Distributed Control Patterns for Intelligent Mechatronic Systems

Majid Sorouri
The University of Auckland,
Auckland, New Zealand.
msor021@aucklanduni.ac.nz

Sandeep Patil
The University of Auckland,
Auckland, New Zealand.
spat251@aucklanduni.ac.nz

Valeriy Vyatkin
The University of Auckland,
Auckland, New Zealand.
v.vyatkin@auckland.ac.nz

Abstract—This paper describes a step-by-step approach for developing modular and reusable logic controllers of mechatronic systems. Following that, it presents the results of employing three distributed control methods, including master-slave, peer-to-peer and independent controllers on a pick-and-place robot as a simple showcase of implementing a distributed control system on industrial applications, using the emerged IEC 61499 standard. Moreover, it addresses possibilities and challenges of using each approach.

***Keywords*— Distributed Control logic, plug–and-play control logic, smart mechatronic components, automation systems, industrial controllers, cyber-physical systems**

I. INTRODUCTION

The design of modular, versatile, and reusable (and ideally "Plug-and-Play") control logic systems has been one of the principal challenges confronted by modern manufacturing industries. This is due to the fact that the conventional control paradigms, which are essentially based on case by case design and centralization of control tasks, lack fundamental capabilities such as agility, scalability, handling complexity, re-configurability and fault tolerance [1], which all hinder this industry to meet the ever increasing expectations of the market. Some of these abilities have been partially provided by other influential players to the manufacturing world. For instance, mechanical engineers' implementation of modern material handling systems with prefabricated modular parts has considerably facilitated assembly, reconfiguration and portability in various industrial applications, including manufacturing [2]. Similarly, over the past decades, single-purpose industrial controllers have been gradually replaced by robust software components providing more flexibility, interoperability and reusability to the manufacturing systems and many hardware vendors are nowadays producing devices that can be utilized in different systems without much customization effort (plug-and-play)[3, 4].

In software engineering, over the last decades, a large amount of research has been devoted to implement and improve logic control design methods and an extensive review of such scientific approaches may be found in [5]. However, in spite of the considerable advances and achievements of all these academic endeavours, these software methods are still far from fulfilling modern industrial demands[6, 7].

It is therefore vital that software designers define suitable design patterns that concentrate on "cross-cutting control logic functionalities and suitably abstracting from application specific details"[8] which will make the design of complex automated systems easier and faster for engineers and practitioners. In fact, the ideal case would be to have intelligent software components that have most of their functionalities "pre-automated by the vendors" [3] and users can conveniently tailor them to their particular application with the least integration effort. Furthermore, these software components would be capable of being distributed throughout the network, regardless of the size of the application, which can be as small as a simple drilling machine [9] or as enormous as a smart grid [10] dominating across a country or even a continent, while each software component is able to interact with the entire system and sharing data and information with them.

II. IEC 61499 STANDARD FOR DISTRIBUTED CONTROL SYSTEMS

Among the current industrial programming standards for industrial controllers, IEC 61499 [11-13] promises significant advantages to the users in terms of simplicity of system level design and more importantly, the possibility of distributing control programs over variant distributed hardware and throughout the entire network [14, 15].

According to the IEC 61499, a function block's interface comprises a number of data and event inputs and outputs. The functionality block's main execution semantics consists of a finite state machine known as 'ECC' (Execution Control Chart) which is composed of a set of states that are connected with Boolean transition conditions and each state can contain one or more actions being defined as algorithms. On an input event trigger, the associated input values are refreshed and depending on the ECC, one or more algorithm(s) may be executed. Finally, after execution of algorithm(s) at each state, output data are generated and the designated event outputs will be issued [9].

Despite a number of advantages promised by the IEC 61499 technology, its industrial acceptance are still limited, which can be partially explained by the lack of proven design methods and patterns for distributed systems software of modular mechatronic components. Hence, in order to migrate sustainably from centralized to distributed control, some questions and concerns need to be addressed including:

- How could larger and more complex systems be incrementally derived from simpler controllers in a generic way [3]?
- How could the controllers' interfaces be defined in a standard way so that controller components can be connected together and reused conveniently [3]?

