IEC 61499 Ontology Model for Semantic Analysis and Code Generation

Wenbin (William) Dai, *IEEE Member*, University of Auckland, New Zealand, wdai005@aucklanduni.ac.nz

Victor Dubinin, *non-member*, University of Penza, Russia, victor_n_dubinin@yahoo.com

Valeriy Vyatkin, *Senior IEEE Member*, University of Auckland, New Zealand, v.vyatkin@auckland.ac.nz

***Abstract* – The IEC 61499 standard is targeting the enhancement of the IEC 61131-3 PLC standard in distributed automation systems improving the reconfigurability, portability and reusability of automation software. This paper proposes a foundation to a novel approach for design the system based on IEC 61499 function blocks. This design process is including design recovery, configuration, semantic analysis and code generation using ontology models as the knowledge base. The IEC 61499 ontology model is provided as well as the semantic enrichment and corrections are described. The semantic analysis for IEC 61499 and the entire framework for code generation will be completed in the future.**

***Index Terms*— IEC 61499, Ontology, Knowledge base, Semantic Analysis, Reasoning, Description Logic.**

I. INTRODUCTION

To handle the increasing design complexity in the automation industry, the distributed reference architecture of the IEC 61499 standard [1] was created as an improvement for the centralized Programmable Logic Controllers (PLC) architecture of the IEC 61131-3 standard [2]. Instead of cyclical execution model defined in IEC 61131-3, the IEC 61499 introduced event-triggered function block execution model. This execution semantic is aimed for providing more efficient processing model for PLCs and also standardizes the inter-communication between various devices. Also the reconfigurability and reusability of the function block based design structure of IEC 61499 is much better than of IEC 61131-3.

It has been proven in the recent research (e.g. [3], [4]) that IEC 61499 can implement modern software engineering design methods, for example object-oriented design patterns, much better than PLC tools and languages. This improves the design efficiency substantially but, on the other hand, results in quite complex graphical programs. Checking their syntactic and, especially, semantic correctness becomes a challenge. This can be partially overcome by applying some "correct by design" methodologies, for example, using the physical structure of the plant as a starting point for automatic code generation. This requires some "semantic glue" between the two worlds: physical object structure and its automation software.

Last but not least, it is important to automate the re-use of existing IEC 61131-3 PLC code in the newly developed IEC 61499 function block solutions. To maximize the reusability of the original IEC 61131-3 PLC code, several redesign methodologies have been investigated in [5], [6] and [7]. However all those methodologies require certain level of extra human efforts to put on the migration process and easily cause inaccuracy of the result. All those approaches are not able to convert the PLC system to function blocks 100% accurate automatically and also the original execution semantics can vary in the new converted function blocks system. Again, a formal semantic link between the concepts of PLC programming and function block programming can simplify the migration task.

To provide such a multi-purpose semantic glue, a knowledge model for representing the related concepts of the function blocks architecture can be of great help. The ontology mechanism [25] can be considered as a proper representation of such a knowledge base. The word "Ontology" is from philosophical study of the nature of being, existence or reality originally and now is widely deployed into the semantic web programming and service-oriented architecture (SOA). Recently this concept is introduced into the automation and control area [12]. Ontologies are majorly applied to describing manufacturing and processing plants. For example, there are some researches on providing a standard ontology for manufacturing domain, for instance, MASON [8]. This paper aims to provide a general ontology model for the IEC 61499 standard which is focusing on analysis of system level design as well as execution semantic recovery from the code level design.

The W3C Consortium published an international standard of ontology language for semantic web – OWL [9]. OWL is an extension of RDF (Resource Description Framework) which is considered as the fundamental data model of semantic web programming. Similar to RDF, OWL is also based on XML. There are three versions of OWL: OWL-Full, OWL-DL and OWL-Lite. OWL-DL is selected as our target language as:

- The complexity of OWL-DL is between OWL-Full and OWL-Lite which is enough for handling the scenarios.
- Reasoning can be applied to the ontology model of OWL-DL which is based on description logic (DL) [10].

