Function Block Implementation of Service Oriented Architecture: Case Study

Wenbin (William) Dai,
Luleå University of Technology, Sweden,
IEEE Member, w.dai@ieee.org

Valeriy Vyatkin,
Luleå University of Technology, Sweden and Aalto University, Finland, *Senior IEEE Member*,
vyatkin@ieee.org

James H. Christensen,
Holobloc Inc,
james.h.christensen@gmail.com

Victor Dubinin,
Penza State University, Russia
victor_n_dubinin@yahoo.com

Abstract – The Service Oriented Architecture (SOA), initially developed for general purpose computing, is becoming increasingly popular in industrial automation, especially due to the growing importance of distributed measurement and control systems, Internet of Things and wireless sensor network infrastructures. There are two serious gaps in the current implementations of SOA for automation: the lack of a proper standard form for representing the logical relation between services at the system level, and, in particular, lack for a proper visual representation format enabling intuitive understanding and reconfiguration of services during the system lifecycle. In this paper, it is shown how these gaps can be filled using the IEC 61499 function block architecture in the most intuitive and natural way. The ideas are illustrated on example of systems, composed form intelligent mechatronic devices. Such features, as software reuse, flexibility, reconfigurability and scalability are demonstrated. The paper shows complementarity of the SOA and IEC 61499.

I. Introduction

Industrial automation experiences the growing importance of distributed measurement and control systems, Internet of Things and wireless sensor network infrastructures. Service Oriented Architecture (SOA) [[1]], initially developed for general purpose computing, is becoming increasingly popular in industrial automation. In the SOA way of thinking, functionalities are encapsulated in services. Services are communicating with others only by using message passing mechanism. A service sends a request message, another service receives the message, executes the service invoked and sends a response message if needed.

There are several research works on applying SOA in the industrial automation domain [[2]-[7]]. However, there are two serious gaps in the current implementations of SOA for automation: the lack of a proper standard form for representing the logical relation between services at the system design level, and, in particular, lack for a proper visual representation format enabling intuitive understanding and reconfiguration of services during the system lifecycle.

The analysis, presented in [[8]], shows that IEC 64199 architecture for distributed automation, is appropriate to apply the SOA paradigm in the industrial automation domain. In this paper, it is further shown how the above mentioned gaps can be filled using the IEC 61499 function block architecture in the most intuitive and natural way. The ideas are illustrated on example of systems composed form intelligent mechatronic devices. Such features as software reuse, flexibility, reconfigurability and scalability are demonstrated.

The paper shows complementarity of the SOA and IEC 61499. The graphical representation of function block networks fits perfectly to the role of top level description diagram of services and relations between them. The rest of the paper is structured as following. In the section II, the basic principles of SOA in automation are provided. Then the idea is illustrated on a case study example in the section III. In the section IV, two important SOA concepts: service orchestration and composition are discussed. Finally, the recommendations of implement SOA in industrial automation using FB are listed.

II. Service-oriented architecture in Automation: basic principles

Main concepts of service-oriented architecture in industrial automation (SOA/IA) can be formulated as follows:

On the logical level:

1) The functionality is encapsulated into atomic elements called services.
2) Services may invoke other services via message passing.
3) All process data (sensors and actuators) are independently accessible and addressable, and access to them is also implemented via services, e.g. Sensor1.GetValue(), or Actuator2.SetValue(1).

On the physical level, these assumptions presume total connectivity of all actors in the automation system via a network, e.g. the Internet or industrial fieldbuses. With this assumption, the allocation of services to particular hardware devices is not important, provided that performance and communication penalties are within acceptable boundaries. In particular, it is widely discussed that some services can be executed right at the embedded computing devices, even within (smart) sensors and actuators, while others can be executed in the Cloud. One of the key features of SOA is that services are loosely coupled. This feature provides extreme flexibility with which services could be discovered and accessed from different execution platforms.

