RESEARCH Open Access

# CIMverter—a template-based flexibly extensible open-source converter from CIM to Modelica

Lukas Razik\* D, Jan Dinkelbach, Markus Mirz and Antonello Monti

From The 7th DACH+ Conference on Energy Informatics Oldenburg, Germany. 11-12 October 2018

\*Correspondence: Irazik@eonerc.rwth-aachen.de; Iukas@razik.de Institute for Automation of Complex Power Systems at E.ON Energy Research Center of RWTH Aachen University, Mathieustr. 10, 52074 Aachen, Germany

#### **Abstract**

Over the last decade, the Common Information Model (CIM), as specified by IEC 61970 / 61968, has become an important document format for the storage of power grid data. Its object-oriented design makes it easily maintainable and extensible for many use cases in the energy sector. As a result, more and more power grid analysis and simulation tools allow the import and export of CIM based power grid data. Unfortunately, many of them are proprietary and therefore not convenient in the research area since their component models and numerical back-ends often cannot be modified by the user. Thus, open-source alternatives are in demand, such as simulation environments based on the popular modeling language Modelica. Therefore, this paper presents our approach of a template based CIM to Modelica converter. The usage of templates makes it easily adaptable for the generation of Modelica system models targeting arbitrary Modelica libraries. The presented approach is implemented in an open-source project called CIMverter, evaluated on a real-world case with two Modelica power system libraries, and validated against a proprietary simulation tool.

**Keywords:** CIM, Modelica, Smart power grids, Energy, Simulations

#### Introduction

The Renewable Energy Directive requires the EU member states to fulfill at least 20% of their total energy needs with a wide range of renewable energy technologies by 2020 (European Commission 2018). More renewables often require smart grid operation with an increasing application of ICT systems exchanging information among one another. The Common Information Model (CIM) was developed to meet the requirements on a well-defined and extensible data format, nowadays used by many entities in the energy sector, such as the European Network of Transmission System Operators for Electricity (ENTSO-E). CIM was standardized in IEC 61970 (Energy Management), IEC 61968 (Distribution Management), and IEC 62325 (Energy Market Communications) (IEC 2012a; 2012b; 2014) as part of the IEC/TR 62357 reference architecture (IEC 2016). Among others, these CIM standards define an ontology, i.e. classes of virtual objects, representing real-world objects, and their relations (Uslar et al. 2012). Most important for this paper is IEC 61970 which defines



things needed for the representation of power grids (i.e. the *topologies* with equipment, lines, etc.), used by grid operators as a software tool independent data format.

Because of the widespread use of CIM based grid topology interchange, power system simulation and analysis tools such as NEPLAN can handle CIM. The problem of such proprietary simulation solutions in the academic research often is an insufficient or unavailable possibility for component model as well as solver modifications. As a consequence many open-source and free power system simulation tools have been developed during recent years as, for instance, MATPOWER which is compatible to the proprietary MATLAB as well as the open-source GNU Octave environment (Zimmerman et al. 2011), and its Python port PYPOWER. Other solutions are programmed in the object- and component-oriented multi-domain modeling language Modelica (Fritzson 2015). Since this allows a declarative definition of the model equations, the Modelica user resp. programmer does not need to transform mathematical models into imperative code (i.e. assignments). Modelica simulations can be executed with the aid of proprietary environments such as *Dymola* and open-source ones such as *OpenModelica* (Fritzson et al. 2006) and *JModelica* (Åkesson et al. 2010) with various numerical backends. Modelica libraries with models for power system simulations are PowerSystems (Franke and Wiesmann 2014) and ModPowerSystems (Mirz et al. 2016). The use of Modelica for power system simulations is not limited to the academia but it is also applied in real operation, especially with CIM based grid topologies as shown in (Viruez et al. 2017). However, in the presented approach an intermediate data format, called IIDM, is used.

The main contribution of this paper is the presentation of our open-source tool called CIMverter that transforms CIM documents into Modelica system models based on arbitrary Modelica libraries, as specified by the user. This allows for executing any kind of Modelica simulations which shall make use of informations stored in CIM documents. To achieve this, CIMverter utilizes a template engine that processes template files written in Modelica, containing placeholders. These placeholders are filled by the template engine with data from the CIM document and combined to a complete system model that can be simulated in a Modelica environment. The use of a template engine leads to encapsulation, clarity, division of labor, component reuse, single point-of-change, interchangable views, and so forth, as stated in (Parr 2004). For instance, this means that in case of many interface changes of a component model, the Modelica user does not need to modify the CIMverter source files but just the templates written in Modelica. Hence, there is no special knowledge of CIMverter's programming language or any Domain-Specific Language (DSL) needed. Furthermore, this paper presents examples on how CIM objects can be mapped to objects of a usual Modelica power system library. Our template based approach can also be used for conversions to formats other than Modelica.

This paper gives a short introduction to data formats as well as the main soft-ware components used in CIMverter followed by an overview of the overall concept. Then it describes how the mapping from CIM to Modelica is performed at top level and on bottom level with the usage of a C++ representation of the Modelica classes in the so-called *Modelica Workshop*. Following this, the approach and implementation is evaluated with the aid of two Modelica power system

libraries and validated with a commercial simulation tool. Finally, related work is discussed and the paper is concluded by a roundup and an outlook of future work.

#### **Fundamentals**

#### **CIM RDF/XML Documents**

The CIM specification is based on the *Unified Modeling Language* (UML), a formalism for graphical object-oriented modeling (Rumbaugh et al. 2004). With UML *class diagrams* and relations between them (e. g. *inheritance, associations, aggregations*, etc.) the CIM standards specify which kind of objects a CIM document can contain and how these objects are interlinked, which is called an *ontology*. The UML class diagrams contain attributes only, as there is no need for function definitions in case of CIM as information model specification. Machine readable CIM drafts are provided by the *CIM User Group* (CIMug) and a brief introduction to CIM is provided by (McMorran 2007). While UML is used for the definition of all CIM classes and their relations, the actual data (i. e. the CIM objects) in CIM documents is encapsulated through RDF/XML.

