OntoPLC: Semantic Model of PLC Programs for Code Exchange and Software Reuse

Yameng An, Feiwei Qin, Baiping Chen, Rene Simon, and Huifeng Wu, *Member, IEEE*

***Abstract*—Regarding Programmable Logic Controller (PLC) development, improving programming efficiency and encouraging code exchange and software reuse is necessary to increase the productivity and safety of smart manufacturing and industrial automation. However, differences between implementations are significant even though all manufacturers claim to conform to the IEC 61131-3 and IEC 61131-10 standards, resulting in incompatibilities inside heterogeneous systems, preventing projects from interoperating between vendors. In this paper, we present an approach that utilizes an ontology-based semantic model named OntoPLC to enable automatic porting of PLC projects between development environments and also prevent significant information loss during the translation process. High-level semantics of PLC projects are added into OntoPLC including software resources, which can be required for software reuse leveraging semantic query. To demonstrate the usefulness of the ontology model, the proposed methodology is applied to a turbine driven boiler feed pump control module.**

***Index Terms*—PLC, IEC 61131-3, Web Ontology Language (OWL), computable semantic model, knowledge representation, interoperability**

I. Introduction

NOWADAYS, due to the development of intelligent manufacturing, industrial control systems based on PLC has been playing an essential part in the industry. In addition to usual functionalities such as sequential relay control, motion control, process control, and networking, PLC has evolved over the years to include sufficient processing power, adequate storage capacity, more data handling methods and communication methods that are approximately equivalent to desktop computers [1]. The IEC 61131-3 standard [2] has specified the syntax and semantics of the five programming languages for PLC. Although almost all PLC vendors claim that their development platforms are compliant with this standard, in reality, none of them is compatible with each other [3], resulting in incompatibility in project exchange. PLCopen wants to accomplish a goal that a project can be transferred from one development environment to another without losing information. The PLCopen TC6 workshop publishes their XML schema for IEC 61131-3, in which interfaces for software tools are defined. Later, the IEC committee overhauls this and develops IEC 61131-10 PLC open XML exchange format [4]. This work has laid a solid foundation for project transformation by providing a unified XML exchange format. However, with the IEC standards only defining the syntax of PLC languages, it is still not possible to provide a computable semantic model that allows automatic code reusing, exchanging, and interaction among heterogeneous platforms in practice.

For instance, both "ADD" functions in CODESYS and OMRON's Sysmac Studio support the operation of adding integers or real numbers. However, the "ADD" function in CODESYS also supports addition between time (TIME), time of day (TOD), date & time (DT) data types with a single operation but different results: TIME + TIME = TIME, TOD + TIME = TOD, DT + TIME = DT; Sysmac Studio uses different operations for different combinations of data types: ADD_TIME for two times, ADD_TOD_TIME for a time and a time of day, ADD_DT_TIME for a time and a date & time. To convert a CODESYS project to Sysmac Studio, the "ADD" function should be translated into different functions according to the input data type. As shown in Table I, other development platforms also have their own implementation of the "ADD" functions. Therefore, the translation process is not trivial. In order to solve these problems, and to promote the reusability of code, we provide a formal computable semantic model in this paper. Semantics aim at providing a common meaning to the terms in a particular domain [5]. If the information has been translated to a unified semantic description, it's exchange and interoperate process among different PLC platforms could be implemented automatically in the semantic layer.

To provide a formal semantic tool, a knowledge model for representing all necessary concepts in a PLC project can be of great help. According to Nambi et al. [6], the ontology mechanism can be considered as a proper representation of such a knowledge base. Ontologies are used to capture knowledge about some domain of interest by providing a shared understanding of a specific domain. Such a shared understanding is necessary to overcome differences in terminology and expression [7]. For example, one application's "FBD" (Function Block Diagram) may be semantically equal to another application's "fbd", so they can be mapped to the same ontology. In this paper, we focus on the formal semantic representation of PLC projects by using Web Ontology Language (OWL), which is a standard ontology language developed by the World Wide Web Consortium (W3C). We refer to the formal computable semantic model as OntoPLC in this paper. OntoPLC focuses on representing the semantics of PLC projects and solving the incompatibility of export-

This work was supported by National Natural Science Foundation of China (Nos. U1609211 and 61972121).

Corresponding author: Huifeng Wu.

Yameng An, Feiwei Qin, Baiping Chen, and Huifeng Wu are with the Institute of Intelligent and Software Technology, Hangzhou Dianzi University, Hangzhou, 310018, China (e-mail: anyamg@pm.me; qinfeiwei@hdu.edu.cn; chenbp@hdu.edu.cn, whf@hdu.edu.cn).

Rene Simon is with the Department of Automation and Computer Sciences, Harz University of Applied Sciences, Wernigerode, 38855, Germany (e-mail: rsimon@hs-harz.de).

TABLE I
FUNCTION "ADD" WITH HETEROGENEOUS SEMANTICS AMONG DIFFERENT DEVELOPMENT PLATFORMS.

Beremiz
(ANY_NUM: IN1, ANY_NUM: IN2) $\Longrightarrow$ (ANY_NUM: OUT)
(TIME: IN1, TIME: IN2) $\Longrightarrow$ (TIME: OUT)
(TOD: IN1, TIME: IN2) $\Longrightarrow$ (TOD: OUT)
(DT: IN1, TIME: IN2) $\Longrightarrow$ (DT: OUT)
Sysmac Studio
(ANY_NUM: IN1, ANY_NUM: IN2) $\Longrightarrow$ (ANY_NUM: OUT)
(ANY_STRING: IN1, ANY_STRING: IN2) $\Longrightarrow$ (ANY_STRING: OUT)
CODESYS
(ANY_NUM: IN1, ANY_NUM: IN2) $\Longrightarrow$ (ANY_NUM: OUT)
(TIME: IN1, TIME: IN2) $\Longrightarrow$ (TIME: OUT)
(TOD: IN1, TIME: IN2) $\Longrightarrow$ (TOD: OUT)
(DT: IN1, TIME: IN2) $\Longrightarrow$ (DT: OUT)
TwinCAT
(ANY_NUM: IN1, ANY_NUM: IN2) $\Longrightarrow$ (ANY_NUM: OUT)
(TIME: IN1, TIME: IN2) $\Longrightarrow$ (TIME: OUT)
(TOD: IN1, TIME: IN2) $\Longrightarrow$ (TOD: OUT)

ing and importing between different PLC platforms based on semantic methodologies. Thus, OntoPLC can be utilized as a computation model to express and semantically enrich PLC project information, and as a software engineering tool for implementing code exchange functionality and enabling software reusability.