III. AIMS AND OBJECTIVES

To address some of the aforementioned concerns and to step toward facilitating distributed control design, this research proposes a systematic approach toward controller design of mechatronic components and as an instance, a design process for a controller block of a cylinder whose instances can be adapted to different control patterns with the least modification effort is described. Following that section, it presents three design approaches using the IEC 61499 function blocks, namely master-slave, peer-to-peer and independent control as well as comparing them with the traditional centralized control. Then, each scenario has been evaluated in terms of design effort, complexity, modularity and reusability in a simple Pick-and-Place Robot developed as a case study.

IV. DESIGN OF A REUSABLE CONTROLLER Function BLOCK (CFB)

In order to design a reusable and application-independent **C**ontroller **F**unction **B**lock (CFB) which is in charge of managing all the behaviours of the mechatronic components assigned to it, the following steps must be taken.

A. Specify Functionality:

The first step to design a CFB is to precisely define its functionalities, which include specifying all the algorithms needed to utilize its actuators. In the specified example shown in Fig 1, the only expected functionality of a double-acting cylinder CFB would be to set two Boolean values associated with two pneumatic valve switches to either ON or OFF.

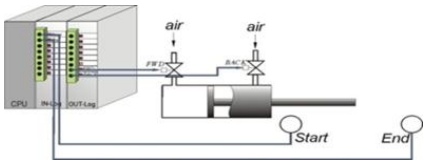

Fig 1: A basic single cylinder control system.

Functionalities of a CFB, appear as algorithms in the ECC states (e.g. GoForward and GoBackward shown in Fig 2) and in this case, are written in Structured Text which is one of the IEC 61131-3 [13] compliant languages. It is important to note that even two mechatronic components which may operate similarly might have different CFB functionalities. For instance, a single-acting cylinder (spring-return), compared to a double-acting cylinder (mentioned above) has a different functionality, as it only has one output and requires setting one Boolean value into true or false. Therefore, physical objects with various functionalities require different controller types.

B. Self-Perception (S):

The term 'self-perception' is used to ascertain the information a CFB receives as input data from its own sensors. In the cylinder example, the two end position sensors (Home and End as depicted in Fig 2), will enable the controller to perceive its location and find out when the cylinder arrives at either end positions. These two states are represented in the ECC as 'CylAtEnd' and 'CylAtHome' (Fig 2). All sensor inputs are connected to an input event ('REQ') which refreshes sensors data, whenever their values are changed. Any change in functionality of a mechatronic device, may also pose some changes in its self-perceptions. For instance, if in an application, a cylinder is assumed to stop in the middle position, then, adding another sensor would be necessary, to notify when the cylinder has reached its middle point.

C. Initialization(I):

In general, initialization is required for any controller block which demands pre-set outputs. In a CFB, as soon as the 'INIT' event is triggered, the 'INIT' algorithm is invoked and it initializes the CFB by giving initial values to the outputs before it is normally executed (Fig 2). Then, the output event ('INITO') will be issued to confirm initialization and if connected to other FBs, trigger their initialization event.

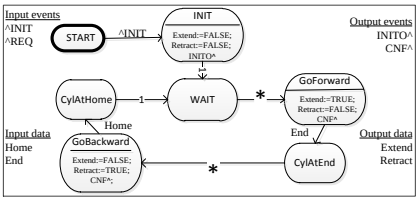

Fig 2: Cylinder controller for the system depicted in Fig 1 (prior to applying any decision condition)

Following the aforementioned steps, so far, the controller has acquired its interface with the plant[3], as well as the required algorithms for its operation . However, it is not yet capable enough to be utilized in any application, since no task has been assigned to it, and as well, it is not known when and under what conditions, these tasks should be performed. This fact is illustrated in Fig 2 by the missing transition conditions, denoted with '*****'. The ECC of the cylinder CFB contains five states, including 'INIT', 'GoForward', 'CylAtEnd', 'GoBackward' and 'CylAtHome', along with the three algorithms 'INIT', 'AlgExtend' and 'AlgRetract'. There is one more state, called 'Wait' which is designated for when the controller is in idling condition. The created states are then arranged in a logical sequence and characterize the initial form of the CFB internal part. Such reusable FBs representing controllers of physical objects may be selected from the repository of function blocks and tailored to different applications which will be discussed further in the following section.

