 and IEC 61970-457. However, the physical dynamic behavior is only represented in a pictorial block diagram description rather than in an explicit mathematical form (i.e. differential and algebraic equations). Thus, in this current state, interoperability can only be unassailable through the comparison of several implementation results, which depend on each individual representation embedded within the individual simulation tools' numerical routines.

With the increased need for coordination between TSOs and to comply with the aforementioned regulations, new analysis tools require unambiguous model representation. Such representation requires a computer-readable, equation-based, and standardized modeling language to describe the dynamics of each component and the power system as a whole.

1.2. Background

The Modelica language [3] is an open, standardized, object-oriented and equation-based language suitable for modeling complex cyber–physical systems. Modelica components from the OpenIPSL library help to describe the mathematical behavior of the power systems components, and they have been validated against common power system software tools as reference PSSE and PSAT [4]. To initialize a Modelica model, start values (i.e. initial guess) for all algebraic, continuous and discrete variables need to be provided [5,6]. This process is part of the numerical routines that a software developer needs to implement, and requires to set proper initial guess values to the components, which can solve the initialization problem before time-domain simulation can be performed.

The models in OpenIPSL defines start values as parameters, they correspond to the "steady-state" variables of algebraic variables in the transmission lines, namely voltage and powers. These values are derived from a power flow solution and other static equipment variables to be provided by the model developer when assembling a network model. These values are required by the available solvers in any Modelica compiler to calculate the initial conditions (i.e. solve the initialization problem), before conducting time-domain simulations. Note that the solution of the initialization problem is specific to each Modelica tool, the use of the Pantelides algorithm being the most common among them [7].

In the literature, there exists a Python library for reading CIM information. The PyCIM project [8] implements the CIM classes and attributes from version 14 and version 15. However, this library is not updated to support the CIM version 16, which includes information for dynamics. Other works such as [9] offer transformation capabilities, from specific modeling tools to CIM. In [10] their work is focused in graph-based algorithms to handle power systems data and topology processing within the CIM representation.

The CIM-2-MODelica tool is a prototype implementation for a model-to-model transformation involving different modeling languages. The concept of Model-to-Model (M2M) transformation [11], based on mapping rules, which associate the naming conventions from the source language with the naming convention of the target language, is implemented with basic eXtensible Modeling Language (XML) schema [12]. In the literature, there are other rule-based programming languages such as ATL [13], which is platform based. However, the XML schema and formats are more comprehensible by power system domain users and platform independent. XML is also used as the syntax format implementation for CIM models. Different Modelica compilers, such as OpenModelica, offer functionality to export Modelica code to XML.

1.3. Contributions

This M2M transformation tool proposes a modular software engine that allows to transform parametric and structural information of a power grid contained within the CIM and transform it into a Modelica model that fully describes the CIM information and it compliments it with behavior in the form of equations. This produces a computable model, i.e. one that can be simulated and analyzed using Modelica tools.

This software architecture proposes the design of a set of classes and methods for processing the information available in different CIM Profiles. A repository for **mapping rules based on CIM v16 and the OpenIPSL Modelica library** for power systems [14,15] is proposed. The design of this mapping repository and software architecture is based on Object-Oriented Programming (OOP) principles. Thus, the design and the current implementation could be reutilized to other programming languages. The internal package distribution of the architecture will be described, as well as the main actions or tasks to be implemented. It will follow the description of the internal class structure of the tool, which considers the main software objects that should be implemented, using other OOP programming languages, to reproduce the functionality described in this paper.

The proposed solution provides means **to aid a Modelica tool in solving the initialization problem**. This is done by proper use of the Modelica language features in each of the variables that should be initialized, within each of the *"initial equation"* section of every Modelica component within a whole power system. Other variables can be specified with the Modelica keyword *"start"*, which is utilized to control the initialization of the required component's variables.