We outline two important steps. First, construct a unified ontology model for PLC project. Second, extract entities in PLC project from its source code files and translate them into OWL instances (ABox-assertion box), whereas corresponding XML schema forms OWL classes (TBox-terminological box). This semantic model also allows us to perform reasoning, inferences, and queries for further implementation.

The paper is organized as follows. The related researches about the reuse of PLC components and ontologies in the industrial automation area are reviewed in Section II. The construction of a unified PLC ontology model is introduced in Section III. And then the implementation process of OntoPLC is presented in Section IV. An automatic translation tool is provided in this section as well. In Section V, we demonstrate the usefulness of the novel model by automating rule-based verification of code exchange and software reuse in a use case. Finally, conclusions and future work are attached in Section VI.

II. RELATED WORK

A. Reuse of PLC Components

Several pieces of related work focus on vendor independence of PLC source codes, i.e., transforming PLC programs from heterogeneous PLC programming tools of different vendors into independent neutral formats. Estevez et al. [8] provide a method that allows transforming PLC source code among different programming tools by using XML technologies as interoperable middleware. Ghosh et al. [9] apply a vendor-independent XML format to emphasize the control logic of PLC source code. Dietz et al. [10] utilize PLCopen XML for the exchange of behavior models of mechatronic components within different simulation tools.

Some approaches in works of literature targeting code generation. For instance, PLCspecif [11] is a formal specification language of reusable PLC components. After the specification of the component's behavior, the automatic generation of Structured Text code is also available. A template-based approach interprets PLC IDE and plant models as graph data structures to generate PLC programs consistent with the software architecture [12]. Another method generates control software in PLCopen XML from process design [13]. The transition utilizes standard data representation of process topology and control software.

Design pattern is another method since it can increase the reusability and modularity of the code. Fuchs et al. [14] apply five typical patterns to analyze dependencies and encapsulations of software units for further reuse. Similar to this, Nenninger et al. [15] put forward a whole-part pattern together with data blocks (DB), which encapsulate functions and function blocks, to enable code reuse.

B. Ontology in Industrial Automation Domain

Ontology provides a formal explicit description of concepts in a particular domain and relations among them. According to Noy et al. [16], the benefits of using ontology are multiple, including sharing a common understanding of the information structure among people or software agents. Recently ontology is introduced into the industrial automation area, where it is majorly applied to describe manufacturing and processing facilities.

Dai et al. [17] propose an ontology model for designing PLC systems based on the IEC 61499 standard, which is a standard about function blocks [18]. The semantic correction and enrichment are described to establish a complete and useful ontology model. Further, mappings from PLC ontology model to function block model and code generation are implemented to migrate from IEC 61131-3 PLC codes to IEC 61499 function blocks automatically [19]. An ontology-driven methodology is present for converting PLC plants into ones managed by system-on-chip or single-board-computers [20]. By mapping the behaviors of sensors and actuators into the ontological schema, the implementation details of the PLC plant is simplified.

Some approaches focus on data integration or model design in the automation domain. CTRLont [21] is an ontology model that formally specifies the domain of control logic in building automation system (BAS). This ontology could be used for automating rule-based verification of designed control logic. Yang et al. [22] provide a method for the automatic generation of smart-grid automation systems software by using an ontology model and ontology transformation. This method relies on the IEC 61850 and IEC 61499 standards. Ryabinin et al. [23] put forward a solution to build custom IoT-based Human-Machine Interfaces (HMI) which utilize a series of ontologies to control the behavior of all the internal modules. The ontology knowledge base is adaptive to different applications.

C. Summary

There are certain limitations on using XML format as a representation of a PLC project. The XML format is designed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2020.2997360, IEEE Transactions on Industrial Informatics

for representing general information and does not provide means for expressing the semantics of data, hence the semantic information related to PLC cannot be expressed in a commonly agreed manner. For example, the connection relationship of the blocks and detailed semantics of function and function block are not commonly defined in XML. Moreover, the nesting of tags does not have standard meaning. In other words, there is no intended meaning associated with the nesting of tags; it is up to each programming environment to interpret the nesting. Collaboration and exchange are supported if there is an underlying shared understanding of the vocabulary.

The XML schema defines document structures while ontology is a domain model providing rich logical constraints. Existing computational models based on ontology related to the industrial automation system are aiming at various application aspects, e.g., data integration, information exchange, model design, and data reuse. These models leverage OWL and other ontology techniques to represent, infer, and align knowledge in respective applications, e.g., IoT, BAS, etc. The ontologies describe the concepts and their relations about communicating information, control logic and devices. Ontology is typically used to break the gap between two manners where either concepts or concept relationships are mismatched in the information exchanging process.

However, there is no model about explicit modeling of the PLC project and these existing models fail to meet a series of demands for a semantic model targeting information exchange and reuse between heterogeneous PLC platforms.

- Explicit representation of PLC project and formal specification of PLC domain knowledge and experience.
- High-level description of PLC project related software resources, e.g., the functionality of a function as an architectural style, and contextual information of a program.
- Extensibility to integrate other ontologies in adjacent domains.
- An automatic tool to extract information to the knowledge base.
- Use of a computable model to enable intelligent applications based on the ontology knowledge base, such as code exchange, code recommendation, code quality evaluation, and software reuse.

To respond to the above requirements and to allow the representation of PLC project in a unified formal manner, we put forward a novel computable ontology model named OntoPLC which will be described in the following sections.

