On Automatic Generation of IEC61850/IEC61499 Substation Automation Systems Enabled by Ontology

Chen-Wei Yang[1], Valeriy Vyatkin[1,2], Arash Mousavi[1], Victor Dubinin[3]
Department of Computer Science, Electrical and Space Engineering
[1]Lulea University of Technology, Lulea, Sweden
[2]Aalto University, Helsinki, Finland
[3]University of Penza, Penza, Russia

chen-wei.yang@ltu.se, vyatkin@ieee.org, arash.mousavi@ltu.se, victor_n_dubinin@yahoo.com

***Abstract*—This paper presents an introductory step in the automatic generation of distributed control software for power distribution automation systems based on Ontology Driven Engineering enabled by industrial standards IEC61850 and IEC61499. The novelty of this approach is the ability of automatically generating the logical connections between the logical nodes. The paper covers the several stages of the transformation process, such as developing the IEC61850 ontology and the ontology transformation rules. The developed IEC61850 ontology includes the logical nodes descriptions and additional contextual relations between the logical nodes which is lacking in the IEC61850 SCL configuration language. Then, the IEC61850 ontology is transformed to an existing IEC61499 ontology, adding classes of IEC61850 logical nodes as IEC61499 function blocks in the IEC61499 ontology. The means of the transformation of the ontologies is based on the eSWRL sematic web rules language, an extension to the rule language SWRL. The end result is the development of an IEC61850 ontology and a set of eSWRL rules which facilitates the ontology transformation.**

Keywords—component; IEC61850; IEC61499; Ontology; eSWRL

I. INTRODUCTION

The so called Smart Grid is the future electricity supply infrastructure which is expected to incorporate communication infrastructure and bi-directional power flow providing real-time information for the actors involved [1]. It is expected the usage of distributed renewable energy resources (Photovoltaic, hydro and wind, etc.) and Distributed Energy Storage Devices (Battery, Plug-hybrid electric vehicles) will increase, transforming the traditional centralized control architecture of the grid into a distributed one [2]. An example of one such system which reflects these changes is the FREEDM system [3]. The proposed FREEDM system allows the integration of DRER resources and introduces decentralized control for fault management and energy management.

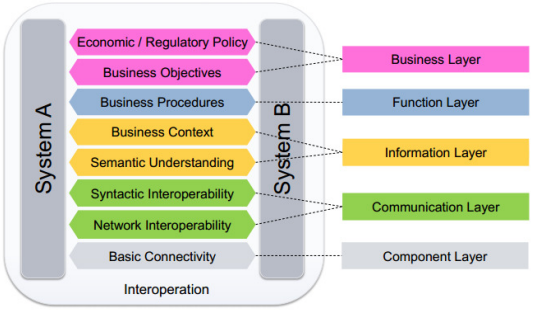

Fig. 1. Reference Smart Grid model from NIST [1]

In the Smart Grid, there will be many intelligent devices from different vendors interacting with one another. Therefore, interoperability is an important requirement to such intelligent devices. In the Smart Grid Reference Model (SGAM) [1] shown in Fig. 1, there are five main interoperability layers identified.

The main focus of this work is on the lower rung of the interoperability layers centered on harmonizing the communication and the component layers. The two standards which are suitable in the harmonization of these two interoperability layers are the substation automation standard IEC61850 and the distributed automation standard IEC61499. IEC61850 was first introduced as a standard for substation automation systems. The aim of the standards is standardize communication in substation automation systems (SAS) enabling interoperability between Intelligent Electronic Devices (IED) from different device vendors. Its advantage in providing a standardized means of communication between distributed IED devices will be greatly beneficial to future smart grid systems. While IEC61850 provides a comprehensive information model of substation automation components, the internal implementation of control elements or the control logic of IEC61850 logical nodes are out of the scope of the standard. However, another open standard, IEC61499, has been proposed in [4] to complement IEC61850 by implementing the internal control of the IEC61850 logical nodes. IEC61499 defines a reference architecture for distributed automation control systems, which comprises of such design artifacts as basic and composite function blocks

(FB). FBs are capable of capturing the IEC61850 hierarchical information model from the logical node level to the basic data level. In addition, it provides a native platform for designing distributed control logic for IEC61850 systems. There has been a number of works following [4-6] illustrating the benefits of the standards harmonization in various application scenarios. This work is a next step in the harmonization effort, aiming at automatic generation of IEC61850 and IEC61499 based control systems using Ontology Driven Engineering (ODE).

