2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

The Application of Service-Oriented Architectures in Distributed Automation Systems

Wenbin (William) Dai, Luleå University of Technology, Sweden, IEEE Member, w.dai@ieee.org
Valeriy Vyatkin, Luleå University of Technology, Sweden and Aalto University, Helsinki, Finland, Senior IEEE Member, vyatkin@ieee.org
James H. Christensen, Holobloc Inc, james.h.christensen@gmail.com

***Abstract* – The complexity and scale of automation industry applications have increased substantially in recent years, posing new challenges to fulfill growing requirements for reusability, interoperability, flexibility and reconfigurability. The adoption of service-oriented architectures (SOAs) could be a feasible solution to meeting these challenges. Based on a comparison between the conceptual frameworks of SOAs versus modern standards for distributed automation, a methodology is proposed for the application of SOAs in the distributed automation domain. The implementation of the Service-Oriented Architecture based PLC execution environment is demonstrated on a case study.**

***Index Terms* — Service-Oriented Architecture, Industrial Automation, IEC 61499, function blocks, IEC 61131-3, PLC, Reusability, Interoperability, flexibility, SOAP, WSDL.**

I. INTRODUCTION

The hardware and software paradigm of programmable logic controllers (PLC) dominates the industrial automation landscape. PLCs are widely used in almost every branch of industry: material handling systems, manufacturing systems, process control, building automation, traffic light control, etc. The software of PLCs is most commonly developed under the guidance of the IEC 61131-3 standard [1] that defines a set of programming languages. However, PLC vendors implement this standard based on their own interpretation and taking into account compatibility with legacy systems. This creates interoperability issues between various PLC platforms. An additional drawback is that the IEC 61131-3 standard is not oriented toward distributed automation systems; the highest level of its software model, the "configuration," is limited to a single PLC device [1] [4]. For this reason, substantial design overhead is incurred in the design of a distributed system using PLCs under the IEC 61131-3 paradigm [2].

The IEC 61499 standard [3], whose 2nd edition has been released recently, presents some approaches to the solution of the existing issues of the IEC 61131-3 standard in designing distributed automation systems [4]. The IEC 61499 standard allows developers to design an abstract system without identifying hardware platforms, thus improving interoperability. Assignment of function blocks to devices can be deferred until deployment. The IEC 61499 standard uses an XML file format to avoid portability issues. Additionally, management commands defined in the standard enable reconfigurability at runtime [5].

In existing implementations of the IEC 61499 standard, interoperability, portability and reconfigurability are partially achieved. There are still some compatibility issues between different IEC 61499 vendors. For example, service interface function blocks (SIFB) developed for one platform in most cases is not portable to another platform. Several features (for instance, create and delete function block instances during execution) are fulfilled for reconfiguration. Approaches to more intelligent actions, such as automatic deployment [11] and automatic fault recovery [5] have been proposed and demonstrated, but are not yet standardized, while other advanced capabilities such as automatic load sharing between controllers are yet to be implemented.

In the general computing domain, the concept of Service-Oriented Architecture (SOA) is used to design distributed networked systems. SOA provides a software design pattern in which software components are only connected via messaging. This loose coupling feature ensures interoperability between various platforms and languages as well as flexibility. Furthermore, each service must register a service contract in the service repository in order to be discoverable from other services. This increases reusability of programs.

This research aims to apply the SOA in the automation domain in order to enable more intelligent control and reduce development and maintenance costs. The rest of the paper is organized as follows: In Section II, relevant research on applying the SOA concept in the industrial automation domain is reviewed. In Section III, SOA principles are compared to both the IEC 61131-3 standard and the IEC 61499. Section IV then presents and discusses examples of the potential applications of SOA using IEC 61499 function blocks. In Section V, the implementation of the SOA based IEC 61499 execution environment is demonstrated. Finally, Section VI presents an evaluation of the feasibility of this approach and lists possible future work.

II. RELATED WORKS

Several researchers discussed recently the ideas of applying SOA in the industrial automation domain.

The feasibility of applying Semantic Web technologies and service-oriented architecture in automation industry has been discussed by Jammes and Smit [6]. Their work illustrates the usefulness of applying SOA and Web services standards to meet the challenges of interoperability, scalability, plug-and-play connectivity and seamless integration of automation systems.