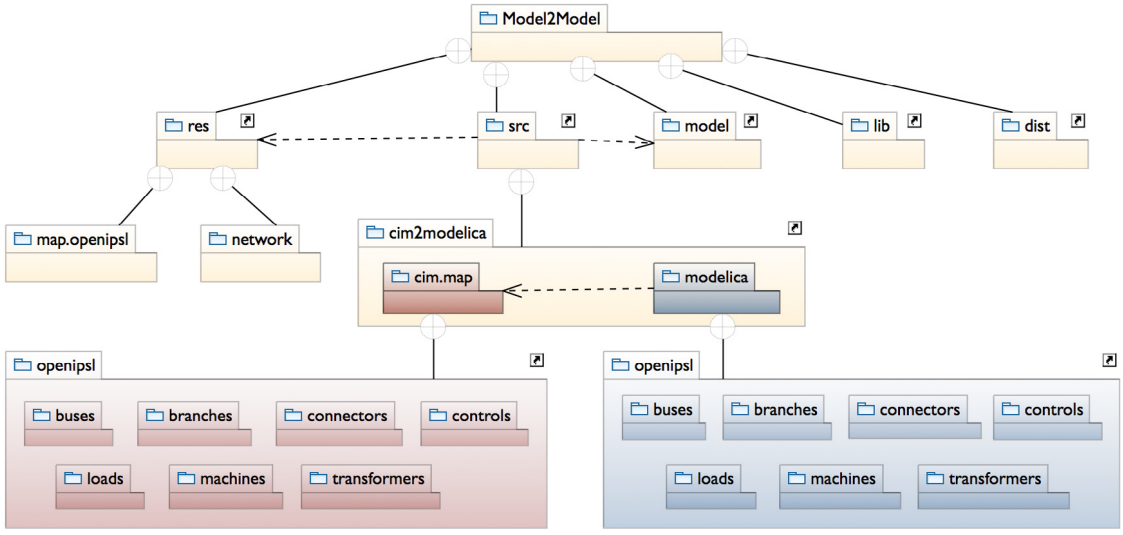

Fig. 1. Package and Folder organization of the proposed software architecture.

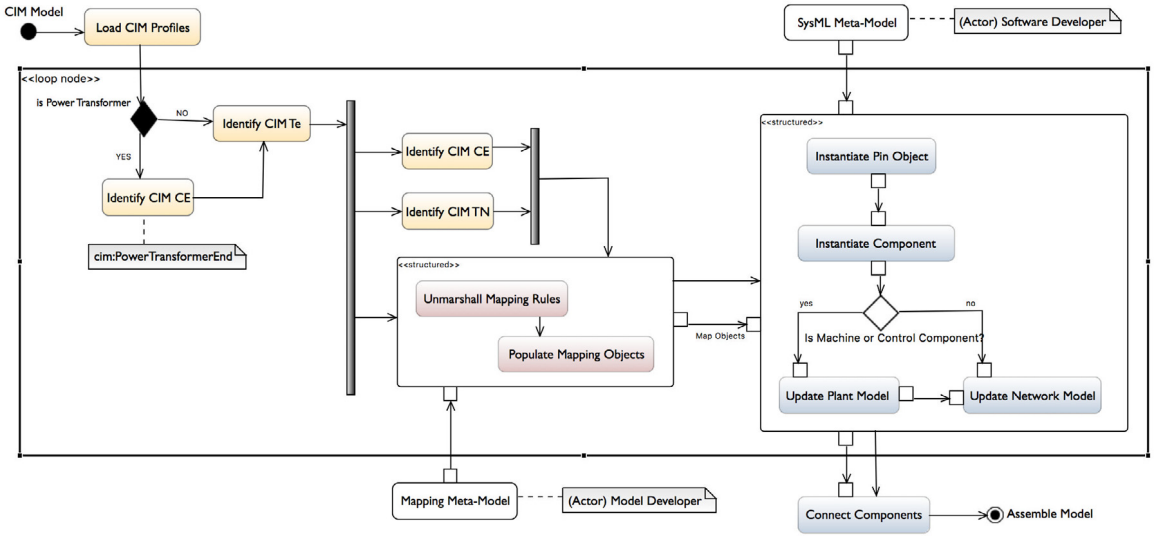

Fig. 2. Actions in white are functionalities for tool enhancements; Actions in the first <<structured>> node perform the mapping operations; and actions in the second <<structured>> node implement the creations of target Modelica model. The Arrows connecting small squares indicate the pass of objects. Arrows connecting the actions indicate control flow through. The actors responsible for each action are enumerated within the comment icon.

The CIM information source provides realistic equipment parameters and power flow solution values, required by those components. This information **establishes the initial guess of the variables in power system components**, which are fundamental for the solvers to calculate the proper behavior of the dynamic models utilized for the time-domain simulations. The M2M transformation process automatically provides the corresponding equipment values, power flow solution and dynamic values, available within the source CIM information profiles, to the corresponding Modelica components.