III. OntoPLC ontology

Since the industrial automation area requires highly accurate knowledge for maintenance and decision making, the ontology is constructed by defining the application first, then defining the application scenario, and subsequently retrieving the corresponding data. In the code exchange and software reuse scenario of this paper, the proposed approach refers to a semantic representation of PLC project and knowledge alignment by using the ontology model. The entire process includes importing PLC projects into the PLC knowledge base, and then automatically map instances from heterogeneous platforms through a unified ontology model by implementing ontology matching and mapping rules. The PLC knowledge base is defined in OWL-DL. An overview of the OntoPLC knowledge base is shown in Fig. 1.

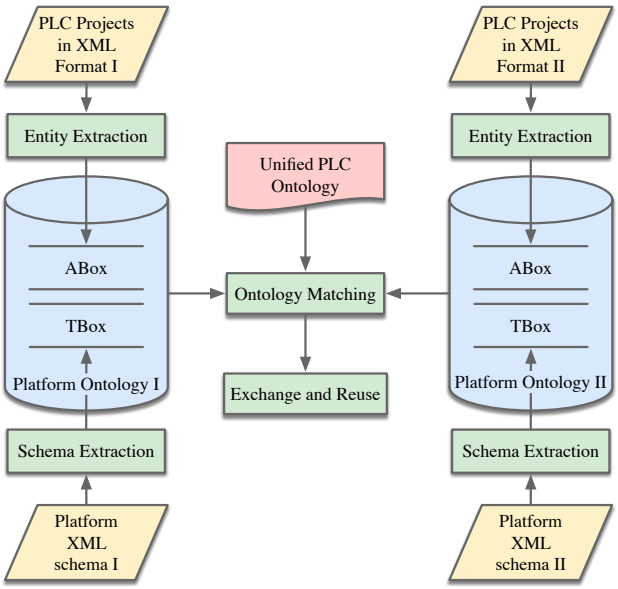

Fig. 1. OntoPLC knowledge base for exchange and reuse of PLC projects between heterogeneous platforms by applying ontology matching through a unified PLC ontology.

A. Unified PLC Ontology Model Definition

We apply a top-down development process starting with the most general concepts in PLC domain and subsequent specialization of these concepts to construct a layered ontology since PLC projects are systematic and have certain structures [16]. The major part of the unified PLC ontology model is shown in Fig. 2. In this ontology, we choose to represent the common structure of PLC projects with high extensibility. For vendor-specific information, such as vendor-specific programming languages and task types, our ontology can easily be extended to include new knowledge.

For ontology editor and viewer, Protégé is the most commonly used tool in academic research for ontology modeling and development. It is an open-source tool originally developed by Stanford University. We will use Protégé as our editor and viewer tool for prototyping the PLC project ontology here.

Central to the ontology is the definition of `Project`, which is the most elementary concept. A project contains essential metadata (`ProjectName`, `CreatedTime`, etc.), vendor-specific information (`VendorSpecificInfo`), runtime config (`Configuration`), and most importantly, `POU`. The Program Organization Unit (POU), written in the five supported languages, Ladder Diagram (LD), Function Block Diagram (FBD), Instruction List (IL), Structured Text (ST), and Sequential Function Chart (SFC), consists of `GraphicElement` and `TextElement`. The terminology is in accordance with the IEC 61131-10 standard.

OWL provides two main types of properties in order to represent semantic relationships. Object properties contain the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2020.2997360, IEEE Transactions on Industrial Informatics