In fact, the widely-used *Extensible Markup Language* (XML) is a text-based formalism for arbitrary human- and machine readable documents (Bray et al. 1997). Unfortunately, the tree structure of XML documents does not allow the link of different XML elements, except for parent-child relationship. As a result, the *World Wide Web Consortium* (W3C) has proposed the *Resource Description Framework* (RDF) which can be used in combination with XML for the representation of arbitrary relations between a <subject> and an <object> by so-called RDF triples of the form "<subject> cobject>"(Pan 2009). In the triple "a car has an engine", the cpredicate> "has" would represent an aggregation between the <subject>, "a car", and the <object>, "an engine". This is the way how links (i. e. instances of aggregations, compositions, etc.), as specified by CIM UML, can be expressed. For the *descrialization* of CIM documents, the open-source *CIM*++ *Descrializer* is used. It reads the CIM document and outputs them in C++ as a list of CIM objects which are interlinked as defined by the RDF statements in the CIM document. More on CIM++ and CIM RDF/XML can be found in (Razik et al. 2018).

#### Modelica

The design of Modelica enables engineers to focus on the formulation of the physical model by the implementation of the underlying equations in a declarative manner (Fritzson 2015). The physical model can readily be implemented without the necessity to fix any causality through the definition of input and output variables, thus, increasing the flexibility and reusability of the models (Tiller 2001). Besides, existing Modelica environments relieve the engineer from the implementation of numerical methods to solve the specified equation system. The concept of component modeling by equations is shown exemplarily in Fig. 1 Listing 1 for a constant power load, which is typically employed to represent residential and industrial load characteristics in electrical grid simulations.

The presented PQLoad model is part of the ModPowerSystems (Mirz et al. 2016) library and is derived from the base model OnePortGrounded using the keyword extends, underlining that the Modelica language supports object-oriented modeling by inheritance. In the equation section, the physical behavior of the model is defined in a

```
model PQLoad "Constant power load"
    extends ModPowerSystems.Base.Interfaces.
        ComplexPhasor.SinglePhase.OnePortGrounded;
    parameter SI.ActivePower Pnom = 0.5e6 "active power";
    parameter SI.ReactivePower Qnom = 0.5e6 "reactive power";
    equation
        Pnom/3 = real(v*conj(i));
        Qnom/3 = imag(v*conj(i));
    end PQLoad;
Fig. 1 Listing 1: Component model of a constant power load
```

declarative manner by the common equations for active and reactive power. The parameters employed in the equations are declared in the PQLoad model beforehand, while the declarations of the complex variables voltage and current are inherited from the base model OnePortGrounded. A complex system, e.g. an entire electrical grid, can be implemented as *system model* by instantiating multiple components and specifying their interaction by means of *connection equations*, see line 25 in Fig. 10 Listing 6. The connect construct involves two connectors and introduces a fixed relation between their respective variables, e.g. between their voltages (equality coupling) and currents (sum-to-zero coupling).

Typically, Modelica environments provide a GUI for the graphical composition of models. After finishing the modeling process, the Modelica code is parsed, analysis and optimization steps are performed to obtain a minimum equation system, and then C code is generated, linked against a numerical library, and executed for the solution of the model (Fritzson 2015).

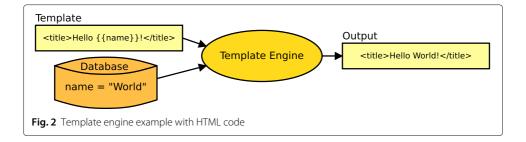
#### **Template Engine**

A *template engine* (also called *template processor* or *template system* and common in web site development) generates the Modelica code. Template engines allow the separation of *model* (i. e. logic and data) and *view* (i. e. resulting code). For CIMverter it shortly means that there is no Modelica code within the C++ source code of CIMverter. To achieve this, template engines have a

- data model: for instance based on a database, a text / binary file, or a container type of the template engine's programming language,
- template files (also called templates): written in the language of the resulting documents together with special template language statements, and
- result documents: which are generated after the processing of data and template files, so-called *expanding*,

as illustrated in Fig. 2, where an example HTML code template with a place holder  $\{\{name\}\}$  is filled with the name from a database, resulting in a complete HTML document. Such place holders are one type of *template markers*.

Two famous template engines which can be used in C++ are *Teng* and *Clear-Silver*. The template engine chosen for CIMverter is *CTemplate* (Silverstein). It is already used by the *CIM++ Unmarshalling Generator* (Razik et al. 2018). Besides, it makes no difference on the language in which the templates are written.



For CTemplate these are just strings containing template markers which can be i.a. of following types:

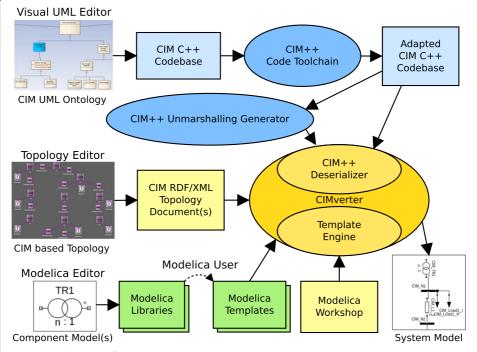
- Variable markers of the form { {VAR} } which are replaced by text from data dictionaries as described later.
- Start and End Section markers of the form { {#SECTION} }...{ { / SECTION} },
   enclosing sections which may appear zero to N times.
- Template include markers of the form { { >TEMPLATE\_INCLUDE } }, advising
   CTemplate to insert and expand another template (*subtemplate*) at the marker's location, which also can be repeated zero to N times in the output with different dictionaries.
- Comment markers of the form { {!Comment Your Code}}.

The data dictionaries provide mappings from *keys* (e.g. name) to *values* (e.g. World). Both, keys and values are strings and it is on the application using CTemplate to provide well formatted values to be inserted. Furthermore, on the one hand not all dictionary values must be used and on the other hand not all markers must exist in a dictionary during expanding. In such cases, the values resp. markers are being ignored by CTemplate. For more details on CTemplate, please refer to (Silverstein).

# Concept

The concept of CIMverter is depicted in Fig. 3. It uses and extends the concept of CIM++ as introduced in (Razik et al. 2018). The CIM UML ontology can be edited by a visual UML editor and exported to a CIM C++ codebase which is not compilable and therefore needs to be completed by the CIM++ code toolchain. The resulting adapted CIM C++ codebase, representing all CIM classes with their relations, is compilable and used by the CIM++ Unmarshalling Generator for the generation of code which is needed for the actual deserialization process of the CIM++ Deserializer. The CIM++ toolchain and the Unmarshalling Generator are applied in an automated way, whenever the ontology is changed. This keeps the CIM++ Deserializer compatible with newest CIM RDF/XML documents.