2. Software description

This software framework acts as a M2M transformation tool that enables the user to create a dynamic Modelica network model, with the suitable initial conditions for time-domain simulations, from the corresponding CIM model. The framework is built upon mapping rules, which connect CIM semantics with the corresponding semantic description of the OpenIPSL library components. The transformation tool works with information models built upon the IEC 61970-301 and IEC 61970-302 packages, which support the CIM v16. In its current version, it supports a limited number of components to represent the basic topology of an electrical network and different combinations of machine and controls for power plants.

3. Software architecture

The software framework is organized in different folders and packages to facilitate the organization of internal resources and class structure (see Fig. 1). The internal class structure is conceived to facilitate the scalability of the transformation tool, allowing the creation of additional packages for new source and target models:

1. The ***dist*** folder to store the generated library .jar files.
2. The ***lib*** folder to store the .jar files that are used within the library implementation.
3. The ***res*** folder, which contains different resources used by the library. The *res.map.openipsl* folder contains the implementation of the mapping rules between the CIM and the

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE model SYSTEM "openipsl_component_map.dtd">
<pwPinMap cim_name="class_name" rfd_id="" rfd_resource=""
          package="openipsl_package_path" name="openipsl_component_name"
          stereotype="connector">
<attributeMap cim_name="attribute_cim_name"
          name="openipsl_parameter_name"
          datatype="Real" variability="parameter" visibility="public"
          flow="0"> value </attributeMap>
<attributeMap cim_name="attribute_cim_name"
          name=" openipsl_parameter_name"
          datatype="Real" variability="none" visibility="public"
          flow="1"> value </attributeMap>
</pwPinMap>
```

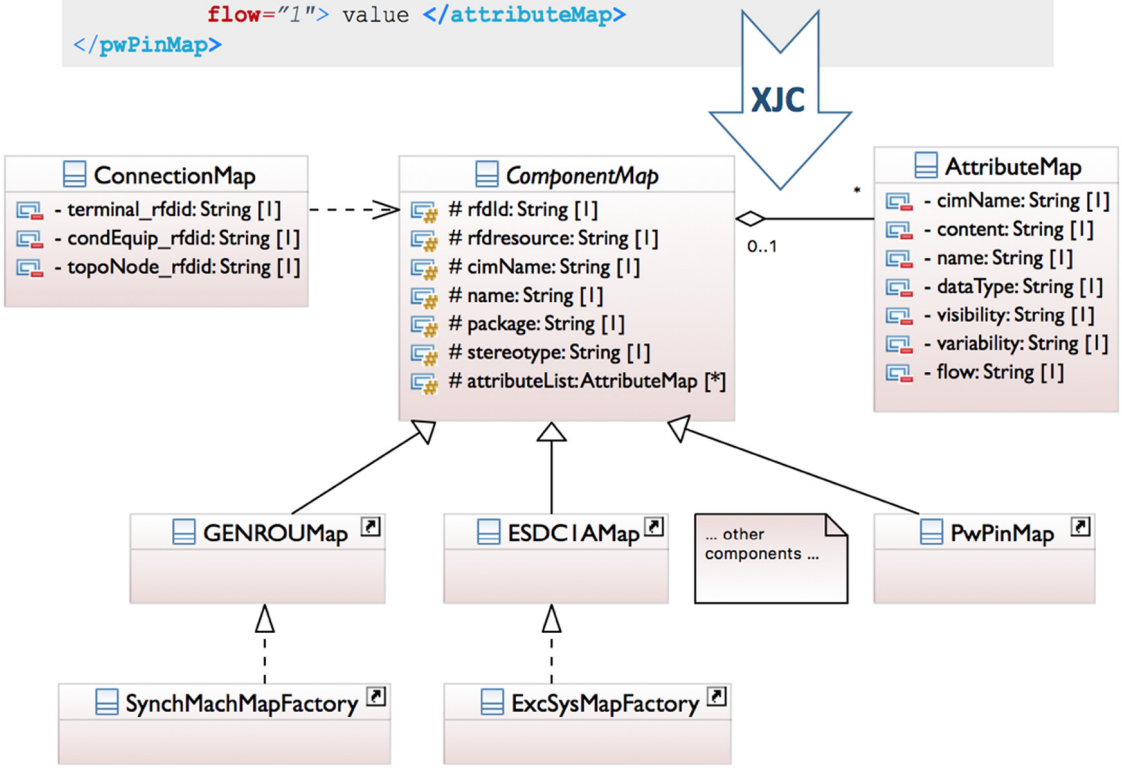

Fig. 3. On the top of the arrow, an example of an XML Mapping rule file with general syntax. On the bottom, the final version of the Mapping Meta-Model class structure resulting from the execution of the XJC command for generating JAVA classes from XML schema.

OpenIPSL. And the *res.network* folder contains any CIM class and attribute used as inputs for the transformation process.
4. The ***src*** folder, which contains the packages and classes to implement the transformation process.
5. The ***model*** folder, which is used by the library to store the target Modelica components, resulting from the transformation process.