Lastra [7] discussed the current trends in industrial

automation, especially in factory automation, concluding that immediate benefits could be provided to factory automation by introducing XML, Web Services and Semantic Web Technologies.

Delamer et al. [8] discussed software abstractions that could facilitate the implementation of loosely-coupled automation systems, proposing a service-oriented architecture and event-based middleware to easily add, remove or replace software components. Web services technologies and ontology are used for the middleware implementation. Lobov et al. [9] investigated applying Semantic Web services based on ontologies into the manufacturing industries. The solution separated responsibilities between various machines from different vendors. Ontology and reasoning are used to define responsibilities for each device in the system. However, neither of these works takes IEC 61131-3 or IEC 61499 into consideration.

Thramboulidis et al. [10] proposed a SOA based engineering support system for IEC 61499 in industrial automation systems, which can be easily extended. Design features are defined as services and published in the public domain. Those features could be reused by other developers. This proposal is only intended for application at the design level.

Overall, one can conclude that there is no existing work on applying the SOA concept to the low level control of PLCs. This paper aims at bridging this gap in order to achieve flexibility and interoperability between various automation systems.

III. SERVICE-ORIENTED ARCHITECTURE PRINCIPLES IN THE AUTOMATION DOMAIN

There are several well-known principles of service-oriented architecture defined in the software engineering domain [12]. In this section, those common principles will be applied to both IEC 61131-3 PLCs and IEC 61499 function blocks. In Fig. 1, all common principles and their relationships of SOA are illustrated. Each SOA principle, and its relationship to the IEC 61131-3 and 61499 standards, is explained in detail below.

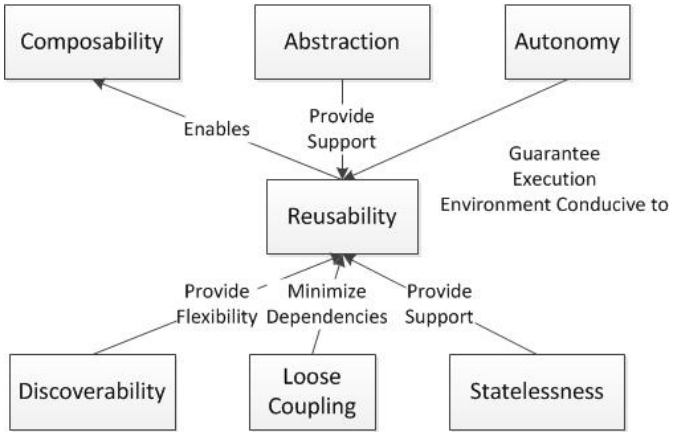


Fig. 1 SOA Principles Diagram.

First of all, *reusability*, the fundamental principle of SOA, is achieved by encapsulating logic into a service, which could be used by more than one service requestor. In the IEC 61131-3 standard, logic could be placed in one of the program organization unit (POU) types: functions or function blocks. Both functions and function blocks are designed for the purpose of reuse; however, the top-level IEC 61131-3 POU, the "program", is not reusable except via the error-prone procedure of copy and paste.In the IEC 61499 standard, all logic must be encapsulated in a function block type (basic, composite or service interface). IEC 61499 function blocks are naturally designed as reusable components, as are all architectural elements up to the device level. Overall, there is no doubt that from a reusability point of view, SOA could be applied to existing IEC 61131-3 and IEC 61499 architecture.

Secondly, services are formally defined by *contracts*. A service contract consists of all primary definitions: general information of service, such as name, type, owner, version, responsibility; functional description including requirements, service operations, how to invoke the service (message components); and additional information like quality of service, security, semantics, description of the service, etc. In the PLC implementation, there is no means for such formal definitions except for comments and descriptions that may be stored inside the PLC code or saved as a separate document using any format. The IEC 61499 standard provides a better solution via the mechanism of service sequences. As illustrated in Fig 2 in an example of the generic IEC 61499 REQUESTER function block, the behaviour of a function block can be described via the graphical language of service primitives. This is complementary to service implementation that can be done via a variety of models and languages.

