On Migration from PLCs to IEC 61499: Addressing the Data Handling Issues

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

***Abstract* – The IEC 61499 Function Block architecture is considered as the next generation of programmable control technology. However, during migration from IEC 61131-3 centralized PLC to IEC 61499 distributed control, the data handling efficiency issue has arisen. This paper uses baggage handling system as case study to propose two different approaches for providing efficient data handling model. The comparison between the object-oriented and service-oriented models is provided as well as the limitations.**

I. INTRODUCTION

Programmable logical controllers (PLC) are widely used in automation of various industries. PLC program execution is cyclic. In a scan cycle, PLCs read all inputs at the beginning of execution scan cycle, then go through every active function and update all outputs at the end of each scan.

IEC 61131-3[1] is the widely adopted international standard for PLC programming. Five standard programming languages are defined in IEC 61131-3: Ladder Diagram (LD), Structure Text (ST), Instruction List (IL), Sequential Function Chart (SFC) and Function Block Diagram (FBD). Each language has its own advantage, for instance, ladder diagram is similar to electric circuitry drawings and is easy for the maintenance electricians to understand the logics. The syntax of Structure Text is similar to high-level programming languages like C. Most software developers can get used to the language syntax within minimal time.

The PLC program execution performance is commonly measured in the scan time. The execution time of each scan is the sum of all instructions' execution times which have been executed in this scan, plus the time required to read from input modules and update from output modules [3]. For many real-time applications, the scan time of the PLC determines whether the control system goal is achieved or not.

The IEC 61499 standard [2] was published in 2005 with the purpose to address specific requirements of distributed systems and to complement the IEC 61131-3 standard. IEC 61499 aims at the improvement flexibility, reconfigurability and interoperability of the current PLC standard. In IEC 61499, the programming overhead for inter-device message exchange is minimal. Messages are easily passed around by using for example publish/subscribe function blocks. The basic element of IEC 61499 applications is a function block. Function blocks can encapsulate code in different PLC languages and be executed when an input event is triggered.
Obviously there is no concept of scan cycle in IEC 61499. The time consumption for running a particular sequence of algorithms is not limited to scan time. The IEC 61499 application will execute the algorithms continuously by driving the downstream function blocks until no further downstream function block is detected in the chain. The total execution time of a program is variable, depending on the particular length of the function block chain sparked by an input event. The comparison of the program execution times between PLCs and Function Blocks will be a critical factor for measuring system performance. This paper will continue from [1] investigating a more efficient approach for migration base on comparing the system performance.

The paper is structured as follows. In section II, data handling mechanisms of both IEC 61131-3 PLC and IEC 61499 are reviewed. Then in section III, the background of baggage handling system (BHS) is provided with an overview of an existing PLC solution. Section IV presents the BHS IEC 61499 model based on object-oriented design and discusses the limitations of this approach. Section V presents another approach of service-oriented model for BHS also discussing the limitations.

II. DATA HANDLING IN IEC 61131-3 PLC AND IEC 61499 FUNCTION BLOCKS

A. Data handing in IEC 61131-3 PLC Program

There are several data types defined in IEC 61131-3: Bit Strings, Integer, Real, Time, Date and Time of day, String, Arrays, Sub Ranges, Derived and Generic [1]. Users also can create customized data type based on the combination of predefined data types or other user defined data types. Variables must be declared in the symbol list before using those data types. There are five different attributes for variables: Global, Local, I/O Mapping, External and Temporary [1]. The global scope variables are accessible in all PLC tasks, while local variables are only accessible in the task where they are declared. I/O mappings are a special type of variables linked to actual inputs and outputs modules in the PLC rack. Global variables are commonly used for constants, databases and variables that must be utilized in variable PLC tasks. Although it is convenient for PLC developers to declare all variables as global, to prevent data being updated accidentally by other tasks in the same PLC program, local variables are still quite popular in industrial practices.

During execution of PLC programs, instruction execution time is controllable by the developers. Time elapsed for running a PLC instruction depends on the operands of this instruction – if the parameters are in the form of large data block, it will take longer to process those data.

B. Data handing in IEC 61499 Function Blocks

Designed as the next generation automation programming paradigm, the IEC 61499 standard also complies with all data types which were defined in IEC 61131-3. Adapter type as another form of user defined data type is also supported in IEC 61499.