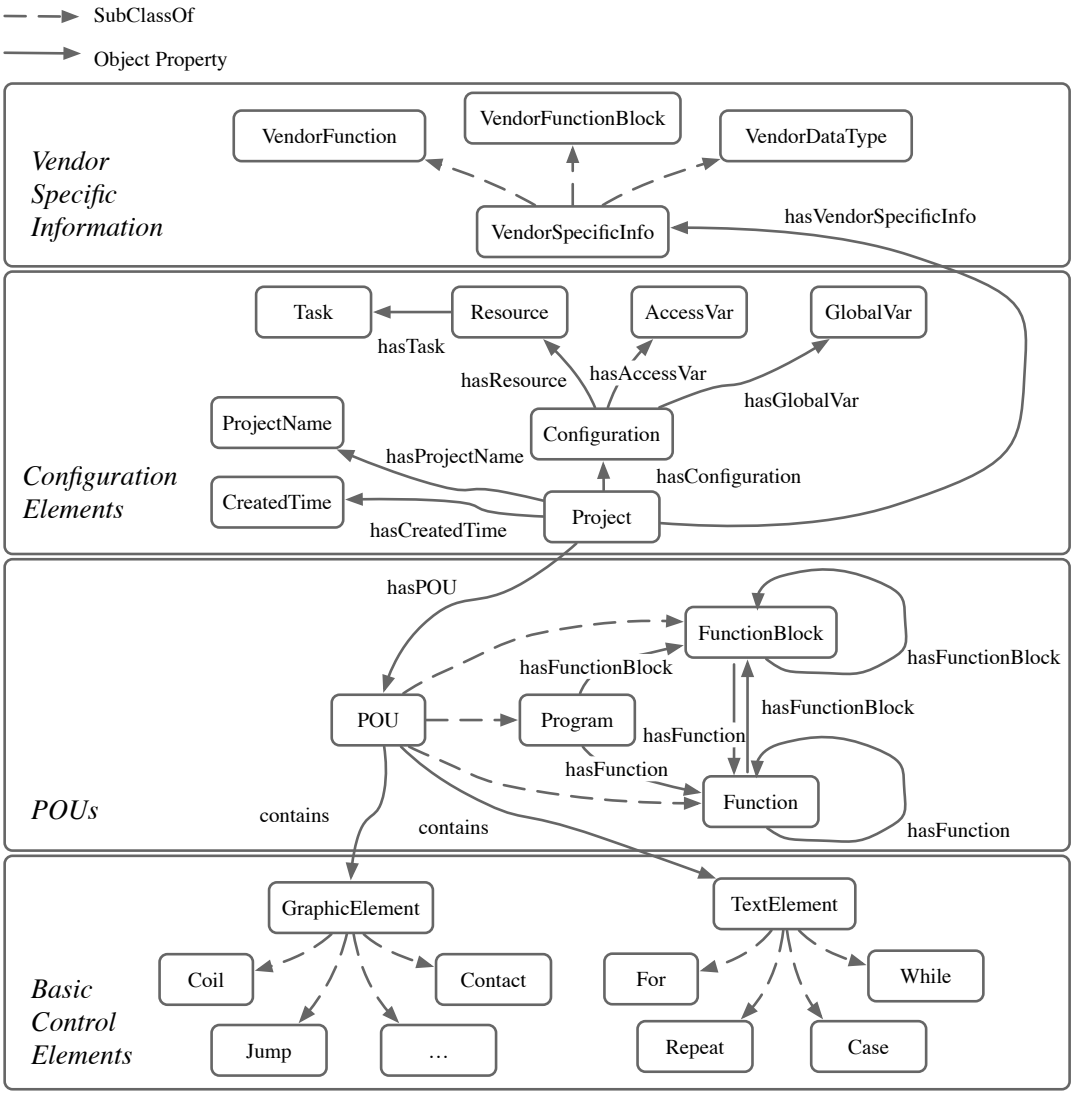

Fig. 2. Class hierarchy of the unified PLC ontology model (partly).

structure information of the PLC project in this ontology, and we define their names in the form of `hasDomainConcept`. Data properties are used to represent the attribute values of elements in PLC project. To represent a constant value of a data type in the attributes of elements, the name of data property is defined as `hasDomainConceptAttribute`, or as `hasConceptValue` if the concept itself is a constant value. Data properties and restrictions (e.g., quantifier restrictions, cardinality restrictions, and hasValue restrictions) in the ontology are omitted in the figure for readability.

B. Evaluation of the Unified Ontology

The unified PLC ontology defined in the previous subsection has the following characteristics.

1) Clarity: This ontology tries to communicate the intended meaning of defined terms as much as possible, and the definitions are objective. Where possible, a complete definition (a predicate defined by necessary and sufficient conditions) is applied instead of a partial definition (defined by only necessary or sufficient conditions). All definitions are documented with natural language.

2) Extendibility: This ontology is designed to anticipate the use of shared vocabulary. Other users are able to define new terms for special uses based on the existing vocabulary, in a way that does not require the revision of the existing definitions. What's more, this ontology is constructed in a modularization way, which is effective for further ontology reuse [24].

3) Minimal encoding bias: The concepts are specified at the knowledge level without depending on a particular symbol-level encoding. According to Gruber [25], encoding bias should be minimized because knowledge-sharing agents may be implemented in different representation systems or different styles of representation.

IV. Implementation of OntoPLC

The construction of the unified PLC ontology in OntoPLC has been discussed in the previous section. In this section, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2020.2997360, IEEE Transactions on Industrial Informatics

present and discuss facets related to the implementing process. The implementation of OntoPLC is summarized in Fig. 3.

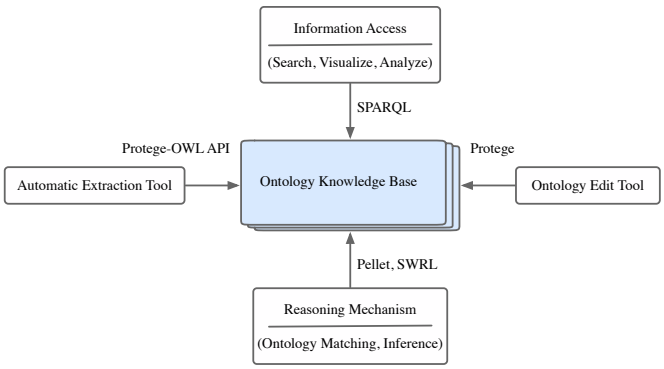

Fig. 3. Implementation of OntoPLC.

A. Translating Schemas and Instances

The transformation of XML schema is performed with a set of predefined mapping rules. We summarize these rules in Table II. XSD file is in the form of a tree structure that starts from "the root" to "the leaves", which is equivalent to the hierarchy of ontology classes. Class (`owl:Class`) is generated from `xsd:element` that has attributes or child XSD elements. Leaf elements that have no attribute together with XSD attribute are mapped to `owl:DataProperty`. For the case when one XSD element contains another element, we assume a "subClassOf" relation or a "has" relation. An XSD element is mapped to `owl:subClassOf` if its parent element contains no literal, otherwise to `owl:ObjectProperty`.

TABLE II
MAPPING RULES DEFINED IN THE TRANSFORMATION METHOD.

XSD	OWL
xsd:element containing at least one xsd:element or at least one xsd:attribute	owl:Class
xsd:complexType/xsd:group	owl:subClassOf, owl:ObjectProperty
xsd:attribute/xsd:element containing neither xsd:element nor xsd:attribute	owl:DataProperty
xsd:minOccurs	owl:minCardinality
xsd:maxOccurs	owl:maxCardinality
xsd:sequence/xsd:all	owl:intersectionOf
xsd:choice	combination of owl:intersectionOf owl:complementOf, and owl:unionOf

The transformation of XML files aims to recognize all the instances declared in the XML file, and then create individuals and property assertions. First, we retrieve the XML tree of the XML file by using a parser. Second, we scan the XML tree and create the OWL construct by using APIs. For each element encountered in the XML tree, an individual is created, and its attributes are obtained and translated. All instances of a PLC project are distinct, which means that any two instances represent two different real-world objects. Conversely, OWL individuals are not inherently distinct. Unless explicitly declared as distinct, two OWL individuals may represent the same real-world object. To translate the PLC project instances correctly, it is then necessary to declare explicitly all the OWL individuals contained in the same file to be distinct by using `owl:AllDifferent` OWL construct. If two or more instances own the same name, increasing serial numbers will be used as suffixes in individual names according to the order of its appearance.

B. Ontology Matching

The goal of ontology matching is to find the equivalence relations between elements of different ontologies. Solving such a matching problem is of key importance to generate semantic descriptors [26]. The heterogeneity in PLC domain is mainly caused by the differences in the terms and attributes. Therefore, we apply a multi-facets method considering both element names and structural aspects on the schema-level, or say TBox level, of the ontology. The algorithm is presented in Algorithm 1. The unified PLC ontology works as a reference model. Preprocessing is required before using the algorithm including removing connecting characteristics and platform-specific prefixes and lowercase all the strings.