V. ADAPTATION OF A REUSABLE CONTROLLER TO A SPECIFIC PROBLEM

In order to design a CFB to be fully operational in a system and able to solve a particular problem, more issues need to be addressed which are as follows.

A. Environment Perception (E):

The environment where a mechatronic component is located and being used may impose some boundaries or limitations on its CFB operation, and the controller must be aware of those conditions known as 'environment-perception' (e.g. information to avoid collision with other objects or machines).

B. Task Perception(P):

The other term 'task-perception', identifies specification of the tasks to be managed by the controller, so that it knows the details of the goal intended to be achieved. A CFB can gather the information needed for these two perceptions, either through direct access to the sensory data, or by receiving it from other controller blocks. Obviously, it is also possible for a controller to pass such information to other blocks. Similar to self-perception data mentioned in the previous section, all input data representing task-perception and environment-perception will be associated with 'REQ' events.

C. Decision:

A controller function block needs to analyse the obtained information (from the plant and other blocks) and based on that information, make the appropriate decisions. The "If-statement" conditions represented by *arrows* in the ECC are considered as *decisions*, due to the fact that they identify when an actuation must take place. For instance, in the ECC illustrated in Fig 2, if in the empty spaces marked with '*', number '1' (always true) is written as a transition condition, the cylinder will continuously move back and forth, or in another case, if an input event is put in both conditions, each time it is triggered it will permit the algorithm to be executed and cause the cylinder to either extend or retract.

D. COMMUNICATION (C):

The last influencing factor in the design of a controller function block taking part in a distributed system is the way it interacts with the other existing controllers of the system. CFBs constantly exchange some events and data between one another to be able to coordinate their behaviours and collaborate to achieve their common goal. In general, these exchanges are of two types: information or command. "Information" consists of all the perceptions that need to be processed prior to any decision making. In contrast, "Command" is obviously what can be immediately followed and no further processing inside the CFB is required.

Following all these steps will guide the developer through the design of a proper controller for a mechatronic component. However, there have always been some difficulties in identifying how many input/output events should be considered for a controller interface, how uniformly they can be named and categorised, so that they can be easily recognised, and finally, how they must be associated to data inputs and outputs. Therefore, to resolve this issue, it is proposed to classify events and consequently their associated data into the following four groups: *Initialisation, Perception, Follow and Order* (Fig 3).

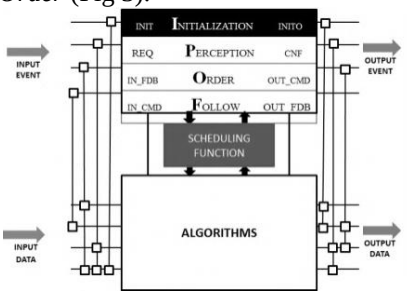

Fig 3: Event classification proposed for the development of a re-usable controller function block interface

*1) Initialization (**I**)*: INIT-INITO events are present in almost all controller blocks and are only connected to the data which is involved in Initialization (briefly discussed in previous section). For those exceptional controllers which do not require any pre-set values, this type of events can be eliminated.

*2) Perception (**P**):* These types of events will be responsible for exchanging information with other blocks as well as with the sensors and actuators. A pair of 'REQ-CNF' event is proposed for this purpose. It must be noted that the REQ event only has to refresh the input data (upon trigger) and must not be used in any ECC transition condition. As for the 'CNF', each time an output data value is changed, a 'CNF' (Confirmation) event will be triggered. In larger applications where a great deal of data is being exchanged among numerous controller blocks, more pairs of 'REQ-CNF' events may be considered, so that each pair can be associated with one cluster of data which is shared with a limited number of blocks to avoid an excessive-event trigger of irrelevant controller blocks.

3) Order events (**O**): 'IN_FDB' and 'OUT_CMD' events are employed when a block sends commands to one or more controller blocks to force them to perform an operation, followed by receiving feedback from them. For every *independent* command (A command is *dependent*, when it is used for several operations), issued by a CFB, *one state* and *one transition* condition must be added to its ECC and by following the process activity planning, be placed between two intended states(e.g. Fig 10). Moreover, given the fact that an event happens at an instant, and its value cannot be retained for a period of time, when a combination of commands is needed as a condition, or when there are other conditions that have to be met before the command is issued, single event command will not suffice and 'OUT_CMD' has to be accompanied with a Boolean *data variable* which will take part in transition condition. This is demonstrated in Peer-to-Peer control design (section VII). Moreover an 'OUT_CMD' event may be associated with more data types to clarify some particular attributes of the command (e.g. running a motor at different speeds).