978-1-4799-4905-2/14/$31.00 ©2014 IEEE

A downside of the SOA concept is the lack of dependability: it is hard to validate system's behaviour without having the explicit system-level "picture" of the entire distributed functionality. Despite some works on illustrating service sequence diagrams, programmers still need to put in manual efforts as it is not a standard part of SOA. The first step in the validation is understanding. Even that is hard without having a visual representation of all related services and interconnections between them. That is why there is ongoing pursuit of a proper visual representation form that could capture such aspects as relation between services and their internal organisation.

The function block architecture of IEC 61499 contains all necessary artefacts to provide a solution for this task as it is conceptually illustrated in Fig. 1. Function block types are considered as service type definitions. Message passing between services is represented by connections between function blocks. There are two types of connections in the IEC 61499 standard: event and data. Data inputs and outputs must be associated with event input and outputs in order to pass values in and out of function blocks. In the SOA view, each event connection is referred to as a message type. Data variables associated with this event are used as the input parameters of the message.

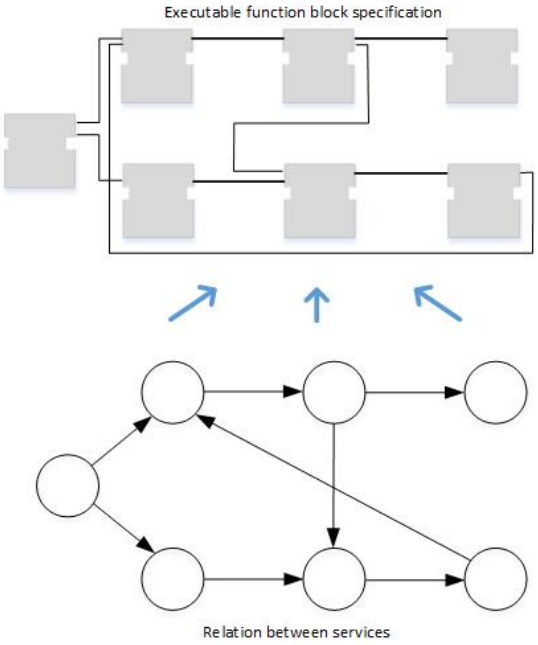

Fig. 1: A function block application generated to implement requirements specified in the form of services.

III. Illustrative Example

In this paper, a family of reference examples with increasing level of complexity will be considered. The first one in Fig. 2 consists of just one linear motion pusher. Once a workpiece is placed in front of the pusher (that is detected by WPS sensor), the desired service of this system is to push the workpiece to the destination sink and retract the pusher to the initial state. The Figure also shows the hardware architecture of this system that fits to the Internet of Things vision: here all sensor and actuator devices are equipped with embedded microcontrollers and network interfaces.

This provision could be regarded as extreme and redundant at the current level of technology development, however it may well be valid as a long term vision. The benefit of such architecture is the ease of integration of various components into a system. In this study, this assumption motivates investigation of the intelligence implementation in a form of device independent services. This way, software components, encapsulating the devices' functionality can be also easier assembled to a working software system. With such a software architecture, the need for having an embedded microcontroller in each physical device, like sensor or actuator, is not compulsory: several services can be combined in one hosting microcontroller, if this is justified by the costs or performance reasons.

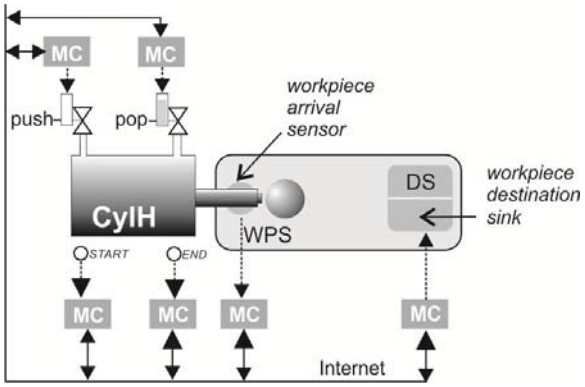

Fig. 2: Workpiece transfer system with one linear motion pusher.

