Virtual61850: A Model-Driven Tool to Support the Design and Validation of Virtualized Controllers in Power Industry

Nadine Kabbara, Timothe Grisot and Jerome Cantenot
EDF R&D Paris Saclay, France

Keywords: Model-Driven Engineering, Metamodels, Eclipse EMF, OCL, IEC 61850, Interoperability, Power Systems.

Abstract: Model driven engineering (MDE) has seen a rising interest by the power industry particularly for supporting application and standards developments. Modern power systems often still involve many manual, error-prone configuration works. Examples include specification developments and configuration of communicating controllers. Recently, new concepts such as virtualized controllers (deployed in virtual machines or containers) have emerged. However, such a concept still remains rather new for power system experts where integrating virtualized controllers into their existing engineered information systems is a non straight-forward task. This study thus proposes to tackle some of the engineering and integration problems faced by this new concept thanks to the benefits offered by MDE. Virtual61850 is an implementation of a model-driven tool for supporting the configuration of future virtualized controllers into the power industry's information systems. It supports basic industry requirements including platform independence, standardized legacy configuration languages (IEC 61850 standard), modularity, and integrated testing and validation. The application was benchmarked for scalable models creation, editing, and validations that are necessary for advanced industrial simulation and field deployments of virtualized controllers.

1 INTRODUCTION

The exploitation of formalized models for the design of software artifacts (e.g. source code, documentation, tests) is the core principle of Model-driven engineering (MDE), a discipline in computer science. The model-centric approach allows the specification and design of software applications at a high level of abstraction, which can then be automatically validated before any platform-specific code generation (Clark et al., 2015).

Every model, in the context of MDE, can be mapped to a higher 'type model' describing all the knowledge of that model (i.e., elements, transitions, connection types). The 'type model', otherwise known as 'meta-model', aims to create an elegant language for representing domain artifacts (abstract syntax). Each syntax is associated with formally defined meanings or semantics that represent the concepts of a domain (composing a Domain-Specific Language DSL) (Kent, 2002).

Meta-modeling is also particularly beneficial for developing precise specifications of domain standards (e.g. systems engineering, telecoms, process modeling) (Clark et al., 2015). It helps enhance the standard's comprehension and implementation by the involved parties. In addition, meta-models provide consistent documentation generation and facilitate the standards' development and maintenance. These benefits were particularly observed by (Andren et al., 2013) for the power industry domain as in the case of the IEC 61850 standard (*Communication networks and systems for power utility automation* (IEC, 2020)).

The motivation for adopting MDE principles in the power industry is driven by the increased interest in smarter electrical grid developments. The high reliance on complex information and communication technologies necessitated utilizing standardized semantics to maintain system-level interoperability of the power grid information systems (Andren et al., 2013).

Currently, there exists different power system standards covering communication, control, or automation fields. They focus on data interoperability in terms of i) data syntax, ii) semantics, iii) and protocols. However, existing tools that allow to configure and manipulate the specified data are inconsistent (Andrén et al., 2017). This thus constraints the standard's complete adoption and integration by the in-

Kabbara, N., Grisot, T. and Cantenot, J.
Virtual61850: A Model-Driven Tool to Support the Design and Validation of Virtualized Controllers in Power Industry.
DOI: 10.5220/0012360600003645

In *Proceedings of the 12th International Conference on Model-Based Software and Systems Engineering (MODELSWARD 2024)*, pages 157-168
ISBN: 978-989-758-682-8; ISSN: 2184-4348

volved parties. Moreover, the heterogeneous tools often rely on non-formalized data which makes adopting an automated approach to manage the development process more problematic.

A study from 2002 showed that over 22% of human configuration errors were observed in configured control devices that are crucial for protecting the power grid in case of faults (Lundqvist and Aabo, 2002). The errors primarily included missing or wrong data settings. A more recent study in 2020 by (Resch et al., 2020) showed that despite the technological advancement (and the existing often proprietary editing tools), the power industry is still highly reliant on many manual, time-consuming, and error-prone engineering efforts.

The tedious efforts are still especially noticed in the case of 1) specification developments, 2) configurations of communicating devices in substations, 3) and data exchanges at the level of the grid control center IT systems (Resch et al., 2020). Often, there is a lack of inherent validations to identify early problems when configuring local control devices. More specifically, validations can be acted upon at three distinct levels: 1) basic file format level, 2) basic data elements and naming level, and 3) semantic level for rich data interpretation (Marcadet and Lambert, 2016). For example, trivial tests such as the configured connections of human-machine interfaces and basic reachability tests still constitute an important sum of the total solution costs due to the dedicated manual resources needed (Resch et al., 2020).

Motivated by these issues, we propose to further enhance the engineering process for new power and control systems developments (based on the IEC 61850 standard) by adopting an MDE approach. The main contribution of the paper is the implementation of a novel model-driven tool for the development of virtualized power system control command applications that supports existing standard data models.

The rest of the paper is organized as follows: Section 2 presents the state of art of MDE in different industries. Section 3 specifically introduces the power industry domain and its data model needs. Section 4 explains the concept of virtualized intelligent devices. Section 5 introduces a high level functional description of the Virtual61850 tool. Section 6 presents the implementation details and general architecture of the tool. Section 7 details the evaluation of Virtual61850 by the creation and editing of instances based on real industrial substation configuration data models. Finally, section 8 concludes the study and its future works.

2 STATE OF THE ART

Many researchers in various industries have previously studied the use of model-driven concepts for improving application development efforts. The authors in (Ledeczi et al., 2003) developed a modeling framework based on a domain language for the simulation of embedded systems. The framework aims to unify embedded application development by abstracting the different tool formalisms and semantics. (Ledeczi et al., 2003) developed metamodels to capture the application design and dataflow (hardware; functionality; mappings) where nonfunctional requirements were added as model constraints using the Eclipse Object Constraint Language (OCL) (OCL, 2023).