A follower CFB must specify to the sender, the boundary, where, a command can be issued effectively and this is normally done by a Boolean feedback signal 'Ready' (whose value is 1, when Ready, and 0 when NotReady). In general, for each independent command being sent to a follower CFB, one feedback must be allocated to identify the availability of the follower to its commander. However, there are exceptions. For instance, it is possible that one feedback be commonly used for two or more dependent commands (e.g. slave cylinder shown in Fig 9), or the case where the follower is always ready to receive commands and the feedback's "True" value will not change (e.g. vacuum CFB in Fig 7). Finally, in commander CBD, the feedback data received must be included in the transition condition which goes to the command state (where the command event is issued).

*4) Follow events (**F**)*: They are inversions of *Order events* and used in a CFB that follows the commands received from other blocks. They will be represented by 'IN_CMD' and 'OUT_FDB' events in the FB interface.

VI. CASE STUDY DESCRIPTION

A pick and place robot as shown in

Fig 4(a) is selected as a case study and consists of four actuators, two double-acting pneumatic cylinders (horizontal), one single-acting pneumatic cylinder (vertical) and one vacuum unit. Along with the actuators, this device is equipped with 11 sensors. Ten of them are position sensors indicating the availability of work pieces on trays and cylinder end positions, and the last sensor detects the vacuum On/Off condition. Each actuator along with its associated sensor/s is considered as one mechatronic object. Hence, the robot is basically composed of four mechatronic components, three cylinder units and one vacuum unit. The following is a list of requirements, facts, and conditions associated with the robot operation which is exploited for identifying task (T)/environment (E) perceptions and of each controller component:

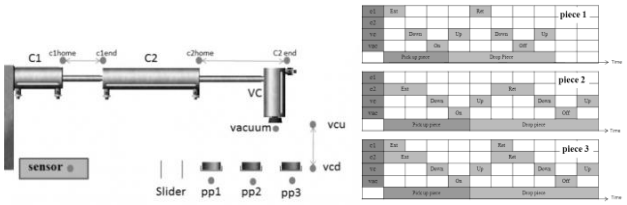

Fig 4: (a) Reference object: pick and place robot. (b) Activity diagram of the pick and place robot

A. The role of the device is to pick up work pieces which are passed from other units into three trays (pp1-pp3), and drop them on a slider. (T)

B. The robot configuration is as illustrated in Fig 4.(E)

C. PP1 can be reached by extension of C1, pp2 is accessible by extension of C2, and finally getting to pp3 will require both cylinders extensions.(E)

D. For safety reasons, and to avoid collision with other devices, the vertical cylinder must start its movement only when C1 and C2 are at standstill condition (their end positions). (E)

E. The precedence and order of picking up pieces from the trays should be 1-2-3. (T)

F. The robot should start its operation when new work pieces are added, and stop when no work piece is available. (T)

VII. Different Control Scenarios

In this section, a centralized solution, along with three distributed control scenarios are presented, which are developed by following the proposed methodology of employing reusable CFBs. Furthermore, the way these different scenarios will reflect on the interface and internal part of each CFB is analysed.

A. Conventional Centralized Controller

In this type of design, a single block controls the entire plant and the control logic is encapsulated in one CFB (Fig 5). Two sets of input-output events are required for this method. The 'INIT-INITO' events initialize the output data, while 'REQ-CNF' events refresh the sensor-actuator data values being exchanged with the plant.

In spite of the magnificent benefits of applying the IEC61499 standard, such as time saving and reduction in design efforts, using it in a centralized approach, seems to have no particular advantage over conventional industrial programming languages, as for any new configuration, such as the one illustrated in Fig 6, or when task descriptions alter, the whole design process must be initiated from scratch. In general, centralized controllers may be suitable for some applications, in the sense that they acquire a single point of control as well as easy integration and optimisation, but have considerable drawbacks, such as having poor scalability, high communication overhead and low resilience to failure [16].

