Ontology-based Design Recovery and Migration between IEC 61499 - compliant Tools

Wenbin (William) Dai, *Graduate IEEE Member*, wdai005@aucklanduni.ac.nz
Valeriy Vyatkin, *Senior IEEE Member*, v.vyatkin@auckland.ac.nz
Victor Dubinin, victor_n_dubinin@yahoo.com

***Abstract* – This paper proposes a new method of semantic design recovery of automation applications, with source code used to generate ontological semantic model of IEC 61499 application. The model can be used for automatic semantic analysis of the system and automatic code generation for various IEC 61499 tools. This method creates a foundation for correct-by-design development tools and automatic migration between different tools. The semantic enrichment and analysis are fulfilled via ontology reasoning and query. This complete design loop is implemented in a software tool including code capture engine, semantic analyzer and code generation engine with the pre-defined IEC 61499 ontology model. The method is demonstrated on a simple control system case study.**

***Index Terms*— IEC 61499, Function Blocks, Ontology, Description Logic (DL), Semantic Analysis, Ontology Reasoning, Code Generation and Design Recovery**

I. INTRODUCTION

The IEC 61499 standard [1] aims at programming next generation of automation systems following PLCs compliant with IEC 61131-3 standard [2]. One of the key features of the IEC 61499 standard is component based design which increases the overall level of design and performance of developers. The component based design gives an opportunity to increase the design level even more by attaching semantic categories to each component. The next generation tools would be able to support programming in terms of semantically rich operations, rather than just blocks. Such design process would require design recovery and semantic check on code compliance with certain design patterns that requires intelligent meta-modelling support.

Another practical problem is migration from legacy systems. In many application domains, the existing base of IEC 61131-3 applications is large and therefore migration strategy is important. The migration IEC 61131-3 PLCs to IEC 61499 function block systems is considered as an intermediate step of introducing distributed control. During the migration process, reusing some PLC code is an essential procedure for cost saving and reduce development time. It has been shown in previous work [3] that the PLC design pattern can be reused in function block systems. Other approaches from [4-7] demonstrate the migration is possible but lots of human efforts are required. A new design approach is still needed to code transform automatically from one platform to another.

The IEC 61499 system design is completed by using an IEC 61499 IDE for instance, FBDK [8] by Holobloc, or NxtStudio by NxtControl [9]. These IDEs provide not only editing functionalities but also function block run-time environments which the developed system could be deployed into supported hardware platforms. The developed IEC 61499 system is also distribution-transparent due to its' component-based organisation. The overall usability of such system is a big step ahead as compared to same implementation in IEC 61131-3 PLCs.

Along with benefits, the active use of IEC 61499 tools for complex application development has revealed some problems. For example, creating instances of function blocks and manually connecting each input and output are still time consuming. Human errors or typos also happen easily during this process. Secondly, though the function block design is self-explanatory, it is still not sufficiently clear for understanding systems' semantics especially to project managers, engineers and technicians with weak programming background. Finally, while FBDK, NxtStudio and other IEC 61499 IDEs are all following the standard, there are still some implementation variations which lead to incompatibility between them. In order to solve those issues, a more generic design approach would be desirable to be seamlessly applicable for all IEC 61499 standard supported tools.

Those issues can be solved by applying the new approach using ontology and description logic proposed in this paper. A function block application can be used as a source to generate platform independent ontology knowledge base. Then, version for a particular platform can be generated automatically from the knowledge base.

The rest of the paper is structured as follows: The general information on the IEC 61499 standard is given in section II. In section III, the details of new design approach is provided. Several reviews of relevant papers are given in section IV. The entire ontology model of IEC 61499 is given from section V. This section also discusses the ontology query engine which is involved in the code generation process. In section VI, the design recovery from code level to knowledge base process is described. Migration from different IEC 61499 platforms are also described here. In Section VII, the syntactic checking and semantic analysis are performed for the recovered design. The code generation procedures are given in Section VIII. A case study is given through the entire paper. In the final part, the paper is concluded and future works are listed.

II. BRIEF INTRODUCTION TO IEC 61499

The artefacts of the IEC 61499 function block architecture are briefly discussed as follows:

Function Block (FB) – is a module with interface that consists of event and data inputs and outputs. The events also will be further referred to as signals. A function block can be invoked only by an input event. There are three types of function blocks: basic, composite and service interface function block.