However, unlike IEC 61131-3 standard, there is no concept of global memory or global variables in IEC 61499 standard. All data variables in IEC 61499 function blocks must be encapsulated in a function block. In PLC function block diagram implementation of IEC 61131-3, a data variable can be declared as input/output or address reference which allows the developer passing variables as pointer into function block diagram, updating values in the function block diagram and returning the value of those variables. In IEC 61499, there is no pointer type, so the corresponding variable shall be declared as separate input variable and output variable. In IEC 61499 function blocks network, each function block is holding its own internal variables. When a specific data variable is requested by an event emitted by another function block, this block shall hand over the value of this variable to the destination function block with an update event signal. If the value is modified by another function block, the new value must be passed back to the owner function block. Otherwise, the Supervisory control and data acquisition (SCADA) system or high level control system cannot trace all data in the system as the owner of the data is changing during runtime.

III. Heavy Data Processing Case Study and Existing Issues - Baggage Handling Systems

The airport baggage handling system (BHS) is an excellent example illustrating the complex data handling process. In a typical airport BHS, massive information has to be processed in a short period to ensure that all bags are safe and delivered to the destination flight on time.

The diagram in Figure 1 illustrates a simple BHS layout with all devices. When the passengers check their bags in at the check-in counters, bag tags will be attached. The flight, passenger and bag information shall be recorded in BHS for further reference. Security currently becomes the most concern issue of the aviation industry. All bags travelling internationally must be screened before loading onto the aircrafts. During the incline screening process, all bags must be screened and tracked during the entire process. In this case, bags positions, screening results and related information must also be kept in the PLC memory. The PLC database is designed as array of thousands bag records. Each bag record is consists of multiple double integers which contains all related information. To achieve tracking of bags, each bag's position in the BHS shall be updated during the operation. This on-belt tracking process requires large memory and powerful PLC processors capable of handling those tasks in milliseconds. All bags passed screening test would be sorted to their designed flights. Bags are also required to be tracked during this sortation process to ensure they are delivered to the correct flight. The sortation system requires a number of powerful PLCs with large memory to process bags in time. In a large airport baggage handling system, commonly the BHS system is divided into several subsystems that each subsystem controlled by a separate PLC. Nodes are communicating with each other via sending messages to other nodes. Bag information is spread around other PLCs through those messages. This communication overhead time is also critical for the BHS performance. The messages must be stored into memory before executing the algorithms to ensure the data are synchronized.

Existing PLCs' performance does not always meet the requirements of the baggage handling system. In a complex system, single PLC may not be powerful enough to handle a particular subsystem efficiently. In order to solve the issues, more PLCs are required and each subsystem is divided into more subsystems. On the other hand, PLCs are communicating with each other for exchanging data during operation. More PLCs means the communication overhead will also be increased. When the number of PLCs reaches a certain level, adding more PLCs will actually decrease the performance due to the communication overhead. According to the site experiences, this is not the best solution to solve this issue. Also adding more PLCs are not practical for cost-efficiency reasons due to their high price.