The paper is structured in the following manner. Chapter II will discuss the related works in IEC61850 and IEC61499. In addition, existing works in automatic code generation of IEC61850 systems are also discussed. Chapter III presents the novel idea of automatically generating IEC61850/61499 systems using ODE. Chapter IV provides a case study, which illustrates the initial transformation process of the proposed solution and the conclusion of the paper is in Chapter V.

II. RELATED WORKS

A. *IEC61850 and IEC61499*

The idea of combining IEC61850 and IEC61499 into a distributed substation automation solution and prototypes of such solution is discussed in [4-6] with a reference example using the function block development platform FBDK. In addition, a co-simulation environment was also proposed between IEC61499 and Matlab Simulink to provide an environment for validating the IEC61850/61499 control systems. The concept of Intelligent Logical Nodes (iLN) is the IEC61499 realizations of the IEC61850 logical nodes as function blocks. The benefits of the iLN concept is that in addition of fully capturing the hierarchical logical node information models in IEC61499 function blocks, the iLN concept also provides a platform where intelligence or logic could be added into the logical nodes providing the ability of decision making for each logical nodes. Thus creating a distributed system where each distributed node (Group of the intelligent logical nodes) is capable of collaborating and making decisions in a distributed manner for smart grid control and protection schemes.

There are existing research works in the substation automation domain which also highlights the viability of such solution. Article [7] illustrates an IEC61850 based protection scheme with GOOSE messaging implemented with IEC61499 and IEC61850 using the IEC61499 development platform nxtStudio. In [8], GOOSE messaging function blocks were developed in the function block development platform ISaGRAF to allow interoperable communication between IEC61850/61499 solutions and commercial IEC61850 Intelligent Electronic Devices (IED). Another solution presented in [9] uses open-source function block development platform 4DIAC and the Matlab Power System Analysis Toolbox (PSAT) to create a smart grid co-simulation environment. All these related works demonstrate the potential of the IEC61850/61499 synergy.

B. *Automatic Generation of IEC61850/ 61499 Systems*

There are existing research works which apply automatic generation to create IEC61850 based control systems [10, 11]. The source of generation is commonly the IEC61850 System Configuration Language (SCL). SCL is a XML based configuration language introduced to allow the exchange of system configuration between IEC61850 systems. There are in total six types of SCL configuration files and each configuration file type is used to configure a different part of the IEC61850 system. Examples of SCL files include the Configured IED Description (CID) for configuring individual IED devices and the System Configuration Description (SCD). The SCD file captures the complete configuration of an IEC61850 system, which includes logical node definitions, communication configurations and individual IED configurations. In most of the existing works where the automatic generation of IEC61850 system is concerned, the SCD file is used as the source of generation since it contains the entire configuration of an IEC61850 system. The sections of the SCD file, which are used for the generation, are the *DataTypeTemplates*, the *IED* section and the *communication* section. The *DataTypeTemplates* contains the definitions of the all logical nodes used in the IEC61850 system. This is essential as the definitions constitute the multi-layer information model of the logical node including the *DataObjects* and the *DataAttributes* of each unique logical node model. The *IED* section captures the configurations of each IED devices in the IEC61850 system. There can be more than one IED device within a system and there will be an *IED* section for each IED in the system. The *communication* section contains the configuration of the communication protocols such as Generic Object-Oriented Substation Event (GOOSE) messaging and Server/Client communication configurations. The communication section configures the entire communication network of the IEC61850 system defining the addressing of each IED devices (sub-networks and networks), the packaging of data for transmission (e.g. GOOSE Dataset) and the signal flow defining the data exchange between logical nodes. As mentioned previously, there are existing works, which automatically generate code for IEC61850 systems. However, there is very little existing work on generating IEC61499 based control systems based on IEC61850 models.

In [10], an open platform for rapid-prototyping protection and control schemes with IEC61850 is proposed. One part of the open platform is to automatically generate low-level communication configurations and the logical node information model. Lower-level communication includes GOOSE messaging (Configuration of GOOSE datasets) and Sampled-Value (SV) messaging. In [10], the SCD file is used as the source of the automatic generation process. The target code is generated in the C programming language. The information models are generated by parsing the SCD XML. The logical node definitions are extracted from SCD and each hierarchical level of the logical node is generated. Data type of each attribute, along with data object types and the logical node types are represented as hierarchical C data structure.