The functionality of a basic function block is defined as a state machine called Execution Control Chart (ECC). The semantics of ECC is similar to Moore finite automata with actions assigned to states. An action consists of an algorithm and output event issuance (either can be omitted). An EC-transition has a condition "clocked" by no more than one event input and having a guard condition that is a predicate over data inputs and internal variables (but no events).

A Composite Function Block is specified by interface and functionality, defined as a network of function block instances interconnected via event and data connections. A service interface function block (SIFB) for the purposes of this paper can be understood as a "black box" whose internal structure is not specified.

On top of function block types, a Function Block Application (FBA) is also a network of function block instances, but it has no interface. An FBA is a highest level structure in the hierarchy of IEC 61499 artefacts considered in this paper.

A system configuration combines instances of device types connected with communication network segments, applications, and their mapping into devices. For instance, in Fig 1 a simple manipulator control system is shown. The manipulator consists of two cylinders controlled by a joystick. Two cylinders are declared as Z1 and Z2 of basic function block type Cylinder. Limit switches for each cylinder are declared as SW1 and SW2 of basic function block type Switch respectively. This function block network is presented as FBA in the system configuration.

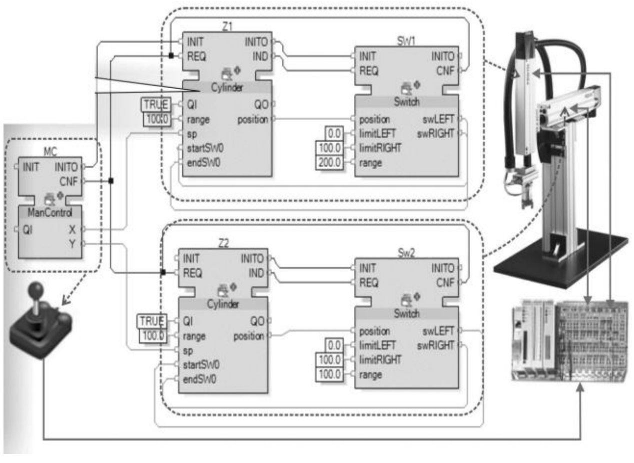


Fig.1. Case Study of Two Cylinders Example.

III. Generic Design Approach for IEC 61499

The proposed design approach aims at automatic design recovery from the code level, automatic syntactic checking, semantic analysis and code generation for various IEC 61499 platforms. The IEC 61499 platforms FBDK and NxtStudio are selected here as the example although there are many other tools available like 4DIAC-IDE [10], FBench [11], CORFU IDE [12]. The proposed design approach is illustrated in Fig.2.

In the top part of Fig.2, source code of an application or system following IEC 61499 and created in NxtStudio or FBDK tool, is imported into the knowledge base by the design recovery engine. The knowledge base is based on ontology and defined in the semantic web language OWL [13]. An ontological knowledge base consists of two parts: T-Box and A-Box. In the ontological terms, definitions are belonging to the taxonomy box (T-Box) which contains general properties of the ontological concepts. When this is applied to the IEC 61499 standard, a T-Box contains definitions of all IEC 61499 keywords (concepts) and the relationships between those IEC 61499 concepts. In terms of ontologies, the system configuration is a concept. A system configuration has some devices which are the also concepts. Those concepts are linked to the system configuration by using object properties. For another example, basic function block is also a concept in ontology. The basic function block type must contain no more than one execution control chart (ECC) is described as basic FB has an object property of exactly one ECC. Those concepts are described in the description logic [14] which ontology is mainly defined in. Various IEC 61499 dialects may have some different naming of same concept or some definitions which only supported by this platform. For example, nxtStudio introduced some keywords in the XML schema that do not exist in the IEC 61499 standard. As a result, the contents of T-Box for nxtStudio and FBDK are not 100% identical.