For ontology editor and viewer, Protégé [11] is the most commonly used tool in academic research for ontology design and development. It is an open source tool originally developed at the Stanford University. Protégé supports not only editing ontology concepts, properties and individuals and display ontology in graphical view but also the ontology query and reasoning based on the description logic (using Plug-ins). We will use Protégé as our editor and viewer tool for prototyping the IEC 61499 ontology here.

The paper is structured as follows: Section II provides a brief

overview of the ontology concept in computer domains which will be used in this paper. The related research works on the IEC 61499 ontology are reviewed in section II. The concepts and processes of the code generation for IEC 61499 are illustrated in section III. In section IV, a complete ontology model for IEC 61499 is discussed where properties are defined for all the IEC 61499 objects. Then semantic correction and enrichment are provided in section V. Based on this model, ontology reasoning and semantic analysis are discussed in section VI. Finally, future work and conclusion are attached in section VII.

II. Related Works

An IEC 61499 ontology is proposed in [12] for semantic analysis of IEC 61499 compliant systems. The function block type ontology model includes basic, composite and service interface function blocks and the system configuration. The ontology model for any function block type includes model of its interface. Along with that, the ontology model of a basic function block includes the Execution Control Chart (ECC) ontology model. In the ECC ontology model, EC state, EC algorithm, EC Transition and EC Transition conditions are defined. Instead of ECC model, the composite function block ontology model includes references to component function block instances along with models of event and data connections. Finally, the system configuration ontology model contains devices, resources, applications, links, mappings and network segments and their object and data properties. That paper provides examples of semantic analysis for IEC 61499 files using description logic and Semantic Web Rule Language (SWRL) [16]. A semantic analysis tool using Protégé plug-in is developed for automatic semantic checking.

In [13], the idea of semantic modelling with ontologies was applied to Automation Objects [22-24]. The automation object concept is based on the IEC 61499 function block model generalizing it to represent mechanical, electrical and software model. The automation object is stored in the form of UML and semantics of automation objects is defined in OWL/XML. To describe semantics of IEC 61499 system models and function block networks, the hierarchy OWL/XML structure is used to illustrate the complete system behaviour from not only software contents but also mechatronics information. The automation object ontology is compatible with the Protégé tool too.

The paper [14] is more focused on reconfigurability of the control system during the real-time execution for multi-agent systems (MAS). To achieve reconfiguration of a manufacturing system on the fly, the IEC 61499 standard best suits the requirements. The approach provided for MAS is the combination of ontological representation for low level control and a UML type format for high level control. The reasoning can be applied to low level control part which provides flexibilities to manufacturing system as well as predicting ability to reconfigure the entire system during the processing.

An ontology based reconfiguration system is also using IEC 61499 in [15]. To achieve rapid reconfiguration during the manufacturing process, the agent is built based on an ontological model. This model contains all knowledge about the manufacturing process. With any change during the physical environment, the agent will react without any human intervention. This automatic reaction is based on ontology query and reasoning. Once the reaction is decided from the agent, the system configuration will be reconfigured immediately to respond the physical change.

III. IEC 61499 Ontology for Design Recovery and Code Generation

The design recovery and code generation framework based on ontology model for IEC 61499 is outlined in figure 1. The design process starts with any IEC 61499 Editor tools, for example, NxtStudio [20] or FBDK [21]. When the system design is completed, the Design Importer is used to import all project files into the knowledge base and create all individuals according to IEC 61499 DTDs or XML schema. Those instances in the knowledge base together with the pre-defined IEC 61499 ontology model will be handed over to the semantic analysis engine to ensure absence of design errors introduced by a human. The verified design will be generated again back into IEC 61499 XML format. The advantages of this framework are:

- Even without any IEC 61499 editor, with instances created in the framework, the IEC 61499 code can be automatically generated.
- All human mistakes and typo errors during the design can be eliminated prior to the testing phase.
- The design is recoverable from the operation code. The code capture engine will use the DL reasoner to detect all individuals from the code. The changes made outside the framework can be read back to A-Box. The premise of this mechanism is the design pattern and instance names are not changed during the manual modifications.
- This framework is independent from any specific editor tool and is compatible with all the tools as well as long as the XML Schema is provided.

