Redesign Distributed IEC 61131-3 PLC System in IEC 61499 Function Blocks

Wenbin(William) Dai, The University of Auckland, New Zealand wdai005@aucklanduni.ac.nz
Valeriy Vyatkin, The University of Auckland, New Zealand v.vyatkin@auckland.ac.nz

***Abstract* – In this paper we discuss the problem of migration from the PLC control to the event-driven and component based architecture of IEC 61499 function blocks. We are using a conveyor system to illustrate three different migration methods: object-oriented in basic function block, object-oriented in composite function block and class-oriented in basic function block. The advantages and limitations are summarized and guideline for migration is provided.**

***Index Terms*—Automation and control systems, Manufacturing automation, IEC 61499, PLC, Migration**

I. INTRODUCTION

With the increasing global competition, manufacturers are more concern to reduce the cost by improving the efficiency of their production systems. Also redundancy and safety are taken more serious consideration in the current automation world due to the increased complexity of the control systems. Distributed automation is considered as an advanced option to increase efficiency and provide redundancy to the manufacturing and logistics systems. However, the current automation paradigm, based on the use of programmable logic controllers (PLC), programmed according to the IEC 61131-3 [1] standard, is not suitable for distributed system as it is defined for centralized control.

IEC61499 [2], seen as the next generation automation systems standard, is designed to cover interoperability, portability and reconfigurability which are missing in IEC 61131-3. The IEC 61499 architecture relies on the event-driven paradigm of execution which seems to be a disruptive technology with respect to the cyclically executing PLC programs. The target benefits of IEC 61499 are manifold: the design related ones include the increased portability of event-driven components, and the execution time ones include possibly better reaction time due to no need in cyclic scan execution of the whole PLC program.

The migration of software from the currently dominating PLC software architectures to the newly emerging component-based architecture of IEC 61499 is a very important issue for the success of the latter. There is no existing guideline clearly illustrating the migration strategy. Manufacturers are struggling with implementation of distributed system in IEC 61131-3 PLCs and they are not able to investigate the IEC 61499 solution due to the lack of resources and appropriate design tools and execution platforms, which increases the risks of the adoption.

The questions of interest are:

- Is it possible to re-use some PLC code in IEC 61499 function block projects?
- What would be the options for migrate the PLC code to IEC 61499?
- What would be the best methodology for migration?
- And, what are the limitations for the IEC 61499 function blocks?

The paper is structured as follows. Section II points out the limitation of the current IEC 61131-3 compliant PLCs for distributed control system. Related work is reviewed in section III. An illustrative example and design methodology are given in Section IV and V for our migration studies. Section VI and VII provide the possible solutions for migration. The discussion including guidelines and limitations for migration from IEC 61131-3 PLC to IEC 61499 function blocks is given in Section VIII. The last section concludes the paper and points out the future work directions.

II. REQUIREMENTS AND LIMITATIONS OF USING IEC 61131-3 PLC IN DISTRIBUTED SYSTEM

In a complex automation system, for instance, material handling system, typically thousands of inputs and outputs pins are required. The input and output PLC modules are placed in different control panels on site. Those control panels are located remotely around the conveyors in order to minimize the length of power and signal cables used. The remote I/Os are connected to the PLCs located in the central control room via a field bus (for instance, ProfiNet [12], ControlNet[13] or Ethernet/IP[14]). This setup gives the best cost-effective model by reducing the hardware cost as much as possible.

Also, the existing PLCs are not powerful enough to handle everything in a single PLC within a reasonably short scan time. In a large material handling system with hundreds of conveyors and other devices, the system has to be divided into several subsystems, each controlled by a PLC.

There are several critical limitations when applying the IEC 61131-3 standard to distributed PLC systems. The first issue is the conflicts of write access for outputs in multiple PLCs. In IEC 61131-3, each remote output module must be owned by a single PLC. All output pins from that output card only shall be turned on or off by that particular PLC. The original purpose of this design is to protect PLCs from unexpected

actions and to ensure synchrony of the data. But this also greatly increases the engineering time to work out the owner of each output module.

Beside the ownership issue, this design model also increases the complexity of software development. As a single output module cannot be owned by multiple PLCs, no more than one PLC can access the same output module. Instead, those output pins required by multiple PLCs must be assigned to one of those PLCs for first step. In the second step, the alias PLC tags of those output pins must be created in all other PLCs which need to access those pins. In the final step, those tags will be copied to the owner PLC of that output card via the field bus. This procedure involves lots of work time and care as the entire process must be done manually.

Beyond that point, copying data over the field bus will significantly reduce the system performance and reliability and increase hardware costs. For time critical systems like airport baggage handling systems, the scan time of PLC must be under a certain threshold to ensure its correct operation. The communication overhead time between PLCs is increased heavily and the system may not behave correctly if updating data over field bus cannot be handled fast enough. Also, the entire reliance on the field bus reduces system's reliability: if any of the field bus connection is lost, the system cannot continue operating until the connection is fixed. Single failure in the system can cost a long downtime and big economical losses for end-users.