Similarly, (Corradini et al., 2022) proposed a model-driven tool to reduce application development efforts for heterogeneous Internet of things (IoT) platforms. A graphical tool for the modeling and dynamic management of Docker (Docker, 2023) containers was developed by (Paraiso et al., 2016). The work thus proposes a model-driven solution to abstract the docker model and its execution environment based on the Open Cloud Computing Interface (OCCI) (OCCI-WG, 2023). Container topology is also validated with Eclipse OCL constraints.

As for the power system industry, adopting a model-driven approach based on the standardized electrical grid data formats was argued by (Brédillet et al., 2010) as a key enabler for the smart grid vision. Authors in (Neis et al., 2019) developed a survey on model-driven practices for power system control applications including the software design (source), code format (target), and their transformation process (source/target). (Neis et al., 2019) observed the dominance of power system-specific modeling techniques (e.g. control theory) and the need for a transparent, explicit methodology to transform specifications into executable functional software. Most work was interested in proposing a comprehensive framework that includes a static and functional specification, especially for engineering control and automation applications.

3 DIGITAL ELECTRICAL GRID DEVELOPMENTS AND INFORMATION MODEL NEEDS

The electrical grid is a cyber-physical system of systems with a hierarchy of control, monitoring, and

management. Its operation requires the streaming of large amounts of data spread across distributed sensing nodes (from large-scale power plants, distributed energy resources (DERs), remote terminal units, etc.) which must be accurately interpreted in real time. To perform a seamless exchange of critical electrical grid data, an equally complex information system is required. Currently, power system information systems are managed by utilities and grid operators that most often respect certain international standards.

However, correctly describing and configuring the necessary data (e.g., process, controller, primary asset data) by different tools becomes a complex task. Adopting a system-level and standards-based approach for power system developments is thought to reduce their overall lifecycle and operational costs by 20% compared to the traditional siloed approach (2019, 2019). To respond to these needs, the IEC introduced different standards semantically defined for power systems: 1) IEC 61850 for power automation, 2) Common Information Model (CIM) for power system operation and planning, and 3) IEC 62056 for metering part (Marcadet and Lambert, 2016).

These standards aim to provide a more efficient, reliable, and safer power system where information can be exchanged without lock-ins from proprietary definitions (CENELEC, 2012). An information model in the context of power systems is a representational set of real-world physical or logical components (e.g. physical primary equipment, logical functional applications, functional parametrizations, etc) that need to be communicated to other system parties (e.g. another device, human operator) in an easily comprehensible format. A library of domain-specific information models has been elaborated in each standard. The libraries include common semantic names used by power system actors with self-describing meta-data covering system topology, measurement location, etc.

The specified information can then be transmitted to multiple devices or hosts with no additional mapping of essential meta-data (e.g. value, unit, scale, etc.). These communicating devices are known per IEC 61850-5 (IEC, 2013) as intelligent electronic devices IEDs and can be defined as: *"a device incorporating one or more processors with the capability to execute application functions, store data locally in a memory, and exchange data with other IEDs (sources or sinks) over a digital link"*.

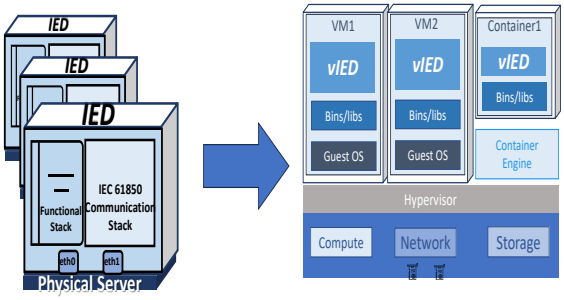

Figure 1: Example of possible vIED architecture based on virtual machines and containers.

4 VIRTUALIZATION OF IEDS

In the following subsections, the idea of IED virtualization and the motivation for using MDE in vIED developments are presented.

4.1 Context & Motivation

The traditional mode of deploying IEDs was on small industrial physical servers. However, more recently, the idea of virtualizing the IEDs and deploying them in virtual machines (VMs) or containers has been a rising research topic (seen in Figure 1) (Kabbara et al., 2022). The advantages of such new deployment lie in reducing the number of devices that need to be operated and managed and hence their capital and operational costs. Moreover, the well-established virtualization environments and tools for Information Technology IT systems can be adjusted and reutilized for virtual IEDs (vIEDs) management.

This includes remote accessibility and monitoring, clones and backups, as well as redundancy and recovery support. Also, the emulated environment of VMs and containers provides the required conditions to perform advanced simulations and tests of IEDs including communication, data models, functionality, etc. Despite these promising new features, the power industry is often hesitant to integrate novel technologies that can easily scale into their existing (rather already functional and critical) information systems.

Moreover, power system engineers often have limited knowledge of purely IT-based technology like virtualization. Their knowledge of power system communication and control also can't be replaced by IT experts on virtualization. This leads to more difficulties in coordinating the work of both parties as in the case of vIEDs developments. Another concern is the heterogeneity of the available tools that support virtualization (e.g. VMWare (VMware, 2023), KVM (RedHat, 2023), Docker (Docker, 2023), etc). The di-

versity of the different tools adds another layer of confusion for power system experts trying to test vIEDs where tool interoperability for large-scale information systems becomes non-evident.

We claim that, up to our knowledge, there is still no generic tool that supports the virtualization process and configuration of virtual IEDs. The main novelty of this study is in facilitating the development and adoption process of IED virtualization by developing a new tool based on MDE principles.

The tool characteristics are:

- Platform independence from any specific virtualization technology
- Supporting the existing IEC 61850 configuration language for IEDs
- Modularity to support cooperation between experts from different domains (Power systems, IT, communications)
- Enhancing the testing and simulation process of vIEDs by automating the generation of configuration files necessary for vIED testing.