Algorithm 1 Mapping heterogeneous PLC ontologies to the unified PLC ontology

Input:
Unified PLC ontology, O; Platform specific ontology, O'.
Output:
Alignment which contains a set of correspondences between two ontologies.

$Correspondences = \{\}$
for each $c \in O$ and $c' \in O'$ **do**
  Calculate the string similarity of concepts c and c' using the Levenshtein distance, $sim\_string(c, c') = 1 - \frac{lev(c,c')}{|c|+|c'|}$
  Calculate the internal structure of c and c' by comparing properties, $sim\_internal\_structure(c, c') = \frac{|c_p \vee c'_p| - |c_p \wedge c'_p|}{|c_p \vee c'_p|}$
  Inquire the datatype similarity of c and c' from predefined datatype similarities, $sim\_datatype(c, c')$.
  $similarity(c, c') = sim\_internal\_structure(c, c') \sqcup sim\_string(c, c') \sqcup sim\_datatype(c, c')$
  if $similarity(c, c') > similarity\_threashold$ **then**
    $(c, c') \rightarrow Correspondences$
  end if
end for
return $Correspondences$

C. Ontology Reasoning

After ontology matching, the semantics of the heterogeneous platforms are mapped to the unified PLC ontology model. However, the matching results are not guaranteed to be accurate. Besides, we have represented the basic-level knowledge of the software resources of the PLC project, but

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TII.2020.2997360, IEEE Transactions on Industrial Informatics

there remains high-level semantics that cannot be retrieved directly. Moreover, in the development and maintenance scenarios, engineers' knowledge and experience are valuable for reference. It would be helpful to formally represent such kind of knowledge and experience (e.g., development cases, feature models, issues with corresponding solutions, test logs, bug reports) so that the new engineers can solve related problems by utilizing the ontology knowledge base.

To solve these problems, SWRL rules are used to carry out ontology reasoning. Note that the SWRL has a lot of limitations and it's not appropriate for all tasks. The specific usages are as follows.

- Formally express engineers' domain knowledge and experience in SWRL rules, e.g., software construction principles, software maintenance method, and software evaluation.
- Infer implicit instances knowledge and complex relations from explicit instances knowledge by using SWRL to retrieve high-level semantics about software resources, devices information, malfunction cases, etc.
- Check and verify the mapping results by using SWRL.

Specific SWRL rules for heterogeneous information fusion and consistency checking are proposed in Section V-B and Section V-C respectively.

D. Protégé Plug-in of OntoPLC

We develop a plug-in of the Protégé editor to integrate and automate the translation process of the XML schema and XML file of a PLC project within the Protégé editor. The translation is performed by a parsing process which is followed by a converting process. Finally, a generator is set for the creation of new axioms. The details of this plug-in are given in Fig. 4. Apart from the automatic translation, we also provide a set of tools including "Add", "Edit", "Delete" for freely manual manipulation.

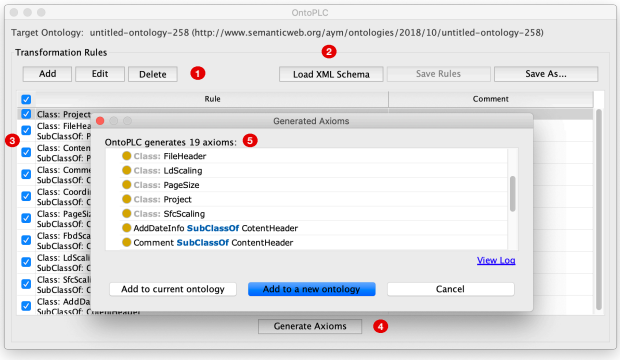

Fig. 4. Protégé plug-in OntoPLC. This plug-in is an automatic engine for converting PLC projects into OWL, then mapping it to the unified PLC ontology. 1) Transformation rules edit panel: Add, edit and delete functionality for manual manipulation; 2) Load XML Schema: Load, parse and convert XSD file for TBox; 3) Transformation rules browser: Show the list of transformation rules; 4) Generate axioms: Trigger the creation of new axioms based on the transformation rules; 5) Axioms preview: Preview new axioms and add them to a new ontology or to the current open ontology.

V. Application of OntoPLC

A. Example of Representing a PLC Project with OntoPLC

PLC projects commonly use five programming languages including two textual languages, i.e. ST and IL, and three graphical languages, i.e. FBD, LD, and SFC. OntoPLC is able to transform all these languages into the ontology model. In this section, an example written in FBD language is given to illustrate the formal semantic representation provided in this paper, since FBD is most widely used.

This program is used to monitor the steam turbine driven boiler feed pump control system, checking if all parameters are in the given ranges including bearing temperature, vibration amplitude and so on. The instant one of these parameters is detected to deviate from the normal range, a fault signal will be output immediately. As shown in Fig. 5, if the measured value of the steam turbine's vibration amplitude or water pump's bearing temperature exceeds the value of VibMax for more than 5s, it will output PumpFault and OverVib. And if the measured value of the steam turbine's temperature or water pump's temperature exceeds the value of TempMax, it will output PumpFault and OverTemp.

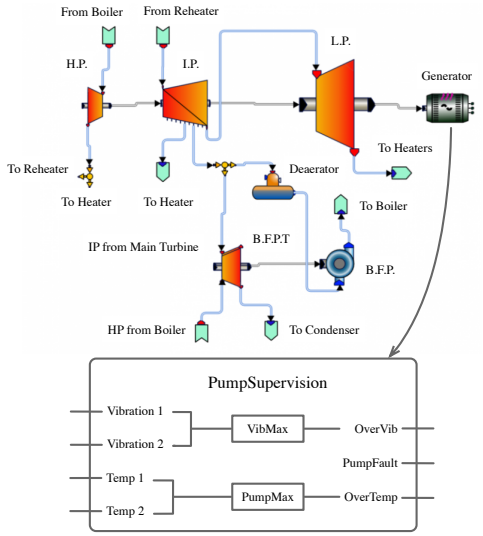

Fig. 5. Turbine driven boiler feed pump control module.