CIMverter uses the CIM++ Deserializer for deserialization of CIM objects from RDF/XML documents to C++ objects. Therefore, CIMverter also includes the adapted CIM C++ codebase, especially the headers for all CIM classes. Due to ongoing development of CIM and the concomitant automated modifications of these headers, one might suppose that the CIMverter development has to keep track of all CIM modifications but in the vast majority of cases a subsequent modification of CIMverter code is unneeded. This is because the continuous development of CIM mostly leads to new CIM classes with



**Fig. 3** Overall concept of the CIMverter project: The upper part shows the automated code generation process from the definition of the ontology by CIM UML to the unmarshalling code generation of the CIM++ Deserializer. The middle part shows the transformation process from a given topology (based on the specified CIM ontology) to a Modelica system model, based on Modelica libraries which are addressed by appropriate Modelica templates

further relations or new attributes in existing classes. Such extensions of existing CIM classes require no changes on CIMverter code using them.

With a Modelica Editor, the component models of Modelica libraries can be edited. In case the interface of a component model is changed, the appropriate Modelica template files have to be adapted by the CIMverter user who wants to generate the complete Modelica system models out of the given CIM topology documents. Thereby, using the template engine with the concomitant model-view separation leads to the following advantages:

- clarity: the templates are written in Modelica with only few kind of template keywords (i. e. markers).
- division of labor: the CIMverter user, typically a person with electrical engineering background and knowledge of Modelica, can adapt the Modelica templates easily in parallel with the CIMverter programmer reducing conflicts during their developments. While the engineer does neither need any C++ programming skills nor any knowledge of CIMverter internals, the programmer does not need to keep CIMverter up-to-date with all Modelica libraries that could be used with CIMverter.
- component reuse: for better readability, templates can include other templates, which can be reused for different component models of the same or further Modelica libraries.
- interchangable views: some Modelica models can be compiled with various options,
   e. g. for the use of different model equations, which can be defined directly in the

- code of the system model. For this purpose, the user can easily specify another set of templates.
- maintenance: changes to the Modelica code to be generated, which are needed e.g. due to changes of component model interfaces, can be achieved by editings of template files in a multitude of cases. Changing a template, by the way, is less riskier than changing a program which can lead to bugs. Furthermore, recompiling and reinstalling of CIMverter is unnecessary.

As already pointed out, some changes to the Modelica libraries require more than a template adaption which is related to the mapping of the deserialized CIM C++ objects to the dictionaries of the template engine used to complete the Modelica templates to full system models.

For a clear mapping between relevant data from the CIM C++ objects to the template dictionaries, the Modelica Workshop was introduced. For each Modelica component, the Workshop contains a C++ class with attributes holding the values to be inserted in the appropriate dictionary, which will be used for the Modelica code fragment expansion of the belonging component within the Modelica system model. The mapping from CIM C++ objects to these Modelica Workshop objects is defined by C++ code. An alternative would have been the introduction of a DSL for a more flexible mapping definition. However, a really flexible DSL would have to support data conversions and computations for data mappings from CIM to Modelica class instances. Despite tools for DSL specification and parser generation etc., the complexity of the CIMverter project would increase. Moreover, CIMverter users as well as the programmers would need to get familiar with the DSL. Both reasons would make CIMverter's maintenance and further development more sophisticated and therefore less attractive to potential developers. For instance, the co-simulation framework mosaik at the beginning also made use of a specially developed DSL for scenario definitions (Schütte 2011) but it was removed later on and now the scenarios are described by Python code, in which mosaik is implemented, as it is more flexible and powerful. The Modelica Workshop and other implementation design aspects, as described in the next sections, shall perform the C++ coded mappings in an intuitive and understandable way, making CIMverter therefore easily extensible by further Modelica component models and libraries.

# **Overall implementation**

As described in the Concept, CIMverter utilizes CIM++ for deserialization of the CIM topology documents (e. g. power grids) for the generation of full system models based on the chosen Modelica library (e. g. ModPowerSystems). C++ was selected as programming language because of the CIM++ Deserializer, with its including CIM C++ codebase, as well as CTemplate, both written in C++. As a static, strong type-checking language with less *Runtime Type Information* (RTTI) capabilities than a dynamic language such as e. g. Python, speculative dynamic typecasts are used for a return of the correct CIM C++ class object. Anyway, the times for the conversion of CIM to Modelica models in comparison to the compile time of the generated Modelica models is negligible. The usage of C++ also allows to look up CIM details in the Doxygen documentation generated from the adapted CIM C++ codebase of CIM++.

CIMverter has a *Command Line Interface* (CLI) and follows the UNIX philosophy of writing one program for one task (McIlroy et al. 1978; Raymond 2003). Therefore, it can be simply integrated into a chain of tasks which need to be performed between the creation of a CIM topology and the simulations by a Modelica environment as realized in the SINERGIEN Co-Simulation project (Mirz et al. 2018).

A configuration file is handled with the aid of the libconfig++ library, where i.a. the default graphical extent of each Modelica component can be adjusted. It also allows the definition of default CIM datatype multipliers (e.g. M for MW in case of IEC61970::Base::Domain::ActivePower) which are not defined in some CIM RDF/XML documents such as the ones from NEPLAN based on the ENTSO-E profile, specified by (ENTSO-E 2018). After these implementation details, in following subsections the main aspects of the overall implementation are presented.

#### Mapping from CIM to Modelica

The mapping from CIM documents to Modelica system models can be divided in three levels of consideration as in (Cao et al. 2015).

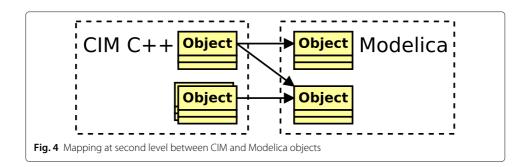
At first level, there are the library mappings. The relevant data from CIM C++ objects, as deserialized by CIM++, is first stored in an intermediate object representation (i. e. in the Modelica Workshop) with a class structure similar to the one of the Modelica library. Hence, for each Modelica library there can be a set of appropriate C++ class definitions in the Modelica Workshop.