We note that the purpose of this study was not to generate any application logic (low-level source code); it is outside the scope of the described process. Also, the initial focus was not on assuring or benchmarking any real-time performance of the vIED instances; the following study is limited to the data modeling specification level.

4.2 Relevant MDE Advantages for vIED Developments

The conformance of an instantiated model to a particular meta-model can be easily validated thanks to predefined syntax (e.g. parsing) and semantic constraints (based on domain rules). Another advantage provided by MDE is its separation between conceptual or platform-independent models and platform-specific models for implementations (code executables) (see Figure 2). This separation, formally specified by the Object Management Group (OMG) (Kleppe et al., 2003; OMG, 2023), helps to increase application portability and reduce development efforts and errors.

A seamless translation between platform-independent and platform-specific models (PIM/PSM) can be done by describing a set of transformational rules with pre/post conditions. The transformation target can itself be a model conforming to an external (or the source) meta-model in the case of a model-to-model transformation (M2M) (QVTo, 2023) or can be a generated code in the case of model-to-text transformation (M2T) (Czarnecki and Helsen, 2006). There exists an open-source community for MDE as part of the Eclipse foundation, which has provided the Eclipse Modelling Framework (EMF) Ecore (Eclipse-EMF, 2023) (implemented in the Java programming language) with several model transformation plugins (e.g. Acceleo for M2T (Acceleo, 2023) and QVTo for M2M).

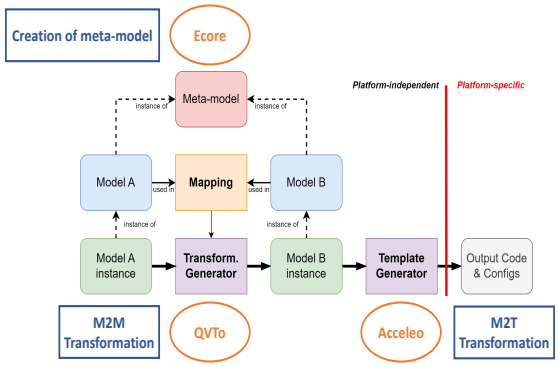

Figure 2: MDE principle and strengths.

5 APPLICATION FUNCTIONALITIES

The main idea was to implement a tool to support the design and validation of virtualized control devices for power grids. The objectives and criteria defined for Virtual61850 are described in Table 1. The overall toolchain can be seen in Figure 3.

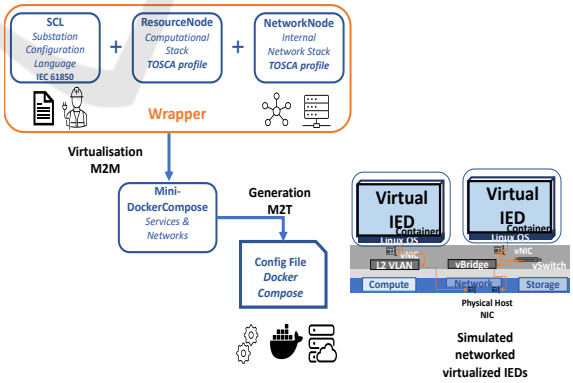

Figure 3: MDE tool chain for IED virtualization followed in Virtual61850 application.

5.1 Virtual61850 Models Integration

The application references various models that are essential to our work on IED virtualization, as listed below in Figure 4. Our specific workflow (the virtual-

Table 1: Objectives and criteria of the Virtual61850 application.

Item	Description
Objectives	
O1	Model navigation, editing support;
O2	Import/export existing specifications (i.e., IEC 61850) into/from a common root model;
O3	Model validations of syntax and semantic (business) constraints;
O4	Automated model transformations to generate platform-specific configurations;
O5	Support the simulation and testing of the generated configuration topology on the target platform;
O6	Reiteration to enhance the quality of the utilized models and their representative information.
Criteria	
C1	Ease of use with validated models;
C2	Modularity (partial) of the tool interface to support new models and transformations additions and modify existing ones;
C3	Scalability to support design and validation of large model instances;
C4	Conformity to standard models.

ization of IEDs) requires dealing with several models: SCL, ResourceNode, NetworkNode, and OpenAPI. Thus, the user needs to manipulate several instances related to different models.

- **SCL (Substation Configuration Language):** a model published as part of the IEC 61850 standard (IEC, 2013). it includes the functional description of the IEDs;
- **ResourceNode:** a model describing the computing resources available for the virtualization process;
- **NetworkNode:** a model allowing to define a virtual network stack between the various virtualized elements;
- **OpenAPI:** a model that describes an API and can be referenced from the ResourceNode, based on the OpenAPI standard (LinuxFoundation, 2023);
- **Wrapper:** serves as a root model that groups instances of SCL, ResourceNode, NetworkNode, and the OpenAPI models;
- **Mini-DockerCompose:** a model that is a simplified representation of the Docker Compose specification.

It is noted that both ResourceNode and NetworkNode models are solutions based on the existing

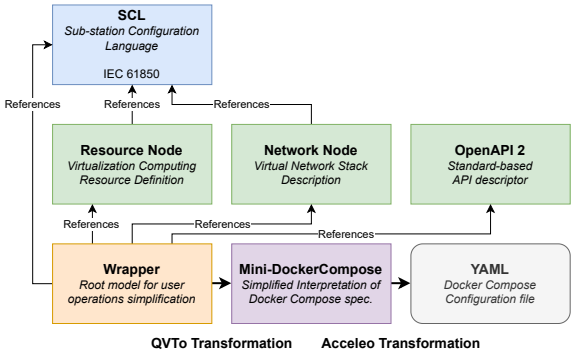

Figure 4: Models and transformations of the ”Virtualization” process.