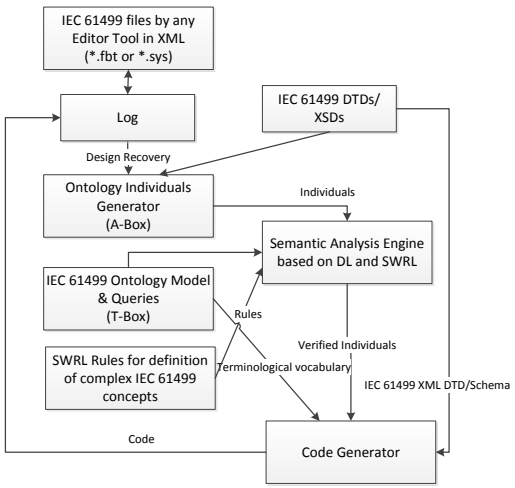

Fig.1. IEC 61499 design framework based on ontology.

IV. Layered IEC 61499 Ontology Model Definition

The IEC 61499 ontology in [12] was manually developed leaving questions of how adequate is it to the text of the standard and, especially, to particular implementations, which

may slightly deviate from the standard or may extend its insufficiently defined parts. Therefore, in this work we propose a layered approach to structuring the ontology, where the base level is automatically generated from XML schema that captures most of syntactic properties and is used directly by IEC 61499 tools for syntactic analysis. This approach promises to have less discrepancies between the code syntax supported by a tool and its semantic analyser. Moreover, the layered approach promises better extensibility of the ontology, or possibility to customize it for a particular dialect or design pattern.

A basic knowledge base comprises two components: a T-Box and an A-Box [10]. T-Box stands for taxonomy box which describes general properties of the concepts. A-Box or assertion box which retains knowledge that is specific to the individuals of the main of discourse. In IEC 61499 terms, T-Box is the knowledge base of all properties and relationships between IEC 61499 concepts. All actually implemented IEC 61499 system configurations and function blocks belong to A-Box. So the IEC 61499 ontology is defined as:

$$K = (T, A)$$

where

- K is the IEC 61499 Ontology Knowledge Base;
- T is the Terminological Axioms (Rules) of IEC 61499;
- A is the Assertional Axioms (Individuals) of IEC 61499.

Here T-Box contains any finite set of IEC 61499 general concept inclusion, so T has the form of

$$C \sqcup D \ (R \sqsubseteq S) \text{ or } C \equiv D \ (R \equiv S) \quad [10]$$

where

- C and D are concepts of IEC 61499;
- R and S are roles of IEC 61499.

Now the concepts of IEC 61499 must be clarified. According to the IEC 61499 standard, there are three major domains of concepts from the programming point of view: Library Elements, Function Block Management Commands and Data Types. Library Elements comprise all IEC 61499 elements including function block types, system configuration, resources, etc. Function block management commands define the communication protocol between IEC 61499 devices. Finally, all data types allowed in IEC 61499 are listed. Overall the IEC 61499 ontology structure is given in figure 1, where three major domains are directly connected to IEC 61499 as top level entity. All sub-domains are defined under major domains:

Library Elements ≡ Common Elements ⊔ FB Types ⊔ Adapter Types ⊔ Resource Types ⊔ System Elements ⊔ Sub-Application Types ⊔ Network Elements.

and

Function Block Management Commands ≡ Common Elements ⊔ FB Types ⊔ Data Types ⊔ Adapter Types ⊔ Requests ⊔ Responses.

As no repeatable concept name is allowed in ontology definition, all the IEC 61499 ontology concepts are named as *DomainName_SubDomainName_ConceptName* to avoid confusion. If there is no subdomain for a domain like DataType, the IEC 61499 ontology concepts are named as *DomainName__ConceptName* instead.