III. Related Works

The importance of the migration from PLC programming architecture (IEC 61131-3) to IEC 61499 has been recently recognized by several researchers, for example [4], [5], [15], [7], [16], [17]. However, no work is known to us specifically tackling the distributed systems design issues. On the other hand, there has been interest developing in automation research to the service – oriented architectures as a possible solution for distributed systems. A few works in this direction are discussed as follows:

In the paper [8], authors pointed out that a service is described by its interface and no concern of actual implementation is required. Also the fundamental concepts of service-oriented control architecture are reconfigurability and interoperability between various devices which is identical to the IEC 61499 standard. XML is recommended as a standard protocol for exchanging configuration or data among the service-oriented system.

In the paper [9], a case study of distributed data warehouse is given using grid services-oriented architecture. In this paper, the issue of data ownership over service-oriented architecture is arisen and data scheduling optimization is also discussed.

In the paper [10], authors provide the execution rules in a service-oriented architecture for distributed system. First-In First-Out (FIFO) and Shortest Queue (SPT) are used as the rules to compare the execution average time of service-oriented architecture program. Earliest start time (EST), earliest finish time (EFT) and same recipe (SR) are applied in the execution as well. The results prove EFT-SPT is the best execution rule for distributed system in SOA.

In the paper [11], authors provide the definition of service-oriented architecture in industrial automation: An SOA is a set of architectural tenets for building autonomous yet interoperable systems. Authors also introduce the SOAP and SOA in web services into the industrial automation and recommend using Internet protocol for automation devices.

One of the design methods introduced in this paper is inspired in a way by the service – oriented architectures: a class of services related to a group of devices is implemented in a single function block. We refer to this design method as *class-oriented* design approach.

IV. Illustrative Example

To illustrate the function block design and execution rules we will use an example of a conveyor system as presented in Figure 1.

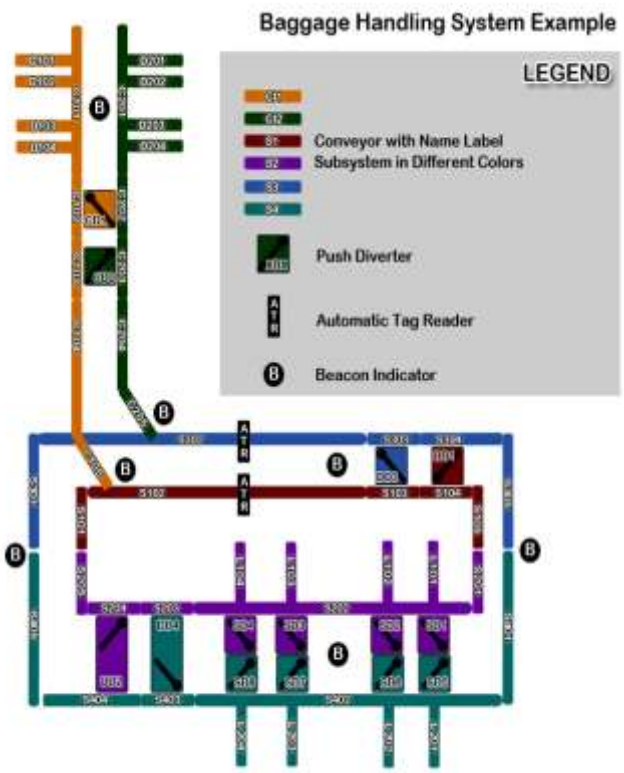

Figure 1. A Conveyor System of a Simple Airport Baggage Handling System.

The sample airport baggage handling system consists of total six subsystems: 2 Check-in system (CI1 and CI2) and 4 Sortation systems (S1 to S4). Those subsystems form two fully redundant systems: CI1/S1/S2 and CI2/S3/S4. There are 6 crossover pushers (CD1 to CD6) which are responsible for diverting bags to the other system. The check-in subsystem CI1 and CI2 are collecting bags from counters and conveying those bags to the sortation lines (S1/S2 and S3/S4). Pushers CD1 and CD2 will divert the bags to the other lines if any downstream conveyor is not operating normally. The sortation subsystems (S1 to S4) are responsible for delivering each bag to the flight it belongs to. In Figure 1, laterals L101 to L108 are used for different flights. Bags will be pushed to their destination laterals by the diverters (DD1 to DD8). Crossover pushers (CD3 to CD6) will divert bags to the alternative line if any downstream conveyor is faulty or the other PLC in the same sortation line is down. For example, when the S4 PLC breaks down, pusher DD3 will automatically divert all bags to S1 line.