ETSI NFV TOSCA standard (ETSI, 2023). The ETSI TOSCA models are specializations of the TOSCA standard (OASIS, 2023) for network function virtualization covering computing resource descriptors, internal and external connection points, links, and lifecycle management and policy metadata. However, we observed that the ETSI TOSCA model was too large and complex to manipulate for our own needs.

The resource descriptors and virtual networking elements were profiled and separated into individual smaller models (ResouceNode, NetworkNode). The profiling was done manually by comparing the IEC 61850 data model contents and filling the missing information that are critical for VM or container deployments. This separation helped us better understand the TOSCA models and allowed us to justify some simplifications, eliminations, and power domain-specific additions. For example, the elements of the ETSI TOSCA model covering life-cycle management and policy were omitted. Also, the separation of virtual links and connections points CP into internal and external ones was simplified to a single generic link or CP. The relationship types were not explicitly modelled but represented within the inherent model and meta-model relationships as per the unified modelling language.

To simplify the handling of different instances, it was decided to introduce a ”wrapper” model that groups all the models within the same node. Using a root model greatly simplifies references between elements of various instances. It avoids the need for inter-document references that are both hard to manage from a developer’s point of view, as well as the source of possible misunderstanding and issues for the user due to multiple file manipulations.

The first implementation targets the Docker platform using Docker Compose specification (DockerCompose, 2023). Docker-compose was chosen as a proof of concept implementation due to its ease of use and support for scalability. However, the modular-

ity developed within Virtual61850 allows it to target other virtualization platforms in the future. This contributes to the achievement of objective O5 of Table 1.

5.2 Editor Behaviour

The application is built following the principles of a classic editor. The user can load and save instances of the various models from and to the file system, as well as view and edit their contents. Model instances are typically stored as XML files. The XML format, although offers a robust structure, is difficult to manipulate by humans. Thus, it was decided that upon opening a file containing an instance of a model, the user would be shown a clean tree view that focuses on the instance's structure and makes it easily understandable.

Along that, when the user selects an element of the tree, the element's properties are displayed as a form in a separate view so that he can view and edit the attributes and references of the selected element. Figure 5 shows a typical state of the application interface as described above. These editor functionalities correspond to objective O1 of Table 1. It is also important to note that when editing a model instance through the editor, the syntax conformity to the supported domain standards is automatically ensured by construction, thus ensuring criteria C4 of Table 1.

5.3 Instances Import/Export Mechanism

Instances of the Wrapper model can become quite large as virtualization projects become more complex. Also, users might want to transfer only relevant information to some experts (i.e., network information to telecom expert). Thus, the application provides an import/export mechanism; it is possible to import and export instances of SCL, ResourceNode, NetworkNode, and OpenAPI models into and from an instance of the Wrapper model. This mechanism is related to objective O2 of Table 1.

5.4 OCL Validation

Model instance validation plays an important role in speeding up the development process as it reduces human errors at the design stage. Eclipse EMF makes it possible to define both business rules and semantic constraints directly in the model, which can then be checked for every instance of the corresponding model. These constraints are defined using OCL and a tool is provided by EMF for validating instances.

The application thus embarks on the OCL validation functionality, meaning the user can select an instance and validate it, that is, check all the business rules on every object of the instance. The tool also displays a report to the user so that he can obtain details on potential OCL rules violations, as well as export it to a text file if needed. An example of a validation report produced by the application is shown in Figure 5. This contributes to the achievement of objective O3 of Table 1.

OCL validation is directly integrated into the application. Each generated model code comes with a special Java class called a validator, containing methods for validating objects related to all classes of the model. When the user selects an object of an instance and validates it, this object is validated as well as all of its "children". The principle of the recursive strategy is based on a depth-first search approach (Goodrich, 2023).

```
//Make sure ports eth nbs are unique
self.ports->forAll(e1,e2 | e1<> e2
    implies e1.eth_number<> e2.
    eth_number)

//Max nb of ports per ConnectionNode
ports->size() < 12

//Only L2 or L3 Data values at the same
     time
self.l2_protocol_data->notEmpty()
    implies self.l3_protocol_data->
    isEmpty()
self.l3_protocol_data->notEmpty()
    implies self.l2_protocol_data->
    isEmpty()

//Only one address at the same time
address->notEmpty() implies goose_se->
    isEmpty() and smv->isEmpty()
goose_se->notEmpty() implies address->
    isEmpty() and smv->isEmpty()
```

Listing 1: Examples of OCL constraints on the Wrapper model.

6 IMPLEMENTATION

To implement the various functionalities described above, it has been decided to develop a heavy client (also called rich client). Eclipse EMF provides the Eclipse Rich Client Platform (RCP) framework, which fits to our needs. RCP also relies on a model-driven approach for building complete graphical interfaces.