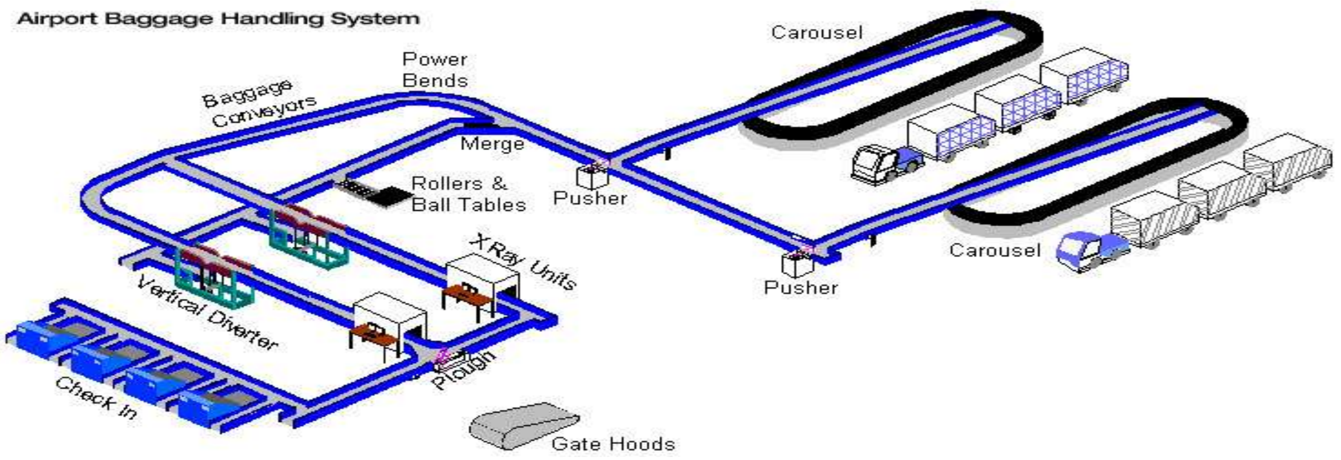

Fig. 1. Baggage Handling System Diagram [10]

IV. IEC 61499 FUNCTION BLOCK MODEL BASED ON OBJECT-ORIENTED DESIGN FOR BHS

In this and the following section, we will investigate how to improve the performance of a current PLC based baggage handling system by migrating to IEC 61499 function blocks. We will compare two design approaches. The code is structured in a modular or object-oriented way similar to the physical structure of BHS, but the internal logic is similar to PLC code for a single conveyor [1]. We will refer to this approach as *object-oriented design*, where the main object – orientation feature is strong encapsulation of data. In the other, the structure is optimized to minimize data exchange between the modules. This approach will be referred to as *class oriented design*.

A. *IEC 61499 BHS Model Resulting from Straightforward Migration from PLC*

To migrate from centralized PLC based control of the baggage handling system from Figure 1 to IEC 61499 distributed function block control, a series of steps shall be taken. One possible methodology, extended from [1] is akin to the object-oriented concept similar to that of high-level programming languages. In this approach, for all physical devices in the system the corresponding function blocks will be generated according to their physical layout, for instance, conveyors, sortation machines or control panels.

For the first step, controllers of all such devices have to be represented by basic function blocks. Figure 2 illustrates the interface and internal algorithms of a basic function block of a standard conveyor. As the function block in IEC 61499 must be triggered by events, REQ input event is used to update all data inputs associated to this block. This event is also the trigger of execution control chart (ECC) inside the basic function block, that is an internal state machine [3] defining the behavior of the block. Once the EC transition condition is true, the ECC will jump to the next state, execute the algorithm for this state, and then turn on the related output events and update all associated data outputs. In this case, a conveyor normally has several states: off, stop, run, economy, and fault. Depending on the signals from data inputs, conveyor will switch between states automatically.

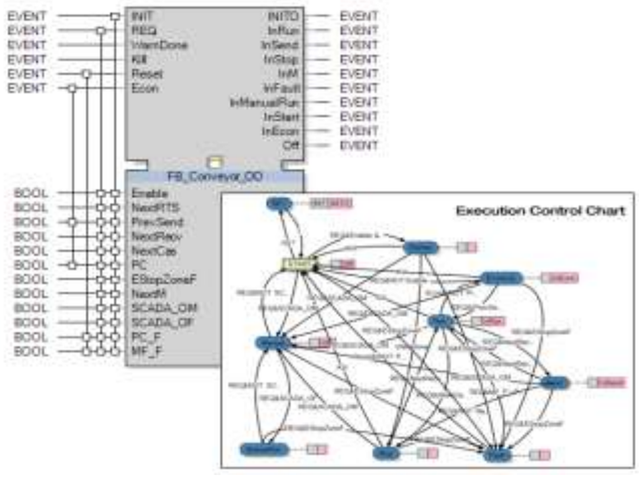

Fig. 2. Basic Function Block of a Standard Conveyor.

Timers are used widely in all PLC applications. For example, a standard conveyor will turn into economy mode when it has kept running for a period of time and no bag was detected. This mechanism is only activated when a timer is set to the predefined time and no bag is seen from all photoelectrical sensors installed on this conveyor. In IEC 61499 format, all delay timers are implemented in a separate service interface function block – E_DELAY.

In order to run the standard conveyor properly, several timers need to be inserted into a composite function block FBCall_Conveyor together with the basic conveyor control function block FB_Conveyor. Figure 3 demonstrates the interface of FBCall_conveyor and internal function block networks. An additional timer is added when the EC transition condition requires. Those FB_Conveyor blocks raise the output event when this particular conveyor instance is in the state to trigger the timer function block. The timer done bit is set and trigger the ECC inside FB_Conveyor to jump states when the predefined value is count up.