Part of the resulting PumpSupervision ontology achieved by applying OntoPLC is given in Fig. 6. This part contains one function GT1 and one function block TON_1. The relationship of connection between variables and blocks are realized by `localId` mechanism. All variables and blocks (i.e. function and function block) have unique `localId` of their own. A block can have an object property `hasRefLocalId` which indicates it has a variable with particular `localId`.

B. Example of Translating a Heterogeneous Graphic Control Across Different Platforms

The problem of translating a heterogeneous graphic control can be divided into two basic problems. The first is how to precisely represent this graphic control and make a unified understanding by using OntoPLC. The second is how to

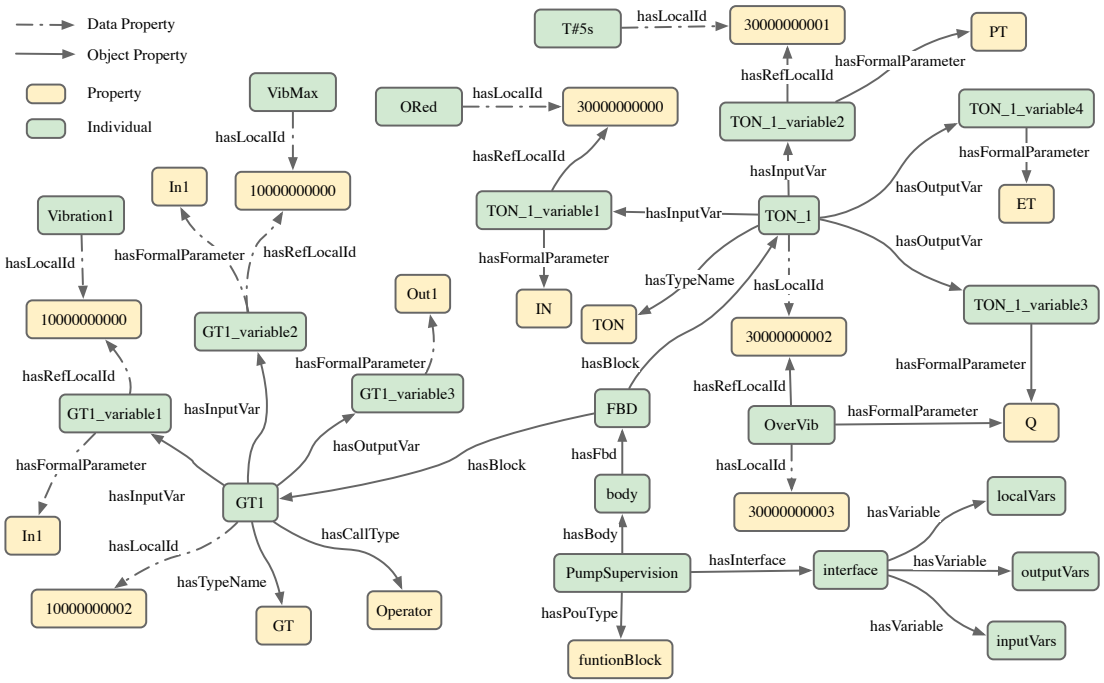

Fig. 6. Ontology model instance of PumpSupervision according to OntoPLC (partly).

match the knowledge of the control on different platforms utilizing SWRL rules. Other graphic controls like LD elements (contact, coil, etc.) and SFC elements (action, step, etc.) can be processed in the same way.

Consider the example of function "ADD" we provide at the beginning. Part of the facts about the function "ADD" in Beremiz are displayed in Fig. 7. This function is used for time-of-day additions accepting one input variable of TIME_OF_DAY type and one input variable of TIME type outputting a variable of TIME_OF_DAY type. Unlike "ADD" functions for digital calculation which can have more than two input variables, "ADD" for time functions have and can only have two input variables. We can learn from Fig. 7 that the key information for the translation process is the input variable type and the output variable type of the function.

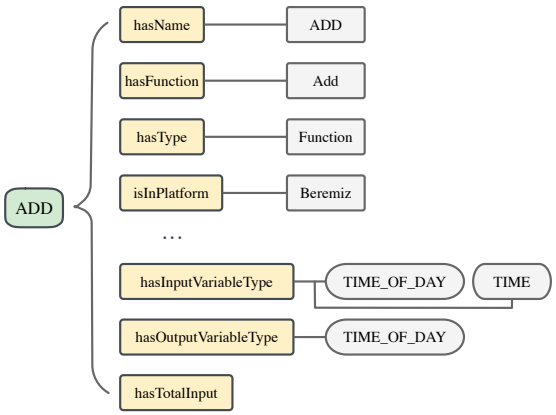

Fig. 7. Facts about the function "ADD" in Beremiz: (TOD: IN1, TIME: IN2) ⇒ (TOD: OUT) (partly). Rectangle nodes with background color represent classes, nodes with blank background indicate individuals. Flattened rectangle nodes represent data types.

TABLE III
FUNCTION "ADD" MATCHING CLOSURE

Beremiz
ADD(ANY_NUM, ANY_NUM, ANY_NUM, ...)
ADD(TIME, TIME, TIME)
ADD(TOD, TOD, TIME)
ADD(DT, DT, TIME)
CONCAT(STRING, STRING, STRING)
Sysmac Studio
ADD(ANY_NUM, ANY_NUM, ANY_NUM, ...)
ADD(STRING, STRING, STRING)
ADD_TIME(TIME, TIME, TIME)
ADD_TOD_TIME(TOD, TIME, TOD)
ADD_DT(DT, TIME, DT)
CONCAT(STRING, STRING, STRING, ...)
CODESYS
ADD(ANY_NUM, ANY_NUM, ANY_NUM, ...)
ADD(TIME, TIME, TIME)
ADD(TOD, TOD, TIME)
ADD(DT, DT, TIME)
CONCAT(STRING, STRING, STRING)
TwinCAT
ADD(ANY_NUM, ANY_NUM, ANY_NUM, ...)
ADD(TIME, TIME, TIME)
ADD(TOD, TOD, TIME)
ADD(DT, DT, TIME)
CONCAT(STRING, STRING, STRING)

To migrate function "ADD" across different platforms including Beremiz, Sysmac Studio, CODESYS, and TwinCAT, firstly, we come up with a matching closure according to a tuple of function name, input variable type, and output variable type. This closure is shown in Table III. The first parameter in each function tuple indicates the output variable type of this function. The remaining parameters indicate the input variable types. The matching target of the function that is in this closure is also in this closure.