Within the *src* folder, the software architecture is organized as follows:

1. The cim2modelica.cim package contains the JAVA classes implementing an API to handle the CIM/RDF [16] implementation from different CIM Profiles.
2. The *cim2modelica.cim.map* package contains the JAVA classes used to implement the class structure of the mapping rules. The transformation tool populates into this class structure the values from the source CIM model, according to the mapping rules (see Fig. 3).
3. The *cim2modelica.modelica* package contains a general class structure to create Modelica component instances of the OpenIPSL library i.e. *cim2modelica.modelica.MOClass*. Under the *cim2modelica.modelica.openipsl* there are specific classes with specific OpenIPSL component attributes i.e. *cim2modelica.modelica.openipsl.controls.es.OpenIPSLExcitationSystem*.

4. Software functionalities

The main algorithm follows a similar procedure as described in [17]: from a ***cim:Terminal*** class, other classes and relations are identified, but with differences in the input data and its processing. Our source CIM profile information contains the topology and the power flow results, in conjunction with the dynamic information of the components. Thus, the CIM model contains the information of the Equipment, Topology, State-Variable and Dynamic Profiles. The transformation tool builds the network component by pairs of ***Terminal–ConductingEquipment*** or ***Terminal–TopologicalNode***, to build the correspondent components, i.e. a *ConductingEquipment* of type *EnergyConsumer* is built as a *Load component*, or a *TopologicalNode* is built as a *Bus component*. In case of Power Transformers, we use the pairs ***PowerTransformerEnd–Terminal*** to build the corresponding *Transformer* component. A detailed functionality workflow of the M2M transformation tool is described in Fig. 2, as follows:

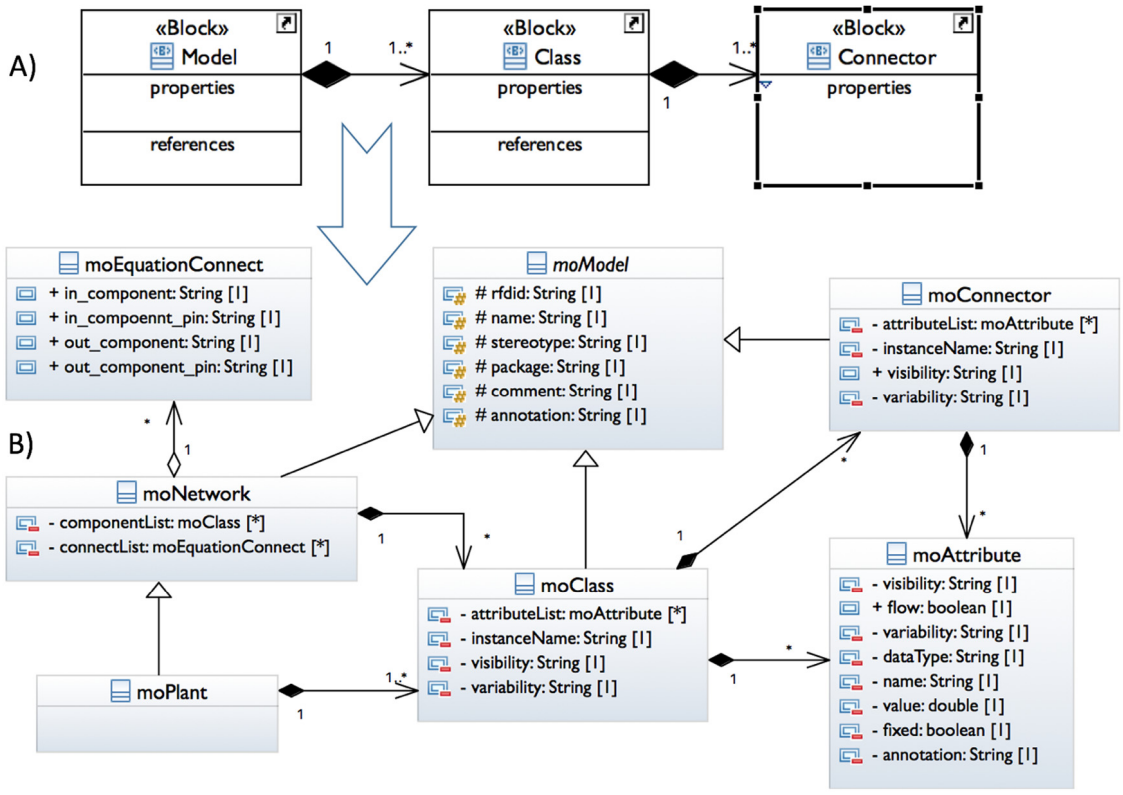

Fig. 4. (A) Shows the block diagram which relates the hierarchy of Modelica class stereotypes and (B) Shows the resulting SysML Meta-Model class structure for the creation of OpenIPSL object instances, power plants and network representations.

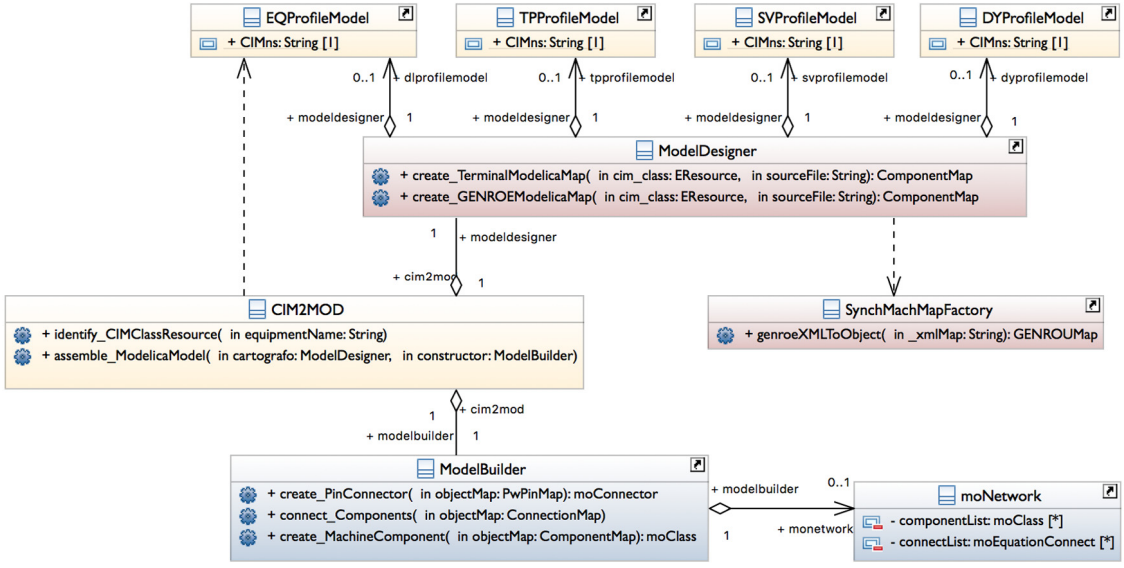

Fig. 5. Class structure representation of the main controller classes, which implement the main logic of the M2M transformation tool. The figure only represents a subset of methods for each class.

1. The actions *Load CIM Profiles* is responsible to load into memory the information from the CIM profiles.
2. To build the network, the actions *Identify CIM Te*, *Identify CIM CE* and *Identify CIM TN* are used to load the connectivity information represented by the EQ and TP Profiles.