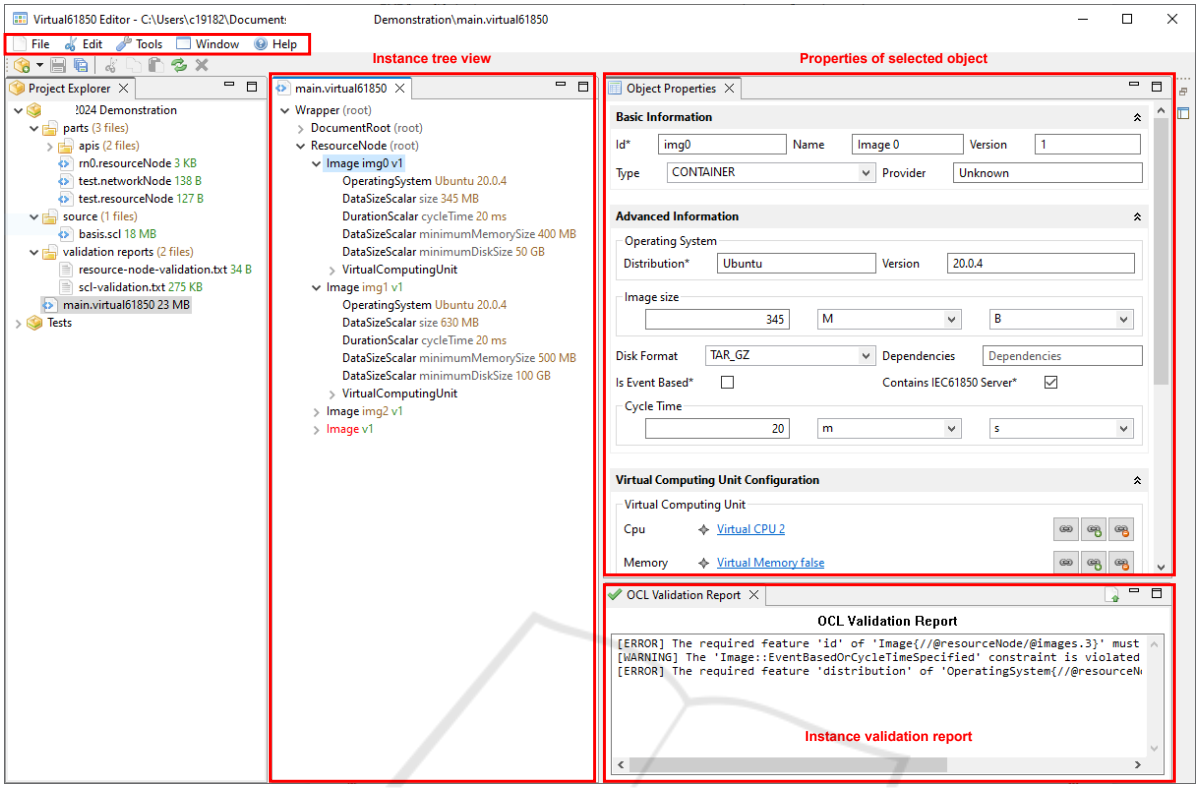

Figure 5: Interface of the Virtual61850 application.

6.1 General Architecture

The framework provides a context system coupled with dependency injection to enhance the ability to separate an application's components into various dedicated services. The use of such a framework helps speed up the development of our proof-of-concept application, and its native compatibility with the various Eclipse tools simplifies implementation.

Models are designed using Ecore, a meta-model that is the central component of EMF. Ecore is capable of generating model code for the Java language, which can then be used in our application to easily manipulate instances of various models. When generating model code, Ecore can also generate utility-related classes like the *Resource Factory* that make it possible to persist model instances to the file system in the form of XML files and load these instances back by parsing the XML content.

Our application needs to display information about the instances the user wants to edit, that is, some kind of view model or forms. Creating forms for editing an object's properties can be a tedious task especially when models contain dozens of classes, with each class potentially containing multiple attributes and references. To speed up this process, EMF provides the EMF Forms framework (Eclipse-EMF, 2023), which automatically generates view models from the model definition file (an instance of the Ecore meta-model) that can be customized using a graphical editor.

Generated view models are compatible with multiple graphical libraries and can be easily integrated into our Eclipse RCP application, as shown in the *Object Properties* part of Figure 5. The following paragraphs describe the implementation choices that allow partial modularity and extensibility of the application covering criteria C2 of Table 1.

6.2 Models and Transformations Integration

The application needs to handle all parts of the root Wrapper model. Internally, the application uses an enumeration to keep track of all managed models. This enumeration contains important fields related to the model, such as the human-readable name, and the root class of the model. Also, it contains the corresponding object validator (used for OCL validation) or the resource factory by generating or parsing the data instances.

This system is a first step towards a fully modular

model integration as described in criteria C2 in Table 1. This contributes to the achievement of objective O4 of Table 1. It is possible to add new supported models, but this must be done manually, that is, by adding an item to the corresponding enumeration. Any newly added model is automatically compatible with the editor, that is, the application becomes fully capable of opening instances of this model, validating instances (if OCL rules are defined), and executing transformations that are compatible with it.

6.2.1 M2M Transformation

QVTo is a model transformation language that is included in the EMF. It is used for specifying and executing model transformations, which consists of transforming an instance of a model into an instance of another model, which is a process that is very frequently used in the context of MDD. QVTo uses a high-level declarative syntax for expressing transformation rules and is based on the OMG QVT specification which ensures interoperability and consistency.

Every QVTo transformation starts by referencing the models that it needs to handle. Then, we define the signature of the transformation that is, its input and output and its main entry point in the main function. The mapping function for creating Docker services from a Wrapper model are seen in Listing 1.

Deployments are more complicated to determine because we need to navigate in the NetworkNode instance of the Wrapper in order to determine the virtual connection points that are associated with our ResourceNode image. For each connection point, we create a container for the service by using the toDockerContainer mapping function.

```
modeltype WRAPPER "strict" uses wrapper
    ('...');
modeltype MINI_DOCKER_COMPOSE "strict"
    uses miniDockerCompose('...');
modeltype RESOURCE_NODE "strict" uses
    resourceNode('...');
modeltype NETWORK_NODE "strict" uses
    networkNode('...');

mapping resourceNode::Image::
    toDockerService() : Service {
        result.name := self.name;
        result.version := "1";
        result.image := self->map
            toDockerImage()->selectOne(
            miniDockerCompose::Image);
        result.deployments := input.
            rootObjects()[Wrapper].
            networkNode.
            virtualConnectionPoints->
            select(vcp | vcp.
            deployedImage = self)->map
                toDockerContainer();
}
```

Listing 2: QVTo M2M transformation example from Wrapper resourceNode to miniDockerCompose models.

An instance of the SCL model contains the definition of the various IEDs installed in a substation. Each of these IEDs must be virtualized, consequently, we need to create a ResourceNode instance in which we define a VM or container image for each IED. A simple transformation that generates a pre-initialized ResourceNode instance was created. Along with creating instances of the Image class, it also pre-fills in some attributes to speed up development and help us support scalability.