B. Distributed Controllers

1) Master-Slave Controllers: In this method of distributed controller design, one controller (master) will take control of one or more controllers and guide them through their operations via sending commands (e.g. work piece manager depicted in Fig 7 and Fig 8). In this case, the slave CFB does not need to make any decision on such operations, and acts solely based on the commands being sent to it. Moreover, it is possible that a controller act as a master with respect to some controllers, and at the same time, be a slave to others [3]. This is the case for the vertical cylinder, as it acts as a slave to the work piece manager and a master to the vacuum unit. (Fig 8 and Fig 10)

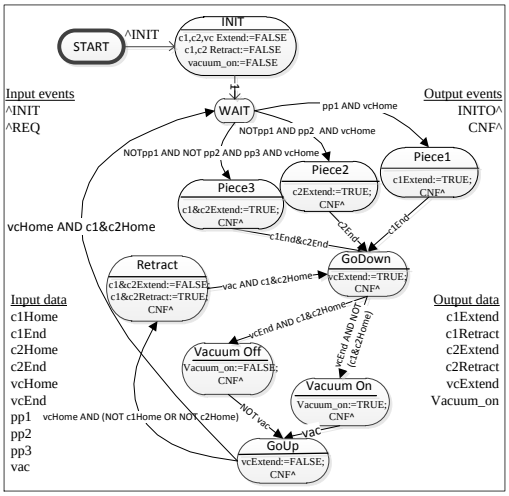

Fig 5: State diagram of a Centralized Controller

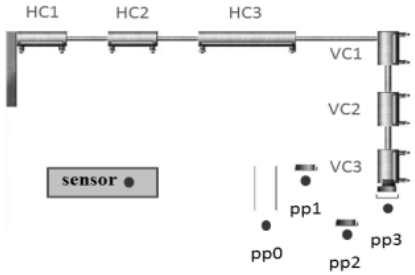

Fig 6: Example of a more complex configuration of a pick and place robot along with work piece trays positioned at various levels [18]

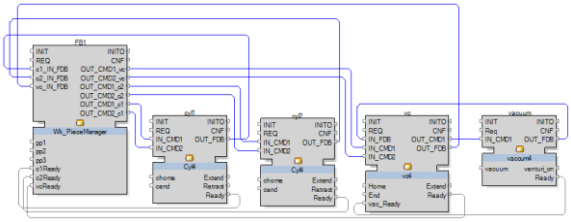

Fig 7: Distributed control of the robot with Master –Slave relation (connections with the plant are emitted for simplicity)

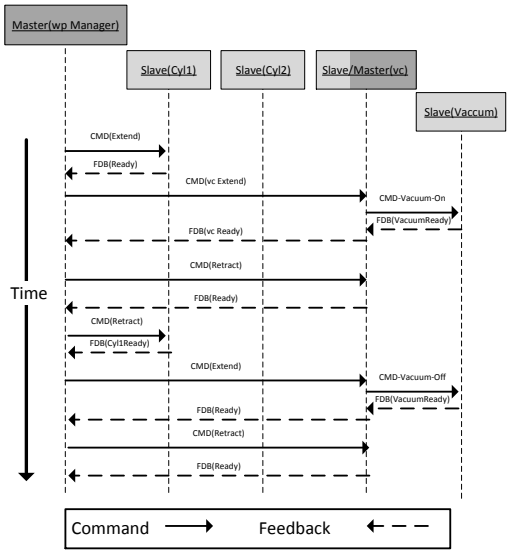

Fig 8: An example of event exchange among master-slave controllers to pick up a work piece from tray one

Fig 9 shows the state diagram for C1 and C2 modelled the same way as formerly described in section V. Due to their slavery nature, they do not require any task perception **(P)** ('REQ', will not be used) and merely follow their master's commands **(F),** ('IN_CMD1' to extend, 'IN_CMD2' to retract and 'OUT_FBD' to announce their readiness to the master). Moreover, in contrast with the vertical cylinder, horizontal cylinders do not command any other CFB, and therefore, do not require order events (O). Fig 9 and Fig 10 clearly depict the differences caused between these two cylinder CFBs, which is derived from their various roles in the control system.