The devices in each subsystem are listed below:

Legends: (# - represents number index for the device)

D### - Check-in Counter
C###/S### - Conveyor
CD# - Crossover Pusher
DD# - Crossover Pusher
SD# - Sortation Pusher
L### - Lateral
ATR# - Automatic Tag Reader

Check-in 1(CI1) – D101 , D102, D103, D104, C101, C102, C103, C104, C105, CD1
Check-in 2(CI2) – D201, D202, D203, D204, C201, C202, C203, C204, C205, CD2
Sortation 1 (S1) – S101, S102, S103, S104, S105, ATR1, DD1
Sortation 2 (S2) – S201, S202, S203, S204, S205, DD2, SD1, SD2, SD3, SD4, L101, L102, L103, L104
Sortation 3 (S3) – S301, S302, S303, S304, S305, ATR2, DD3
Sortation 4 (S4) – S401, S402, S403, S404, S405, DD4, SD5, SD6, SD7, SD8, L201, L202, L203, L204

The subsystem redundancy is listed in the table below.

Original Subsystem	Destination Subsystem	Via Push Name
CI1	CI2	CD1
CI2	CI1	CD2
S1	S2	DD1
S2	S1	DD2
S3	S4	DD3
S4	S3	DD4

Table 1. Redundancy Crossover Pushers.

V. DESIGN METHODOLOGY

First, let us describe the PLC network design. In each subsystem, a main control panel is installed so that all cables from field devices are wired into the panel. Remote I/O modules located inside the main control panel are connecting to the field devices. All PLCs are sitting in a single rack in the main control room and communicating with remote I/O via Ethernet. Figure 2 illustrates this PLC network.

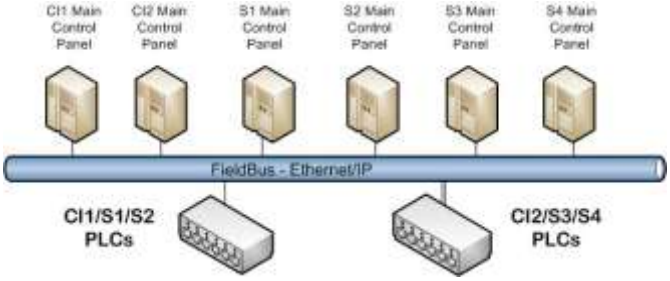

Figure 2. PLC Network Diagram for BHS.

One PLC is assigned for each subsystem. In each PLC, an individual subroutine is created in ladder diagram (LD) for each device. During each scan, PLC executes all the subroutines one by one. Figure 3 provides the structure of the PLC code. It is common in PLC code design that a function is developed for each device to provide control, service or interface. Then data tags required by each device are created and those data tags are considered as the instance of that device. The function will go through each device in order. This design follows in a way the service-oriented approach.

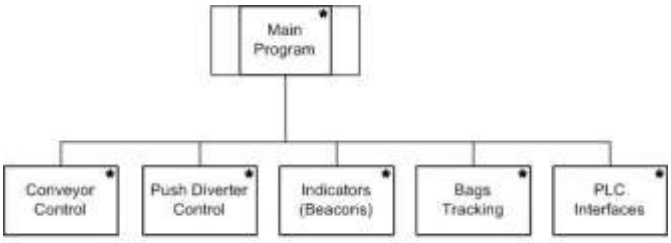

Figure 3. PLC Code Structure for BHS.

If we take a further look at the layout drawing of the BHS, beacons are installed all around the conveyors to indicate when a fault occurs or issue a warning that the system is starting. Some beacons are placed at the joint of two subsystems. Those beacons are designed to relay the status of downstream conveyors which belong to another subsystem. Also there are some beacons shared by two subsystems such as B101 at the check-in area which are shared by CI1 and CI2.

In the IEC61131-3 PLC code, those beacons must be assigned to a single owner PLC. In Figure 4, the beacon B101 is lit on when conveyor C101 in CI1 or conveyor C201 in CI2 is in fault state. The physical beacon output is wired into the PLC output module that is owned by CI1 subsystem. When the conveyor C201 malfunctions, the B101_On output in CI2

must be copied to CI1 over PLC interlink or via a field bus (ControlNet or Ethernet). Extra ladder logic code in CI1 is required to turn beacon B101 on when B101_on signal is raised from either CI1 or CI2.

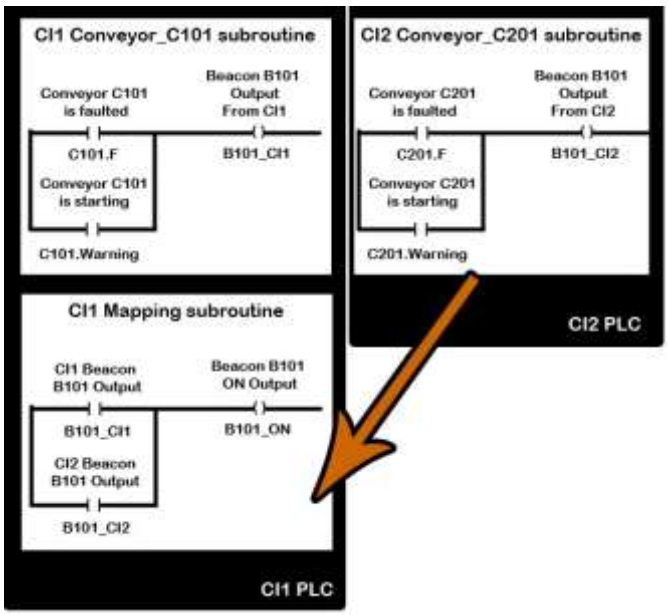

Figure 4. Beacon Indicators in IEC 61131-3 PLC Code

Now let us consider how the PLC code could be encapsulated into event-driven function block of IEC 61499. There are two design principles that can be applied: object-oriented approach or class-oriented architecture.