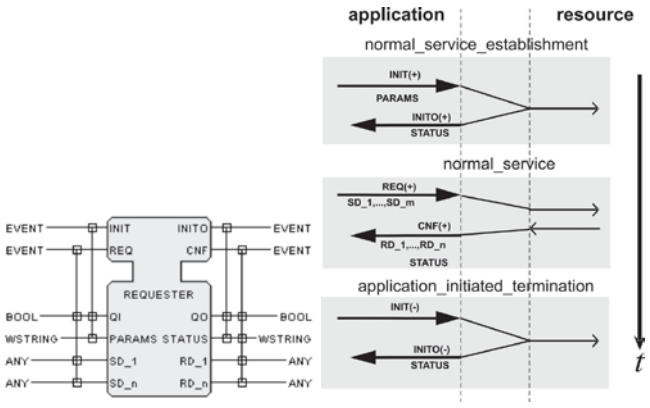


Fig. 2 Example of service primitives in IEC 61499 REQUESTER Service Interface Function Block.

Loose Coupling is another key principle of SOA. Services are independent from other services, which can only communicate with each other via messaging. Loosely coupled services ensure future expansion from various vendors with different technologies. The existing PLC POUs are tightly coupled. Variable values are passed directly into and out from functions and function blocks. Since variables could be stored in global memory of PLCs, whenever such a variable is processed by a POU, a new value must be

returned to the PLC global memory. On the other hand, IEC 61499 function blocks can communicate only via event and data connections that can be seen as a model of message passing. All variables must be kept inside a function block instance, implying the absence of global memory. Therefore the IEC 61499 function blocks can be considered as loosely coupled.

Loosely-coupled services enable another character of SOA – *Abstraction*. Services hide their (potentially complex) logic from the external environment, being a kind of black box. A system in SOA can be represented by services and communications between them. The action of an IEC 61499 function block can be invoked only by events, which makes it perfect for representing invocation of services by other services. Abstraction hides complexity of the underlying logic from users and developers. IEC 61131-3 PLCs provide some degree of abstraction by hiding logic inside functions and function blocks. Similar structures are provided in IEC 61499. At the application level, there only exist function block instances and connections between those instances.

Next, services are *Composable* with other services. The main purpose of this feature is to ensure services could be utilized effectively whenever required by other services, that is a form of reusability. In terms of composability, both IEC 61131-3 PLCs and IEC 61499 FBs are perfectly in-line with SOA. A new POU could be built by encapsulating the existing ones.

However, a certain range of logic can only be accessed by a service during execution time; this is called *Autonomy*. There are two types of autonomy: the service has total control over the underlying logic or services have clear boundaries but may share some underlying logic. In PLCs, functions can be shared between POUs. In order to use an IEC 61131-3 function block type, a new instance must be initialized whenever is required. This is also the case for IEC 61499 function block types. Function blocks from both standards have full control of their underlying logic.

Statelessness requires that only a minimum amount of state information shall be kept in the service. The service is stateless when there is no message present. The main reason is to ensure reusability of services and scalability for the application. PLC POUs are stateless if all status data are kept in the global memory. When a PLC POU is invoked, variable values are passed into this POU. New values will be updated back to the global memory and POU itself remains stateless. The same design approach is certainly achievable in IEC 61499. In a IEC 61499 FBs a service can be associated with an input event and an algorithm invoked upon the event. Inside the algorithm, the corresponding design pattern (supported by tools) can prohibit certain uses of internal and output variables that would violate statelessness of the service.

The last principle is *Discoverability*. Services shall have abilities to discover other services in order to avoid creating duplicated services or logic on the service level. Also, services are discoverable via service repository on the SOA level. There is no such feature available in the IEC 61131-3 standard. All functionalities are predefined during the design stage. The earlier discussed mechanism of service sequences in IEC 61499 creates a great pre-requisite for implementing service discovery mechanism. It can be accompanied by open device management services of IEC 61499 that allow querying devices with open XML based protocol in order to discover the list of function block instances located in the device and their descriptors. However, some means of retrieving the service sequences needs to be provided. This could be done either by extending the management services, or by defining a separate service that can access a repository containing the IEC 61499 XML definitions of function block types, which include their service sequences.