These master and slave controller blocks can be fully re-used in a new system configuration such as the one shown in Fig 6. There will be more such CFBs added to the controller design and the only changes will be involved in the work piece manager. The changes to this block can also be automated (created dynamically) using a meta-model representation of the whole system. Given an input similar to

Fig 4(b), the work piece manager can be dynamically generated.

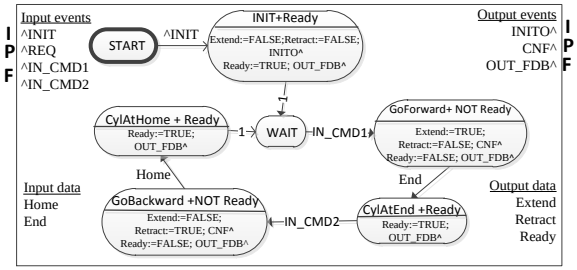

Fig 9: Controller of the horizontal cylinders being implemented as slaves to work piece manager

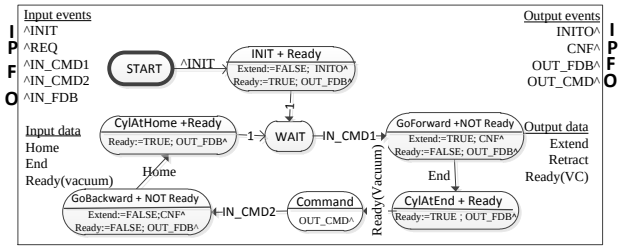

Fig 10: Controller of the vertical cylinder implemented as Master for vacuum unit and slave for work piece manager

2) Peer-to-Peer Controllers: This approach can be considered as bilaterally apply a Master-Slave relationship for two CFBs in a way that *order* ***(O)*** and *follow* ***(F)*** events merge together in a way that CFBs only send commands to each other, while each command acts simultaneously as a feedback to the other block' command, and vice versa, as illustrated in Fig 11 - Fig 14. This method is very useful, particularly for objects that are subject to some restrictions by their adjacent function blocks in a way that one action is permitted or blocked by other action(s) of an adjacent block and contrariwise [3].

In the case study, the two horizontal cylinders, 'C1' and 'C2' permit the vertical cylinder VC to extend; alternatively the VC allows C1 and C2 to retract. Also, as the VC arrives at its end position, it orders vacuum unit to either turn ON (pick up) or OFF (drop) and once its condition is changed, sends a command to VC to retract.

The controllers for C1 and C2 are slightly different in a sense that each responds upon the presence of different work pieces at the trays. For instance, to pick up a work piece from pp2, C1 bypasses the GoForward state (does not extend) and just permits the vertical cylinder to extend, whereas cylinder Two has to extend before permitting the VC to extend. Therefore, to maintain simplicity and uniformity of controller function blocks, both task perceptions **(P)** are provided in one function block. Following that, C1 and C2 will be instantiated from the same CFB, and then, distinguished by passing different ID numbers to them as an input (Fig 11). This method can be applied conveniently to other applications when one universal block is designed to be instantiated in various environment conditions.

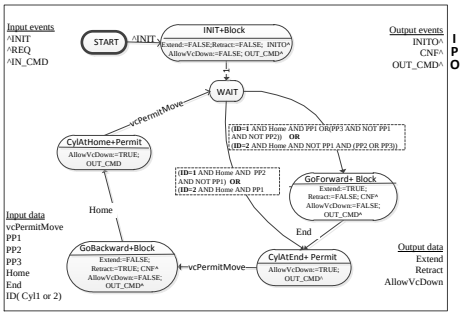

Fig 11: Controller of horizontal cylinder in Peer-to-Peer relationship with the vertical cylinder

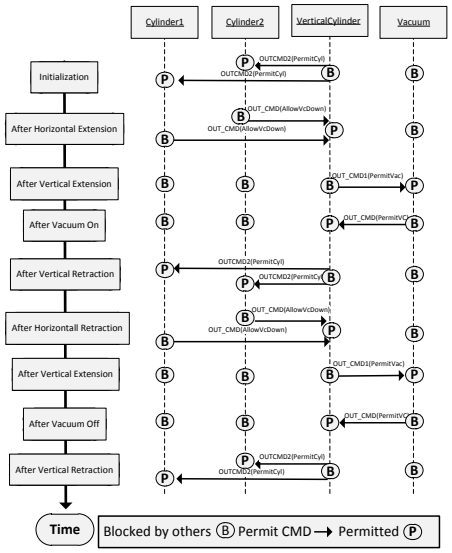

Fig 12: An example of event exchange among Peer to Peer controllers when they aim at picking up a work piece from tray one