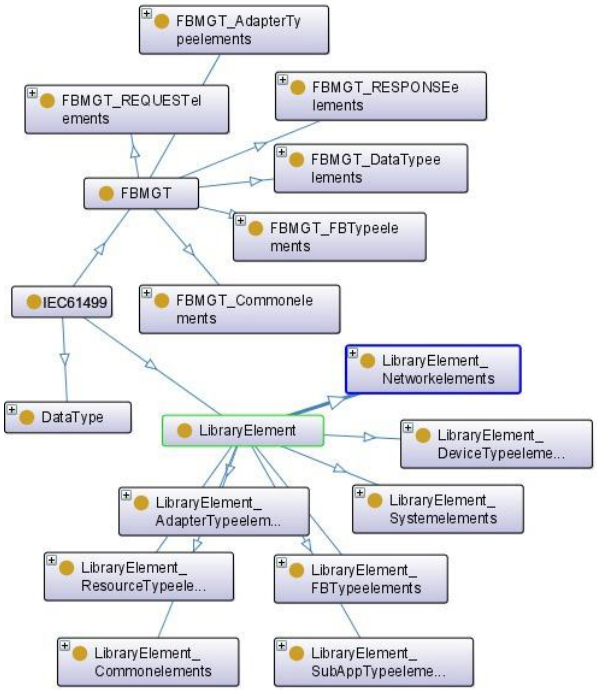

Fig.2. IEC 61499 Ontology Model Overview.

An IEC 61499 concept, or in another word – element, are linked to other elements via some description logic relationships. In the term of OWL, these description logic relationships are called properties. There are two types of properties in the OWL ontology: Object Property and Data Property. An object property is used to describe a property value that refers to another object. Correspondingly, the data property is used when the property value refers to the actual data literal or a data type. In addition, extra information shall be stored in annotation properties. A concept $C_i$is defined as:

$$C_i = (D_i, PD_i, PO_i, PA_i, DT_i, f_i)$$

where

- D_i is a set of IEC 61499 Ontology Concept Elements related to C_i;
- PD_i is a set of Data Properties belonging to C_i;
- PO_i is a set of Object Properties belonging to C_i;
- PA_i is an Annotation of C_i;
- DT_i is a set of Data Types used by definition C_i;
- f_i is a constructor which builds a DL expression from D_i, PD_i, PO_i, DT_i for the concept C_i.

As this ontology is mainly used for semantic analysis and code generation, the object properties are mainly containing the hierarchy of the IEC 61499 code structure. When using an object property to represent an element requiring another element, the name of this object property is defined as *Has_DomainName_SubDomainName_ConceptName*. To complete this object property, domains and ranges are compulsory. The domains are the classes where this object property will be used from and ranges are the classes where this object property will be applied to. An object property is capable to be used in multiple locations in the same ontology model. Data properties are utilized to represent the attribute values of elements in IEC 61499. When using a data property to present the constant value of a data type in the attributes of elements, the data property is naming as *Has_DomainName_SubDomainName_ConceptName_AttributeName* or *Has_ConstantValue_DomainName_SubDomainName_ConceptName* when the element itself is a constant value. Similar to object property, domain and range

are required as well. In the data property, domains are the locations where this data property will be used from and ranges are the pre-defined data types in the XML Schema and OWL.

The idea of IEC 61499 properties ontology definition will be illustrated on the concept of FBType of IEC 61499. The keyword FBType is a function block type of basic, composite or service function block in IEC 61499. A function block type comprises of Basic FB type or FB Network (Composite FB) or Service (Service Interface FB) associated with an interface regardless function block types. Beyond those essential parts, there might be some extra details including compiler information, version information, etc. For the data properties, a function block must have a name and may have some comments.

First step of creating a class is to create all data properties used in this class. For instance, FBType must have a name of data type String (Characters) in description logic:

$$\exists \text{ Has\_FBType\_Name.String}$$

And then convert into OWL format in Fig. 3:

```
<rdfs:subClassOf>
  <owl:Restriction>
    <owl:onProperty rdf:resource="#Has_FBType_Name"/>
      <owl:qualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
      </owl:qualifiedCardinality>
    <owl:onDataRange rdf:resource="#String"/>
  </owl:Restriction>
</rdfs:subClassOf>
```

Fig.3. IEC 61499 Data Property Example of FBType.