In [12], a tool chain for smart grid automation framework for the design of IEC61850/IEC61499 is proposed with automatic generation being the core of the framework. The tool chain, also known as SysGRID, takes the SCL XML as an input to automatically generates IEC61850/IEC61499 systems. The automatic generation process is ruled based by parsing the SCD XML to an equivalent IEC61499 XML. Hierarchical logical nodes' information models are generated as (composite) function block types called Intelligent Logical Nodes (iLN) [6].

A shortcoming of the SysGRID tool chain is the lack of ability to create meaningful logical connections between the logical node function block instances in the generated IEC61499 application. Logical connections between function blocks define the control and data flow between blocks and are essential for complete definition of the function block application's semantics. This work aims to take the automatic generation a step further to allow the logical connections between the logical node function blocks to be automatically connected.

III. NOVEL APPROACH

A. *Ontology as the basis of IEC61499 code generation*

There are existing works where ontology has been applied in the IEC61850 research domain. The majority use of ontology currently is for the harmonizing IEC61850 and the Common Information Models (CIM) which includes IEC61970 and IEC61968 as shown in the work [13]. However, ontology has not yet been applied in the application of automatic generation of IEC61850 substation automation systems. In the domain of IEC61499, ontology is already used for the purpose of automatic generation of IEC61499 control systems. Article [14] utilizes ontology to facilitate the transformation of IEC61499 systems from traditional IEC61131-3 PLC systems. In [15], ODE was used as the basis for the automatic generation of IEC61499 control codes for a Baggage Handling System.

B. *Contextual Relations Between Logical Nodes*

Even though IEC61850 provides a comprehensive library of semantic models known as logical nodes, there is a lack of concrete semantic information attached to the logical nodes. This was intentional on the part of the TC57 working group to ensure interoperability between the logical nodes. Since IEC61499 is a block diagram based language, logical connections between logical nodes showing flow of data is important in the design of IEC61499 control systems.

In the first edition of the IEC61850 standard, the data variables defined in each logical node could be classified as settings, control and output. The classification of variables as input variable is intentionally avoided to preserve interoperability from the communication point of view. From the communication's point of view, it is not necessary to standardize input variables. Interoperability can be achieved by standardizing variables, such as: settings, status and output variables. For example, between the logical nodes TCTR (Current Transformer Logical Nodes) and PIOC (Instaneous Overcurrent Logical Node), there is no clearly defined relationships between the two logical nodes. However, relations between the two logical nodes can be defined from a particular context. In the context of overcurrent protection, TCTR will have a direct logical connection to the PIOC logical nodes as the PIOC logical node requires the measurement readings from the current transformer logical node TCTR to determine whether an overcurrent fault has occurred. Therefore, simply having an ontological model of an IEC61850 configuration derived from the SCL configuration is not enough, contextual information which provides logical relations between logical nodes in IEC61850 based protection schemes is also necessary in order to infer the relations between the logical nodes. The advantage of using ontology as the system container is that it is able to capture semantic relationships between the IEC61850 components (I.E. the logical nodes) whereas XML is only able to capture the syntactal relationships between the IEC61850 components. In addition, the semantic correctness of the transformed target IEC61499 ontology can be checked by using semantic rules defined in the form such as SQWRL.

C. *Automatic Generation of IEC61499 Driven by ODE*

The idea of using ODE for the purpose of automatically generating IEC61499 systems was first proposed in [15]. The Semantic Web Rule Language (SWRL) is a language used to express rules and logics for the Semantic Web. Due to the rule of monotonicity, SWRL cannot be used to modify existing information in ontology. This means that it is not possible to create new instances, delete existing instances or relations between existing instances. This is problematic when generating IEC61499 code as new instances of function blocks, input and output interfaces need to be dynamically created. To overcome these limitations, an extension to the SWRL language called eSWRL was proposed in [15] which supports the creation and deletion of classes, object properties and class instances. The justifications and advantages of using eSWRL for the purpose of ontology transformation can be found at [15].