The next step is matching the knowledge across platforms using SWRL rules. For instance, the ADD function in Fig. 7 implies ADD(TOD, TOD, TIME) in Beremiz. Consider migrating this function from Beremiz to Sysmac Studio. According to the matching closure in Table III, this function is equivalent to ADD_TOD_TIME(TOD, TIME, TOD) in Sysmac Studio. This equivalent relation specified in SWRL rules is expressed Rule 1. The order of the functions' input variables is not significant.

Rule 1 Identify equivalent "ADD" function.

$$\begin{aligned} &Function(?\,x) \wedge hasName(?\,x, \text{“}ADD\text{”}) \\ &\wedge hasPlatform(?\,x, \text{“}Beremiz\text{”}) \\ &\wedge hasOutputVariableType(?\,x, \text{“}TOD\text{”}) \\ &\wedge Function(?\,y) \\ &\wedge hasName(?\,y, \text{“}ADD\_TOD\_TIME\text{”}) \\ &\wedge hasPlatform(?\,y, \text{“}SysmacStudio\text{”}) \\ &\wedge hasOutputVariableType(?\,y, \text{“}TOD\text{”}) \\ &\Rightarrow equivalentTo(?\,x, ?\,y) \end{aligned}$$

In a test example, a Beremiz ontology contains 276 items and a CODESYS ontology contains 196 items. As a result, 67 out of 69 equivalent items are mapped. The precision of our algorithm is 100% and the recall is 97.1%.

C. Consistency Checking After Translation

It is necessary to check and verify the translation results by automating a rule-based verification method because the semantic consistency may be lost during the translating process. For instance, connectivity is an important feature of graphic control elements like functions in PLC project. Since the importing and exporting platforms of the translating process could have different numbers of input variables or output variables of the equivalent functions, this might lead to the change of the connectivity feature of the function after translation. A connected function may become partly connected. This kind of error is difficult to be resolved automatically. It is then required to implement consistency checking and correction by verification rules.

The semantics of the features are determined by boundary conditions. For example, each input variables and output variables of a connected function is linked to other elements. We express such conditions in SWRL rules in Rule 2 so that ontology inference engines like Pellet [27] can understand and then verify the feature instances.

Rule 2 Identify connected function.

$$\begin{aligned} &Function(?\,f) \wedge hasVariable(?\,f, ?\,var) \\ &\wedge hasConnection(?\,var, ?\,c) \Rightarrow Connected(?\,f) \end{aligned}$$

Partly-connected functions contain both linked variables and unlinked variables. This condition is specified in Rule 3.

After the consistency checking of SWRL rules, the developed plug-in tool deletes or corrects the inconsistent and incompatible semantics of the instances.

Rule 3 Identify partly-connected function.

$$\begin{aligned} &Function(?\,f) \wedge hasVariable(?\,f, ?\,var_1) \\ &\wedge hasConnection(?\,var_1, ?\,c_1) \\ &\wedge hasVariable(?\,f, ?\,var_2) \\ &\wedge (hasConnection = 0)(?\,var_2, ?\,c_2) \\ &\wedge differentFrom(?\,var_1, ?\,var_2) \\ &\Rightarrow PartlyConnected(?\,f) \end{aligned}$$

VI. Conclusion and Future Work

This paper presents an approach to enable the formal unified semantic representation of PLC projects, which could be used for code exchange and software reuse. The main motivation is that different vendors have different implementations for IEC 61131-3 projects, and they can not be reused in other environments not only from file exchange format (syntactic) aspect but also from the semantic aspect. We provide a unified PLC ontology model as middleware and then present the implementation of automatically translating XML schema and PLC projects into the ontology-based knowledge model. The translation process converts the original tree structure to an object-oriented graph structure. This will inevitably lose the original tree structure and will cause the loss of the original information, for example, the hierarchy of the XML file order of the XML elements. Nevertheless, OntoPLC ensures all necessary concepts are maintained correctly and consistently. By using OntoPLC, PLC programs generated by multiple vendor platforms of different models could be targeted to the unified PLC ontology model, semantic search of PLC controls could be performed by using semantic tools, code reuse and exchange among projects could be achieved, and PLC application development could be faster than ever. Meanwhile, heterogeneous ontologies are matched to the unified PLC ontology model by using a multi-facets ontology matching algorithm. Mismatches in PLC projects from different vendor platforms are aligned utilizing ontology reasoning and SWRL rules.

Future work includes improving the unified PLC ontology model to contain different aspects of engineering semantics of POUs. Then this model could be used for code quality evaluation, code summarization, code question answering, and code recommendation. This research work will continue with semantic reasoning, retrieval, and analysis by using techniques and tools such as graph transformation, SWRL, and SPARQL. The mapping rules between ontologies for code exchange will be completed in the future.

References

[1] H. Wu, Y. Yan, D. Sun, and R. Simon, "A customized real-time compilation for motion control in embedded PLCs," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 2, pp. 812–821, 2019.

[2] IEC TC 65/SC 65B - Measurement and control devices, "Programmable controllers - Part 3: Programming languages," International Electrotechnical Commission, Standard IEC 61131-3:2013, Feb. 2013.

[3] M. B. Younis and G. Frey, "Formalization of PLC programs to sustain reliability," in *IEEE Conference on Robotics, Automation and Mechatronics, 2004.*, vol. 2. IEEE, 2004, pp. 613–618.