A. Object-Oriented Approach

In the object-oriented approach by [18,19], function blocks are used as the fundamental programming unit that contains all related functions, variables and interfaces of a single device (object). Similar to the general concept of object-oriented programming, when creating a new system in IEC 61499, a separate instance of the function block would be created for each device in the system (for example, creating a FB instance of conveyor for each conveyor in the system). Eventually, those FB instances are connected together according to the physical layout position order. It is easy to demonstrate that the composite function block illustrated later in Figure 8 (under the stated assumptions) will have the behavior equivalent to the original PLC executing the ladder diagram cyclically for each device.

B. Class-Oriented Architecture

In the class-oriented approach, the basic function block is still the basic element of the program. Similarly to the object-oriented approach, here the function block represents a class of devices and encapsulates all its functionalities. The difference is that a single instance of this function block serves all the devices of this class (for instance, all conveyors in the system will be processed in the same function block, or a particular group of such conveyors).

In other words, all data and algorithms providing one particular service are encapsulated in a function block. This model ensures each function block is able to provide a service without invoking any other function blocks.

We will illustrate both solutions by implementing them in nxtStudio IEC 61499 Editor [3].

VI. OBJECT-ORIENTED SOLUTION

According to the object-oriented design method, the functionality of each physical device (conveyor, pusher, beacon, etc.) is implemented in a basic function block.

The IEC 61131-3 subroutines/functions are encapsulated into algorithms of the FB. The conversion to algorithms is, however, not straightforward. We will illustrate the method using the conveyor control example in the baggage handling system context.

A conveyor controller is typically designed as a finite state machine (FSM). The common states for a conveyor are off, startup, stop, run and fault. The finite state machine jumps states based on the changing values of PLC inputs. After the current state is switched, the logic of the new current state is activated and associated outputs are emitted. The FSM is implemented in a PLC programming language, for example in ladder logic diagram which may look like illustrated in Figure 5. The conveyor control subroutine divides into two major parts: FSM and outputs. In the FSM part, the transitions of states are defined. The subroutine will jump to the outputs part and regenerate output status based on the state of FSM.

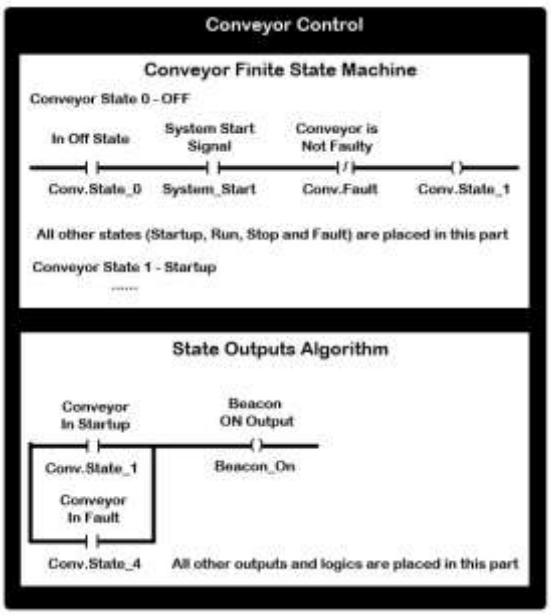

Figure 5. Conveyor control code structure in IEC 61131-3.

To migrate this conveyor control into the IEC 61499 format, there are two approaches by applying the object-oriented concept.

A. Convert PLC Code into ECC and EC State Algorithms

The first choice is to convert the original FSM that precedes the PLC Code (refer to [20] for FSM recovery from PLC code) into the execution control chart (ECC) inside basic function block. The ECC is a state machine which is switching states based on input events triggering.

For that, however, the original FSM needs to be known, or it can be recovered from the PLC ladder logic, for example following the ideas of [21],[22].

First design step is to create one EC State for each state from the original finite state machine. In our case, we will create 5 EC states in the new conveyor basic function block: *off, startup, stop, run* and *fault*. Secondly, the conditions of state transitions are mapped directly to the EC transitions conditions. For example, in the PLC code, the premise for jumping from *off* to *startup* state that there is no fault on the conveyor and start button is pressed is set as the new EC transition condition between EC State off and startup. Finally, the state actions are placed into the EC state algorithms. In figure 6, when the system is starting up, the beacon and sounder indicator will be turned on for warning. The beacon and sounder output command is lit in the *startup* state attached algorithm in ECC.