The support of major SOA principles in PLCs and IEC 61499 is given in the table 1 below.

TABLE I.
SOA Principle Support in IEC 61131-3 and IEC 61499

SOA Principles	IEC 61131-3	IEC 61499
Reusability	Yes	Yes
Service Contract	No	Partially
Loose Coupling	No	Partially
Abstraction	Partially	Yes
Composability	Partially	Partially
Autonomy	Partially	Partially
Statelessness	Possible	Possible
Discoverability	No	Partially

Reusability is supported in IEC 61131-3 PLCs. There is certain support of abstraction, composability, autonomy and statelessness in PLCs. But formal definition (service contract), loose coupling and discoverability are missing from the current standard. There are more similar principles between the IEC 61499 standard and the service-oriented architecture: all SOA principles are fully or partially supported by this architecture. Overall, the IEC 61499 standard is more feasible for adoption of the SOA.

IV. Mapping between SOA and the IEC 61499 Standard

In the previous section, the feasibility of applying SOA principles to IEC 61131-3/IEC 61499 was investigated. As stated in the introduction, the motivation of this work is to improve flexibility, and interoperability. In order to achieve this goal, the SOA principles will be applied to the IEC 61499 standard.

A standard SOA design pattern for the IT industry is shown in Fig. 3 [12]. The top layer is defined as a highly abstracted diagram, which is used to describe business

processes (for example, UML, flow chart, etc.). The middle layer is the service interface layer. The service interface layer provides a lower abstraction view of how processes defined in the process layer could be mapped to services. The bottom layer is the application layer where services are actually deployed to multiple applications.

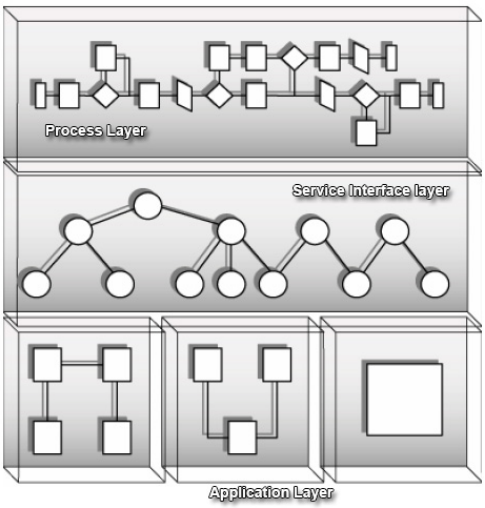


Fig. 3 SOA Layer Definition.

The flow-chart describing the functionality of the "Process layer" refers to features defined in functional design specifications for a particular automation system or machine. The service interface layer refers to actual overall low level control design of control logics. The application layer reflects the mapping between function blocks (services) and actual IEC 61499 resources (hardware/devices). The IEC 61499 standard fully covers this feature and is well supported in all function block IDEs as FBDK [15], 4DIAC IDE [16], NxtOne [17].

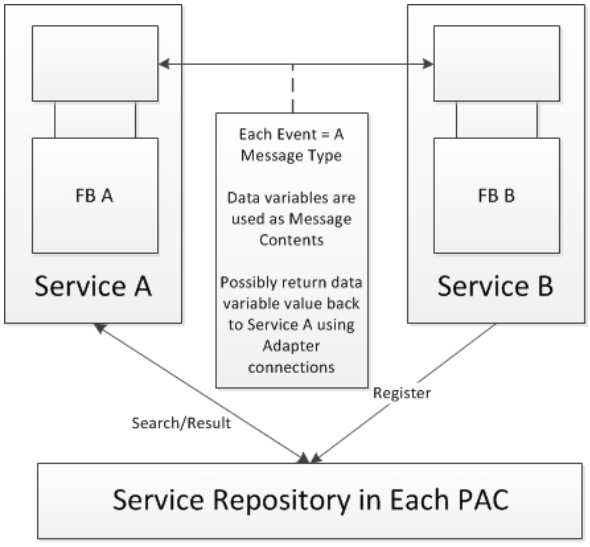


Fig. 4. SOA in IEC 61499 approach.