[4] IEC TC 65/SC 65B - Measurement and control devices, "Programmable controllers - Part 10: PLC open XML exchange format," International Electrotechnical Commission, Standard IEC 61131-10:2019, Apr. 2019.

[5] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[6] S. A. U. Nambi, C. Sarkar, R. V. Prasad, and A. Rahim, "A unified semantic knowledge base for IoT," in *Internet of Things (WF-IoT), 2014 IEEE World Forum on*. IEEE, 2014, pp. 575–580.

[7] G. Antoniou and F. Van Harmelen, *A semantic web primer*. MIT press, 2004.

[8] E. Estevez, M. Marcos, E. Irisarri, F. Lopez, I. Sarachaga, and A. Burgos, "A novel approach to attain the true reusability of the code between different PLC programming tools," in *2008 IEEE International Workshop on Factory Communication Systems*. IEEE, 2008, pp. 315–322.

[9] A. Ghosh, S. Qin, J. Lee, and G.-N. Wang, "An output instruction based PLC source code transformation approach for program logic simplification," *Informatica*, vol. 41, no. 3, 2017.

[10] M. Dietz, J. Sippl, and R. Schmidt-Vollus, "Concept for an interoperable behavior library for virtual commissioning using PLCopenXML," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 1433–1436.

[11] D. Darvas, E. B. Viñuela, and I. Majzik, "PLC code generation based on a formal specification language," in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*. IEEE, 2016, pp. 389–396.

[12] Y. Pavlovskyi, M. Kennel, and U. Schmucker, "Template-based generation of PLC software from plant models using graph representation," in *2018 25th international conference on mechatronics and machine vision in practice (m2vip)*. IEEE, 2018, pp. 1–8.

[13] N. Papakonstantinou, J. Karttunen, S. Sierla, and V. Vyatkin, "Design to automation continuum for industrial processes: ISO 15926–IEC 61131 versus an industrial case," in *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2019, pp. 1207–1212.

[14] J. Fuchs, S. Feldmann, C. Legat, and B. Vogel-Heuser, "Identification of design patterns for IEC 61131-3 in machine and plant manufacturing," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 6092–6097, 2014.

[15] P. Nenninger and T. Puchstein, "Software design patters in IEC 61131–3 systems: A case study," in *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE, 2018, pp. 795–800.

[16] N. F. Noy, D. L. McGuinness *et al.*, "Ontology development 101: A guide to creating your first ontology," 2001.

[17] W. Dai, V. Dubinin, and V. Vyatkin, "IEC 61499 ontology model for semantic analysis and code generation," in *Industrial Informatics (INDIN), 2011 9th IEEE International Conference on*. IEEE, 2011, pp. 597–602.

[18] IEC TC 65/SC 65B - Measurement and control devices, "Function blocks - Part 1: Architecture," International Electrotechnical Commission, Standard IEC 61499-1:2012, Nov. 2012.

[19] W. Dai, V. N. Dubinin, and V. Vyatkin, "Migration from PLC to IEC 61499 using semantic web technologies," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 44, no. 3, pp. 277–291, 2014.

[20] M. Cristani, F. Demrozi, and C. Tomazzoli, "ONTO-PLC: An ontology-driven methodology for converting PLC industrial plants to IoT," *Procedia Computer Science*, vol. 126, pp. 527–536, 2018.

[21] G. F. Schneider, P. Pauwels, and S. Steiger, "Ontology-based modeling of control logic in building automation systems," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 6, pp. 3350–3360, 2017.

[22] C.-W. Yang, V. Dubinin, and V. Vyatkin, "Ontology driven approach to generate distributed automation control from substation automation design," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 2, pp. 668–679, 2016.

[23] K. Ryabinin, S. Chuprina, and K. Belousov, "Ontology-driven automation of IoT-based human-machine interfaces development," in *International Conference on Computational Science*. Springer, 2019, pp. 110–124.

[24] C. Legat, C. Seitz, S. Lamparter, and S. Feldmann, "Semantics to the shop floor: towards ontology modularization and reuse in the automation domain," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 3444–3449, 2014.

[25] T. R. Gruber, "Toward principles for the design of ontologies used for knowledge sharing?" *International journal of human-computer studies*, vol. 43, no. 5-6, pp. 907–928, 1995.

[26] F. Qin, S. Gao, X. Yang, M. Li, and J. Bai, "An ontology-based semantic retrieval approach for heterogeneous 3D CAD models," *Advanced Engineering Informatics*, vol. 30, no. 4, pp. 751–768, 2016.

[27] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: science, services and agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

Yameng An received B.S degree in Computer Science and Technology in Hangzhou Dianzi University, China in 2018. Since then she has been working towards M.S degree in Hangzhou Dianzi University. Her research interests include embedded system, motion control, and IIoT.

Feiwei Qin received the Ph.D. degree in Computer Science and Technology from Zhejinag University, Hangzhou, China, in 2014. He is currently an Associate Professor with School of Computer Science and Technology, Hangzhou Dianzi University, Hangzhou, China. His research interests include artificial intelligence, knowledge representation, graphics and image process, and CAD.

Baiping Chen received his B.Sc. and M.Sc in Computer Science and Technology from Hangzhou Dianzi University, Hangzhou China, in 2002 and 2005. He is currently a teacher at the Institute of Intelligent and Software Technology, Hangzhou Dianzi University, Hangzhou, China. His research interests include embedded systems, software development methods and tools, intelligent control & automation.

Rene Simon received the Ph.D. degree in engineering from Otto-von-Guericke University, Magdeburg, Germany, in 2001. He is a professor of control systems at the Department of Automation and Computer Sciences, Harz University of Applied Sciences, Wernigerode, Germany. His major research fields include engineering of automation systems, especially industrial controllers. He is chairman of PLCopen and project leader IEC 61131-10 Ed. 1.0.

Huifeng Wu (M'18) received the Ph.D. degree in computer science and technology from Zhejiang university, Hangzhou, China, in 2006. He is currently a professor in the Institute of Intelligent and Software Technology, Hangzhou Dianzi University, Hangzhou, China. His research interests include industrial internet of things, software development methods and tools, software architecture, embedded system, intelligent control & automation.