Object mappings are part of the second level. There are not just one-to-one mappings, es illustrated in Fig. 4. Sometimes, several CIM objects are mapped to one Modelica object resp. component, such as the IEC61970::Base::Wires::PowerTransformer. There are also CIM objects like IEC61970::Base::Core::Terminal (electrical connection points, linked to other CIM objects) which are not mapped to any Modelica component models.

Parameters and unit conversions are performed at the third level between the CIM C++ objects and the Modelica Workshop objects. Examples are voltages, coordinates, and so forth. The next section faces the second and third level mappings as part of the Modelica Workshop but before, the CIM object handling is explained.

# **CIM Object Handler**

The CIMObjectHandler is in charge of the CIM objects handling. Figure 5 Listing 2 shows a part of its main routine ModelicaCodeGenerator. Topological nodes have a central role in CIM topologies of power grids. Therefore, finding a TopologicalNode



```
ctemplate::TemplateDictionarv *dict =
        new ctemplate::TemplateDictionary("MODELICA");
       for(BaseClass *Object : this->_CIMObjects) {
         if(auto *tp_node = dynamic_cast < TPNodePtr > (Object)) {
           BusBar busbar = this->TopologicalNodeHandler(tp_node, dict);
           std::list<TerminalPtr>::iterator terminal_it;
           for(terminal_it = tp_node->Terminal.begin();
               terminal_it != tp_node->Terminal.end(); ++terminal_it) {
             if(auto *power_trafo = dynamic_cast<PowerTrafoPtr>(
                            (*terminal_it)->ConductingEquipment)) {
               Transformer trafo =
                 PowerTransformerHandler(tp_node, (*terminal_it),
                                         power_trafo, dict);
               Connection conn(&busbar, &trafo);
               connectionQueue.push(conn);
Fig. 5 Listing 2: Snippet of the routine ModelicaCodeGenerator
```

(saved as tp\_node), a busbar object of the Modelica Workshop class BusBar is initialized with it. busbar is needed later on, for the connections of all kind of conducting equipment (i. e. power grid components) that is connected to it.

Then, the inner loop iterates over all terminals of the found tp\_node and checks which kind of ConductingEquipment is connected by the respective terminal to the tp\_node. In case of a PowerTransformer, a trafo object of the Modelica Workshop class Transformer is initialized with the data from the PowerTransformerHandler. Furthermore, a new connection between the previously created busbar and the trafo is constructed and pushed on a queue of all connections. These steps are performed for all other kinds of components, which is why the ModelicaCodeGenerator calls handlers for all of them.

The tp\_node with the terminal connected to the regarding component (here: trafo) are passed to the appropriate component handler (here: PowerTransformerHandler). Besides, the handler also gets the main template directory dict, called "MODELICA". Within a handler, the conversions from the required CIM C++ object(s) to the Modelica Workshop object trafo are performed. Furthermore, a subdirectory (here called "TRANSFORMER" used for the Transformer subtemplate, see e.g. Fig. 8 Listing 4) is created and linked to the given main template directory (see Fig. 7 Listing 3).

Some conversions are related to the graphical representation of the CIM objects. This is because a graphical power grid editor, which can export CIM documents, can link a  ${\tt IEC61970::Base::DiagramLayout::DiagramObject}$  to each component, with information about the position of this component, i. e. (x, y)-coordinates, in the coordinate system of the graphical editor. Since the coordinate system of the CIM exporting editor (e. g. NEPLAN) can differ from the one of the Modelica editor (e. g. OMEdit), the coordinates are converted by following code lines:

```
t_points.xPosition = trans_para[0]*x + trans_para[1];
t_points.yPosition = trans_para[2]*y + trans_para[3];
```

For reasons of flexibility, the four parameters trans\_para can be set in the configuration file and in case of NEPLAN and OMEdit are defined by |1,0,-1,0|. Furthermore, the NEPLAN generated CIM documents have several DiagramObject instances linked

to one component. To avoid multiple occurrences of the same component in the Modelica connections diagram, the middle point of these DiagramObject coordinates is calculated. This middle point then defines the component's position in the Modelica connections diagram.

Another conversion must be performed for the instance names of Modelica classes which are derived from the name attribute of the CIM object and may not begin or contain certain characters. Each such object derives its name attribute from the elementary IEC61970::Base::Core::IdentifiedObject superclass. More on the electrics related conversion details will be given in the next section.

### Modelica workshop implementation

In Fig. 5 Listing 2, different CIM object handlers (e.g. PowerTransformerHandler) return appropriate Modelica Workshop objects which represent components of the targeted Modelica library. It should be stated at this juncture that CIM is not only related to power grid components and, for instance, also includes energy market players (e.g. Customer), Asset, and so forth. Moreover, as presented in (Mirz et al. 2018), CIM also can be extended by further classes of different domains. Hence, the Modelica Workshop does not need to be reduced to power grid components, even though the current Modelica Workshop is related to components for power grid simulations. This is due to ModPowerSystems as first Modelica library targeted by the CIMverter converter. Nonetheless, the current Modelica Workshop can be used as is for the utilization of another Modelica library as presented in the Evaluation. To avoid reimplementations, each Modelica Workshop class representing a Modelica component, such as Slack or Transformer, inherits from the so-called ModBaseClass.

#### Base Class of the Modelica Workshop

All Modelica components need an annotation information which defines the visibility of the component, its extent, rotation, etc. Each Modelica Workshop class, inheriting from ModBaseClass, therefore has an annotation member holding the annotation data in a form as used in the Modelica component's annotation statement. For this purpose, ModBaseClass also holds several member functions which combine the annotation data to well structured strings as needed for the template dictionary used for filling the annotation statements of all Modelica template files as the annotation statements of all Modelica components have the same structure and the same markers (see lines 12-14 and 20-22 of Fig. 10 Listing 6).

For the Modelica statements which differ between different Modelica components (see lines 8-11 and 16-19 of Fig. 10 Listing 6) there exists a virtual function set\_template\_values. In each of the component subclasses this function will be overridden with a specialized one which sets all markers that are needed for a complete filling of the belonging Modelica component template, such as presented in Fig. 8 Listing 4.