To build the entire conveyor control system, all FBCall_Conveyor function blocks shall be connected in the same direction as the bags flow. This ensures each conveyor understands the status of the upstream and downstream

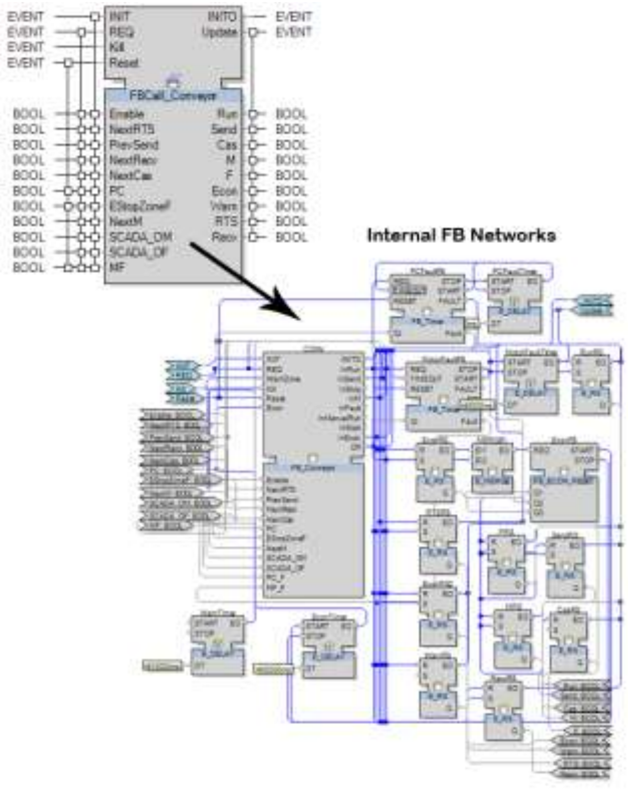

Fig. 3 FBCall_Conveyor Composite Function Block.

conveyor to provide correct system behavior.

As discussed in the previous section, bags are required to be tracked through the entire BHS. In order to record all information, a baggage database function block FBBag_DB is essential. Each bag in the system has a unique ID and all details of this bag are recorded in the database including the time entering/exiting the system, IATA tag, screening status, current bag position, destination and etc. This function block provides both a read and write interface and keeps database in the internal memory.

When a bag enters the system, a new unique ID will be assigned and this bag record will be stored into the first conveyor tracking model. When this conveyor is running, the tracking model will calibrate the latest bag position and update it into the bag record. A FIFO (First-in First-Out) type stack memory is built in the FB_Track.

When the photoelectrical sensor is triggered at the most downstream position of the conveyor, the first bag record in the stack will be handed over to the downstream conveyor when it is leaving the conveyor. All other bags that are still in this tracking model will shift one position up in the stack. Those tracking function block models are connecting with each other in the same order of conveyor control function blocks. Also after the bag details are updated during the operation, this bag record must be synchronized with the database. The major reason for updating bag record in real time with database is that the baggage handling system requires communication with high level control systems and all data must be stored in the high level control system database. Except those major control function blocks, HMI functions and Inputs/Outputs updating functions in the PLC program are also converted to individual function blocks. The entire solution overview can be seen from Figure 4.

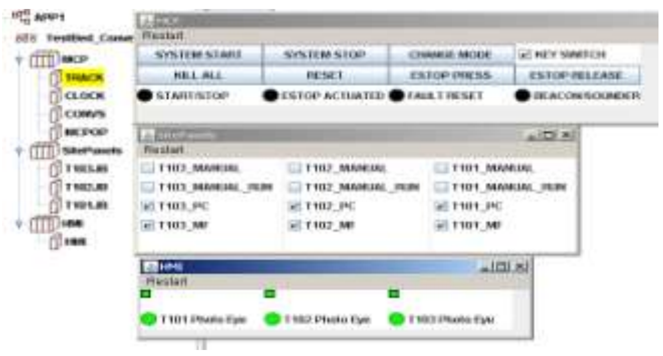

Fig. 4. Baggage Handling System OO Example in IEC 61499.

B. Limitations of the object-oriented design approach

The most concern about the object-oriented approach is data handling efficiency. As mentioned in the previous section, there is no global memory concept in IEC 61499. Instead, all data variables are encapsulated within basic function blocks.