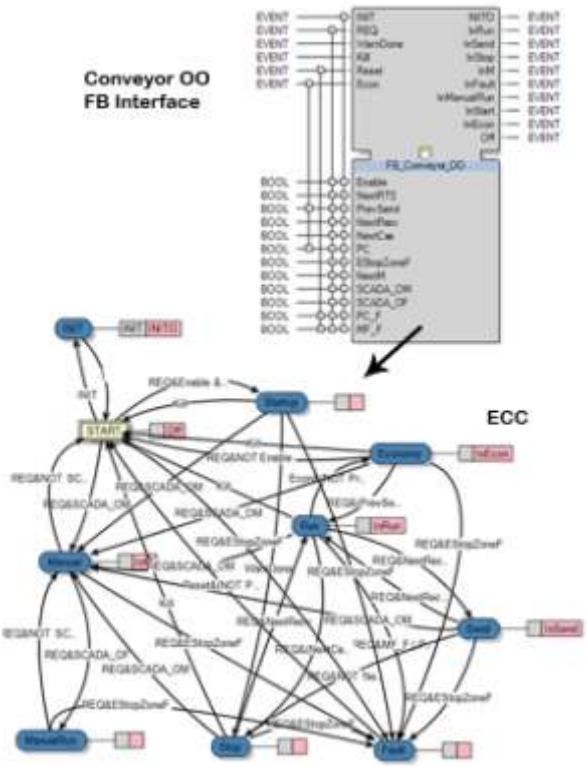

Figure 6. Conveyor Basic Function Block - Converting PLC state machine to ECC

B. Reuse Entire PLC Code

In this method, no PLC code change is required at all, as it is encapsulated into the algorithm REQ. The new basic function block is a top level entity for detecting the changes of inputs when the REQ event is sent from the I/O capture Service Interface FB. When an input change happens, the basic function block will jump into the state REQ processing request. After the algorithm is executed, the new output data shall be updated. In Figure 7, the FB interface is identical with the previous model. But inside ECC, there are only two EC states - IDLE and REQ states. When the photo eye on the conveyor is flushed or any other inputs change their value, the FB wakes up from IDLE state, starts executing the algorithm in REQ state. Once the process is completed, the FB remains in IDLE state until any other input changes.

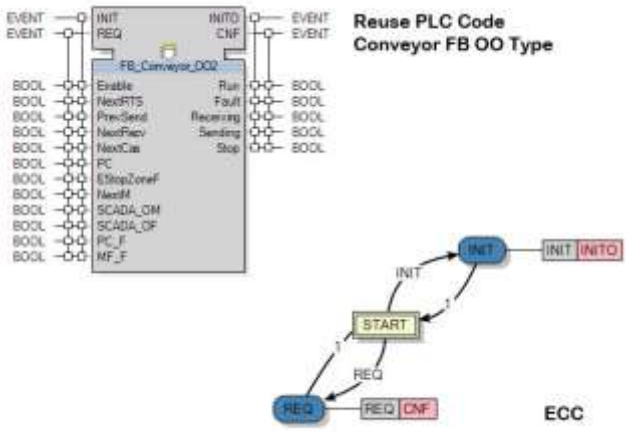

Figure 7. Conveyor Basic Function Block - Reuse PLC Code

After converting all subroutines/functions into function blocks, a composite function block is used to resemble a circuit of logic elements as in Figure 8. Function blocks are connected together relied on the physical layout. In this case, a single conveyor is connected to both upstream and downstream conveyor function blocks.

For deployment, an IEC 61499 – compliant device is used for each subsystem which only contains conveyors and other devices belonging to this subsystem. Those devices are distributed and interoperable with each other. IEC 61499 compiler will dynamically allocate IEC 61499 devices into various resources (processors). After all function blocks are compiled into executable binary files, this FB library will be deployed to those processors. The FB instances are created using the management commands (defined in IEC 61499) sent from the programming tool to the remote processors. The processors are updating their I/O data via remote I/O modules connected via field buses like Ethernet.

Figure 8 demonstrates the IEC 61499 solution for CI1 subsystem. Function block instances of C101 to C105, D101

to D104 are created respectively. FB_Inputs are connected to every function block for updating inputs status as well as the FB_Database which provides global variables and constants. The push diverter CD1 and beacon B101 are also converted into function blocks in the new IEC 61499 models.