6.2.2 M2T Transformation

Acceleo is a M2T transformation language tool that is used to generate textual artifacts such as source code, documentation or configuration files from models. Acceleo allows you to define templates and rules that specify how to transform model instances into text output.

Automatically generating Docker Compose YAML configuration files from an instance of the Mini-DockerCompose model supports an iterative development process, where models can be improved based on feedback and testing. Changes to the models can then be easily propagated by re-generating code, for example. The principle of the transformation mainly relies on generating the correct YAML tree for Docker Compose configuration files with all possible data found in the Compose instance. In case the Mini-DockerCompose instance contains at least one network, we write the networks' top-level element. Then, for each network, we add an entry to this element which corresponds to the name of the network. Eventually, we add another layer with the properties of the corresponding network. Similar code is used for defining Docker volumes and configs.

```
[template public generateElement(
    compose : Compose)]
[comment @main/]

[if (compose.networks->size() > 0)]
networks:
[for (network : Network | compose.
    networks)]
  [network.name/]:
  [for (property : Property | network.
      properties)]
    [property.key/]: [property.value/]
  [/for]
[/for]
```

```
[ / if ]
```

Listing 3: Acceleo M2T transformation example from miniDockerCompose models to .yml config file.

6.3 Internal Services Modularity

Various features of the application were implemented using a service system, that is, an implementation that is separated in a dedicated package. This is the case for functionalities such as OCL validation, QVTo and Acceleo transformations executors, instance persistence, and the embedded program launcher. Each service can be used in any part of the application by using the dependency injection mechanism combined with Eclipse context annotations. This contributes to the achievement of objective O6 of Table 1.

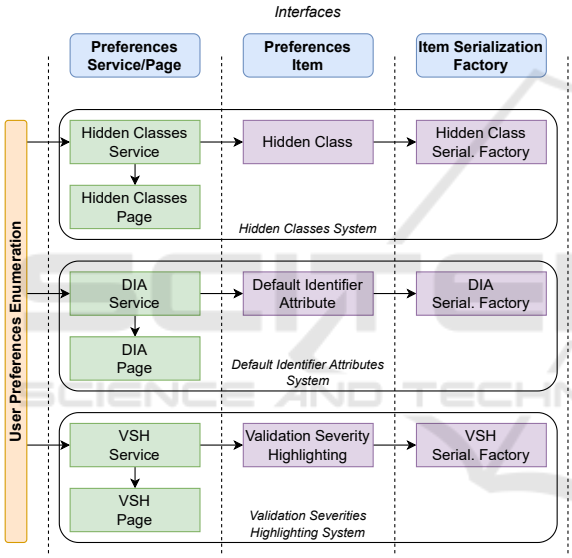

Figure 6: User Preferences system overview.

6.4 Extensible User Preferences System

The application provides a simple user preferences system so that the user can customize how the application behaves. As shown in Figure 6, three preferences have been implemented: 1) the possibility to hide certain classes in the instance tree view, 2) customizing which attributes are used to display a name in the instance tree view, and 3) how validation errors and warnings are highlighted.

The RCP framework provides a way to persist data across application launches called persisted state, which our implementation uses to store user preferences. However, the only way to store data is by using key-value pairs, both the key and the value being strings. We could have decided to implement our own user preferences persistence system, but using a mechanism that is directly integrated within Eclipse RCP was much simpler and less time-consuming.

In case one wants to store objects and not only primitive types, it is thus necessary to implement a serialization system. One important and useful aspect of the Eclipse preferences system is the notion of scope. The application uses two different scopes: the default scope which is useful for providing a default initial configuration, and the instance scope which contains the actual user preferences. This system mainly relies on one class and two interfaces that are described below.

- *Preferences Manager*: This class is in charge of providing a standard way to access, store, and delete key-value pairs in the preferences. As the Eclipse preferences system works using a node tree, each preferences type stores its data in a specific and exclusive node. The class implements three main methods: `get`, `put`, and `remove`. It is capable of accessing both the default and instance scope.
- *Preferences Item Serialization Factory*: This is an interface that is used to implement the serialization factory for each preferences item type. This is where the programmer implements the serialization mechanism. It defines three methods: `getInstance` (this is intended to be a singleton), `serialize` and `unserialize`.
- *Preferences Item*: This is an interface that is used to implement all preferences items. It only defines one method `serialize` that generally just calls the `serialize` method from the corresponding preferences item serialization factory.

6.5 Embedded Java Programs

Sometimes our users might need to use external tools. In the case of our ”Virtualization” process, users may need to use a Java program called ”Genconfig” provided as a `*.jar` file. This program is used to generate additional configuration files for the IEC 61850 communication of IEDs based on the SCL instances (.scd files) (MzAutomation, 2023). The `*.jar` file has been included in the application’s resources and is executable by the user from the interface.

The application source code contains an enumeration that holds items that reference each embedded Java program. It contains all the information needed to execute the program including its name and how to retrieve the parameters from the user.

The application integrates a service, the *Embedded Program Launcher*, that is responsible for

prompting the user for the various parameters required by the program (as defined in the enumeration item) and then launching the execution of the program using the provided parameters. A new separate process is created alongside the application, which can be interrupted at any time by the user. The service also reports back any errors that might occur during the program's execution.

7 PoC EVALUATION

The Virtual61850 tool was first utilized for basic testing purposes of different sets of instances. Virtual61850 source code builds (Linux and Windows), manuals, and test example public data have been anonymously archived on the public Zenodo page. We invite the reviewers to view and test our application by following the tutorial found on Zenodo [1].

7.1 Instance Example

The stability and ease of use of the application as per criteria C1 were observed. In general, we noticed a very stable performance for the Windows build version while the Linux build had some minor instability (originating from Eclipse for Linux).