In high level object-oriented programming languages (e.g. C++ or Java), class members are defined in the proper scope (in a specific class), and those local members are accessible from public methods (functions) [5]. In IEC 61499 terms, data variables are encapsulated inside a basic function block, and those variables will be read or written via internal algorithms, having access to data inputs/outputs. The object-oriented design is proved to be suitable for PCs. The strong CPUs and large memory of computers are capable to handle complex data processing. Also the execution time is not so critical for a PC program. Users can wait an extra couple of seconds until the program finishes the execution. But for PLC, execution speed is the most significant factor of the real-time system. Baggage handling system is one of those typical real-time high performance systems requiring huge data processing in a short time. The most important feature of PLC is reliability. Most PLCs are running under much lower frequency than PCs but will never fail even under extra weather conditions. The memory structures of existing PLCs are mostly linear structures. For two or more dimensional arrays or indirect addressing in PLC application, the execution time is significantly longer than the flat code structure. The processor must decode the indirect address first before start processing the data. Under the object-oriented design, the nested structures are applied through the entire IEC 61499 applications. The execution performance might be affected seriously.

Beyond the processing speed issue, all data are stored locally in the object-oriented approach. When another function block requires a bit of data, this data has to be requested out of the original function block. Refer to Figure 5, when FB_Track_T101, FB_Track_T102 or FB_Track_T103 requires a single bag record from FB_Database, the bag records must be handed over to those function blocks individually. This is extremely inefficient in practice. If this bag record is required by every function block in the system, a data connection with an update event must be linked to every function block in the system.

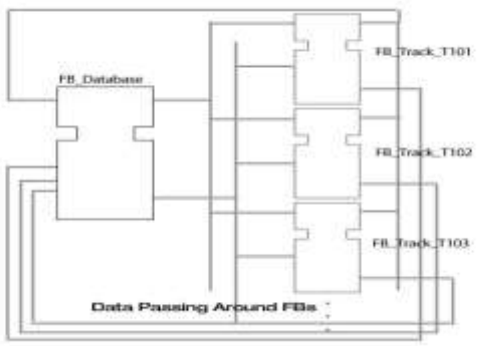

Fig. 5. Inefficient Data Handling in IEC 61499.

In some worse cases, if all function blocks may update this value and return the new value back to the original function block, a pair of input event and data must be added to the original function block. According to the example in Figure 7, after the bag record has been updated, this record must be handed over back to FB_Database. This doubles the connections and increases the processing time. Those inputs cannot be combined as the data merging is not allowed - “Each data input of a component function block can 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” and “Each data output of the composite function block shall be connected from exactly one data output of exactly one component function block, or from exactly one data input of the composite function block“ [3].

In our view, the restriction on data arc merge can be “loosened”, allowing the merge if sources of all merging arcs have the same data type. A stronger condition can require also an event arc from each of the source FBs, but it is not necessary, since an input event sent to the destination FB can ensure sampling of the data input. Thus, the merge of data arcs can look like in Figure 6. The same idea can be implemented using adapter connections that can combine both event and data arcs. Then the merge of adapter arcs needs to be allowed.

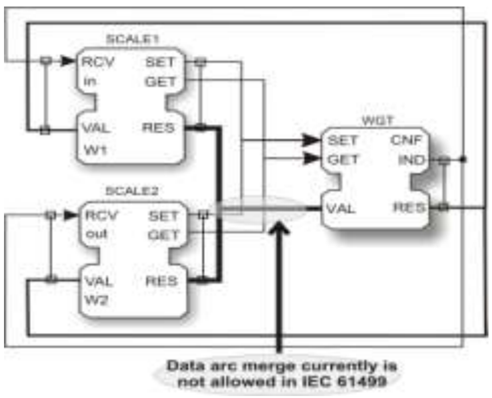

Fig. 6. Proposed extension of IEC 61499: Data Arcs Merging.