Next step is to create all object properties for this class. An FBType either has a Basic Function Block description or Function Block Network (Note: For simplicity we use here "short" concepts properties names and do not consider Service Interface FB):

$$(=1 \text{ Has\_FBNetwork.FBNetwork} \mid =1 \text{ Has\_BasicFB.BasicFB})$$

and then convert into OWL format in Fig. 4.

```
<rdfs:subClassOf>
  <owl:Class>
    <owl:unionOf rdf:parseType="Collection">
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Has_BasicFB"/>
        <owl:onClass rdf:resource="#BasicFB"/>
        <owl:maxQualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxQualifiedCardinality>
      </owl:Restriction>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Has_FBNetwork"/>
        <owl:onClass rdf:resource="#FBNetwork"/>
        <owl:maxQualifiedCardinality
          rdf:datatype="&xsd;nonNegativeInteger">1
        </owl:maxQualifiedCardinality>
      </owl:Restriction>
    </owl:unionOf>
  </owl:Class>
</rdfs:subClassOf>
```

Fig.4. IEC 61499 Object Property Example of FBType.

The figure 5 is a simplified graphical version of the FBType concept. One sees from figure 5 that, an FBType individual must have exactly one interface, name, maximum one service, identification, version information, compiler information and comments, and maximum one Function Block Network or Basic Function Block description.

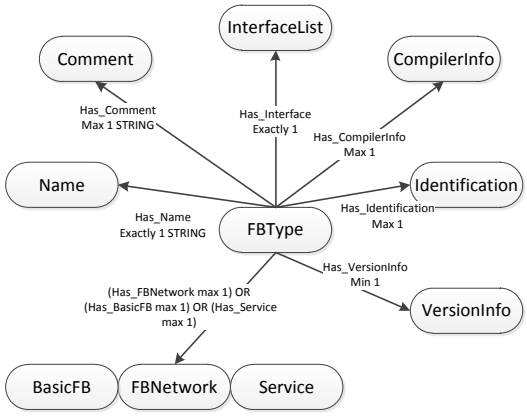

Fig.5. FBType Concept Ontology.

In order to quickly define the IEC 61499 ontology T-Box, an automatic converting methodology is required to avoid human errors during manual processes.

According to [17], a XML file schema is possible to be converted to OWL file automatically. A DTD/XSD to OWL engine is developed for automatic generation of ontology from XML Schema. The IEC 61499 standard provides 3 standard DTDs for Library Elements, Function Block Management Commands and data types to clarify the IEC 61499 XML format. Each DTD file is considered as a domain in ontology and converted and combined by the DTDtoOWL engine into a single IEC 61499 ontology model. The mapping process is presented as following:

1) Each DTD document is considered as a domain in the ontology. DTD Elements are grouped into sub-domains based on the catalogue mentioned previously.
2) Each DTD Element is mapped to an OWL class. In order to give a unique ID to each class, domain and sub-domain names must be added as prefixes for class ID.
3) The hierarchies of the DTD Elements are mapped to the object properties and if the Element only has constants, they are mapped to the data properties straight away by using prefix "*Has_ConstantValue_*". In a standard DTD document, there are some symbols to indicate occurrence of an element as stated in Fig. 6:

Symbol	Comment
*	Declaring Zero or More Occurrences of an Element
+	Declaring Minimum One Occurrence of an Element
?	Declaring Zero or One Occurrences of an Element

Fig.6. DTD Element Occurrence Symbols [16].

The OWL keyword owl:QualifiedCardinality is used to represent those occurrence symbols as showing in Fig. 7:

Symbol	Mapping in OWL
*	Owl:minQualifiedCardinality = 0
+	Owl:minQualifiedCardinality = 1
?	Owl:maxQualifiedCardinality = 1

Fig.7. DTD Element Occurrence Symbols Mapping in OWL.

4) The attributes of an element are mapped to data properties. There are two sorts of attributes:

#REQUIRED and #IMPLIED. Required attribute is mapped to Owl:QualifiedCardinality and with exactly 1 in quantity. Implied attribute means not necessary appear in the XML and better map to Owl:maxQualifiedCardinality = 1.