The case study used in [15] was a baggage handling system. For this paper, a similar approach to code generation is applied, but the application domain is substation automation control using IEC61850 and IEC61499. Due to size limitations, only the first stage of the automatic generation process is discussed in this paper, covering the development of the IEC61850 ontology and the ontology transformation rules used to transform the IEC61850 ontology to the IEC61499 ontology using eSWRL. The IEC61499 ontology will be adopted from the existing work, which contains a generic (domain neutral) ontological description of an IEC61499 system not including elements of IEC61850. The *Prolog*-based component of the transformation process, which implements the eSWRL rules, is not covered in this paper.

IV. CASE STUDY: OVERCURRENT PROTECTION

A. Sympathetic Tripping Feeder Model

The case study used to illustrate the initial transformation process is the Sympathetic Tripping case from [7]. For illustrative purposes, only one feeder branch will be considered for this paper. The protection scheme, however, will still be adopted from the one shown in [7]. The simplified feeder plant model is shown in Fig. 2. The feeder plant consists of a current transformer and a circuit breaker. The associating IEC61850 logical nodes for this feeder are the Current Transformer logical node TCTR, the Instantaneous Overcurrent protection logical node PIOC, the Tripping Condition logical node PTRC and the Circuit Breaker logical node XCBR.

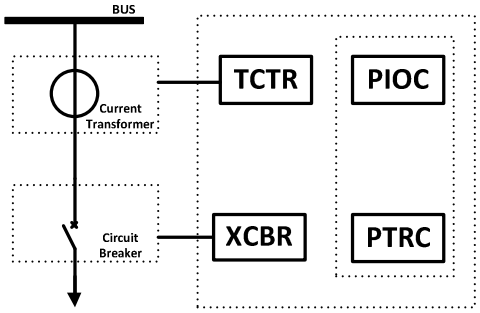

Fig. 2. Simplified Feeder Plant Model (SympatheticTrip) and the associating Logical Nodes.

The protection scheme is adopted from the one shown in [7] as illustrated in Fig. 3. The current Transformer TCTR will send periodic current readings to the Overcurrent Protection logical node PIOC. The signal, which contains the current reading, is the *TCTR.AmpSv_instMag_f* signal. The PIOC will compare the internal overcurrent threshold reading against the received current reading from the TCTR logical node. If the received current reading is above the pre-set threshold, then the PIOC logical node will generate a *PIOC.Op_general* signal to the Trip Conditioning logical node PTRC.

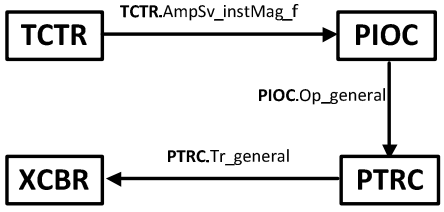

Fig. 3. IEC61850 Overcurrent Protection Signal Passing (Sympathetic Trip).

The PTRC logical node will check its internal condition of tripping. In this scenario, an overcurrent occurrence is enough for the tripping condition to be satisfied. Once the trip condition is satisfied, the PTRC logical node will generate a *PTRC.Tr_general* trip signal to the Circuit Breaker logical node XCBR. The circuit breaker will open or close depending on the value of the *PTRC.Tr_general* signal. The *PTRC.Tr_general* signal is of type Boolean and actuates the circuit breaker to open when the *PTRC.Tr_general* value is true and to close when the *PTRC.Tr_general* value is false.

B. Ontology of IEC61850 Logical Nodes

The IEC61850 ontology of the case study plant model is implemented in the ontology editor Protégé [14]. The simplified ontology model of the IEC61850 logical node is shown in Fig. 4.

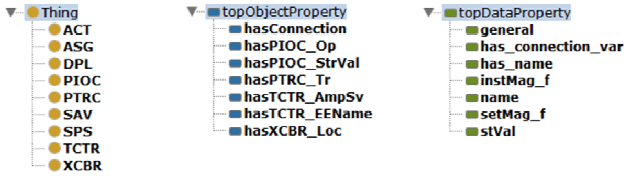

Fig. 4. Ontology of IEC61850 logical nodes in Protégé showing Classes (Left), Object Properties (Mid) and Data Properties (Right)