3. The actions *Unmarshall Mapping Rules* and *Populate Mapping Objects* collect the methods that are used to load in memory the necessary CIM values. The values from the EQ, SV and DY profiles are defined within the mapping rules.
4. The actions *Instantiate Pin Object* and *Instantiate Component* collect the methods to create OpenIPSL component's instances with the CIM values that are stored into memory.
5. The actions *Update Network Model* and *Update Plant Model* collect the methods to create the resulting network representation with the connections between its components. The methods from the action *Updated Plant Model* are conceived to make the distinction from regulating equipment and machines from the rest of the components.

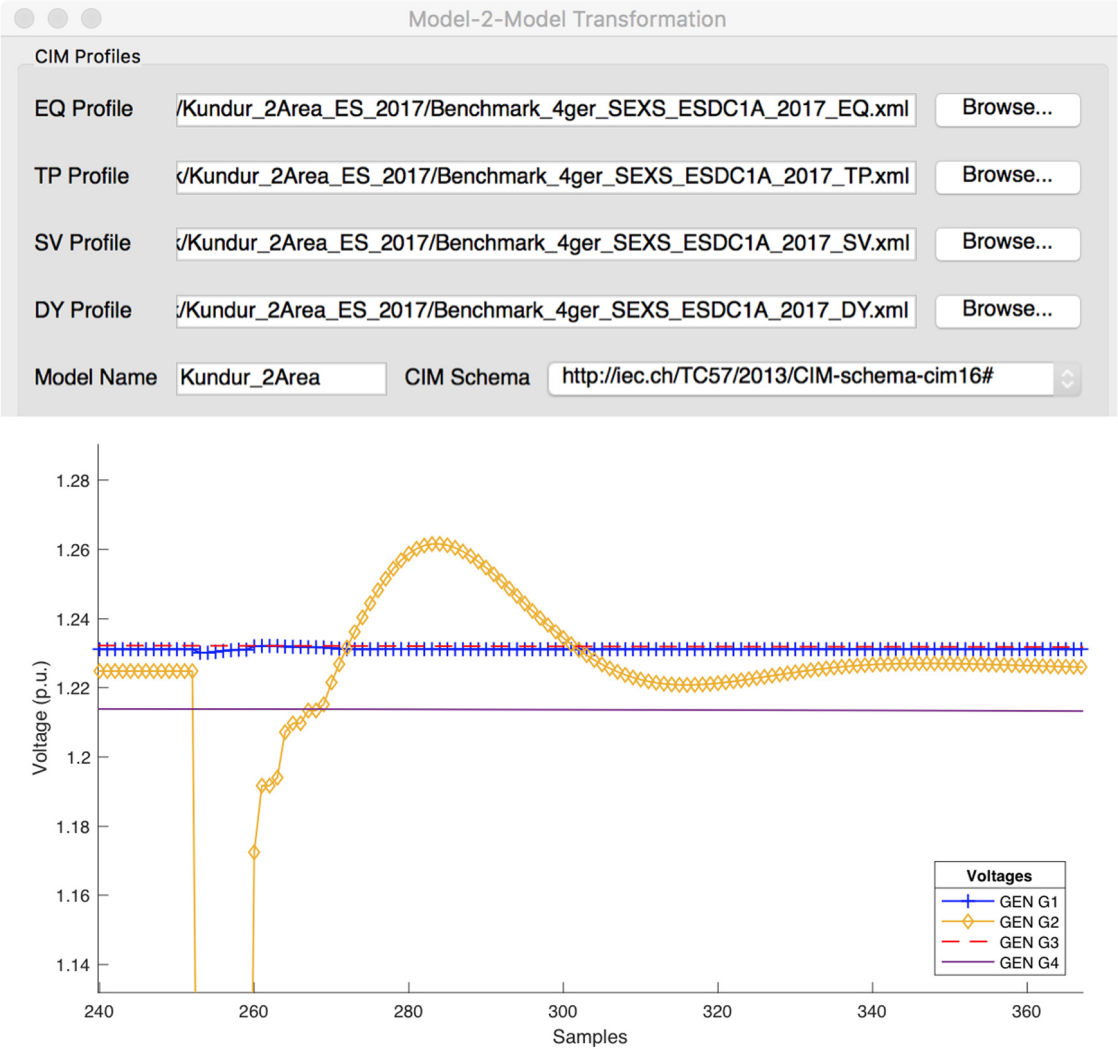

Fig. 6. GUI for M2M transformation (on the top) and simulation results from a Modelica Compiler (on the bottom).

6. The *Connect Components* action creates the Modelica connect equations to form the final network topology.
7. Outside the main loop, the actions *Mapping Meta-Model* and *SysML Meta-Model* define the class structures' implementation of CIM-OpenIPSL mapping rules and OpenIPSL class components that are required to enhance the transformation tool [18,19].

5. Software implementation

The transformation tool is implemented with different class structures that implement the methods of the functionalities described in Fig. 2.[1]

1. The *Mapping Meta-Model* functionality is designed to create the XML mapping rules and JAVA class structure. This class structure is obtained applying the JAXB technology, which provides and API to convert XML schemas into JAVA classes, and vice versa [20]. An example of the resulting class structure is shown in Fig. 3. From the XML mapping rules, classes such as *GENROUMap* are automatically generated with the JAXB library. *ComponentMap, AttributeMap* and *ConnectionMap* classes are the implementation result for the *Mapping Meta-Model* functionality.
2. The *SysML Meta-Model* functionality is conceived to design and implement a class structure based on the block diagram definition for Modelica objects stereotypes: ***model, class*** and ***connector***. Fig. 4 shows this class structure, used to create a network representation with the connector and components instances and their attributes.

On top of the class hierarchy, Fig. 5 describes the design of the control classes and their relationship with some of the internal classes, either to handle the CIM information and to handle the classes to create the Modelica code.