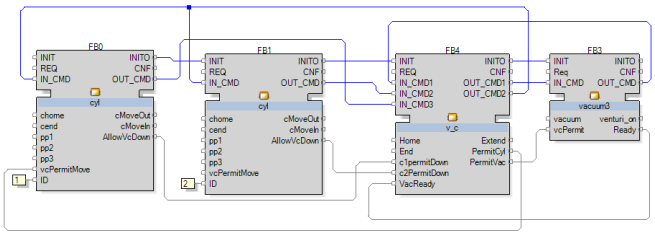

Fig 13: Distributed control of the robot in Peer-to-Peer relation

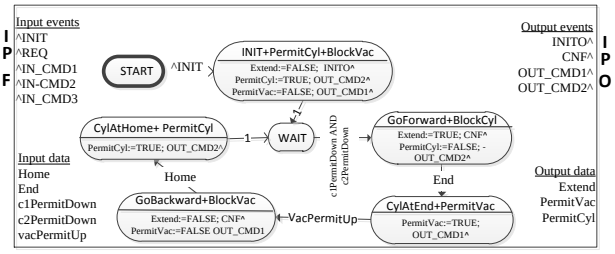

Fig 14: Controller of the vertical cylinder while having Peer-to-Peer relationship with other C1 & C2, and master relationship to Vacuum

As it was mentioned in section V, order events (O) can be associated with a Boolean data variable and in c1 and c2 (Fig 11), 'AllowVcDown' is the variable associated with 'OUT_CMD' and in the vertical cylinder, "PermitVac" and "PermitCyl" are the Boolean data associated with the (O) event 'OUT_CMD1' and 'OUT_CMD2' respectively (Fig 14).

3) Independent and distributed controllers: In this method, the CFSs are provided with as many sensor data as they require being able to thoroughly perceive their task and working environment. In this method, although the control logic is distributed among several CFBs, they perform independently and do not communicate with each other. Hence, no **(O)** and **(F)** events are used in such controller blocks and as depicted in Fig 15 - Fig 16, each CFB has a different structure, and as a result will not be reusable. Implementation of such design method in small applications is easier when compared with centralized approach; however, it would become very complex for large systems, as the designer should consider all operation states for designing each controller.

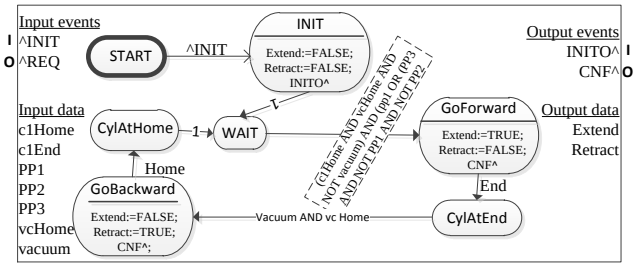

Fig 15: Independent and distributed controller of cylinder1

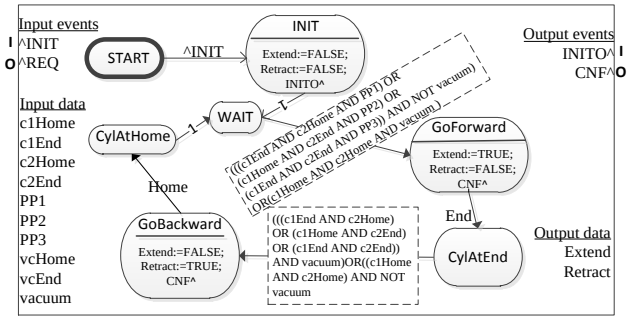

Fig 16: Independent and distributed vertical cylinder controller

VIII. CONCLUSIONS

In this paper, a systematic method of designing re-usable controller function blocks of mechatronic components is described. Also different approaches were proposed to adapt CFBs conveniently into distributed control systems. All the aforementioned control logic approaches were successfully implemented in NxtStudio®1.5[17] environment. In all selected methods, the robot had identical performance. Finally, this study described the challenges and solutions ahead of implementing plug-and- play control systems. Although the complexities of most industrial applications are much greater than the described example, it is highly believed that the same approach can be pursued to facilitate implementation of distribution control systems and thereby benefit from its numerous advantages over centralized systems.