Further in the paper, several more complicated modular mechatronic examples will be considered that are composed of the same mechatronic parts and whose (much more complicated) behavior can be "composed" from the same set of services.

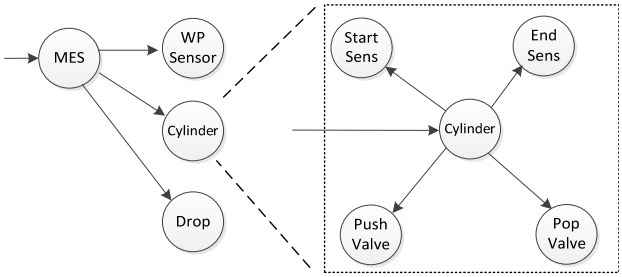

Fig. 3: Graphical representation of a service orchestration.

A. Specification and implementation of the control

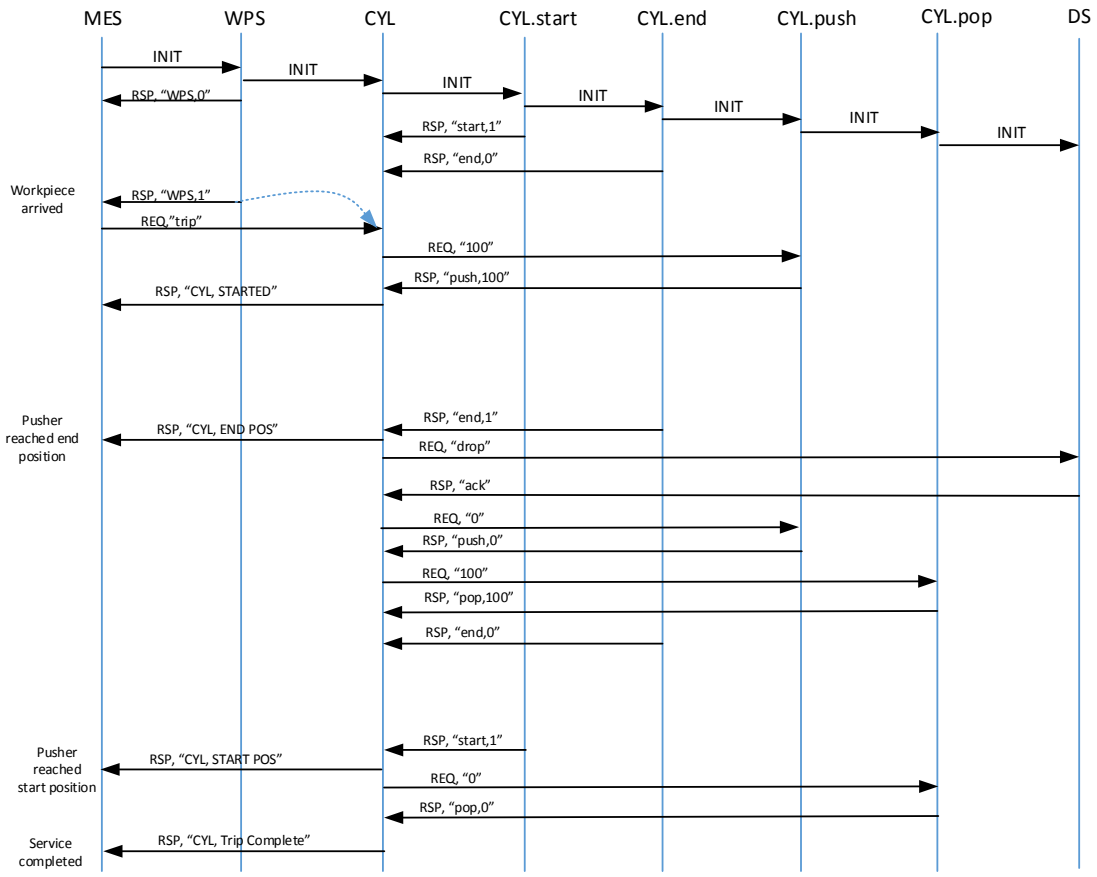

Fig. 4 Trace of communication between the mechatronic components towards execution of the "trip" service.