According to the rules above, the developed DTD to OWL engine is able to generate the complete IEC 61499 ontology T-Box. Finally any instance of a function block type (*.fbt) or a system configuration (*.sys) belong to the A-Box.

V. Semantic Correction and Enrichment For IEC 61499 Ontology Model

The semantic information contained in the *initial* ontology generated from the XML schema of IEC 61499 is quite limited. Also the terms used by DTDs are not 100% equivalent to the terms defined in the IEC 61499 standard text. Therefore some of the IEC 61499 standard terms are totally missing in the generated ontology model. In order to establish a useful ontology model for semantic analysis, semantic enrichment is required, which consists of two parts: syntax correction and execution rules enrichment.

First, all syntax must be 100% accurate. In the DTD/XSD based ontology, a function block type FBType includes maximum one function block network or one basic function block and maximum one service. Indeed, this is syntactically correct but semantically incorrect. A function block type must be a basic, composite or service interface function block. According to the DTD (reflected in the generated FBType ontology definition), an FBType can have one FB Network and also a Service which only exists in Service interface function block. To avoid the confusion, three sub-concepts of FBType are introduced here: BasicFBType, CompositeFBType and ServiceInterface FBType. The definitions are illustrated in Figure 8. According to the textual syntax of the IEC 61499 standard, the basic FB Type only includes exactly one BasicFB and common parts of FBType. Same for composite and service interface function block, only one FBNetwork or Service is allowed.

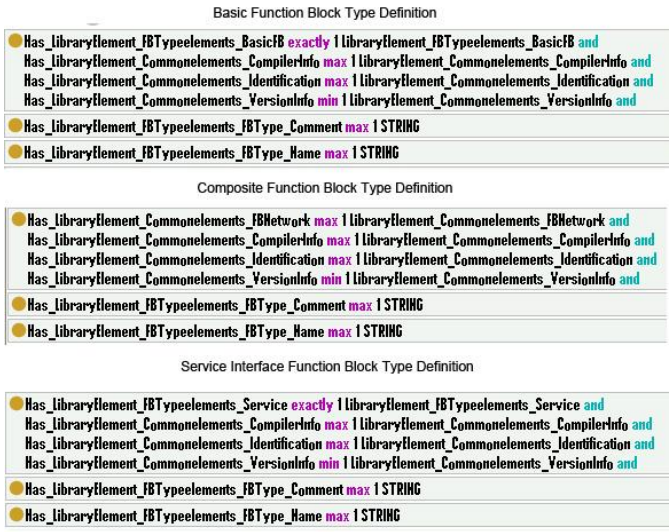

Fig.8. New Sub-FBTypes in Protégé.

Beyond function block types, for an EC action attached to an EC State, the definition of ontology is that one or more algorithm and one or more output can be attached to an EC action. Indeed, the IEC 61499 standard textual syntax states that an EC action can have at least one algorithm or output. The initial ontology model also allows none of these to be present which contradicts the standard.

Secondly, the execution rules must be inserted into the ontology model. For instance, in the generated ontology model, event, data and adapter connections are defined as a common connection attribute which has the source and the destination. Unfortunately, there is no limitation of quantities of source and destinations that can be connected in the generated ontology model. However, it is defined in the IEC 61499 standard text that each data input of a component function block can only be connected to no more than one data output of exactly one other component function block, or to no more than one data input of the composite function block. The extra data properties must be added for checking there is only one data input connecting to one data output and the source and destination data variables are the same data type or the data types can be cast.

Also some of the data attributes in the initial ontology model actually refer to some classes but are imported just as a data variable with the class name. In the ontology definition, a component function block has a function block type with a data property of string type. So, in the ontology term - FB Has_LibraryElement_Commonelements_FB_Type exactly one string is allowed. But actually this attribute FB_Type refers to the element FBType already existed in the ontology. For semantic analysis, an object property of equivalent class is created for linking the attribute type of FB to the FBType element.

Last but not the least, detailed connection definitions between events, data, adapters and sub-applications are missing from the original DTDs and the generated ontology model. According to the IEC 61499 standard textual syntax, all sorts of connections are clearly defined the source and destination of each end. Each end of the connection also specifies the function block instance name and input/output variable instance name. In our initial ontology model, this is just a string variable. No semantic information about any particular connection is included. Those details including sources and destinations are added as extra object properties in the ontology model to describe how the input/output of function block instances are linked together.