3 The *ModelDesigner* class acts as a controller object to *unmarshall* the mapping rules and populate the values from the source model into the *Mapping Meta-Model* structure. It aggregates different control classes that implement an API for reading CIM values from the available CIM profile files. The *ModelDesigner* class uses factory objects, such as the *SynchMachMapFactory*, to instantiate the adequate JAVA class from the corresponding mapping rule loaded.

4 The *ModelBuilder* class acts as a controller object to create the instances of the *SysML Meta-Model* structure. It uses the mapping classes containing the source model values as

[1] Further details on the development of the transformation tool are given within the developer's guide which can be found in the tool repository's docs folder.

input parameters, to create the component instances and connections forming the target network model.

5 The *CIM2MOD* class acts as the main controller object. This class implements the main logic of the transformation tool. It uses the API for reading the EQ CIM profile, to navigate through the Terminals and related *ConductingEquipment* and *TopologicalNode* RDF resources and load their values, located within the other profiles.

6. Illustrative example

To illustrate the usage and results of the application of the transformation tool, the *Klein–Rogers–Kundur Two-Area power network* [21] is used. The transformation tool receives as inputs the source CIM information divided into different CIM profiles that contains the information from the network model and generates as output an equivalent target Modelica components. The outputs result in a file system with the main model file, which contains the auto-generated Modelica model, and additional files that contain the Modelica implementation of power plant representations. The case in Fig. 6 shows an auto generated Two-Area Modelica network,[2] simulated in OpenModelica with a Runge–Kutta solver, with a fix time-step of 0.02. This scenario represents the behavior of the network with the presence of a fault event at the Bus connected to generator 2, with an impedance of 0.11j Ω, at $t = 5$ during 0.1 s.

7. Conclusions

This work shows the software architecture organization and main implementation of the M2M transformation, tool, based on the CIM and Modelica languages for the information exchange and simulation of CIM-compliant power system dynamic models. The proposed implementation follows a modular and scalable design and implementation for mapping rules and network generation that can be extended to support other modeling languages. The proposed implementation considers JAVA as the main programming language, and use of libraries to process XML and CIM/RDF files. Moreover, the implementation is open to any Object-Oriented Programming language, with support to the mentioned semantics.