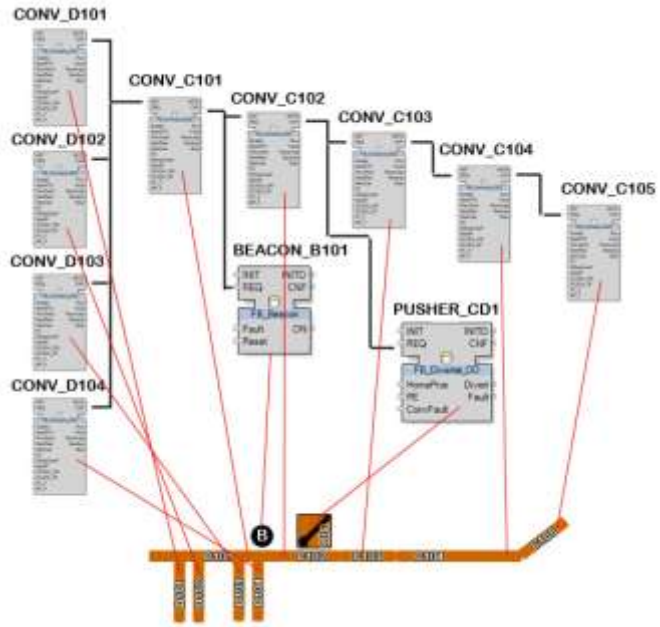

Figure 8. Composite FB Network for CI1 Integration Solution.

This solution required the development of several extra basic/service interface FB types, described as follows:
FB_Database – stores all global variables and constants. This FB_Database is required by all FBs in the network.
FB_Inputs – updates all physical inputs from the Input modules. This service interface function block is invoked every 50ms to read the values of all inputs.
FB_Outputs – update all physical outputs to the output modules. This service interface function block is called by other function blocks when there is a request for output change.

In this object-oriented approach example, a FB instance can be found in the composite function block for each device. Two feasible solutions for migration are also provided: reusing PLC code or converting PLC code to ECC with algorithms. Figure 9 provides an overview for the CI1 system in IEC 61499.

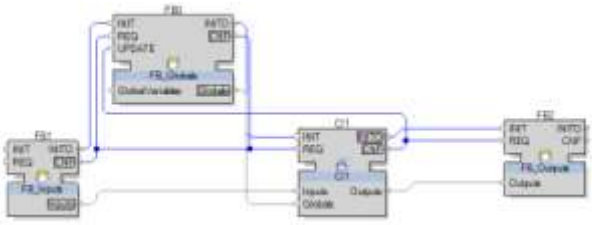

Figure 9. Overview of the entire CI1 in IEC 61499.

VII. Class-Oriented Solution

The other possible integration solution is using the class-oriented architecture. Instead of creating a FB for each device, a basic function block is created for a class of such devices. For instance, FB_Conveyor in Figure 10 is servicing all conveyor control related algorithms. The FB interface is similar to the previous object-oriented model but with an array of the conveyor inputs. When some inputs change for the conveyors, the service interface function block FB_Inputs will update those values to FB_Conveyor immediately. According to the ECC, with REQ event raises, FB_Conveyor shall move to REQ state and read all inputs values into the local conveyors data which are stored as local variables inside the FB_Conveyor. Once the update process is completed, the ECC shall move to the PROCESS state, and inside the PROCESS state algorithm, conveyor logics will be looped for every conveyor in the memory. All original PLC code is reused[1] in the PROCESS state algorithm without any major modification. The only step is to replace all conveyor instance variables in the algorithm with an indexed data array of conveyor (for example, Conv1 is now replaced with Conv[1]). The upstream and downstream conveyors data are easily accessible as its local to the current conveyor.

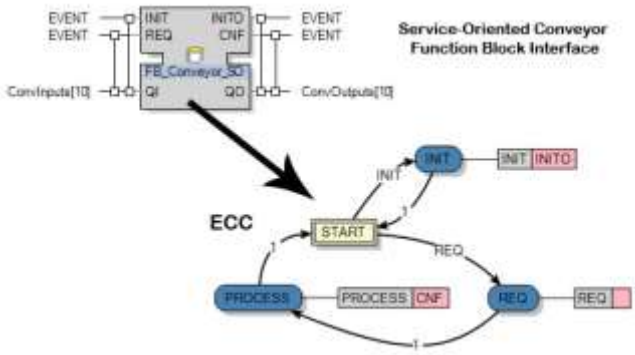

Figure 10. Class Oriented Conveyor Basic Function Block.

Similar to the object-oriented solution, FB_Inputs and FB_Outputs are still required for gathering inputs data and generating output data. To fulfill distributed control, an IEC 61499 device is created for each subsystem which includes, a class function block for each device class (or a subgroup within the class). Those class function blocks contains all data required and only be invoked when the input data changes. Figure 11 illustrates the system overview for class-oriented CI1 subsystem. FB_conveyor contains all conveyors including D101 to D104 and C101 to C105.

[1] Provided that the PLC code has been already designed in a modular way and there is no direct use of variables of other conveyors within the code of a single conveyor (instead they are referred to via a kind of reference table).