The functional components of this system in terms of SOA can be represented as a service orchestration graph, where nodes represent services and directed edges represent the direction of service requests, as shown in Figure 3. Service *orchestration* is a kind of service composition that assumes a central coordination entity. Alternatively, *service choreography* relies on ability of services to coordinate in a purely distributed way without a central coordinator. Each service request may be accompanied by a confirmation message, going in the reversed direction, but it is not shown in the diagram.

The behaviour of "linear motion pusher" system is implemented as orchestration of three services: workpiece sensor, cylinder and drop, interacting with the Manufacturing Execution Systems (MES). The latter is service requester, orchestrating the behaviour of the three components into the system's behaviour. The Cylinder component, in turn, is represented as a composite service, having its own, internal orchestrator that translates external service requests into a sequence of service requests to its component sensors and actuators.

The operation of the system is specified by the sequence diagram in Fig. 4, and its function block implementation is presented in Fig. 5.

As one can see from the sequence diagram, the operation is initiated by the arrival of a workpiece to the position in front of the pusher. This is detected by the WPS sensor that sends a notification message to the MES service. This message can be seen as a reply to the service request by MES that was implicitly made at the initialisation (i.e. by the INIT message). After receiving the notification, the MES requests the service "trip" from the cylinder. The cylinder, in turn, requests the push valve to be opened at 100%. The valve replies with a confirmation reporting its status (100%), and so on. When the pusher reaches the rightmost position, the end position sensor sends a notification to the cylinder that can be seen either as an independent service request, or a reply to the INIT service request.

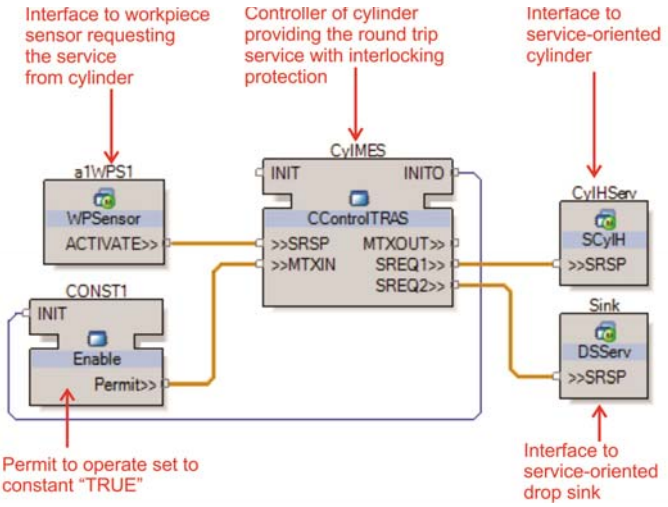

Fig. 5: Function block implementation of service-oriented intelligence in the linear pusher system.

The FB diagram in Fig. 5 is further specifying the abstract picture of Figure 3, although conceptually it bears the same information on the system's architecture. Each function block corresponds to one of the graph nodes in Fig. 3, but the semantics of connecting links is refined. The channels, connecting the CylMES function block with the FBs representing workpiece sensor, cylinder and sink, contain more detailed information on the communication protocol than arrows in the abstract diagram in Fig. 3. These are implemented by the IEC 61499 mechanism of adapter connections, described in the next section.

The function block CControlTRAS has been designed with special provisions for implementing interlocking by using the ring token mutual exclusion algorithm. For that purpose an adapter input MTXIN and output MTXOUT are reserved. Their use will be discussed in the next section.

B. Adapter based implementation of service connections

The one-line connections between circles representing services in Figure 3 are implemented by one-line adapter connections between function blocks in Fig. 5. The mechanism of adapter connections is explained in Fig. 6 on example of the MES -> Cylinder connection. The connection implies signal and data flow in both directions, although the services are requested by MES from Cylinder. The Cylinder can reply to the request immediately (e.g. Request is accepted), or later, (e.g. Request is completed).