In the service interface layer, a function block is reflected as a service entity. Each event connection is considered as a message type. All data variables associated with an event are used as input parameters for the message type. The IEC 61499 function block network could be imagined as a service sequence diagram. Indeed, IEC 61499 function block network in service sequence diagram format provides a more compact view of overall system design. Between two function blocks (as services), only event connections (as message flows) should be displayed. All data variables are hidden from the function block interface; this data could be considered to comprise all the variables associated with the event by a standard WITH declaration in the FB type definition. A message path is shown to indicate the output event from the FB A to the input event from the FB B.

Up to this point, all basic SOA principles are fulfilled in the IEC 61499 standard. However, there is no mechanism implemented in the IEC 61499 standard in order to discover other distributed nodes and understand their functionalities. Discoverability could improve intelligent control by providing improved support for downtime-less redundancy, dynamic reconfiguration, automatic load share between distributed nodes, etc. In order to discover a service from the network, a service repository is compulsory. A service repository deployed in each device on the application layer ensures all services (function blocks) are registered as soon as been created. When a function block triggers an event request (message) to another function block, it will search the destination address of the target function block on the service repository. This SOA structure ensures discoverability and interoperability between various implementation of IEC 61499 platforms.

The one main principle of the SOA is that services could compose other services to create new services – called service orchestration. This is achievable in the IEC 61499 standard by using the composite function block type. The composite function block encapsulates basic function blocks, service interface function blocks or even composite function blocks, which form a function block network. The IEC 61499 standard allows function block type definitions to be downloaded to a device during execution. To create a new function block definition during runtime, IEC 61499 management commands could be sent from IEC 61499 IDEs, but, in principle, from other function blocks as well. On the other hand, services could be decomposed when they are no longer needed by processes. Fortunately, function block instances and type definitions can be deleted also via management commands.

V. IMPLEMENTATION AND CASE STUDY

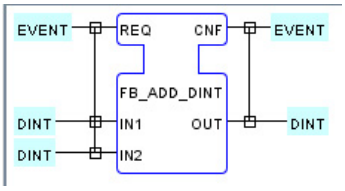
In order to prove the concepts described in the previous section, a SOA-based IEC 61499 execution environment (Compiler + Runtime) is needed. In this section, some preliminary works on the SOA-based IEC 61499 execution environment is illustrated.

The first step is to compile each function block into a collection of services. The proposed mechanism is as follows: function blocks are compiled to dynamic link libraries (DLL) which could be created as services using a standard web services platform such as Microsoft .Net Framework and windows communication foundations (WCF). During the compilation, the interface part of each

function block definition is converted to a Web Service Definition Language (WSDL) [19] file, which can be published at service repositories. In the WSDL 2.0, several properties are defined to describe a service including interfaces, bindings, services, element declarations and type definitions. Type definitions contain data type definitions and are compulsory in the WSDL. These would be used in the declaration of data input and output variables are declared in this part for IEC 61499 function blocks. The WSDL interface part is used to define messages that are available in this service. These definitions would be used to declare event inputs and outputs of function blocks including the WITH declarations of the associated input and output variables, respectively.

Protocol and data format are defined in the binding part of WSDL. In the IEC 61499 standard, protocols used between function blocks are not specified. It provides flexibility to developers to use any protocol by inserting extra communication function blocks in between. For each event, the type of message channel could be specified which represents a protocol.