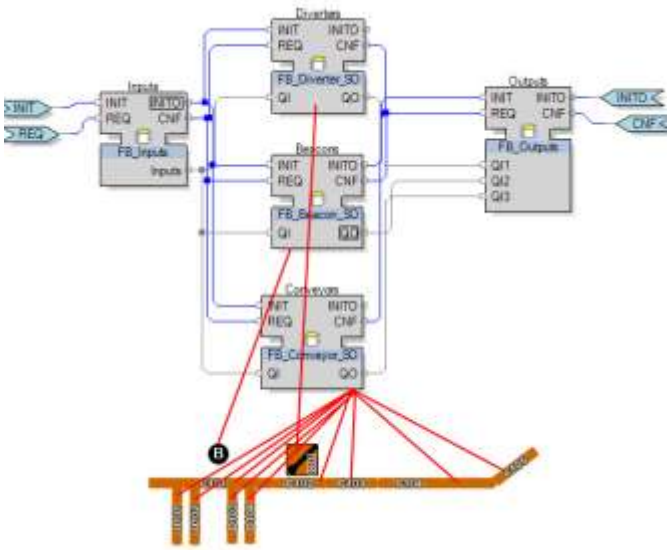

Figure 11. Class Oriented for CI1 Integration Solution.

VIII. DISCUSSION

In the object-oriented design, modular design concept is applied so that each physical device represented by a function block instance. It is convenient to identify each device in the function blocks network. This provides great convenience for debugging and easy understood by maintenance technicians or system operators.

This approach can be also beneficial in terms of performance, as the event flow can be directed only to those FBs which are necessary to be involved in the reaction to a certain input event in the environment.

And, this approach well fits to the vision of [18] when each part of the physical machinery is equipped with its own embedded controller running the corresponding controller FB.

A typical PLC program code implementation is done either in sequential logic or using finite state machine. If the PLC code is controlling a series of actions or is sequential without nested structures, such PLC code is eligible for the reuse of the entire program in EC State algorithms. Converting the entire PLC code into the IEC 61499 as in [4] requires a lot of re-work time on coding and cannot achieve fully identical result. Alternatively, re-use the entire PLC code is a much more cost-effective method. This minimizes the engineering time during the migration process [5]. This is also beneficial for IEC 61131-3 developers to quickly pick up IEC 61499 design and helping manufacturers migrating IEC 61131-3 to IEC 61499 with minimum cost.

The class-oriented architecture provides a different solution for converting the PLC code into function blocks. The key principle here is to consider the basic function block as representing a class of devices or encapsulating a class of services. The class FB stores all related data locally so no global variables or constants is required. This solves the issue that there is no definition of global variables and constants in IEC 61499 [7] which are commonly used in IEC 61131-3.

IEC 61499 is invented to introduce distributed control into the existing automation world. Most of the current studies for IEC 61499 function block implementation are using the object-oriented concept due to the fact that physical machines are distributed and each machine is controlled by a processor. However, the class-oriented design may fit better to the migration of PLC code into IEC 61499 FB. In this case, the global variables still can be used in the program. All programs running on the IEC 61131-3 standard PLCs are executed cyclically. When a scan starts, PLC will read out all input modules data into the PLC memory, go through the entire active algorithms, and update all output modules at the end of each scan. Scan time is a critical factor of measuring the system performance. Missing input data due to the slow scan time would cause unexpected system behaviors. Those abnormal behaviors cannot be easily detected by engineers and also create safety issues on site. As IEC 61499 is based on event-driven function blocks, there is no concept of scan cycle time. Instead, FB_Inputs emits event whenever any input changes. This event will invoke the FBs downstream.

In the object-oriented model, a device is recognized as an object and all data and methods are local within this function block instance. However, in a typical material handling system (for instance, airport baggage handling system), a conveyor cannot operate on its own. The statuses of upstream and downstream conveyors as well as the conveyors in the same zone are also required to determine the operation mode of each individual conveyor. As those status data are kept in their own conveyor function block instances, data connection must be created to pass all statuses variables back. Also the status of this conveyor might be requested by multiple conveyors as well. As a result, numerical data connections are established between all conveyor function block instances. During system operation, the extra overhead time for updating statuses is required that causes increase in the reaction computation. Therefore, the object-oriented approach may be not the best option for all distributed system designs.

In the class-oriented design, conveyor status data, methods and relevant conveyor status data are all local and no overhead time for updating data variables is necessary. During the execution, the entire input data will be copied into this function block at a fixed period interval. After the input variables have been updated, those conveyors in the function block which have one or more updated inputs will be processed and related outputs will be regenerated after the execution of algorithms is completed. This is similar to IEC 61131-3 program in PLC but different in the sense that in PLC all conveyor function blocks will be executed regardless of change of inputs. Compared to the object-oriented model,

there is absolutely no overhead communication time in the class-oriented model. This will increase the efficiency significantly. Beside the overhead time, in normal IEC 61131-3 PLC approach, if a variable is required by more than one program, that variable will be declared as a global variable. In IEC 61499, in order to obey the fundamental concept of encapsulation, there is no definition of global variables or constants. In the object-oriented approach, those global variables and constants must be stored in a separate function block. Then this function block will communicate with every block which has a demand for global variables and constants. This also creates inefficiency in execution. This issue is avoided in the class-oriented model, as all the required data variables are available locally..

