2014 IEEE International Conference on Systems, Man, and Cybernetics
October 5-8, 2014, San Diego, CA, USA

Service-Oriented Distributed Control Software Design for Process Automation Systems

[1]Wenbin (William) Dai *IEEE Member*, [2]Jukka Peltola, [1,2]Valeriy Vyatkin *IEEE Senior Member*[1,] Cheng Pang *IEEE Member*

[1]Department of Computer Science, Electrical and Space Engineering, Luleå University of Technology, Sweden
and [2]Department of Information and Computer Systems in Automation, Aalto University, Finland
w.dai@ieee.org, jukka.peltola@aalto.fi, vyatkin@ieee.org, cheng.pang@ltu.se

***Abstract* — Improving software development efficiency and reusability of existing programs is an important topic for industrial automation. Recently, modern software paradigms have been introduced into the automation domain such as object-oriented programming using the IEC 61131-3 standard and component-based design based on the IEC 61499 standard. In this paper, another software paradigm – service-oriented architecture is applied to the software design for automation programs. The proposed SOA pattern is based on a multi-layered structure with enhanced reusability and flexibility. A case study of water heating system is implemented to prove the concept of SOA paradigm.**

Keywords* — *Service-Oriented Architecture; Component-based design; Object-Oriented Programming; IEC 61131-3 PLCs; IEC 61499 Function Blocks.

I. INTRODUCTION

With increasing complexity of industrial automation systems, software design of automation programs has experienced two evolutions. In the old days, control logic was implemented in electrical circuits and automation systems consisted of combinations of relays, cam timers, drum sequencers and dedicated closed-loop controllers. The design and development of such logic is extremely complicated and time consuming. There is no way for engineers to identify correctness of control logic except by testing with real automation systems.

In early programmable logic controllers (PLC), programs were written in proprietary languages. In 1990s, ladder logic, which resembles a schematic diagram of physical relay logic, was invented for general-purpose automation programs. In 1993, the first edition of international standard for PLC – IEC 61131-3 was published and recently revised third edition was released in 2013 [1]. Four programming languages ladder logic (LD), structure text (ST), instruction list (IL) and function block diagram (FBD) plus a state machine based language Sequential Function Chart (SFC) is defined in the IEC 61131-3 standard. Programming organization units (POU) – functions and function blocks are also introduced by the IEC 61131-3 standard. As a result, control logic could be encapsulated in POUs and reused for future. The efficiency of PLC software design and development is improved significantly by using those languages and POUs. Object-oriented programming (OOP) is also introduced to PLCs in the latest edition of IEC 61131-3 [2].

Nowadays, physical size and functionalities of manufacturing plants are continuously increasing and computing power provided by PLCs cannot match the requirements. It is often not feasible to control such automation systems by a central PLC. Instead, a better performance could be achieved if the control logic was distributed across several PLCs collaborating with each other. However, the IEC 61131-3 standard's architecture does not fit to the distributed system architecture. To address the shortcomings of IEC 61131-3, another international standard, IEC 61499 [3] was published in 2005 and revised in 2012 in order to assist development of distributed automation systems.

The IEC 61499 standard paves the way to practical use of component-based software design paradigm (CBD) in industrial automation domain as exemplified in [4]. As defined in the IEC 61499 standard, all control logic must be encapsulated in function blocks. A function block is a software component, which could be reused by mapping its interface. The paradigm shift is to build automation programs from a standard library rather than start from scratch every time. Automation programs can be generated by configuring required software components [5]. However, managing a large software component library is also a challenge. Besides, programs must be regenerated if any modification to the interface of a software component is required.

In order to improve flexibility and reusability of distributed automation programs, service-oriented architecture (SOA) is considered as the enabler for future collaborative cloud-based industrial automation [6]. Loosely coupled software units in the SOA ensure high flexibility and interoperability. More importantly, in SOA the description of the entire automation process could be integrated within control logics. The process description provides better understanding of system overview and improves designer's productivity.

This paper aims to provide a guideline on how to apply the SOA paradigm into the software design of distributed automation programs to improve flexibility and efficiency. The rest of the paper is organized as follows: In the section II, related works of industrial automation software design

978-1-4799-3840-7/14/$31.00 ©2014 IEEE

approaches and paradigms are reviewed. Then, comparison of various programming paradigms in automation domain, namely object-oriented programming, component-based design and service-oriented architecture, is provided in the section III. Then, a case study of water heating system is introduced in the section IV and implemented using the SOA paradigm. Finally, in the section V and VI, the discussion of how to apply SOA paradigm in the industrial automation is summarized and concluded.

II. Related Works

Software design paradigms are popular research topics in industrial automation domain. One famous software paradigm widely applied in the general information and communication technologies (ICT) is the object-oriented programming. Many research works have been dedicated to application of this concept in the automation domain. For example, Young et al. [7] proposed an object-oriented model for programming control systems based on unified modeling language (UML). Control systems are modeled using UML so that all processes are mapped to functions in object definitions. The UML model is then transformed into control code.