The design of the mapping was first proposed in [18], and then was further explained with a proof of concept of methodology for a model-to-model transformation workflow described in [19]. With this implementation, instances of OpenIPSL components, with proper power flow solutions and equipment values from a CIM model, can be automatically generated and connected to form a Modelica network model, suitable for dynamic simulation studies. The instantiation process complies with the proof of concept presented in the Annex F of the new CGMES 2.5 standard guide lines [22]. It corresponds to the IEC 61970-600 definition, which proposes the use of Modelica to exchange user-defined models.

In future work, the internal package distribution of the software architecture will be described in detail, as well as the main features that were crucial in the implementation herein. Such information would allow to extend the tool or to re-implement it to take advantage of other programming or modeling languages.

The M2M is in constant maintenance of the Github repository that hosts the code. Better implementations of the main algorithm for a general identification of CIM classes is under investigation and development.

[2] Further details on the use of the transformation tool are given within the user's guide which can be found in the tool repository's docs folder.

Acknowledgments

The support of the following funding bodies is acknowledged: the EU funded FP7 project iTesla and the EU funded ITEA3 project OpenCPS.

References

[1] Ivanov Ch, Saxton T, Waight J, Monti M, Robinson G. Prescription for interoperability: Power system challenges and requirements for interoperable solutions. IEEE Power Energy Mag 2016;14(1):30–9. http://dx.doi.org/10.1109/MPE.2015.2485798.

[2] Uslar M, Specht M, Rohjans S, Trefke J, Gonzalez JM. The common information model CIM: IEC 61970, 61968 and 62325. Springer Heidelberg; 2012.

[3] Fritzson P. Principles of object-oriented modeling and simulation with Modelica. Wiley-IEEE Press; 2003.

[4] Vanfretti L, Rabuzin T, Baudette M, Murad M. Itesla power systems library (iPSL): A Modelica library for phasor time-domain simulations. SoftwareX 2016. http://dx.doi.org/10.1016/j.softx.2016.05.001, [On-Line],

[5] León G, Halat M, Sabaté M, Heyberger JB, Gómez FJ, Vanfretti L. Aspects of power system modeling, initialization and simulation using the Modelica language. In: PowerTech, 2015 IEEE Eindhoven, Eindhoven; 2015, p. 1-6. http://dx.doi.org/10.1109/PTC.2015.7232504.

[6] Vanfretti L, Adib Murad MA, Gómez FJ, León G, Machado S, Heyberger JB, Petriteneud S. Towards automated power system model transformation for multi-TSO phasor time domain simulations using Modelica. In: IEEE PES innovative smart grid technologies Europe, Ljubljana; 2016.