Further member variables of ModBaseClass hold the name of the object and the specified units information, whose default values are set in the configuration file. The object's name is read from the name attribute of the CIM class IdentifiedObject. Besides, it accumulates objects of the CIM class DiagramObjects, where the objects rotation and points on the GUI coordinate systems are stored.

CIM	Contained / Accumulated	Modelica Workshop	
PowerTransformer	Member Variables	Transformer	
PowerTransformerEnd <sub>1</sub>	BaseVoltage->	Vnom1	
	nominalVoltage.value $*m_V$		
${\tt PowerTransformerEnd}_2$	BaseVoltage->	Vnom2	
	nominalVoltage.value * $m_V$		
${\tt PowerTransformerEnd}_1$	ratedS.value $\star m_P$	Sr	
${\tt PowerTransformerEnd}_1$	r.value	r	
${\tt PowerTransformerEnd}_1$	x.value	х	
$r \cdot \frac{Sr}{V_{nom1}^2} \cdot 100$ $\sqrt{r^2 + x^2} \cdot \frac{Sr}{V^2} \cdot 100$		Pcur	
$\sqrt{r^2 + x^2} \cdot \frac{\varsigma_r}{\iota^2} \cdot 100$		Vscr	

Table 1 CIM PowerTransformer to Modelica Workshop Transformer mapping

The left column shows the primary and secondary PowerTransformerEnd which accumulate further CIM objects, as listed in the middle column, holding the information needed for the initialization of the Transformer attributes as listed in the right column. The constants  $m_V$  and  $m_P$  stand for the voltage and power value multipliers. The bottom of the table shows that additionally two conversions are needed to calculate the rated short circuit voltage  $V_{SCI}$  and the short circuit losses  $P_{CUI}$  in percent

# **CIM to Modelica Object Mapping**

One of the most interesting mappings is from the CIM PowerTransformer to the Modelica Workshop Transformer class, as presented in Table 1. The PowerTransformer consists of two or more coupled windings and therefore accumulates objects of the class PowerTransformerEnd which represent the connectors of the PowerTransformer (FEIN Aachen e.V. 2018). Further important mappings implemented in the Modelica Workshop are listed in Table 2.

# **Component Connections**

After the instantiations of all components in the Modelica system model, the connections must be defined as well. In Fig. 5 Listing 2 for each newly created component a connection (i. e. instance of Connection class) to the corresponding busbar is created. Therefore, a function template of Connection with the signature

```
template<typename T> void cal middle points(T *component);
```

is called in the constructors of Connection and computes one or two middle points between the endpoints of the connection line. The four different cases for the middle points are illustrated in Fig. 6.

Furthermore, the connectors of the different components can vary between different Modelica libraries. Therefore, the connector names can be configured in a separate configuration file, called connectors.cfg, which is included in the directory of the belonging Modelica template files. Its settings are read by all Connection constructors,

**Table 2** Excerpt of further important mappings from CIM to ModPowerSystems as implemented in the Modelica Workshop

CIM ModPov		
TopologicalNode ExternalNetwoorkInjection	ModPowerSystems Slack	
ACLineSegment	PiLine	
TopologicalNode EnergyConsumer SvPowerFlow	PQLoad	



**Fig. 6** Connections with zero, one, and two middle points between the endpoints. The endpoints are marked with circles

combined, and fed into the dictionary which is used for filling the connections subtemplate, included by the main template file. The final Modelica code generation will be exemplarily presented in the next section.

#### **Evaluation**

For evaluation of the approach and implementation, we show exemplary templates as well as the resulting Modelica models. To demonstrate the flexibility and applicability of CIMverter, we use two different power system libraries, the ModPowerSystems and the PowerSystems library. Besides, we validate the simulation results obtained with the generated models against the commercial simulation tool NEPLAN.

The main Modelica template defines the overall structure of the Modelica system model and contains markers for component instantiations and connection equations, Fig. 7 Listing 3. The inserted subtemplates hold information regarding the library and package from which the models are taken, e.g. see line 1 in the corresponding subtemplates, Fig. 8 Listing 4 (for ModPowerSystems) and Fig. 9 Listing 5 (for PowerSystems), of the Transformer model. As use case, we generate the components for a steady-state simulation of a symmetrical power system in balanced operation. For the ModPowerSystems library, we utilize models from the PhasorSinglePhase package, since complex phasor variables and a single phase representation are functional for this type of simulation. In case of the PowerSystems library, we perform the simulation with models from the AC3ph package, obtaining comparable results by considering the dq0 transform in the synchronously rotating reference frame. Other types of simulation might be performed

```
{{#HEADER_FOOTER_SECTION}}model {{GRID_NAME}}
       {{/HEADER_FOOTER_SECTION}}
       {{#SYSTEM_SETTINGS_SECTION}}
       inner ModPowerSystems.Base.System
         {\{NAME\}\}(freq\_nom(displayUnit = "\{\{FNOM\_UNIT\}\}") = \{\{FNOM\}\})}
       annotation(Placement(visible = {{VISIBLE}}),
         transformation(extent = {{TRANS_EXTENT_POINTS}},
         rotation = {{ROTATION}})));
       {{/SYSTEM_SETTINGS_SECTION}}
       {{>PQLOAD}}
       {{>TRANSFORMER}}
       equation
       {{>CONNECTIONS}}
       {{#HEADER_FOOTER_SECTION}}
       end {{GRID_NAME}}; {{/HEADER_FOOTER_SECTION}}
Fig. 7 Listing 3: Main Modelica template related to ModPowerSystems, including several sections (e.g.
SYSTEM SETTINGS) and subtemplates (e.g. PQLOAD)
```

by changing package and model names accordingly in the subtemplates. The considered Transformer subtemplates, Fig. 8 Listing 4 and Fig. 9 Listing 5, contain markers to define primary and secondary nominal voltage as well as rated apparent power. The interface of the ModPowerSystems component specifies the Transformer's electrical characteristics by rated short circuit voltage  $V_{sc,r}$  and short circuit losses  $P_{cu,r}$ , while resistance R and reactance X are defined for the PowerSystems component.