The hierarchical model of the IEC61850 logical node is made up of three levels. The top level is the logical node. Each logical node is made up of *n*- instances of data objects, which makes up the middle layer. Each data object is made up of *n*- data attributes, which makes up the lower layer. As can be seen in Fig. 4, there are nine classes. The *TCTR, PIOC, PTRC* and the *XCBR* classes are the logical node classes. The *ACT, ASG, DPL, SAV* and the *SPS* class are the data object classes. The data attributes of the data objects are implemented as data properties. The individuals (in ontology, instances are known as individuals) of the logical nodes are linked to the data objects via object properties with the *has-* prefix. For example, the TCTR logical node contains a data object called *AmpSv* of data object type *SAV*. The *TCTR* individual and the *SAV* individual are linked via the *hasTCTR_AmpSv* object property. The two important property of note are the *hasConnection* object property and the *has_connection_var* data property. The *hasConnection* object property between two logical node individuals indicates that a logical connection exists between the two logical nodes. The *has_connection_var* data property of *n* logical node individual indicates the input variable that needs to be created in order to allow other logical node to make the logical connection. An example is the TCTR and the PIOC logical nodes as shown in Fig. 3, the TCTR logical node needs to send the current reading *AmpSv_instMag_f* value to the PIOC logical node. The *hasConnection* object property will be created between the TCTR individual and the PIOC individual indicating that TCTR has a logical connection from itself and the PIOC logical node. In addition, the *has_connection_var* will be added as a data property to the PIOC class individual with the *AmpSv_instMag_f*, indicating that an input variable called *AmpSv_instMag_f* need to be created on the input interface of the PIOC function block.

C. Logical Node Dependent Input Variables

In the original iLN proposal in [6], each iLN function block has mirroring input and output variables. That is all data variables defined in the IEC61850 logical node type are made as input and output variables of the iLN function block. For example, the PIOC logical node has the *Op_general* variable to indicate that an overcurrent condition has been

detected.When implemented as an iLN, the input of the PIOC logical node function block will have an input named *Op_general_in* and a function block output variable named *Op_general_out* as shown in Fig. 5. It makes sense to have the *Op_general_out* variable as it indicates that an overcurrent condition has been detected and the intended use is to have this information passed onto trip conditioning logical node PTRC. However, it makes very little sense to have the *Op_general_in* as an input variable.

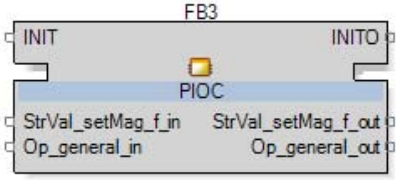

Fig. 5. iLN function block of PIOC with mirroring input and output interface

As discussed previously in chapter III, data variables within each logical node can be classified as settings, controls and Outputs variables. Firstly, let's categorize these variables as either input or output variables in terms of IEC61499 function blocks. Outputs variables are usually logical node status variables, measured values or metered values that need to be passed onto other connecting logical nodes. In the context of IEC61499 function blocks, these types of variables are likely to be output variables. Settings are variables, which are necessary to parameterize the initial conditions of the logical node and are not likely to change its value after the initialization stage. When implemented as a function block, settings variables are likely to be input variables. Control variables are binary or setpoint values, which are usually external control signals that are modified remotely or manual control via HMI panels. In terms of IEC61499 function block, control variables would also be input variables to the function block. Although both settings and control variables are input variables when implemented as function blocks, what these two types of input variables have in common is that these types of variable do not require information from its connecting logical node. Therefore, it is possible to simply parameterize these types of variables and contextual relationships play almost no part in these variables.

What is of interest are the input variable which requires logical connections from other logical node as these variables usually do not exist in the connecting logical node to allow the logical connection to take place. That is the input variables are dependent on the logical nodes, which are connected to. This can be illustrated by the two iLN function blocks in Fig. 6.

The iTCTR function block contains two data variables. The *EEName_name* variable is a description variable, which describes the name of the TCTR logical node and it is an input variable to the function block. The *AmpSv_instMag_f* is a measured variable which measures the periodic current readings and it is an output variable of the function blocks. The iPIOC logical node contains three variables, but only two of the variables are defined in the PIOC class. The setting variable *StrVal_setMag_f* sets the Overcurrent Threshold value and it is an input variable to the function block. The *Op_general* variable is a status variable which is an output variable of the function blocks.