However, copying data around function blocks in any case will slow down the execution speed of the application. The slow execution time shall cause the baggage handling system failing. For example, the bag positions cannot be tracked, communication messages cannot be processed in time or sortation system cannot sort any bags as not enough time is given. Except the data handling efficiency, data synchronization is another critical issue which must be considered in object-oriented design. Referring to Figure 7, if a bag record is passed from the database to two function blocks, one is for updating the bag tracking position and the other is for updating bag screening status. After the bag record is duplicated and updated in both tracking update and screening update block, both values will be returned back to the database function block. In this case, the first bag record returned to the database will be overwritten by the second bag record. This shall cause this particular bag record out of sync. Depending on the occurrence of updating event, either the

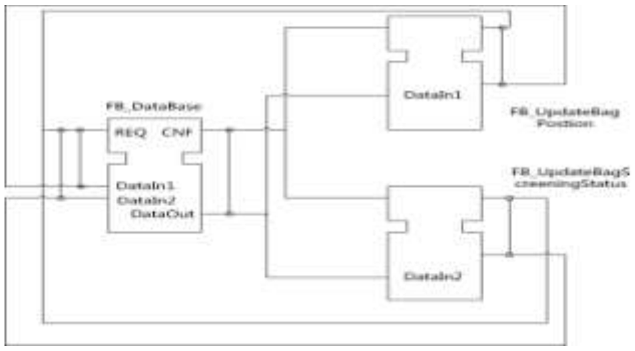

Fig. 7. Data Out of Sync Example in IEC 61499.

new bag position or the new bag screening status will be lost. Although this depends on the execution semantics which is not in this research scope, this situation can be totally avoided already at the design stage. For function block semantics and execution model, please refer to [6] and [7].

Overall, the object-oriented approach seems to bring heavy performance overheads for migration of PLC programs to IEC 61499 FBs.

V. New Approach – IEC 61499 Function Block Model based on Class-Oriented Design

Due to the issues discovered in the previous section, another type of high level programming concept – service-oriented design will be introduced in this section. Service-oriented architecture comprises unassociated, loosely coupled units of functionality that have no calls to each other embedded in them – in other words, grouping reusable, composable and interoperable in the same service [9]. This is majorly used in the web services approach [11]. In the IEC 61499 term of the Class-oriented design, all reusable and interoperable functions should be grouped in a single function block. For instance, all conveyors shall be included in a single function block and all data related to conveyor control shall be local to each other.

Our sample conveyor system from Fig.1 will be redesigned under principals of class-oriented design as follows.

First, all FBCall_conveyor function blocks are grouped together into a single function block – FB_Conveyor. The interface and internal algorithm are given in Figure 8. The physical Inputs status would be updated in a fixed period, for example, 50ms. In every update cycle, an update event would activate the FB_Conveyor to refresh the Input status and update those values inside FB_Conveyor. Once the update is completed, the algorithm will be executed which would go through individual conveyors one by one in the system. An internal memory stack is designed to store all conveyor status. The update state algorithm will pick up each conveyor data from internal memory, go through the conveyor control logics and generate output status. Those outputs module status finally shall be passed back to actual physical output modules. Following the conveyor control function blocks, all tracking function blocks shall be formed into a single function block. Similar to conveyor control function blocks, in every update

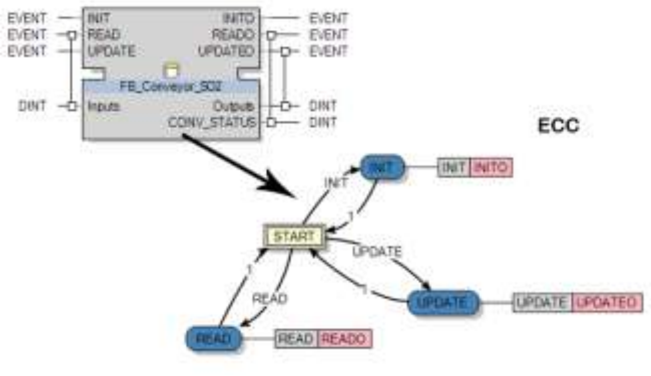

Fig. 8. Class-Oriented FB_Conveyor and ECC.

cycle, an event is raised for reading encoder counts and all photoelectrical sensor status into the FB_Track. Inside FB_Track block, an internal memory stack is used to keep all bag tracking records same as of conveyor function blocks. When the ECC jumps from the *idle* to the *update* state, all bag positions shall be updated one by one in the internal memory. After all bag records are refreshed, all photoelectrical sensor status shall be checked. If any bags arrive at the end of a tracking model, the bag record will be virtually handed over to next tracking model by modifying the status in the memory. No extra handover process is required in this case.

All other parts including HMI function blocks and control panels shall follow the identical guiding principle to be converted into a single function block. A time stamp function block is attached to the each end of function block chain in object-oriented model and service oriented model to compare the execution time. Table 1 below listed the execution in milliseconds for both models. The result demonstrates the service oriented model executes faster under 3 conveyors system with 3 tracking zones. When the number of conveyors and tracking zones goes up, the time difference will be much more significant.