Finally, the service section of WSDL defines end points of services and which interface defined in this service should be applied. The end points indicate where services are placed. The addressing information, such as names, IP addresses and URLs for each function block instances is stored in the endpoints so other services can access via these addresses.




```
<?xml version="1.0"?>
<wsdl:description xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:wsoap= "http://www.w3.org/ns/wsdl/soap" >
  <wsdl:documentation>
    FB_Example.wsdl
  </wsdl:documentation>

  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="IN1" type="xsd:integer"/>
      <xsd:element name="IN2" type="xsd:integer"/>
      <xsd:element name="OUT" type="xsd:integer"/>
    </xsd:schema>
  </wsdl:types>

  <wsdl:interface name="FB_ADD_DINT_Interface" >
    <wsdl:operation name="ADD_DINT"
      pattern="http://www.w3.org/ns/wsdl/in-out"
      style="http://www.w3.org/ns/wsdl/style/iri">
      <wsdl:input messageLabel="In" element="IN1" />
      <wsdl:input messageLabel="In" element="IN2" />
      <wsdl:output messageLabel="Out" element="OUT1" />
    </wsdl:operation>
  </wsdl:interface>

  <wsdl:binding name="FB_ADD_DINT_Binding"
    interface="FB_ADD_DINT_Interface"
    type="http://www.w3.org/ns/wsdl/soap"
    wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
    <wsdl:operation ref="ADD_DINT"
      wsoap:mep="http://www.w3.org/2003/05/soap/mep/soap-response"/>
  </wsdl:binding>

  <wsdl:service name="FB_ADD_DINT_Service"
    interface="FB_ADD_DINT_Interface">
    <wsdl:endpoint name="FB_ADD_DINT_Endpoint"
      binding="FB_ADD_DINT_Binding"
address="http://localhost/Service/FB_ADD_DINT/"/>
  </wsdl:service>

</wsdl:description>
```

Fig. 5 WSDL Example for IEC 61499 Message between FBs.

An example of a simple function block which returns the sum of two values read from sensors and its service definition is illustrated in Fig. 5. Data input and output variables are defined in the type section. Then those variables are used in the interface definition. Finally, the binding and the service are declared.

The next important part is communication between services. Each event connection inside function block designs is considered as a SOAP message type [18]. Data connections associated with this event connection are placed in the SOAP message content. SOAP messages allow a two-way communication: request and response. In the same manner as functions in programming languages, a service provider can provide response messages back to the service requestor after execution is completed. However, IEC 61499 event and data flows are unidirectional, which means that a service call with reply is not supported. In order to implement the response message, a new event connection is created to provide a response channel to the service requester. Data connections are attached to this event connection when return values are required by the service requester. However, this will cause massive event and data connections in the function block design. *Adapter connections* as defined in the IEC 61499 standard can be used directly to represent such bidirectional communication, and the associated *service sequence diagrams* of the corresponding *adapter types* can be used directly to represent each of the service messaging interactions. Hence, SOA communications can be fully represented using existing IEC 61499 elements.

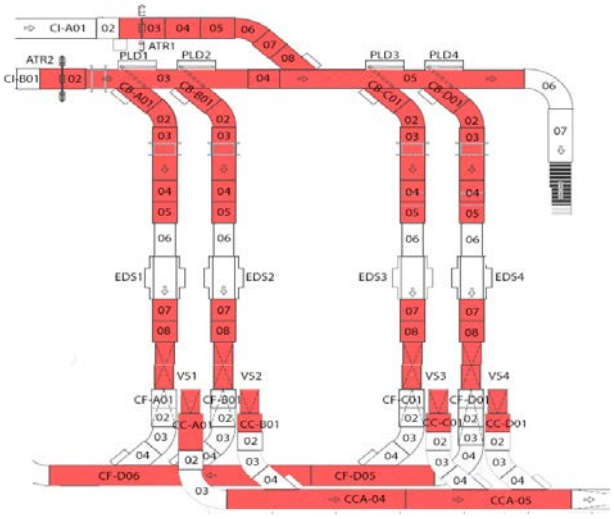


Fig. 6 BHS Screening Subsystem Layout.

A screening subsystem of an airport baggage handling system (BHS) is used as the case study example as shown in Fig. 6. Baggage units are inducted from two in-feed conveyor lines into the screening BHS subsystem. The subsystem includes four screening lines. Bags are equally distributed to four lines by plough diverters. An exposure detection system (EDS) X-Ray machine is the core part of each screening line. Bags cleared from the security check by EDS machines will be diverted to downstream sortation

subsystems. Suspect bags will be sent to further investigation.

The FB implementation of the BHS subsystem is given in Fig. 7. All function blocks are compiled into services and registered on the corresponding service repository in each PLC. PLC1 and PLC2 are running half of the screening lines individually. Another controller - PLC3 - is responsible for feed-in lines.