In this paper, a single adapter type "service" is used to implement all services. Its interface includes two event inputs and two event outputs, associated with dedicated data inputs and outputs. This is sufficient to implement interaction between a service requester and a service provider, including service invocation, reply to the invocation, along with subsequent messages from the service provider with service execution results, etc.

Internal organization of controller

The orchestration of component services into functionality of a cylinder is implemented as a composite function block type CControlTRAS illustrated in Fig. 7. It has one input service interface (adapter socket) for communication with the workpiece sensor, and two interfaces for requesting services from the cylinder and the sink.

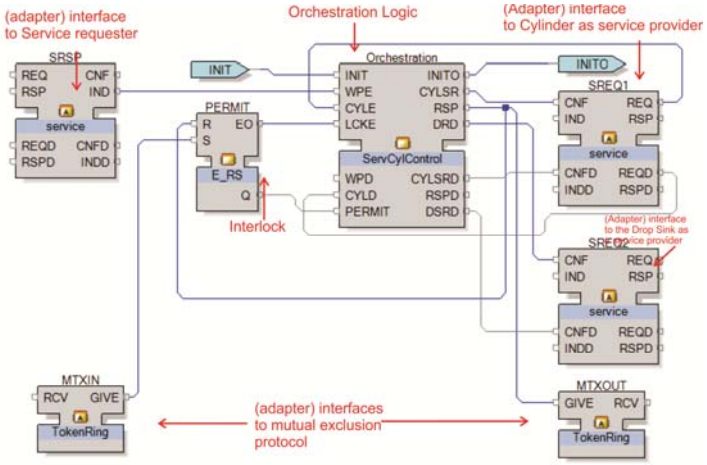

Fig. 7: Internal structure of the service orchestration function block (controller).

The logic of service orchestration is defined in the function block type ServCylControl as a state machine, presented in Fig. 8.

IV. Composition and Orchestration

A. Interlocking

The control logic of a single cylinder, presented in the previous section, was capable of automatically resolving conflicts with other intelligent mechatronic devices without

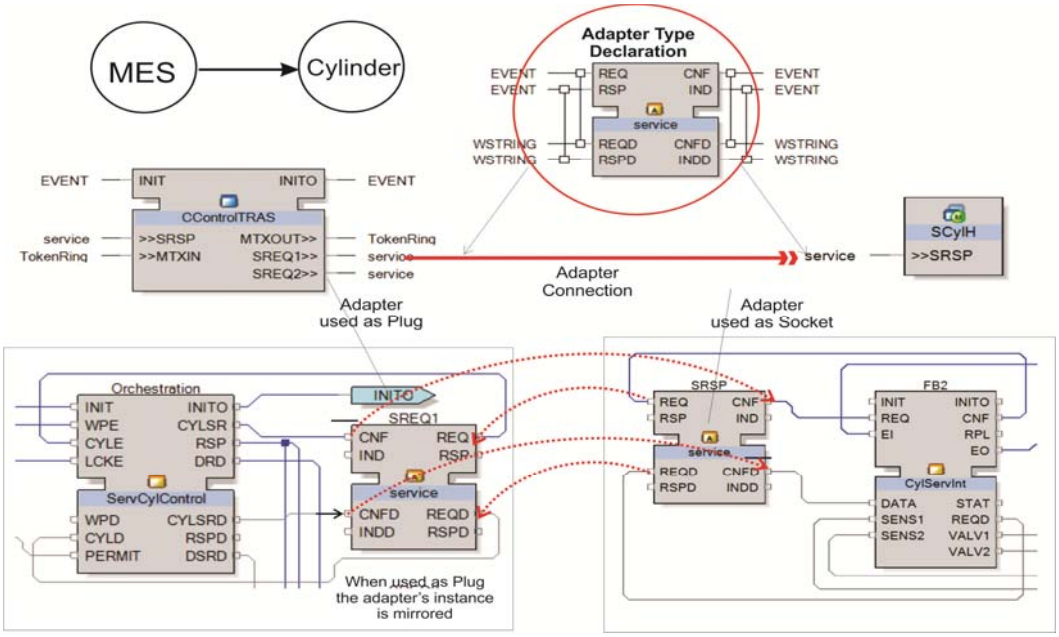

Fig. 6: Implementation of the abstract service request link by means of service adapters.

additional coordination logic.