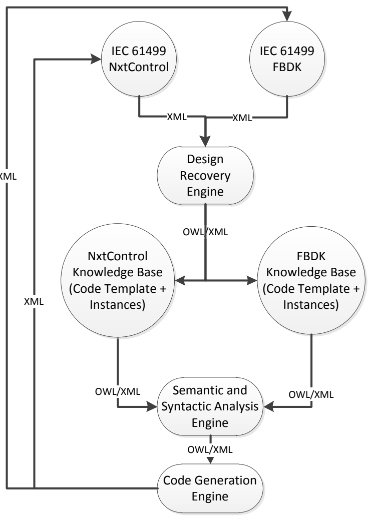

IEC 61499 NxtControl
IEC 61499 FBDK
XML
XML
XML
Design Recovery Engine
OWL/XML
NxtControl Knowledge Base (Code Template + Instances)
FBDK Knowledge Base (Code Template + Instances)
XML
OWL/XML
OWL/XML
Semantic and Syntactic Analysis Engine
OWL/XML
Code Generation Engine

Fig.2. IEC 61499 Design Loop.

In the other part of those knowledge bases, instances are stored into the assertion box (A-Box) which consists of knowledge that is specific to the individual system design. In the IEC 61499 terminology, an A-Box retains all system configurations, settings, parameters and function block instances. That information is saved as ontology individuals in the A-Box.

The XML source code of FB applications created in NxtStudio or FBDK is interpreted by the design recovery engine during the import process and stored into the related knowledge base in the OWL format which relies on XML as well.

At this point, the knowledge base contains all essential information required for code generation. However, there is still another step before the actual code generation can be performed. The syntactic checking and semantic analysis must be performed prior to code generation to ensure the generated code is syntactically 100% compliant with the target platform and the system execution behaviour is not changed during the design process. This task is handled by the syntactic and semantic analysis engine in the tool chain. All instances data in the A-Box are verified by this engine. If any issue is detected by the analysis engine, manual changes are allowed for correcting the design files.

The last step in the design chain is the code generation. Based on the target platform, respected code templates and related instances are combined by the code generation engine and output as IEC 61499 XML files. The generated code is capable to be opened again in the target platform IDE and immediately deployed using its mechanisms of compilation and deployment. If any change is made again in the IDE during the deployment and testing, the modified files can be re-imported again into the knowledge base. All changes made outside the tool chain are captured completely back.

IV. Related Works

Peltola et al. [15] investigated migration from PLC to IEC 61499 on example of batch process control applications. The original equipment is controlled by PLCs and the program is written in sequential flow chart (SFC). The SFC is a visual programming language provided in the IEC 61131-3 standard which naturally supports state machines way of thinking. The authors provide an approach to map the hierarchy of the system design into function block networks and convert from IEC 61131-3 SFCs into IEC 61499 ECCs. The test result proves it is feasible but all the works described in this paper are completed manually. The ideas of mapping components from IEC 61131-3 to IEC 61499, proposed in that paper, have motivated some of the solutions proposed in this work and also some of our future work that includes IEC 61131-3 into the loop.

Several design recovery methodologies are illustrated by Falcione et al.[16]. Authors focus on recovery the ladder logic (LD) and SFC program from programmable logic controllers (PLC). The design recovery algorithms for LD and SFC are provided. The results are represented in a graphical way which is further to be expressed in math equations. But this approach is not suitable for code generation as the result of this approach is not using an open standard format which could be easily interpreted.

Another design recovery approach is provided by [17]. For the re-engineering process of design recovery, authors suggest a formal specification language concept mapping language (CML) as a storage media. Terminology, expressions and grammar definition for the CML are provided. The expression definition of CML is similar to description logic used by ontology but transformations are still done manually. To achieve automatic transformation, this approach is also not applicable.

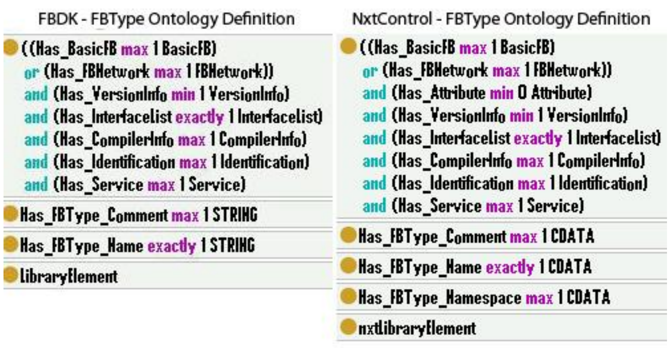
In [18], the authors of this paper developed a knowledge base using ontology for semantic analysis and code generation. In the ontological based database, all object properties and attributes are imported from the source code automatically. There are some essential semantic corrections and enrichments are required to correctly express the IEC 61499 system in the ontology model. In this paper, we continue using that model in the next section with extended flexibilities to support more platforms rather than just the standard IEC 61499 XML.