TABLE I
EXECUTION TIME COMPARISON BETWEEN CLASS-ORIENTED AND OBJECT-ORIENTED MODEL

Model	Average Execution Time(ms)	Max Execution Time (ms)
Object-Oriented	15ms	16ms
Class-Oriented	10ms	12ms

[a]The test is based on 3 conveyor and tracking models simulating in FBRT(Java) running on a PC(2.4Ghz CPU, Dual Core)., 1ms is the minimum time Java can catch on a computer.

Comparing to the object-oriented model, class-oriented model is more efficient in data handling. No actual data handover process has taken place during the execution. For limited power PLCs, this will significantly reduce scan time and speed up the entire program execution. All data are still local attached to the function blocks that no global variable is required. This is compliance with the current release of IEC 61499 standard. As all services are actually happening in the basic function block algorithms for the service-oriented model, the overall structure of function blocks network is much cleaner and tidier compared to the object-oriented model. It is beneficial for developers to have better understanding of the design in short time and easily deploy to other projects without any major work.

However, this concept cannot be directly applied to composite function blocks now. As no internal memory is available in composite function blocks, data variables still have to be passed around function blocks inside the composite function blocks. This will also affect the execution efficiency of the entire program. To apply class-oriented design in IEC 61499 efficiently, the basic function block should be considered as the fundamental design unit. The data variables which will be exchanged between basic function blocks inside composite function block must be optimized. The minimum number of data variables passing around the function block network shall provide maximum efficiency for data handling process.

There are some other options for solving data handling issues on implementation level. For example, a service interface function block can be used to access the real time database. Then all function blocks can read from/write to database via that SIFB. Alternatively, the distributed data processing can be applied to speed up the process. Those possibilities will be investigated further in the future.

VI. CONCLUSION

This paper discussed a key issue during migration from IEC 61131-3 PLC centralized control to IEC 61499 function block distributed control – data handling efficiency. The object-oriented design of IEC 61499 does not best suits the real-time complex data handling systems. Instead, a class-oriented design is given and proven to better suit this system. Class-oriented approach provides an efficient use of system resources and easier understanding and better performance compared to object-oriented approach especially for real-time high performance systems. Finally, to apply the class-oriented approach to all generic cases, the data handling in composite function block must be further investigated to fit the service-oriented model better.

VII. REFERENCES

[1] W. Dai, V. Vyatkin, "A Case Study on Migration from IEC 61131 PLC to IEC 61499 Function Block Control", 7th International IEEE Conference on Industrial Informatics, (INDIN'09), Cardiff, June 2009

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

[3] IEC 61499, Function Blocks, *International Standard*, First Edition, 2005

[4] G.Michel, Programmable Logic Controllers – Architecture and Applications, John Willey & Sons, London, 1992.

[5] J.W. Webb and R.A. Reis, Programmable Logic Controllers: Principles and Applications, Prentice-Hall, Engelwood Cliffs, NJ, 1999.

[6] P. Benjamin, Types and Programming Languages, MIT Press. ISBN 0-262-16209-1, section 18.1 "What is Object-Oriented Programming?"

[7] C. Sunder, A. Zoitl, J.H. Christensen, M. Colla, T. Strasser, "Execution Models for the IEC 61499 elements Composite Function Blocks and Subapplication", 2007 5th IEEE International Conference on Industrial Informatics, Volume 2, 23-27 June 2007 Page(s):1169-1175.

[8] V. Vyatkin, V. Dubinin, "Sequential Axiomatic Model for Execution of Basic Function Blocks in IEC 61499", 2007 5th IEEE International Conference on Industrial Informatics, Volume 2, 23-27 June 2007 Page(s):1183-1188.

[9] H. Channabasavaiah, H. Tuggle, "Migrating to a service-oriented architecture" IBM Developer Works, 16 Dec 2003.

[10] Baggage Handling System Sample Graph, Retrieved from http://www.robson.co.uk/Airport_conveyor_solutions/Baggage_handling_conveyors.htm, 14th Dec 2009.

[11] D.K. Barry, "Web Services and Service-Oriented Architectures, Part 1 Service-oriented architecture overview", Published by Morgan Kaufmann Publishers, ISBN 1-55860-906-7.