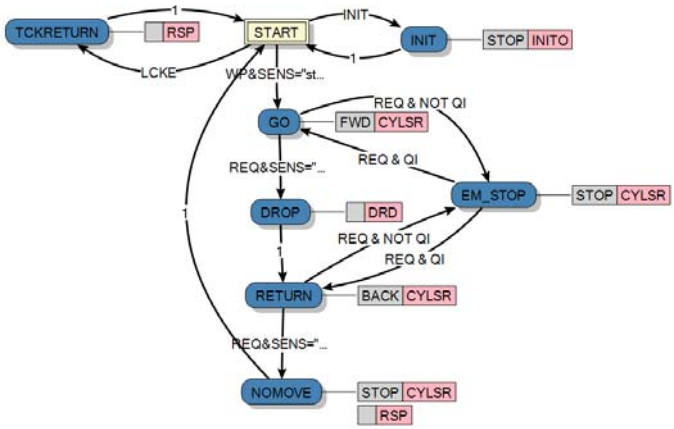

Fig. 8: State machine implementation of the orchestration logic.

For example, when the linear pushers are composed into a system as the one in Fig. 9, there is a potential clash area where the cylinders may collide, if they enter this area simultaneously (which may happen if work pieces appear simultaneously, or nearly simultaneously).

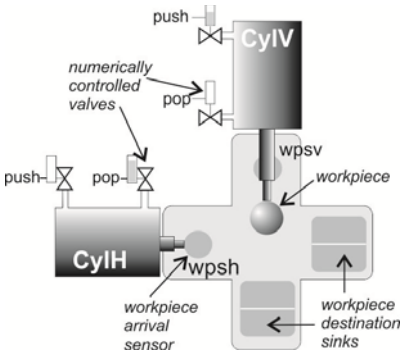

Fig. 9: System of two cylinders.

The corresponding function block application is shown in Fig. 10. As one can observe, this application consists of two applications for a single cylinder pusher case. The interlocking condition is implemented by connecting the MTXIN and MTXOUT of the FBs into a ring. This way the ring token protocol of mutually exclusive access of a cylinder to the operation area is implemented.

B. Scalability and orchestration of more complex behaviour

A more complicated class of problems related to the integration of SO- systems can be illustrated on example based on four linear pushers in Fig. 11. Here, again, it is assumed that each actor (pusher) has its autonomous behaviour (workpiece delivery to the destination) performed under constraints of not clashing with other counterparts. A simple service orchestration application is shown in Fig. 12. It will enable operation of the system and interlocking between the cylinders.

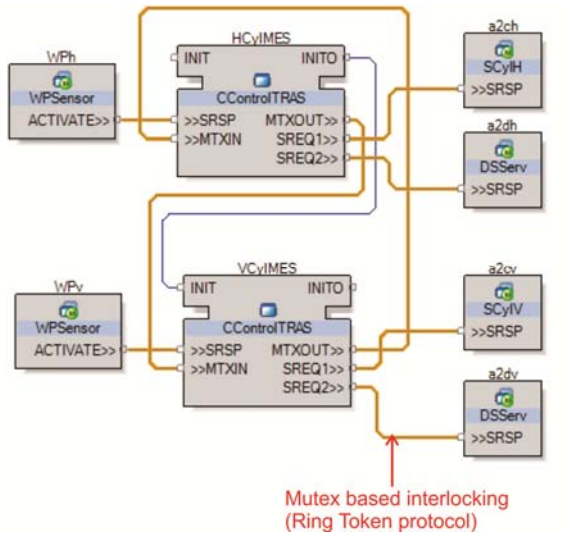

Fig. 10: FB implementation of the two cylinder system with interlocking implemented via the ring token protocol.