V. IEC 61499 Knowledge Base Generation

Knowledge bases are the core part of migration between various IEC 61499 platforms proposed in this paper. As shown in Fig. 2, each knowledge base consists of two parts: code templates and instances. The code template part contains base functions which will be reused as “components” in the function block design. In the IEC 61499 terms, all function block types (basic FB, composite FB and service interface FB) are considered as the fundamental reusable programming organization units (POU) in IEC 61131-3. Those fundamental POUs - function block type definitions are imported into the code template part in the knowledge base. The second part of the knowledge base corresponds to the system configuration. All information regarding the declarations of function block type instances, event and data connections between function block instances and the deployment configurations (for instance, devices and resources allocation for function block instances) are belonging to this part.

The T-Box definitions for NxtStudio and FBDK knowledge bases must be created prior to applying the migration procedure. The IEC 61499 standard is using XML representation as the standard code format. XML schema or a DTD are required to validate syntax of an XML file. FBDK uses the DTD provided within the IEC 61499 standard. NxtStudio is also using the standard DTD but with extended features. To generalize the ontology rules for various XML schemas and DTDs, an automatic conversion from DTD to ontology OWL is implemented to avoid time consuming and error prone manual conversions.

The rules of converting standard DTDs to OWLs are provided in [19]. In the paper [18], a general approach for auto generation ontology from the IEC 61499 standard DTD is given in details as well as the semantic corrections and enrichment for the DTD based ontology model. This approach is suitable not only for the IEC 61499 standard DTD but for any DTD. The ontology T-Box generation of FBDK is also given in [18]. By using the same approach, the ontology T-Box of NxtControl can be created automatically as well.



FBDK - FBType Ontology Definition

```
● ((Has_BasicFB max 1 BasicFB)
  or (Has_FBNetwork max 1 FBNetwork))
  and (Has_VersionInfo min 1 VersionInfo)
  and (Has_Interfacelist exactly 1 Interfacelist)
  and (Has_CompilerInfo max 1 CompilerInfo)
  and (Has_Identification max 1 Identification)
  and (Has_Service max 1 Service)
● Has_FBType_Comment max 1 STRING
● Has_FBType_Name exactly 1 STRING
● LibraryElement
```

NxtControl - FBType Ontology Definition

```
● ((Has_BasicFB max 1 BasicFB)
  or (Has_FBNetwork max 1 FBNetwork))
  and (Has_Attribute min 0 Attribute)
  and (Has_VersionInfo min 1 VersionInfo)
  and (Has_Interfacelist exactly 1 Interfacelist)
  and (Has_CompilerInfo max 1 CompilerInfo)
  and (Has_Identification max 1 Identification)
  and (Has_Service max 1 Service)
● Has_FBType_Comment max 1 CDATA
● Has_FBType_Name exactly 1 CDATA
● Has_FBType_Namespace max 1 CDATA
● nxtLibraryElement
```

Fig.3. FB Type Ontology Definition in FBDK and NxtControl.

Similar to the standard IEC 61499 DTD, each DTD element is mapped to an ontology class. Hierarchies of the DTD elements are described using the ontology object properties "Has_Subelements". And the attributes of each DTD element is created as the ontology data properties "Has_ElementName_AttributeName" with its value. In Fig. 3, the generated T-Box rules for function block type are given from both FBDK and NxtControl. Comparing the object properties, one sees a new DTD element "attribute" introduced in NxtControl, which can apply to FBType element.. Another additional data property is 'namespace'.

VI. Design Recovery from Code

After both T-Box definitions are created, the migration process starts with the design recovery from the code level. The first phase of design recovery is to capture the design pattern of the source IEC 61499 system. The design recovery engine searches through the system configuration and lists all device types, resource types and function block types included. Then it searches items in each device type and tries to capture any embedded resource type or function block type which is not currently in the lists. This search procedure repeats with each device type, resource type and composite function block type. All function blocks utilized in the internal FB network of a composite function block type shall also be added to the function block type list.