VI. Semantic Analysis of IEC 61499

Unlike syntax check, semantic analysis of programs cannot be completely performed by software tools. However, many modern compilers implement some semantic analysis functions. However, the methods of such analysis are usually hard coded in the compilers. In this work we propose a more configurable approach, when the semantic check engine will be able to check properties specified in the ontology model resulted from the syntax correction and semantic enrichment. Semantic analysis is mainly for checking variable, syntax, data type, function and linking are semantically correct or valid. To fulfil semantic analysis of this ontology model, reasoning is applied. The key of ontology reasoning is to test consistency of an ontology concept. From [19], to build a reasoner for the ontology, the procedures are summarized as:

- Build a Tree-View model of the ontology concept;
- Decompose this concept syntactically by applying the Tableau algorithms;
- For nondeterministic rules in the Tableau algorithms, a search is necessary;
- Ensure no clash occurs for rules and stop when no more rules applicable.

Following those principles, semantic analysis for an IEC 61499 system design starts with the ontology reasoning query. A description logic expression shall be built which is containing all properties of this particular ontology concept. For example, to analyse a basic function block design, the description logic expression shall be rearranged as:

$$\text{BasicFB} \sqsubseteq \text{FBtypeelements} \sqcap \leq 1\ \text{Has\_ECC.ECC} \sqcap \leq 1\ \text{Has\_InternalVals.InternalVars} \sqcap \exists\ \text{Has\_Algorithm.Algorithm}$$

This description logic formulae states that a class of basic function block descriptors is a sub-class of intersection of the following four classes: 1) the class of function block type elements; 2) a class where every individual has maximum one ECC, 3) a class where every individual has maximum one internal variable list; 4) a class where every individual has some algorithms. To complete this tree-view model, each concept involved here must be attached with its own description logic formulae. Such as for ECC,

$$\text{ECC} \sqsubseteq \text{FBtypeelements} \sqcap \exists\ \text{Has\_ECState.ECState} \sqcap \exists\ \text{Has\_ECTransition.EC\_Transition}$$

Description logic expressions need to be passed to the semantic analysis engine (*reasoner*). The engine will perform the automatic semantic analysis test based on the query ontology concept. The engine's structure is shown in figure 9.

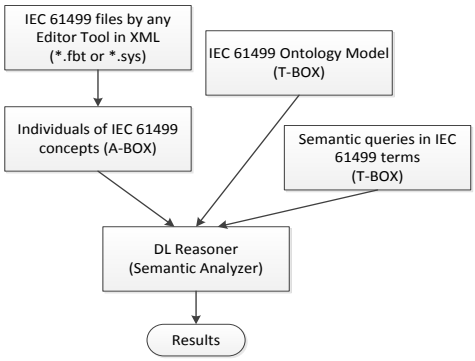

Fig.9. Semantic analysis engine for IEC 61499 ontology.

VII. Conclusions And Future Work

This paper presented an IEC 61499 knowledge base using the ontology model. This knowledge base of IEC 61499 is generated from the IEC 61499 XML schema files and all relationship between IEC 61499 concepts are properly defined. The ontology model is enhanced by semantic corrections and enrichment. This ontology model is suitable for the semantic analysis and the code generation for systems using the IEC 61499 standard.

This research work will continue with completing the IEC 61499 semantic analyser based on DL and SWRL. The semantic analysis shall automatic detect any semantic errors. The code generator for IEC 61499 will be developed as well as the code capture engine. This will complete the design framework for IEC 61499.