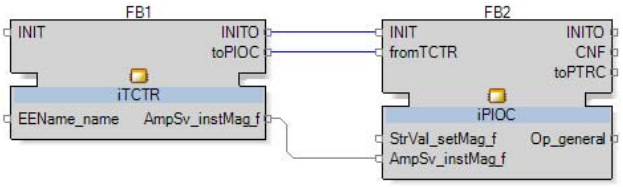

Fig. 6. TCTR (Left) and PIOC (Right) implemented in IEC61499 Function Blocks

As described in the overcurrent protection scheme in Fig. 3, the TCTR logical node sends *AmpSv_instMag_f* measured value to the PIOC logical node to check the measured values against the preset threshold value *StrVal_setMeg_f*. However, the PIOC logical node does not have an *AmpSv_instMag_f* variable defined under the PIOC logical node class. Therefore, it is necessary to create the *AmpSv_instMag_f* variable as an input to the PIOC logical node to allow a logical connection between the TCTR and the PIOC logical node. This is shown in Fig. 6 with the *AmpSv_instMag_f* as an input variable to the PIOC function block. Since the *AmpSv_instMag_f* variable is not defined under the PIOC class definition, it will not be described in the SCL configuration file. Therefore, it is necessary to have additional contextual information such as a protection scheme shown in Fig. 3 to add further contextual relationships between logical nodes that are missing in the SCL description. Logical Node variables such as settings or measured values can be automatically generated based on the SCL description, but input variables, which rely on their connected logical node cannot be created using the SCL configuration. Thus it are the logical node dependent input variables that are of utmost interest as it is necessary to create these variables first (even though they are not defined under their logical node class) before the logical connections can be made.

D. eSWRL Rules for Transforming IEC61850 ontology to IEC61499 Ontology

Note only a small part of entire IEC61499 ontology is used as a target ontology in this case study. It are domain-specific function blocks of type *iTCTR_FB* and *iPIOC_FB, has_interface, is_part_of_interface, event_connection*, and *data_connection* object properties, and *has_name* data property.

There are four main rules, which are used to transform the IEC61850 ontology to an existing IEC61499 system represented in eSWRL. The eSWRL rules will be used to create FB class, FB instances, function block interfaces and connections between the function blocks. The TCTR and the PIOC logical node will be used to demonstrate each of the eSWRL rules. The rules are as follows:

1. The first rule set is to create generic function block classes for logical nodes in the target IEC61499 ontology. That is, for each logical node class in the IEC61850 ontology, the corresponding class will be

created in the IEC61499 ontology. To create a new class, the *createClass* [15] operator can be invoked. The syntax for the *createClass* operator is:

```
eswrl:createClass({?<variable name> | <class
name>})
```

To create TCTR FB class (called *iTCTR_FB*), the following rule can be used.

```
eswrl:thereIsClass(TCTR) ->
eswrl:createClass(iTCTR_FB)
```

2. The second rule is to create an instance of the newly created *iTCTR_FB* class if an corresponding individual exists in the IEC61850 ontology. To create a new instance, the *createInstance* operator will be used. The syntax for the *createInstance* operator is:

```
eswrl:createInstance({?<variable name> |
<class name>}, {?<variable name> | <class
instance name>})
```

To create the new instance, the following rules can be used.

```
TCTR(?tctr_ln), has_name(?tctr_ln, ?nm) ->
eswrl:createInstance(iTCTR_FB, ?tctr_fb),
has_name(?tctr_fb, ?nm).
```

It should be noted that a link between an instance of logical node (LN) class and the corresponding instance of FB class is established on the basis of equality of their names.

3. The third rule set is to create interfaces of logical node function blocks. To illustrate this set of rules, PIOC logical node will be used since PIOC has an input variable from TCTR logical node. In addition, this type of input variables will also have a complementary event input created. E.g. for PIOC logical node, *AmpSv_instMag_f* input variable will have a complementary event input called *fromTCTR* created. To create the function block interfaces, the same *createInstance* operator is used. The rules for creating the interface of PIOC function block are as follows:

   The first step is to create the *INIT* (Initialization) and *INITO* (Finish Initialization) event input and output.

```
iPIOC_FB(?pioc_fb) ->
eswrl:createInstance(FBinterface,
?pioc_fb_interface),
has_interface(?pioc_fb,
?pioc_fb_interface),
eswrl:createInstance(FBevent, ?e1),
has_name(?e1, "INIT"),
eswrl:createInstance(FBevent, ?e2),
has_name(?e2, "INITO"),
is_part_of_interface(?e1,
?pioc_i_interface),
is_part_of_interface(?e2,
?pioc_i_interface).
```