With all files using in the IEC 61499 system configuration are captured, the second phase of the design recovery process is to import all function block types into the code template part of the knowledge base based on the XML schema. The import process starts with checking if this function block type already exists in the knowledge base. If this is a new function block type, the engine will go through all XML elements and create a respective individual for each element in the ontology knowledge base. The relationships between elements are established using ontology object properties. Those object properties are attached to the ontology node and named as *<Has_ElementName>*. Attributes of each node are defined as the ontology data properties with a name of *<Has_ElementName_AttributeName>*. Normally, an XML node is defined as:

```
<Node Name="NodeName" AttributeName="AttributeValue">
   <ChildNode Name="ChildNodeName" ChildAttributeName="ChildAttributeValue" />
</Node>
```

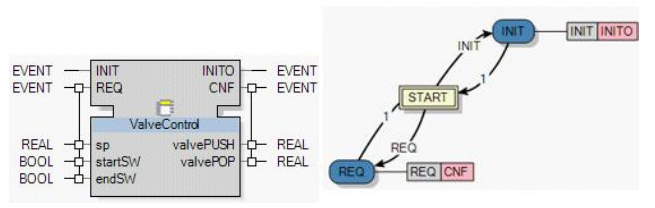
The generic mapping from the original XML format above to the OWL representation of any imported individual is defined as:

```
<owl:NamedIndividual rdf:about="NodeName">
   <rdf:type rdf:resource="Node" />
   <Has_ChildNode>ChildNodeName</Has_ChildNode>
    <Has_Node_AttributeName >AttributeValue</Has_Node_AttributeName>
</owl:NamedIndividual>
<owl:NamedIndividual rdf:about="ChildNodeName">
   <rdf:type rdf:resource="ChildNode" />
   <Has_ChildNode_AttributeName >ChildAttributeValue
      </Has_ChildNode_AttributeName>
</owl:NamedIndividual>
```

If this function block type already exists in the knowledge base, an update to the existing individual is taken place instead of creating a new OWL individual. This process also flattens the nested IEC 61499 structure without losing the system hierarchy. The flat ontology knowledge base is convenient for query and sort.

The FB Type Valve control of the cylinder control system is selected as the example illustrated in Fig. 4. The ValveControl is a basic function block type with two event inputs INIT/REQ and two event outputs INITO/CNF. There are two states in the ECC. In the INIT state, all values are cleared back to zero. In the REQ state, PUSH and POP valve positions are recalculated according to the current position (sp), start switch (startSW) and end switch (endSW) inputs values.

As shown in the bottom part of Fig. 4, the ValveControl is created as an owl individual with a resource type of FB Type. Referring to the ontology definition in Fig. 3, the object properties of FBType - BasicFB, Identification, VersionInfo, CompilerInfo and Interface list are linked to the related sub node individuals. The data property Has_FBType_Name retains the value of ValveControl as the function block type name. Similar to the ValveControl FB, all other FBs and the system configuration of the cylinder control demo can be recovered from the code-level by applying the same rules.



```
<owl:NamedIndividual rdf:about="#FBType_ValveControl">
   <rdf:type rdf:resource="#FBType" />
   <Has_Identification  rdf:resource="#FBType_ValveControl_Identification" />
   <Has_VersionInfo rdf:resource="#FBType_ValveControl_VersionInfo" />
   <Has_CompilerInfo rdf:resource="FBType_ValveControl_CompilerInfo" />
   <Has_InterfaceList rdf:resource="#FBType_ValveControl_InterfaceList" />
   <Has_BasicFB rdf:resource="#FBType_ValveControl_BasicFB" />
   <Has_FBType_Name rdf:datatype="STRING">ValveControl
   </Has_FBType_Name>
</owl:NamedIndividual>
```

Fig.4. Example of OWL Individual for the ValveControl FB.

VII. Syntactic and Semantic Analysis using IEC 61499 Ontology Model

Before the actual code generation can be performed, there are two more steps required. The first step is to ensure the target ontology model for code generation is syntactically correct. The syntax of IEC 61499 files are checked via standard XML parser with pre-defined IEC 61499 DTD files. After all files are checked the syntax using the XML parser, this generated function block system can be opened and edited by FBDK or NxtStudio, respectively.