VIII. References

[1] IEC 61499, Function Blocks, *International Standard*, 2005

[2] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Second Edition, 2003

[3] V. Vyatkin, J. Christensen , J. Lastra, "OOONEIDA: An Open, Object-Oriented Knowledge Economy for Intelligent Distributed Automation", *IEEE Transactions on Industrial Informatics*, vol. 1, 2005

[4] A. Zoitl, V. Vyatkin, "IEC 61499 Architecture for Distributed Automation: the 'Glass Half Full' View", *IEEE Industrial Electronics Magazine*, 3(4), pp. 7-23, 2009, doi: 10.1109/MIE.2009.934789

[5] W. Dai, V. Vyatkin, "Redesign Distributed IEC 61131-3 PLC System in IEC 61499 Function Blocks", *15th IEEE International Conference on Emerging Technology and Factory Automation*, 13-16 September 2010, Bilbao, Spain.

[6] M. Wenger, A. Zoitl, C. Sunder, H. Steininger, "Semantic Correct Transformation of IEC 61131-3 Models into the IEC 61499 Standard", *2009 IEEE Conference on Emerging Technologies & Factory Automation*, Mallorca, ISBN:978-1-4244-2728-4, 7 Pages.

[7] T. Hussain and G. Frey, "Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 2005, pp. 3984-3989

[8] S. Lemaignan, A. Siadat, J.Y. Dantan, A. Semenenko, "MASON: A Proposal For An Ontology of Manufacturing Domain", *Proceedings of the IEEE Workshop on Distributed Intelligent Systems*: Collective Intelligence and Its Application (DIS), 2006.

[9] Web Ontology Language – OWL: http://www.w3.org/TR/owl2-overview/

[10] F. Badder, D. Calavanese, D.L. McGuinness, D. Nardi and P.F. Patel-Schneider, "The Description Logic Handbook, Theory, Implementation and Applications, 2nd Edition.", *Published by Cambridge University Press*, 2007, ISBN 978-0-521-87265-4.

[11] Protégé, a free, open source ontology editor and knowledge base framework, http://protege.stanford.edu/

[12] V. Dubinin, V. Vyatkin, "Ontology of IEC 61499 function blocks" , *International conference "Contemporary information technologies"* (CIT'10), Penza, December, 2010

[13] O. Orozco, J. Lastra, "Adding Function Blocks of IEC 61499 Semantic Description to Automation Objects", *IEEE International Conference on Emerging Technologies and Factory Automation*, Prague, Czech Republic, September, 2006, pp 537 – 544.

[14] W. Lepuschitz, A. Zoitl, M. Valee and M. Merdan, "Towards Self-Reconfiguration of Manufacturing System Using Automation Agents", *IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews*, Vol. 41, No. 1, January 2011

[15] Y. Alsafi , V. Vyatkin, "Ontology-based Reconfiguration Agent for Intelligent Mechatronic Systems in Flexible Manufacturing", *International Journal of Robotics and Computer Integrated Manufacturing*, DOI: 10.1016/j.rcim.2009.12.001, 2009

[16] SWRL – A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/

[17] P. Thuy, Y. Lee, S. Lee, "DTD2OWL: Automatic Transforming XML Documents into OWL Ontology", *2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, 16–18 Aug 2009, ISBN: 978-1-60558-710-3

[18] DTD Document, W3C Standard, Retrieved from http://www.w3schools.com/dtd/dtd_elements.asp

[19] I. Horrocks, "Description Logic Reasoning", Lecture Notes from University of Manchester, UK, 2005.

[20] nxtControl GmbH, nxtControl - Next generation software for next generation customers [Online, 2009, June]. Available: http://www.nxtcontrol.com/

[21] Function Block Development Kit, from Holobloc Ltd, Available: http://www.holobloc.com/

[22] Automation Objects for industrial-process measurement and control systems – IEC SB3/TC 65, working draft, 2002.

[23] Brennan, R.W., Ferrarini, L., Lastra, J.M. and Vyatkin, V. (2006) 'Automation objects: enabling embedded intelligence in real-time mechatronic systems', *Int. J. Manufacturing Research*, 1(4), 379–381

[24] Vyatkin V., Karras, S., Pfeiffer, T., H.-M. Hanisch, Dubinin V., Rapid Engineering and Re-configuration of Automation Objects Using Formal Verification, *Int. J. Manufacturing Research*, 2006, l(4), 382–404

[25] Ontology General Definition, http://semanticweb.org/wiki/Ontology