   The second step is to add PIOC specific variables *StrVal_SetMag_f* (Setting) and *Op_general* (Status).

```
iPIOC_FB(?pioc_fb),has_interface(?pioc_fb
, ?pioc_fb_interface)->
eswrl:createInstance(FBvariable, ?d1),
has_name(?d1, "StrVal_setMag_f"),
is_part_of_interface(?d1,
?pioc_fb_interface),eswrl:createInstance(
FBvariable, ?d2), has_name(?d2,
"Op_general"), is_part_of_interface(?d2,
?pioc_fb_interface).
```

   The third step is to add the *AmpSv_instMag_f* as an input variable from TCTR logical node. This variable is determined in the rule implicitly by means of the instance of PIOC LN class. In addition, it is also necessary to create a complementary *fromTCTR* input event for *AmpSv_instMag_f* input variable.

```
PIOC(?pioc_ln), iPIOC_FB(?pioc_fb),
has_name(?pioc_ln, ?nm),
has_name(?pioc_fb, ?nm),
has_connection_var(?pioc_ln, ?connVar),
has_interface(?pioc_fb,
?pioc_fb_interface) ->
eswrl:createInstance(FBvariable, ?v),
has_name(?v, ?connVar),
eswrl:createInstance(FBevent, ?e),
has_name(?e, "fromTCTR"),
is_part_of_interface(?e,
?pioc_fb_interface),
is_part_of_interface(?v,
?pioc_fb_interface).
```

4. The fourth rule is to make the connection between *AmpSv_instMag_f* output variable of TCTR function block and *AmpSv_instMag_f* input variable of PIOC function block. Firstly, it is necessary to make the event connection between INITO event output of TCTR function block to INIT event input of PIOC function block.

```
TCTR(?tctr_ln), PIOC(?pioc_ln),
hasConnection(?tctr_ln, ?pioc_ln),
iTCTR_FB(?tctr_fb),
has_name(?tctr_ln, ?nm1), has_name(?tctr_fb,
?nm1), iPIOC_FB(?pioc_fb),
has_name(?pioc_ln, ?nm2), has_name(?pioc_fb,
?nm2), has_interface(?tctr_fb,
?tctr_fb_interface), has_interface(?pioc_fb,
?pioc_fb_interface),
is_part_of_interface(?e1,
?tctr_fb_interface), has_name(?e1, "INITO"),
is_part_of_interface(?e2,
?pioc_fb_interface), has_name(?e2, "INIT") -
> event_connection(?e1, ?e2).
```

Secondly, *AmpSv_instMag_f* variable from TCTR logical node will be connected to *AmpSv_instMag_f* variable on the PIOC logical node.

```
TCTR(?tctr_ln), PIOC(?pioc_ln),
hasConnection(?tctr_ln, ?pioc_ln),
iTCTR_FB(?tctr_fb), has_name(?tctr_ln, ?nm1),
has_name(?tctr_fb, ?nm1),
iPIOC_FB(?pioc_fb), has_name(?pioc_ln, ?nm2),
has_name(?pioc_fb, ?nm2),
has_interface(?tctr_fb, ?tctr_fb_interface),
has_interface(?pioc_fb, ?pioc_fb_interface),
is_part_of_interface(?e1,
?tctr_fb_interface),
has_name(?e1, "toPIOC"),
is_part_of_interface(?e2,
?pioc_fb_interface),
has_name(?e2, "fromTCTR"),
is_part_of_interface(?d1,
?tctr_fb_interface),
has_name(?d1, "AmpSv_instMag_f"),
is_part_of_interface(?d2,
?pioc_fb_interface),
has_name(?d2, "AmpSv_instMag_f") ->
event_connection(?e1, ?e2),
data_connection(?d1, ?d2).
```

V. CONCLUSION

This is an initial work in proposing an ODE framework in the automatic generation of IEC61850/61499 systems. Ontology is adopted for the purpose of transformation as existing SCL configuration does not contain concrete contextual information between the logical nodes. Expressing IEC61850 systems in ontology allows the integration of protection related information from protection schemes. The addition of a protection scheme which specifies contextual relations between the logical nodes enables logical connections to be automatically created between the logical nodes when implemented as function blocks, which are lacking in existing IEC61850/61499 based code generators. The initial stage of the transformation process involves transformation IEC61850 ontology to an existing IEC61499 ontology, creating an IEC61499 ontology which describes the generic structure of an IEC61499 system and function blocks with IEC61850 logical node interfaces and information of logical connections. The ontology transformation uses eSWRL, an extension to the semantic web rule language SWRL, which enables the creation of ontology classes, instances and properties. The IEC61499 ontology is then subject to further transformation via *Prolog* to the final IEC61499 control system.