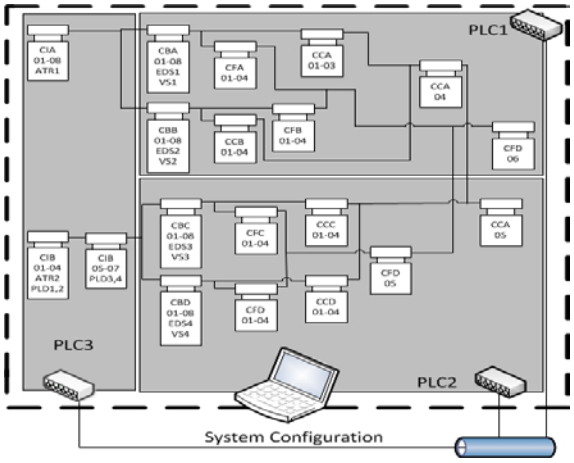


Fig. 7 BHS Screening Subsystem FB implementation.

The preliminary results demonstrate that by combining SOA with IEC 61499, the flexibility and interoperability of IEC 61499-compliant PLCs could be substantially improved.

VI. Conclusions And Future Work

With the goal of improving reconfigurability, flexibility and interoperability for distributed automation systems, the feasibility of applying the service-oriented architecture into the industrial automation domain has been investigated. An analysis of mapping SOA principles in both IEC 61131-3 PLCs and IEC 61499 function blocks was performed. The result indicates the IEC 61499 standard is more suitable than IEC 61131-3 for implementation of service-oriented architectures. A mapping of elements of the SOA into an IEC 61499 standard-compliant application has been presented.

Future work is planned to investigate formal rules for mapping SOA into the IEC 61499. Features such as plug and play and dynamic reconfiguration using SOA for IEC 61499 function blocks also need to be investigated. The compiler and the runtime based on the SOA should be further enhanced in order to test advanced features such as dynamic reconfiguration and downtime-less redundancy.

VII. References

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

[2] W. Dai, V. Vyatkin and J. Christensen, "Essential Elements for Programming of Distributed Automation and Control Systems", *IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*, Cagliari, September, 2013.

[3] IEC 61499, Function Blocks, *International Standard*, International Electrotechnical Commission, Geneva, Switzerland, Second Edition, 2012

[4] V. Vyatkin, "IEC 61499 as Enabler of Distributed and Intelligent Automation: State of the Art Review", *IEEE Transactions on Industrial Informatics*, 7(4), 2011, pp. 768-781

[5] T. Strasser and R. Foschauer, "Autonomous Application Recovery in Distributed Intelligent Automation and Control Systems", *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, Vol. 42, Issue 6, Page 1054 – 1070, 2012.

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

[7] J. Lastra and I. Delamer, "Automation 2.0: current trends in Factory Automation", *IEEE International Conference on Industrial Informatics*, Page 1321 - 1323, 2008.

[8] I. Delamer and J. Lastra, "Loosely-coupled Automation Systems using Device-level SOA", *IEEE International Conference on Industrial Informatics*, Vol. 2, Page 743 - 748, 2007.

[9] 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*, page 2104 – 2108, 2008.

[10] K. Thramboulidis, G. Koumoutsos and G. Doukas, "Towards a Service-oriented IEC 61499 compliant Engineering Support Environment", *IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*, Page 758 - 765, 2006.

[11] A. Zoitl, W. Lepuschitz, M. Merdan, M. Vallee, "A real-time reconfiguration infrastructure for distributed embedded control systems", *IEEE International Conference on Emerging Technology and Factory Automation (ETFA)*, Page 1 - 8, 2010.

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

[13] Unified Modeling Language [Online], available: http://www.uml.org

[14] Joiner Associates Staff, "Flowcharts: Plain & Simple: Learning & Application Guide", ISBN 978-1884731037, 118 Pages, 1995.

[15] FBDK – Function Block Development Kit [Online], available from http://www.holobloc.com/

[16] 4DIAC, An open source IEC 61499 IDE and runtime [Online, 2011], available from http://www.fordiac.org

[17] nxtControl GmbH, nxtControl - Next generation software for next generation customers [Online, 2009, June]. Available: http://www.nxtcontrol.com/

[18] SOAP – Simple Object Access Protocol [Online 2000], Available: http://www.w3.org/TR/soap/

[19] WSDL – Web Services Description Language [Online 2001], Available: http://www.w3.org/TR/wsdl/