Figure 7 shows an example from the start to end configuration file generated. The process starts by importing an existing SCD file describing the communication interfaces and data model of the IED as per IEC 61850. Both generic (non-real) SCD files and real scd files from a digital substation commissioning project were included. Each IED was mapped to an image with its appropriate resource and network descriptions within a Wrapper model.

The editor allows to perform a navigation of the model thanks to a tree view. We note that in real digital substations, a system topology with over 200 vIEDs can be needed. The time to create a proper instance is hence significantly reduced compared to manual manipulations. The M2M transformation maps each image to a service to be ran. Finally, an M2T transformation allows us to generate the platform-specific configuration, which in our case was a docker-compose YAML file, that can be used by the power system engineer for further testing the vIEDs' environment and their performance.

[1] anonym. (2023). Virtual61850 application. Zenodo. https://zenodo.org/records/10041282

7.2 Application Quantitative Analysis

In order to evaluate the performances of the developed application, we ran several benchmarks each related to a specific metric. Two major metrics were identified including instance size and validation time. These metrics can help us determine how well criteria C1 from Table 1 is considered.

Each metric presented in Tables 2 and 3 is the average of 10 samples. A sample is created by launching the application, opening a specific file, and eventually executing a transformation. All samples were produced on a machine with a 1.60 GHz Intel Core i5-8265U processor, 8 GB 1800 MHz DDR3 memory running on Windows 10 Enterprise version 10.0.19045 and OpenJDK 17.0.6.

7.2.1 Handling Large-size Instances

In the context of our motivating example, instance files can become as large as a few hundred megabytes in the case of large substation commissioning projects. Thus it is interesting to evaluate how our application behaves in these cases as per the requested criteria C3 in Table 1. Table 2 below describes the results we obtained in terms of memory consumption in various scenarios. For each scenario, the application is either in an idle state or with a single opened instance or file of the specified size.

These results show that as the manipulated instance grows in size, the memory consumption grows as well but in a more pronounced way (for instance, opening a 1 MB instance uses 34 MB). This growing factor seems to decrease as the instances become bigger: only 610 MB is used for a 50 MB instance, so a factor of 12. In the case of files, the memory consumption also increases although approximately half as much as for instances. Again, as files become bigger, the memory consumption factor becomes smaller.

From a user point of view, manipulating large-size instances should not be a real issue, as the memory consumption stays at an acceptable level when editing instances of a few dozen megabytes. However, it is argued that the benchmarking has only been conducted for instances with limited and restricted model sizes. Therefore, the current results are not yet generalizable to larger instances.

7.2.2 Validation Time

Another challenge when dealing with large-size instances is the duration of the OCL validation. As it is a critical feature of our application, it is important that it can stay usable when working with instances of several hundred megabytes. Table 3 displays the

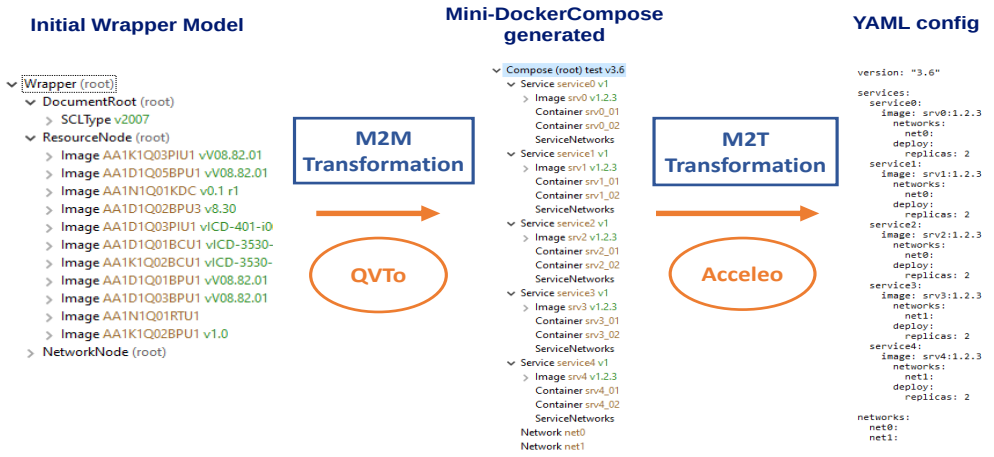

Figure 7: Instance example of Wrapper model transformed into miniDockerCompose through M2M and a .yaml config file with M2T by importing an existing SCD file.

Table 2: Memory usage in typical scenarios.

Scenario	Memory	/ Idle
Idle	225 MB	//
1 MB instance	259 MB	+ 34 MB
18 MB instance	583 MB	+ 358 MB
50 MB instance	835 MB	+ 610 MB
1 MB file	237 MB	+ 12 MB
10 MB file	299 MB	+ 74 MB
100 MB file	864 MB	+ 639 MB

execution times measured in various scenarios with instances of different sizes. In general, the validation times are highly dependant on the number and time of constraints under test. The results obtained were for the Wrapper model with increasing number of constraints as the size of instance increases. We note that the validation times are relatively acceptable for our proof of concept study, but merit further optimization to obtain a more rapid toolchain behaviour.

Table 3: OCL validation duration in typical scenarios.

Scenario	Validation Time
1 MB instance	0.40 s
10 MB instance	2.02 s
18 MB instance	4.01 s
50 MB instance	6.61 s

8 CONCLUSION

This paper presented how a Model-Driven Engineering approach could be beneficial for facilitating the adoption of virtualized controllers by the power industry. Virtual61850, a proposed tool for editing and configuring virtualized power controller instances was detailed and benchmarked based on some industrial requirements. These requirements include platform independence, legacy standards support, validation times and error reductions, as well as scalability and support of large instances.