[7] Pantelides C. The consistent initialization of differential-algebraic systems. SIAM J Sci Stat Comput 1988;9(2):213–31. http://dx.doi.org/10.1137/0909014.

[8] Python implementation of the Common Information Model. PyCIM [On-line]. https://pypi.python.org/pypi/PyCIM/15134.

[9] McMorran AW, Ault GW, Morgan C, Elders IM, Mcdonald JR. A common information model (CIM) toolkit framework implemented in Java. IEEE Trans Power Syst 2006;21(1):194–201.

[10] Ravikumar G, Khaparde SA. A common information model oriented graph database framework for power systems. IEEE Trans Power Syst 2017;32(4):2560–9. http://dx.doi.org/10.1109/TPWRS.2016.2631242.

[11] Brambilla M, Cabot J, Wimmer M. Model-driven software engineering (MDSE) in practice. USA: Morgan & Claypool; 2012, p. 9–11,

[12] eXtensible Markup Language (XML) [On-line]: https://www.w3org/XML/.

[13] Jouault Frédéric, Allilaire Freddy, Bézivin Jean, Kurtev Ivan. ATL: A model transformation tool. Sci Comput Program 2008;72(1–2):31–9. http://dx.doi.org/10.1016/j.scico.2007.08.002,

[14] Winkler Dietmar. Electrical power system modelling in Modelica – Comparing open-source library options. In: Proceedings of the 58th SIMS September 25th - 27th, Reykjavik, Iceland. http//dx.doi.org/10.3384/ecp17138263. Online: http://www.ep.liu.se/ecp/138/035/ecp17138035pdf.

[15] Baudette Maxime, Castro Marcelo, Rabuzin Tin, Lavenius Jan, Bogodorova Tetiana, Vanfretti Luigi. OpenIPSL: Open-instance power system library — Update 15 to iTesla power systems library (iPSL): A Modelica library for phasor time-domain simulations. SoftwareX 2018;7:34–6. http://dx.doi.org/10.1016/j.softx.2018.01.002,

[16] Resource description framework for semantic web. 2014. [On-line]. Available https://www.w3org/RDF/.

[17] Shukla S, Yao Mark G. An efficient method of extracting network information from CIM asset model. In: 7th IEEE innovative smart grid technologies conference; 2016. http://dx.doi.org/10.1109/ISGT.2016.7781190.

[18] Gómez FJ, Vanfretti L, Olsen SH. Binding CIM and Modelica for consistent power system dynamic model exchange and simulation. In: 2015 IEEE power & energy society general meeting, Denver, CO; 2015. http://dx.doi.org/10.1109/PESGM.2015.7286434.

[19] Gómez FJ, Vanfretti L, Olsen SH. CIM-compliant power system dynamic model-to-model transformation and Modelica simulation, IEEE Transactions on Industrial Informatics, PP (99) 1-1. http://dx.doi.org/10.1109/TII.2017.2785439.

[20] JAVA Architecture for XML Binding (JAXB) [On-line]. http://www.oracle.com/technetwork/articles/javase/index-140168.html.

[21] Kundur P. Power system stability and control. New York: Mc-GrawHill; 1994.

[22] ENTSO-E CIM Inter-Operability Tests, 2015. [On-line]. Available: https://www.entsoe.eu/major-projects/common-information-model-cim/interoperability-tests/Pages/default.aspx.