In our use case, we model the benchmark system described in (Rudion et al. 2006), which is a medium-voltage distribution network with rural character. Integrated components are a slack bus, busbars, transformers, Pi lines, and PQ loads. An extract of the resulting Modelica system model generated from the CIM data with the presented CIM converter shows Fig. 10 Listing 6. The system model of the benchmark grid was additionally generated for the use of the PowerSystems library, simply by switching from the ModPowerSystems to the PowerSystems template set. The connection diagrams of the resulting models, Fig. 11, show the same grid topology involving the respective components from both libraries. For the validation of both Modelica system models, they were built and simulated. Afterwards, the simulation results were compared with the ones of the proprietary simulation tool NEPLAN, Table 3.

# Related work and discussion

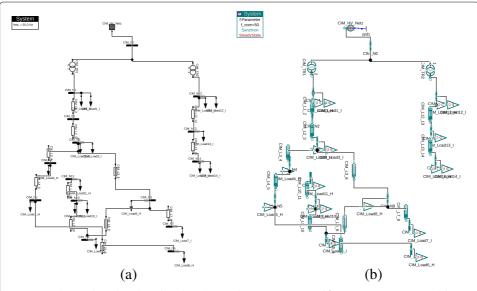
CIM itself as well as the conversion from CIM to different formats for power systems simulations is subject to scientific research for about 15 years (McMorran et al. 2004; Popovic et al. 2007). Since CIM is a flexible but comparatively complex format suitable for huge data sets, there is ongoing research on processing and storage of CIM based data such as grid topologies (Pavković et al. 2017; Ravikumar and Khaparde 2015; Requardt et al. 2017).

Only in the last few years, efforts have been made to power system related modeling in Modelica (Franke and Wiesmann 2014; Casella et al. 2016). Moreover, converters from various document formats to Modelica were developed in the past. In (Cao et al. 2015), for instance, a flexible model transformation from *Building Information Models* (BIM) to

```
model modpowersystems_mv_benchmark_grid
2
      inner ModPowerSystems.Base.System
       System(freq_nom(displayUnit = "Hz") = 50.0)
4
      annotation(Placement(visible = true,
       transformation(extent = \{\{0.0, -30.0\}, \{30.0, 0.0\}\},
5
6
        rotation = 0)));
8
     {\tt ModPowerSystems.PhasorSinglePhase.Loads.PQLoad}
9
        CIM_Load12_H(Pnom(displayUnit = "W") = 15000000.000,
        Qnom(displayUnit = "var") = 3000000.000,
10
        Vnom(displayUnit = "V") = 20000.000)
11
12
     annotation (Placement(visible = true,
        transformation(extent = \{\{-8.0, -8.0\}, \{8.0, 8.0\}\},
13
        rotation = 0, origin = {237.1,-107.8})));
14
15
16
     {\tt ModPowerSystems.PhasorSinglePhase.Transformers.Transformer}
17
        CIM_TR1(Vnom1 = 110000.000, Vnom2 = 20000.000,
        Sr(displayUnit = "W") = 40000000.000,
18
        Pcur = 0.63000, Vscr = 12.04000)
19
20
      annotation (Placement(visible = true,
21
        transformation(extent = \{\{-8.0, -8.0\}, \{8.0, 8.0\}\},
22
        rotation = -90, origin = \{86.0, -64.3\}));
23
24
   equation
25
     connect(CIM_NO.Pin1,CIM_TR1.Pin1)
      annotation(Line(points={{153.80,-40.00},{153.80,-56.15},
26
27
                               {86.00,-56.15},{86.00,-72.30}},
28
        color = {0,0,0}, smooth = Smooth.None));
29
   end modpowersystems_mv_benchmark_grid;
```

**Fig. 10** Listing 6: Medium-voltage benchmark grid (Rudion et al. 2006) as converted from CIM to a system model based on the ModPowerSystems library

different Modelica Libraries is presented. Actually, the approach handles SimModel documents which are a kind of BIMs and based on pure XML (without RDF) which allows the direct use of an XML data binding framework for XML data unmarshalling. The mapping rules are specified in an XML format and the Modelica code generation is hard coded in Python scripts.



**Fig. 11** Medium-voltage benchmark grid (Rudion et al. 2006) as converted from CIM to a system model in Modelica based on the **a** ModPowerSystems and **b** PowerSystems library

**Table 3** Excerpt from the numerical results for node phase-to-phase voltage magnitude and angle regarding the medium-voltage benchmark grid

Grid	Grid NEPLAN		ModPowerSystems		PowerSystems	
Node	V  [kV]	∠V [°]	V  [kV]	∠V [°]	V  [kV]	∠V [°]
$\overline{N_0}$	110.000	0.000	110.000	0.000	110.000	0.000
$N_1$	19.531	-4.300	19.532	-4.268	19.532	-4.268
N <sub>10</sub>	18.828	-4.900	18.828	-4.852	18.828	-4.852
$N_{11}$	18.825	-4.900	18.826	-4.852	18.826	-4.852

The models based on the ModPowerSystems and PowerSystems libraries yield equal results using the *Dymola* environment and *dassl* solver. The results deviate mariginally from the reference results obtained with the proprietary tool NEPLAN, which might be explained by numerical rounding and different solution methods

Recently, the software project *cim2modelica* was published (Gómez et al. 2018). It allows for defining direct mappings from CIM object attributes to the ones in Modelica by a DSL as it is not possible in CIMverter. The disadvantages of this approach are: a programmer as well as the Modelica user have to make familiar with the DSL and it allows no data conversions, as it can be completely flexibly performed in CIMverter by C++ code within the Modelica Workshop. Currently, cim2modelica supports the OpenIPSL Modelica library only and it was not proofed if the mapping approach can be used on any other existing Modelica library or for more complex mappings such as in case of the PowerTransformer. Besides, it does not process any diagram related data which is why no proper connection diagrams of the outputted system model can be printed by the Modelica environment.

### **Conclusion and outlook**

This paper presents an approach for the transformation from CIM to Modelica. The mapping of CIM RDF/XML documents to Modelica system models is based on a CIM to C++ deserializer, a Modelica Workshop representing the Modelica classes in C++, and a template engine. CIMverter, the implementation of this approach, is flexible enough to address arbitrary Modelica libraries as presented by the generation of system models for two power system libraries. In case of ModPowerSystems, there is no need of modifying the mappings as implemented in the CIM object handlers while switching to the Power-Systems library. Also, the Modelica Workshop classes are compatible with both libraries. Subsequently, the generated system models simulated with a Modelica environment are successfully validated against a common power systems simulation tool. CIMverter has already been successfully applied in the research area of power grid simulations as, for instance, in (Dinkelbach et al. 2018).