The semantic analysis is mainly for checking behavioral properties which contains not only the correctness of connections of FBs and mapping of variables but also the execution semantics [20] like detecting event chain loop. The semantic rules are defined in SQWRL [21]. SQWRL is a query language to get information from the ontology similar to SQL database. SQWRL takes a standard SWRL [22] rule antecedent and effectively treats it as a pattern specification for a query. A single semantic rule is satisfied if the query result contains the individual name query asks for. In this paper, only one example is given to provide some brief overview. Detailed semantic rules can be found in [23].

One of the semantic rules is that a system configuration is semantically correct if for those event connection chains that are initiated from an E_CYCLE SIFB, no event connection loop exists in any event chain. One of the common IEC 61499 design issues is the infinite event loop chain. This problem may arise when there is a feedback loop in event connections between a group of FBs. This situation is exacerbated if event source E_CYCLE is connected to one of FBs in the group. As shown in Fig. 5, once the output event EO occurs, the system will keep looping through FB_ADD and FB_SUB infinitely. A new event EO is raised at the E_CYCLE output every 100ms so the number of events that need to be processed will be increasing with time. This semantic rule ensures that such situation would not occur.

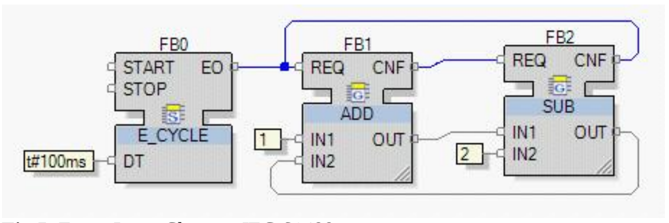


Fig.5. Event Loop Chain in IEC 61499.

To detect this situation, first step is to list all event inputs in the system configuration by using the SQWRL query:

```
EventInputs(?EventInputs) ^ Has_Event(?EventInputs, ?Events) ^
Has_Event_Name(? Events, ?EventNames) -> sqwrl:select(?EventNames)
```

For each event input in the result list, a search through event connections is performed for checking indefinite event loop. The next step is to detect whether this event input will trigger an event output of this function block unconditionally. If searching through the basic function block type, this event input must be used as the only condition on this EC transition and emit an event output. The SQWRL expression for getting input events used as the only EC transition condition and its destination state name is:

```
ECTransition(?ECTransitions) ^ Has_ECTransitionCondition(?ECTransitions,
?ECTranConds) ^ swrlb:stringEqualIgnoreCase(?ECTranConds, <EventName>) ^
Has_ECTransition_Destination(?ECTransitions, ?ECDests) ->
sqwrl:select(?ECTranConds, ?ECDests)
```

If the previous query returns the event input, then another SQWRL expression is used to check if this event input also emits an output event:

```
ECState(?ECStates) ^ Has_ECState_Name(?ECStates, ?ECStateNames) ^
swrlb:stringEqualIgnoreCase(?ECStateNames, <ECDest>) ^
Has_ECAction(?ECStates, ?ECActions) ^ Has_ECAction_Output(?ECActions,
?ECOutputs) ^ swrlb:stringEqualIgnoreCase(?ECOutputs, <EventName>) ->
sqwrl:select(?ECoutputs)
```

If an output event is triggered by this event, this output event will be continuously checking until the end of the event chain or finally loop back to the target input event. Similar procedures are used for composite function blocks. For service interface function block, instead of ECCs, service sequences and transactions are searched.

According to the system configuration in Fig. 1, event chain loops are detected from switch SW1 to cylinder Z1 and switch SW2 to cylinder Z2. However, as there is no indefinite event loop detected in this application. The chance of infinite event loop occurrence is very low as this application is completely event-triggered with no repeated input source. More details could be found in [24].