A graphical interface used as a proof-of-concept tool editor for configuring virtualized controllers was developed. It is based on Eclipse EMF and integrates various models (61850 communication, resources, and networking descriptors, OpenAPI) as well as validation and transformation engines (based on OCL, QVTo, and Acceleo technologies). The application provides a set of functionalities that satisfy the various objectives regarding model editing, validation, and transformations (O1..O6) initially set, and has been designed to be easily extensible for future developments, following a modularity logic as specified by criteria C2.

The validation by design capabilities can prevent configuration errors from an early design stage of the development process. This is highly useful when integrating and testing new concepts as virtualized controllers within a well defined and iterative toolchain process. Also, the possibility of supporting scalable large instances was validated thanks to measuring both memory consumption from the application and validation time.

Future works include implementing a tool to synchronize between the model environment and the Docker Engine API so that the user can simulate the virtualized controllers directly from the application. Additional Model-to-Model transformations allowing users to virtualize controllers using virtual machines or other virtualization technologies can be added to adapt to multiple platforms. Also, the multi-threaded OCL validation mechanism can be reworked by im-

plementing a thread pool and load balancing system to precisely adjust the number of threads and address the unbalanced tree issue. Finally, it is possible to support a mechanism that could convert instances created using an old version of a model to instances compatible with the newest version of this model. This encourages even further the iterative development principle.

REFERENCES

2019, I. E. R. (2019). Standards pour les smartgrids itech edf r&d e. lambert.

Acceleo (2023). Acceleo. https://wiki.eclipse.org/Acceleo/Getting_Started.

Andren, F., Strasser, T., Rohjans, S., and Uslar, M. (2013). Analyzing the need for a common modeling language for Smart Grid applications. In *2013 11th IEEE International Conference on Industrial Informatics (INDIN)*, pages 440–446, Bochum, Germany. IEEE.

Andrén, F., Strasser, T., and Kastner, W. (2017). Engineering Smart Grids: Applying Model-Driven Development from Use Case Design to Deployment. *Energies*, 10:374.

Brédillet, P., Lambert, E., and Schultz, E. (2010). Cim, 61850, cosem standards used in a model driven integration approach to build the smart grid service oriented architecture.

CENELEC (2012). Smart grid coordination group smart grid reference architecture.

Clark, T., Sammut, P., and Willans, J. (2015). Applied Metamodelling: A Foundation for Language Driven Development (Third Edition). arXiv:1505.00149 [cs].

Corradini, F., Fedeli, A., Fornari, F., Polini, A., and Re, B. (2022). X-IoT: a model-driven approach for cross-platform IoT applications development. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, Virtual Event. ACM.

Czarnecki, K. and Helsen, S. (2006). Feature-based survey of model transformation approaches.

Docker (2023). Docker inc. https://www.docker.com/.

Docker-Compose (2023). Docker compose. https://docs.docker.com/compose/.

Eclipse-EMF (2023). Eclipse emf forms. https://eclipse.dev/ecp/emfforms/.

ETSI (2023). Etsi-nfv-sol001. https://forge.etsi.org/rep/nfv/SOL001.

Goodrich, M. T. (2023). Depth-first search. https://www.ics.uci.edu/~goodrich/teach/cs260P/notes/DFS.pdf.

IEC (2013). Communication networks and systems for power utility automation – part 5: Communication requirements for functions and device models.

IEC (2020). Communication networks and systems for power utility automation – part 7-1: Basic communication structure – principles and models. In *IEC TR 61850*.

Kabbara, N., Nait Belaid, M. O., Gibescu, M., Camargo, L. R., Cantenot, J., Coste, T., Audebert, V., and Morais, H. (2022). Towards software-defined protection, automation, and control in power systems: Concepts, state of the art, and future challenges. *Energies*.

Kent, S. (2002). Model driven engineering. In *International conference on integrated formal methods*, pages 286–298. Springer.

Kleppe, A., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*.

Ledeczi, A., Davis, J., Neema, S., and Agrawal, A. (2003). Modeling methodology for integrated simulation of embedded systems. *ACM Transactions on Modeling and Computer Simulation*.

LinuxFoundation (2023). Openapi. https://www.openapis.org/.

Lundqvist, B. and Aabo, Y. (2002). The cost benefit of modern Substation Automation in Electrical High Voltage Installations, abb automation.

Marcadet, D. and Lambert, E. (2016). Riseclipse: Why working at the model level is better for validating data conforming to iec standards. In *2016 Power Systems Computation Conference (PSCC)*.

MzAutomation (2023). lib61850. https://github.com/mz-automation/libiec61850.

Neis, P., Wehrmeister, M. A., and Mendes, M. F. (2019). Model Driven Software Engineering of Power Systems Applications: Literature Review and Trends. 7:177761–177773.

OASIS (2023). Oasis-tosca. http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html.

OCCI-WG (2023). Occi. https://2020.standict.eu/standards-watch/occi-12-open-cloud-computing-interface-%E2%80%93-infrastructure.

OCL (2023). Ocl. https://projects.eclipse.org/projects/modeling.mdt.ocl.

OMG (2023). Meta-modeling and the omg meta object facility (mof). https://www.omg.org/ocup-2/documents/Meta-ModelingAndtheMOF.pdf.

Paraiso, F., Challita, S., Al-Dhuraibi, Y., and Merle, P. (2016). Model-Driven Management of Docker Containers. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE.

QVTo (2023). Qvto. https://projects.eclipse.org/projects/modeling.mmt.qvt-oml.

RedHat (2023). Kvm. https://www.linux-kvm.org/page/Main_Page.

Resch, J., Schuiki, B., Schöndorfer, S., Brandauer, C., Panholzer, G., Pröstl Andrén, F., and Strasser, T. I. (2020). Engineering and validation support framework for power system automation and control applications. (8).

VMware (2023). Vmware. https://www.vmware.com/fr.html.