Overall, the class-oriented architecture avoids using global variables and constants at the top level design. Re-use of the entire PLC code minimizes the migration time. No data is required to be passed around the function blocks, which will reduce the execution time of the program. Those features suit better for distributed system control.

IX. Conclusion And Future Work

The IEC 61131-3 PLCs do not best suit the distributed control systems. We proposed two options for migration from IEC 61131-3 PLC control to the IEC 61499 distributed Function Block control: the object-oriented and the class-oriented architectures. Benefits and drawbacks of both approaches have been presented and compared.
The future work will concern with benchmarking the performance of IEC 61499 running the migrated IEC 61131-3 programs. Also the behavior of the resulting IEC 61499 programs must be verified.

X. Acknowledgements

The authors are grateful to nxtControl GmbH, and personally to Horst Mayer for providing the trial licenses for nxtStudio – the IEC 61499 programming environment and to Arnold Kopitar, Gernot Kolleger and Walter Oberndorfer for providing valuable ongoing support.

XI. References

[1] IEC 61131-3, Programmable controllers - Part 3: Programming languages, *International Standard*, Second Edition, 2003
[2] IEC 61499, Function Blocks, *International Standard*, First Edition, 2005
[3] nxtControl GmbH, nxtControl - Next generation software for next generation customers [Online, 2009, June]. Available: http://www.nxtcontrol.com/
[4] M. Wenger, A. Zoitl, C. Sunder, H. Steininger, "Transformation of IEC 61131-3 to IEC 61499 based on a model driven development approach", 2009 7th IEEE International Conference on Industrial Informatics, 23-26 June 2009, Page 715-720.
[5] 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
[6] V. Vyatkin, "Reuse of PLC Code and adding Safety features in IEC 61499", Research Results Report, University of Auckland, 2010
[7] SOAP Specifications, retrieved from http://www.w3.org/TR/soap/ on 12th April 2010.
[8] W. Dai, V. Vyatkin, "A Case Study on migration from IEC 61131-3 PLC to IEC 61499 Function Control", 2009 7th IEEE International Conference on Industrial Informatics, 23-26 June 2009, Page 79-84.
[9] J. Mendes, P. Leitao, A.W. Colombo, F. Restivo, "Service-Oriented Control Architecture for Reconfigurable Production Systems", 2008 6th IEEE International Conference on Industrial Informatics, 13 – 16 July 2008, Page 744 – 749.
[10] P. Wehrle, M. Miquel, A. Tchounikine, "A Grid Services-Oriented Architecture for Efficient Operation of Distributed Data Warehouses on Globus", 2007 21st International Conference on Advanced Networking and Application, 21-23 May 2007, Page 994-999.
[11] Y. Wang, S. Turner, "Rule Execution in a Service-Oriented Architecture for Distributed Simulation on the Grid", 2008 7th International Conference on System Simulation and Scientific Computing, 10-12 Oct 2008, Page 1151-1158.
[12] F. Jammes, H. Smit, "Service-Oriented Paradigms in Industrial Automation", IEEE Transactions on Industrial Informatics, Vol. 1, No. 1, FEB 2005.
[13] PROFINET, Retrieved from http://www.profinet.com
[14] ControlNet, Retrieved from http://www.controlnet.org
[15] Ethernet/IP, Retrieved from http://www.ethernet-ip.org
[16] M. Hirsch, C. Gerber, H, Hanisch, V. Vyatkin, "Design and Implementation of Heterogeneous Distributed Controllers According to the IEC 61499 standard – A Case Study", 2007 5th IEEE International Conference on Industrial Informatics, 23-27 June 2007, Page 829-834.
[17] W. Dai, V. Vyatkin, "On Migration from PLCs to IEC 61499: Addressing the Data Handling Issues", 2010 8th IEEE International Conference on Industrial Informatics, 13-17 July 2010.
[18] G. Shaw, P. Roop, Z. Salcic, "Reengineering of IEC 61131 into IEC 61499 Function Blocks", 2010 8th IEEE International Conference on Industrial Informatics, 13-17 July 2010.
[19] G. Black, V. Vyatkin, "Intelligent Component – based Automation of Baggage Handling Systems with IEC 61499", *IEEE Transactions on Automation Science and Engineering*, 2009, 6(3), doi: 10.1109/TASE.2008.2007216
[20] V. Vyatkin, P. Roop, Z. Salcic, J. Fitzgerald, "Now That's Smart!", IEEE Industrial Electronics Magazine, 2007 Volume 1, Issue 4, Page 17-29.
[21] H.J. Kim, R. Harms, G. Seliger, "Automatic Control Sequence Generation for a Hybrid Disassembly System", IEEE Transactions on Automation Science and Engineering, 2007 April, Volume 4, Issue 2, Page 194 -205.
[22] Falcione, A. and Krogh, B. (1993), Design Recovery for Relay Ladder Logic. IEEE Control Systems, 13, pp. 90-98.