VIII. Code Generation Based on IEC 61499 Ontology Model

Once the ontology model of the IEC 61499 system passes both the syntax check and semantic analysis, the final step is to generate the code for the target IEC 61499 FB system. The generation process starts from the system configuration. All data properties and object properties of the system configuration node in the ontology is listed by the code generation engine. An XML root node <System> is created and all data properties are mapped to the XML attributes of the root node. The next step is to loop through all object properties and create sub nodes named as object properties. The data properties of each object property are attached as the XML attributes as well. This procedure is repeated again for every sub node until no more object property is found in the ontology model. Also if a function block type, resource type or device type is discovered during the code generation process, the same procedure will apply to the function blocks, resources and devices. Instead of using <System> as the root node, IEC 61499 keywords <FBType>, <ResourceType> and <DeviceType> are used respectively.

During the generation process, the data validation is also performed simultaneously. Data validation is used to check the data correctness for the selected target platform. For instance, date format used in the FBDK is “month/day/year” but NxtStudio is using “year-month-day”. When generating the FBDK code and the attribute data type is date, the value must compliance with the suggested format. If the data type is not matched and can be converted to the target data type, the data casting will be automatically applied to this attribute value.

The real challenge is to generate FBDK code with imported ontology individuals from NxtStudio. The mapping between two ontology definitions is introduced to solve this issue. The ontology mapping [25] is a technique which is used for merging two ontologies together or to find correspondences between semantically related entities of different ontologies. The ontology linking between NxtStudio and FBDK are created by using name based checking. As the NxtStudio knowledge base is the extended version of the FBDK one, all FBDK ontology concepts can be found in the NxtStudio ontology. The individual mapping from NxtStudio to FBDK is done by comparing the keywords. During the conversion process, if a concept name is found in both ontology definitions, this individual will be copied to the FBDK knowledge. If this is a NxtStudio special implementation, the action will be ignored.

In Fig. 6, the generated FBDK version of cylinder control is given. Comparing to the NxtStudio version in Fig. 2, all FB Types, event connections, data connections and constants are successfully converted automatically. However, the HMI FB will not be able to display the interface properly in FBDK due to different HMI implementations between FBDK and NxtStudio. The generated system configuration is still capable for running under FBRT (Function Block Runtime within FBDK).

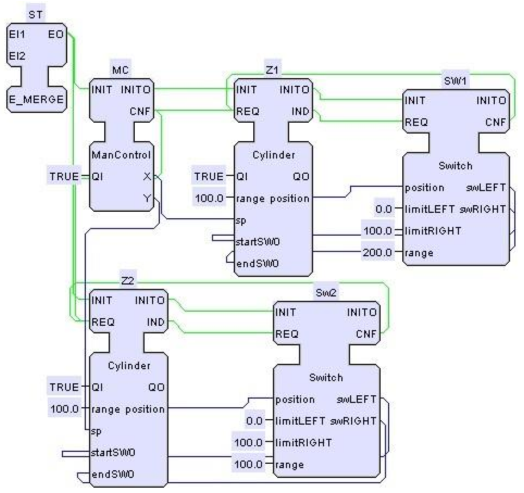


Fig.6. Generated FBDK version of Cylinder Control.

When applying the mapping from FBDK to NxtStudio, the major issue is that some ontology nodes required by NxtStudio are not available in FBDK. The solution for that is to add in a default value for those nodes. This is represented by a data property in the knowledge base:

VarDeclaration (?Var1) ^ Has_VarDeclaration_DefaultValue(?Var1, Value)

If a node is not compulsory in the NxtStudio ontology model, the default value will be null. Otherwise, a default value will be written into Value (for example, the default value for INT data type is 0 and the default value for REAL data type is 0.0).

If anything is not correct due to the generation process, the FB design can be changed using the corresponding IEC 61499 IDE. The design recovery engine is able to capture any manual change and store back into the knowledge base automatically.

IX. CONCLUSIONS

A new ontology based semantic design recovery method is developed in this paper. Its application is demonstrated for seamless migration between IEC 61499 standard compliant tools, but its potential is much greater. Once the parts of source code have been semantic classified into various semantic categories, each of them can be easier migrated to the corresponding target code.

Code level design recovery, semantic analysis and code generation are achieved by using common knowledge base defined in the ontology. Specific details of multiple platforms can be imported into the knowledge base and be outputted as other formats for various platforms. Both the syntactic and semantic check is performed before the code generation to ensure generated codes are compatible with target platforms. A case study has been conducted to prove the automatic transformation from one platform to another is feasible.