REFERENCES

[1] CEN-CENELEC-ETSI, ""Smart Grid Coordination Group (SGCG) Reference Architecture Smart Grid" report of CEN-CENELEC-ETSI."

[2] A. Ipakchi and F. Albuyeh. (2009) Grid of the future. *IEEE Power Energy Magazine*. 55-62.

[3] A. Huang, "FREEDM system - a vision for the future grid," in *Power and Energy Society General Meeting, 2010 IEEE*, 2010, pp. 1-4.

[4] V. Vyatkin, G. Zhabelova, N. Higgins, M. Ulieru, K. Schwarz, and N. K. C. Nair, "Standards-enabled Smart Grid for the future Energy Web," in *Innovative Smart Grid Technologies (ISGT), 2010*, 2010, pp. 1-9.

[5] N. Higgins, V. Vyatkin, N.-K. C. Nair, and K. Schwarz, "Distributed Power System Automation With IEC 61850, IEC 61499, and Intelligent Control," *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on,* vol. 41, pp. 81-92, 2011.

[6] G. Zhabelova and V. Vyatkin, "Multiagent Smart Grid Automation Architecture Based on IEC 61850/61499 Intelligent Logical Nodes," *Industrial Electronics, IEEE Transactions on,* vol. 59, pp. 2351-2362, 2012.

[7] C.-W. Yang, G. Zhabelova, V. Vyatkin, N.-K. Nair, and A. Apostolov, "Smart Grid automation: Distributed protection application with IEC61850/IEC61499," in *Industrial Informatics (INDIN), 2012 10th IEEE International Conference on*, 2012, pp. 1067-1072.

[8] J. Xu, C.-W. Yang, G. Zhabelova, V. Vyatkin, and S. Berber, "Towards Implementation of IEC61850 GOOSE Messaging in IEC61499 Environment," in *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, 2013.

[9] T. Strasser, M. Stifter, F. Andren, D. Burnier de Castro, and W. Hribernik, "Applying Open Standards and Open Source Software for Smart Grid Applications: Simulation of Distributed Intelligent Control of Power Systems," in *IEEE Power & Energy Society General Meeting 2011*, 2011.

[10] S. M. Blair, F. Coffele, C. D. Booth, and G. M. Burt, "An Open Platform for Rapid-Prototyping Protection and Control Schemes With IEC 61850," *Power Delivery, IEEE Transactions on,* vol. 28, pp. 1103-1110, 2013.

[11] F. Andren, T. Strasser, and W. Kastner, "Towards a common modeling approach for Smart Grid automation," in *Industrial Electronics Society, IECON 2013 - 39th Annual Conference of the IEEE*, 2013, pp. 5340-5346.

[12] G. Zhabelova, C.-W. Yang, and V. Vyatkin, "SysGrid: IEC 61850/IEC 61499 based engineering process for Smart Grid automation design," in *Industrial Informatics (INDIN), 2013 11th IEEE International Conference on*, 2013, pp. 364-369.

[13] R. Santodomingo, J. A. Rodriguez-Mondejar, and M. A. Sanz-Bobi, "Ontology Matching Approach to the Harmonization of CIM and IEC 61850 Standards," in *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, 2010, pp. 55-60.

[14] W. Dai, V. N. Dubinin, and V. Vyatkin, "Migration From PLC to IEC 61499 Using Semantic Web Technologies," *Systems, Man, and Cybernetics: Systems, IEEE Transactions on,* vol. PP, pp. 1-1, 2013.

[15] V. Dubinin, V. Vyatkin, and C.-W. Yang, "Automatic Generation of Automation Applications Based on Ontology Transformations," presented at the Emerging Technologies & Factory Automation (ETFA), 2014 IEEE 19th Conference on, 2014.