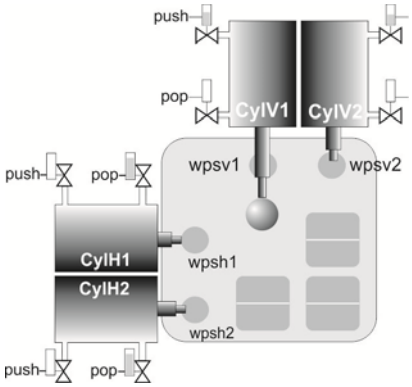

Fig. 11: System composed of four linear pushers.

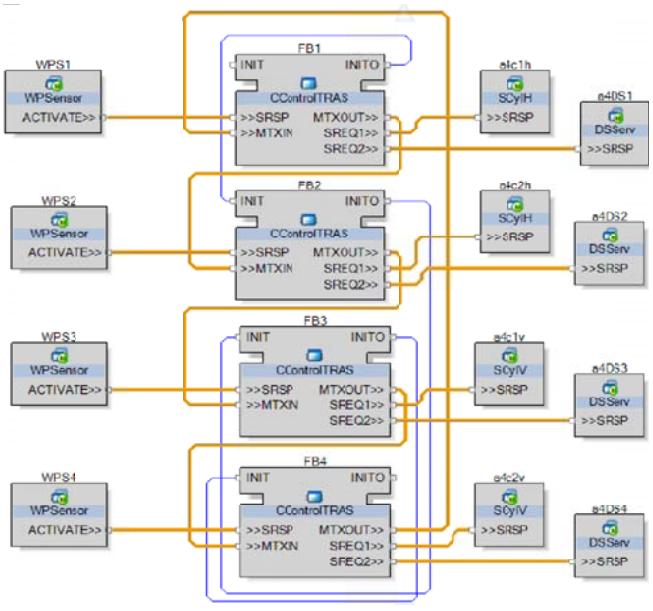

Fig. 12: Function block diagram for the four cylinder system with interlocking.

In automation systems, a more complicated logic of operation of basic actors is often desired. For example, in our example, this can be a more complex trajectory of workpiece motion, requiring collaborative effort of two pushers. This will be implemented as a composite service, resulted from orchestration of basic services of pushers. The implementation architecture can resemble the one in Fig. 13. Here the "Orchestrator" function block type receives notifications from the sensors and requests services from cylinders.

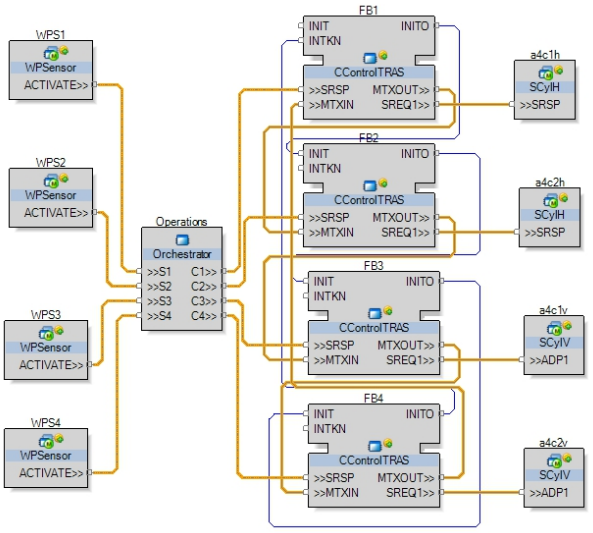

Fig. 13: Architecture implementing orchestrated services.

Capabilities of IEC 61499 tools allow for immediate simulation and validation. The simulation of the implemented case study is shown as illustrated in Fig. 14.