This method will be extended to migration of IEC 61131-3 PLC applications to IEC 61499 with a similar approach based on ontology based knowledge base.

X. REFERENCES

[1] IEC 61499, Function Blocks, *International Standard*, First Ed., 2005
[2] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Second Edition, 2003
[3] W.Dai, V. Vyatkin, "Redesign distributed IEC 61131-3 PLC system in IEC 61499 function blocks", *15th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, Bilbao, Spain, 2010, Page 1 – 8.
[4] C. Gerber, H.-M. Hanisch, and S. Ebbinghaus, "From IEC 61131 to IEC 61499 for Distributed Systems: A Case Study", *EURASIP Journal on Embedded Systems*, Volume 2008, Article ID 231630, 8 pages, doi:10.1155/2008/231630
[5] C. Sünder, M. Wenger, C. Hanni, I. Gosetti, H. Steininger, J. Fritsche, „Transformation of existing IEC 61131-3 automation projects into control logic according to IEC 61499", *IEEE International Conference on Emerging Technologies and Factory Automation*, Hamburg, Germany, September, 2008
[6] O.J.L. Orozco, J.L.M. Lastra, "Agent-Based Control Model for reconfigurable manufacturing systems", *2007 12th IEEE International Conference on Emerging Technology and Factory Automation* , 25-28 September 2007, Page 1233 - 1238.
[7] K. Thramboulidis, G. Koumoutsos, G. Doukas, "Towards a Service-Oriented IEC 61499 compliant Engineering Support Environment", *IEEE 11th International Conference on Emerging Technologies and Factory Automation*, 2006, Page 758 – 765.
[8] FBDK – Function Block Development Kit [Online], available from http://www.holobloc.com/
[9] nxtControl GmbH, nxtControl – Next generation software for next generation customers [Online, 2009, June], http://www.nxtcontrol.com
[10] 4DIAC, An open source IEC 61499 IDE and runtime [Online], available from http://www.fordiac.org
[11] FBench, An open source IDE for IEC 61499 [Online], available from http://www.ece.auckland.ac.nz/~vyatkin/fbench/
[12] CORFU, Function block development environment [Online], available from http://seg.ee.upatras.gr/corfu/dev/index.htm
[13] Ontology General Definition [Online], available from http://semanticweb.org/wiki/Ontology
[14] 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
[15] J. Peltola, J. Christensen, S. Sierla, K. Koskinen, "A Migration Path to IEC 61499 for the Batch Process Industry", *5th IEEE International Conference on Industrial Informatics*, 2007, Volume 2, Page 811 – 816
[16] A. Falcione, B.H. Krogh, "Design recovery for relay ladder logic", *IEEE Control Systems Magazine*, 13(2), 1993 Page 90 – 98
[17] W. Lim, J.V. Harrison, P.A. Bailes, A. Berglas, "Design Recovery through formal specification", *Australian Software Engineering Conference*, 1998, Page 22 – 31.
[18] W.Dai, V. Dubinin, V. Vyatkin, "IEC 61499 Ontology Model for Semantic Analysis and Code Generation", *9th IEEE International Conference on Industrial Informatics*, 26 – 29 July 2011, Lisbon, Portugal
[19] 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
[20] V. Vyatkin, "The IEC 61499 Standard and its Semantics" – *IEEE Industrial Electronics Magazine*, 3(4), 2009
[21] M. O'Connor, A. Das, "SQWRL: a Query Language for OWL", *OWL: Experiences and Directions (OWLED), Fifth International Workshop 2009*, Vol 529.
[22] SWRL: A Semantic Web Rule Language Combining OWL and RuleML[Online], retrieved from http://www.w3g.org/Submission/SWRL/
[23] W.Dai, V. Dubinin, V. Vyatkin, "Semantic Analysis of IEC 61499 Systems Using Automatically Generated Ontological Models", *IEEE Transactions on Industrial Informatics*, 2011, submitted
[24] V. Dubinin, V. Vyatkin, "Cycle detection in IEC 61499 function block systems using ontologies", *International conference CIT'11*, Penza, May, 2011
[25] J. Euzenat, P. Shaviko, "Ontology Matching", *Published by Springer*, 2007, ISBN:3-540-49611-4