It is obvious that the current implementation can also be used for conversions into other formats than Modelica even with the current Modelica Workshop as the introduced template markers can be used in every file format. Therefore, the Modelica Workshop could be cleaned up and extended to a general *Power Systems Workshop*, addressing data formats used by other power system analysis and simulation tools.

Additionally, the current middle point calculations for the Modelica connections diagrams could be improved by the usage of a graph layout library such as *Graphviz* (Ellson et al. 2001). This would allow CIMverter to equip the outputted document with proper diagram data even if the CIM topology to be converted contains no diagram data at all.

# **Availability and requirements**

• **Project name:** CIMverter

Project home page: http://fein-aachen.org
Operating system: platform independent

• Programming language: C++

• **Other requirements:** CIM++, CTemplate, libconfig++, etc.

• License: GNU General Public License v3.0

• Any restrictions to use by non-academics: see GPL v3.0

#### **Abbreviations**

BIM: Building information models; CIM: Common information model; CIMug: CIM user group; CLI: Command line interface; DSL: Domain-specific language; EA: Enterprise architect; ENTSO-E: European network of transmission system operators for electricity; RDF: Resource description framework; RTE: Round-trip engineering; RTTI: Runtime type information; UML: Unified modeling language; W3C: World wide web consortium; XMI: XML metadata interchange; XML: Extensible markup language

#### Acknowledgements

We thank Taojun Li, Maximilian Kißgen, and Achim Völker for the co-development of CIMverter.

#### Funding

This research was funded by the German Federal Ministry of Education and Research (BMBF) within the SINERGIEN and ENSURE research projects under funding codes 03EK3567B and 03SFK1CO. Publication costs for this article were sponsored by the Smart Energy Showcases – Digital Agenda for the Energy Transition (SINTEG) programme.

#### Availability of data and materials

The datasets used and /or analysed during the current study are available from the corresponding author on reasonable request.

# **About this Supplement**

This article has been published as part of *Energy Informatics* Volume 1 Supplement 1, 2018: Proceedings of the 7th DACH+ Conference on Energy Informatics. The full contents of the supplement are available online at https://energyinformatics.springeropen.com/articles/supplements/volume-1-supplement-1.

#### Authors' contributions

LR leads the development of CIMverter and was the major contributor in writing the manuscript. JD is a developer of CIMverter as well as ModPowerSystems and performed the power system simulations for the Evaluation. MM assisted in the development of CIMverter and is the founder of ModPowerSystems. AM is the leader of the aforementioned research projects at the Institute for Automation of Complex Power Systems. All authors read and approved the final manuscript.

#### Authors' information

Lukas Razik received his diploma (German equivalent of a master's degree) in Computer Science from RWTH Aachen University, Germany in 2013. After graduation he worked as research associate at the Institute for Software and Tools for Computational Engineering and the Institute for Experimental Molecular Imaging, both at RWTH Aachen University, in the field of computational differentiation methods. Since 2015 he is with the Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University, where he is a Ph. D. student. His current research interests focus on high-performance and cloud computing approaches in the area of energy management and distribution systems as well as power grid simulations.

Jan Dinkelbach received his M. Sc. degree in Electrical Engineering, Information Technology and Computer Engineering from RWTH Aachen University, Germany, in 2016. Since 2017 he pursues his Ph. D. at the Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University. His research focuses on modeling of electrical grid components and the development of power system simulators.

Markus Mirz received his M. Sc. degree in electrical power engineering from RWTH Aachen University, Aachen, Germany, in 2014. Since 2015 he is research associate and Ph.D. student at the Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University. His current research interests are modeling and simulation of power system components as well as co-simulation.

Antonello Monti received his master's degree (summa cum laude) and his Ph. D. in Electrical Engineering from Politecnico di Milano, Italy in 1989 and 1994 respectively. He started his career in Ansaldo Industria and then moved in 1995 to Politecnico di Milano as Assistant Professor. In 2000 he joined the Department of Electrical Engineering of the University of South Carolina, USA as Associate and then Full Professor. Since 2008 he is the director of the Institute for Automation of Complex Power System within the E.ON Energy Research Center at RWTH Aachen University. Dr. Monti is author or co-author of more than 300 peer-reviewed papers published in international Journals and in the proceedings of international conferences. He is a Senior Member of IEEE, Associate Editor of the IEEE System Journal, Associate Editor of IEEE Electrification Magazine, Member of the Editorial Board of the Elsevier Journal SEGAN and member of the founding board of the SpringerOpen Journal Energy Informatics. Dr. Monti is the recipient of the 2017 IEEE Innovation in Societal Infrastructure Award.

### **Competing interests**

The authors declare that they have no competing interests.