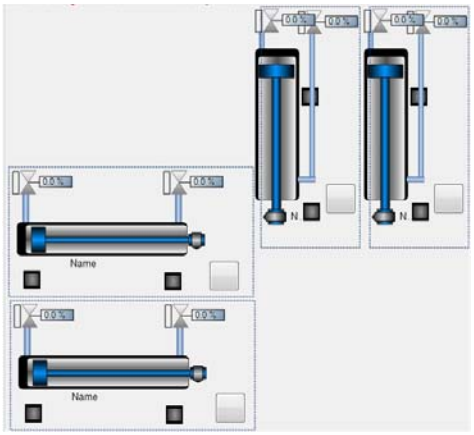

Fig. 14: Visualisation of the simulation of four pushers.

V. Discussion and Conclusions

In this case study, it was proposed to use IEC 61499 function blocks architecture as a convenient graphical modelling language for applying design level service-oriented architecture concept for automation systems. The application of SOA during software design allows quick prototyping of systems in an intuitive way. Moreover, the IEC 61499 tools give a lot of freedom in deploying a function block application to a network of distributed control devices.

The proposed design method with FB enjoys many benefits of SOA, main of which is flexibility. For example, substituting a pneumatic pusher with an electric pusher, or changing a type of valve from proportional to discrete would not require any change in the application's logic, provided that both types of cylinders support the same services. Adding and removing of one pusher would be convenient even during the operation of the rest of the system due to the loose coupling feature of the proposed SOA design paradigm.

SOA also simplifies adding more functionality on top of the existing, as it was illustrated in Fig. 13. Instead of showing only software components with interconnections, semantic information such as description of automation processes could be illustrated in front of users.

The conducted case study on application of IEC 61499 function block architecture for implementation of service-oriented architecture has fully proven the feasibility and benefits of this approach. It enables modelling distributed SO-systems composed of heterogeneous controllers, including SO PLC, such as S1000 [[9]], and supports the plug-and composition of intelligent mechatronic devices equipped with intelligent sensors and actuators, implementing the Internet of Things vision.

VI. References

[1] T. Erl, "Service-oriented architecture: concepts, technology and design", *Prentice Hall Professional Technical Reference*, 760 pages, 2005

[2] F. Jammes and H. Smit, "Service-Oriented Paradigms in Industrial Automation," *IEEE Transactions on Industrial Informatics*, vol. 1, pp. 62 - 70, 2005

[3] A. Lobov, F. Lopez, V. Herrera, J. Puttonen and J. Lastra, "Semantic Web Services framework for manufacturing industries", *IEEE International Conference on Robotics and Biomimetics (ROBIO 2008),* pp. 2104-2108, 2008.

[4] K. Thramboulidis, G. Koumoutsos and G. Doukas, "Towards a service-oriented IEC 61499 compliant engineering support environment", *IEEE Conference on Emerging Technologies and Factory Automation (ETFA'06),* pp. 758-765, 2006.

[5] T. Cucinotta, A. Mancina, G. Anastasi, G. Lipari, L. Mangeruca, R. Checcozzo, and F. Rusina, "A Real-Time Service-Oriented Architecture for Industrial Automation", *IEEE Transactions on Industrial Informatics*, vol.5, no.3, pp.267 - 277, 2009.

[6] S. V. Ragavana, I. K. Kusnantoa, and V. Ganapathyb, "Service Oriented Framework for Industrial Automation Systems", *Procedia Engineering*, Volume 41, 2012, Pages 716–723

[7] Mendes, J.; Leitão, P.; Restivo, F.; Colombo, A. W. (2009) - Service-oriented agents for collaborative industrial automation and production systems, *4th International Conference on Industrial Applications of Holonic and Multi-Agent Systems*, Linz, Austria. p. 1-12

[8] W. Dai, V. Vyatkin and J. Christensen, "The Application of Service-Oriented Architectures in Distributed Automation Systems", *IEEE Conference on Robotics and Automation (ICRA'14)*, 2014, accepted, Hong Kong

[9] Service oriented controller S1000, Inicotech, online: http://www.inicotech.com/s1000_overview.html