IX. FUTURE WORKS

In continuation of this research these methods will be examined on more complex applications, with population of more mechatronic objects having multi communication links, and also some controllers will be implemented in parallel to study how fault tolerance should be dealt with in distributed control logic. We are already in progress for a follow up paper on how this method of plug-and-play modules may be used for formal verification of a similar system. The same case study is being used in that paper as well.

REFERENCES

[1] G. Black and V. Vyatkin, "Intelligent Component-Based Automation of Baggage Handling Systems With IEC 61499," *Automation Science and Engineering, IEEE Transactions on*, vol. 7, pp. 337-351, 2010.

[2] E. Schweikardt and M. D. Gross, "Learning about Complexity with Modular Robots," presented at the Proceedings of the 2008 Second IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning, 2008.

[3] V. Vyatkin, S. Karras, T. Pfeiffer, and H. M. Hanisch, "Rapid Engineering and Re-Configuration of Automation Objects Using Formal Verification," *International Journal on Manufacturing Research*, vol. 1, pp. 382-404, 2006.

[4] L. Penebo and B. Hansson. *Plug and Play in Control Loop Design-ABB Automation Technology Products* Available: http://www05.abb.com/global/scot/scot296.nsf/veritydisplay/30d46d3f3466b41fc1256d43004286ac/$file/3BSE028758_-_en_PLUG_AND_PLAY_IN_CONTROL_LOOP_DESIGN.pdf

[5] G. Frey and L. Litz, "Formal methods in PLC programming," in *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, 2000, pp. 2431-2436 vol.4.

[6] M. Bonfe and C. Fantuzzi, "Application of object-oriented modeling tools to design the logic control system of a packaging machine," in *Industrial Informatics, 2004. INDIN '04. 2004 2nd IEEE International Conference on*, 2004, pp. 569-574.

[7] C. Maffezzoni, L. L. Ferrarini, and E. Carpanzano, "Object-oriented models for advanced automation engineering - modular modeling in an object oriented database," *Control Engineering Practice*, vol. 7, pp. 957-968, 1999.

[8] E. Faldella, A. Paoli, A. Tilli, M. Sartini, and D. Guidi, "Architectural design patterns for logic control of manufacturing systems: The generalized device," in *Information, Communication and Automation Technologies, 2009. ICAT 2009. XXII International Symposium on*, 2009, pp. 1-7.

[9] V. Vyatkin, *IEC 61499 Function blocks for embedded and distributed control systems design*: OOONEIDA, 2007.

[10] V. Vyatkin, G. Zhabelova, N. Higgins, M. Ulieru, K. Schwarz, and N. K. C. Nair, "Standards-enabled Smart Grid for the future Energy Web," in *Innovative Smart Grid Technologies (ISGT), 2010*, 2010, pp. 1-9.

[11] International Electrotechnical Commission - IEC61499, "Function Blocks for Industrial Process Measurement and Control Systems – Part 1: Architecture," ed. Geneva: Tech. Comm. 65, Working group 6, 2005.

[12] V. Vyatkin, *IEC 61499 Function Blocks for Embedded and Distributed Control Systems Design* vol. 2: ISA 2007.

[13] T. Schmidberger and A. Fay, "A rule format for industrial plant information reasoning," in *Emerging Technologies and Factory Automation, 2007. ETFA. IEEE Conference on*, 2007, pp. 360-367.

[14] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State of the Art Review," *IEEE Transactions on Industrial Informatics*, 2011 2011.

[15] V. Vyatkin, S. Karras, and T. Pfeiffer, "Architecture for automation system development based on IEC 61499 standard," in *Industrial Informatics, 2005. INDIN '05. 2005 3rd IEEE International Conference on*, 2005, pp. 13-18.

[16] F. Somers, "Hybrid: unifying centralised and distributed network management using intelligent agents," in *Network Operations and Management Symposium, 1996., IEEE*, 1996, pp. 34-43 vol.1.

[17] nxtControl. ((2010, 10/05). nxtStudio.). Available: www.nxtcontrol.com