#### Publisher's Note

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

#### Published: 10 October 2018

#### References

- Åkesson J, Årzén KE, Gäfvert M, Bergdahl T, Tummescheit H (2010) Modeling and optimization with Optimica and JModelica. org—Languages and tools for solving large-scale dynamic optimization problems. Comput Chem Eng 34(11):1737–1749
- Bray T, Paoli J, Sperberg-McQueen CM, Maler E, Yergeau F (1997) Extensible Markup Language (XML). World Wide Web J 2(4):27–66
- Cao J, Wimmer R, Thorade M, Maile T, O'Donnel J, Rädler J, et al. (2015) A flexible model transformation to link BIM with different Modelica libraries for building energy performance simulation. In: Proceedings of the 14th IBPSA Conference
- Casella F, Bartolini A, Pasquini S, Bonuglia L (2016) Object-oriented modelling and simulation of large-scale electrical power systems using Modelica: A first feasibility study. In: Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE. IEEE. pp 6298–6304
- Dinkelbach J, Mirz M, Schlösser T, Monti A (2018) Hosting Capacity Improvement Unlocked by Control Strategies for Photovoltaic and Battery Storage Systems
- Ellson J, Gansner E, Koutsofios L, North SC, Woodhull G (2001) Graphviz—open source graph drawing tools. In: International Symposium on Graph Drawing. Springer. pp 483–484
- ENTSO-E (2018) COMMON INFORMATION MODEL (CIM) MODEL EXCHANGE PROFILE 1. Available from: https://docstore.entsoe.eu/Documents/CIM\_documents/Grid\_Model\_CIM/140610\_ENTSO-E\_CIM\_Profile\_v1\_UpdatelOP2013.pdf. Accessed 13 May 2018
- European Commission (2018) Renewable energy European Commission. Available from: https://ec.europa.eu/energy/en/topics/renewable-energy. Accessed 28 May 2018
- FEIN Aachen e.V. (2018) Doxygen generated webpages of CIM++ Adapted CIM\_SINERGIEN Codebase: PowerTransformer Class Reference. Available from: http://cim.fein-aachen.org/libcimpp/doc/IEC61970\_16v29a\_IEC61968\_12v08/classIEC61970\_1\_1Base\_1\_1Wires\_1\_1PowerTransformer.html. Accessed 31 May 2018
- Franke R, Wiesmann H (2014) Flexible modeling of electrical power systems—the Modelica PowerSystems library. In:
  Proceedings of the 10 th International Modelica Conference; March 10-12; 2014; Lund; Sweden. 096. Linköping
  University Electronic Press. pp 515–522
- Fritzson PA (2015) Principles of object oriented modeling and simulation with Modelica 3.3. 2nd ed. John Wiley & Sons Fritzson P, Aronsson P, Pop A, Lundvall H, Nystrom K, Saldamli L, et al. (2006) OpenModelica-A free open-source environment for system modeling, simulation, and teaching. In: Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control, 2006 IEEE. IEEE. pp 1588–1595
- Gómez FJ, Vanfretti L, Olsen SH (2018) CIM-compliant power system dynamic model-to-model transformation and modelica simulation. IEEE Trans Ind Inf 14(9):3989–3996
- IEC (2012a) IEC 61968-11:2013 Application integration at electric utilities System interfaces for distribution management Part 11: Common information model (CIM) extensions for distribution
- IEC (2012b) IEC 61970-301:2012 Energy management system application program interface (EMS-API) Part 301: Common Information Model (CIM) base
- IEC (2014) IEC 62325-301:2014 Framework for energy market communications Part 301: Common information model (CIM) extensions for markets
- IEC (2016) IEC/TR 62357-1:2016 Power systems management and associated information exchange Part 1: Reference architecture
- McIlroy MD, Pinson E, Tague B (1978) UNIX Time-Sharing System: Foreword. Bell Labs Tech J 57(6):1899–1904 McMorran AW (2007) An Introduction to IEC 61970-301 & 61968-11: The Common Information Model. Univ Strathclyde
- McMorran AW (2007) An Introduction to IEC 61970-301 & 61968-11: The Common Information Model. Univ Strathclyde 93:124
- McMorran AW, Ault GW, Elders IM, Foote CE, Burt GM, McDonald JR, Translating CIMXML (2004) power system data to a proprietary format for system simulation. IEEE Trans Power Syst 19(1):229–235
- Mirz M, Netze L, Monti A (2016) A multi-level approach to power system modelica models. In: Control and Modeling for Power Electronics (COMPEL), 2016 IEEE 17th Workshop on. IEEE. pp 1–7
- Mirz M, Razik L, Dinkelbach J, Tokel HA, Alirezaei G, Mathar R, et al. (2018) A Cosimulation Architecture for Power System, Communication, and Market in the Smart Grid. Hindawi Complexity
- Pan JZ (2009) Resource Description Framework. In: Handbook on Ontologies. Springer. pp 71–90
- Parr TJ (2004) Enforcing strict model-view separation in template engines. In: Proceedings of the 13th international conference on World Wide Web. ACM. pp 224–233
- Pavković V, Čapko D, Vukmirović S, Erdeljan A (2017) Modeling power system data using NoSQL database. In: Telecommunication Forum (TELFOR), 2017 25th. IEEE. pp 1–4
- Popovic DS, Varga E, Perlic Z (2007) Extension of the Common Information Model With a Catalog of Topologies. IEEE Trans Power Syst 22(2):770–777
- Ravikumar G, Khaparde S (2015) CIM oriented graph database for network topology processing and applications integration. In: Power Engineering Conference (UPEC), 2015 50th International Universities. IEEE. pp 1–7
- Raymond ES (2003) The art of Unix programming. Addison-Wesley Professional
- Razik L, Mirz M, Knibbe D, Lankes S, Monti A. (2018) Automated deserializer generation from CIM ontologies: CIM++—an easy-to-use and automated adaptable open-source library for object deserialization in C++ from documents based

on user-specified UML models following the Common Information Model (CIM) standards for the energy sector. Comput Sci Res Dev 33(1):93–103

Requardt B, Wende-von Berg S, Wagner T, Toebermann C, Braun M (2017) Modular system architecture for processing of CIM. In: International ETG Congress 2017; Proceedings of. VDE. pp 1–6

Rudion K, Orths A, Styczynski ZA, Strunz K (2006) Design of benchmark of medium voltage distribution network for investigation of DG integration. In: Power Engineering Society General Meeting, 2006. IEEE. IEEE. pp 6–pp

Rumbaugh J, Jacobson I, Booch G (2004) Unified Modeling Language Reference Manual, The (2Nd Edition). Pearson Higher Education

Schütte S (2011) A Domain-Specific Language For Simulation Composition. In: ECMS. pp 146–152

Silverstein C CTemplate System Documentation. Available from: https://htmlpreview.github.io/?https://github.com/OlafvdSpek/ctemplate/blob/master/doc/index.html. Accessed 31 May 2018

Tiller M (2001) Introduction to physical modeling with Modelica. Kluwer Academic Publishers, Boston

Uslar M, Specht M, Rohjans S, Trefke J, González JM (2012) The Common Information Model CIM: IEC 61968/61970 and 62325-A practical introduction to the CIM. Springer Science & Business Media

Viruez R, Machado S, Zamarreño LM, León G, Beaude F, Petitrenaud S, et al. (2017) A Modelica-based Tool for Power System Dynamic Simulations. In: Proceedings of the 12th International Modelica Conference, Prague, Czech Republic, May 15-17, 2017. 132. Linköping University Electronic Press. pp 235–239

Zimmerman RD, Murillo-Sánchez CE, Thomas RJ (2011) MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education. IEEE Trans Power Syst 26(1):12–19

# Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ▶ Immediate publication on acceptance
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ▶ springeropen.com