Werner describes the object-oriented extensions for the IEC 61131-3 standard in [8]. Object-oriented programming concepts (OOP) including methods, access specifiers (public, private and protected), inheritance and polymorphism (extends, interface and implements) are added in the latest edition of the IEC 61131-3 standard. The advantages of OOP such as better program structure, extensibility and reusability are illustrated in the paper.

There is no proven evidence that all characteristic OOP features mentioned above are equally valuable in industrial automation software development. Even in the general purpose ICT world, the usefulness vs. harm of inheritance and polymorphism is often disputed. The conceptual framework for object-oriented design of automation software was introduced in [9] with implementation based on IEC 61499 standard. IEC 61499 supports most important features of such design, such as encapsulation and reuse of automation code models: state machines as well as legacy PLC languages [10]. Combined with the abilities of being executed immediately and on distributed platforms these features represent an interesting set of opportunities for efficient automation software development.

A number of researchers have used the IEC 61499 models for more efficient design, that is often referred to as object-based or component-based, to differentiate it from the object-oriented programming. For example, Zoitl et al. [11] propose guidelines and patterns for building hierarchical automation systems using IEC 61499 as the modeling language. The structuring principles for hierarchical control architectures using the IEC 61499 language are illustrated in this paper. The component-based design is achieved by mapping layered elements into function blocks.

Black et al. [12] provides a case study of airport baggage handling system based on the component-based architecture and the IEC 61499 standard. For each device, e.g. a conveyor, there is a software component type that is taken from a library and instantiated. The component-based design is proven as scalable, reconfigurable and fault tolerant. Furthermore, intelligent self-configuration is also feasible by applying the component-based design.

A method of applying component-based design to intelligent control of batch processes using IEC 61499 and ISA S88 is proposed by Ivanova et al. [13]. The control logic is encapsulated to IEC 61499 function blocks and can be reused for different scenarios. The scheduling of logic components is implemented as an IEC 61499 FB (Function Block).

Cengic et al. [14] propose a framework for component based distributed control software also using the IEC 61499 standard. Automation components are introduced, which can be hierarchically embedded to create new components. The system hierarchy is presented by encapsulation of multi-layered automation components. The framework aims at platform independence, thanks to the IEC 61499 compliance.

Overall, one can observe a variety of design approaches in automation that are motivated by mechatronic modularity of physical systems and object-oriented modeling and programming. The object-oriented design of automation systems is implemented using UML in most existing works. The component-based design is natively suitable for IEC 61499.

III. Modern Software Paradigms in Automation: Object-oritend Programming vs Component-based Design vs Service-Oriented Architecture

Industrial automation programming methods have been continuously influenced by modern software engineering paradigms [15]. One of the popular topics is to introduce the Object-Oriented Programming concept into industrial automation. The key of the OOP is the object [16]. Objects are defined as software entities that encapsulate *data* and *methods* of data processing. In industrial automation, software objects are often related to physical objects, aka *devices,* such as conveyors, sensors and machinery [17]. Devices are controlled by control signals that can be generated by its associated software objects directly, or through a more complex invocation chain by other software objects. For example, in a baggage-handling system (BHS), a conveyor section could be waked up by a *request to start* signal sent from an upstream conveyor, when it starts moving. The use of OOP in automation domain aims at increase of code reusability. Automation software could be built from a library of device classes with specified relations between devices. The system could be extended easily by inserting more instances of objects. If any functionality of a machine needs to be modified or a new type of machine is brought in, device classes could be adjusted rapidly to meet new requirements by using inheritance and polymorphism of the OOP concept. One should note, however, that using polymorphism at execution time is associated with an extra computational cost, and is therefore often prohibited in

automation and embedded systems. Besides, it may hamper determinism of computations. Therefore, automation systems developers often prefer using only design time inheritance, deriving software components from the existing ones by extension of their data, methods and interfaces.

The approach that can be referred to as component-based (or object-based) design is not concerned with the techniques for derivation of components from other components (inheritance, polymorphism), but focusing on methodologies of components creation, composition and execution. The IEC 61499 standard perfectly matches the concept of component-based design with a function block modelling a component. The function blocks are connected to other function blocks via predefined interfaces (event and data connections). A component can also be composed of other components, which enables modelling of hierarchical structures that can help in reusing design of entire subsystems, such as screening subsystem that consists of X-Ray screening machines and automatic tag readers.

With rapid adoption of Ethernet-based communication at the factory floor, the service-oriented architecture (SOA) is becoming especially appealing as a framework for distributed automation software systems. The SOA-based architecture is a set of software components whose interface descriptions can be published and discovered [18]. Service providers and consumers are loosely coupled to ensure minimum dependencies between services. The interaction between services is defined in the service contract, which can be published to the service repository. Such interaction can be also represented using the artefacts of IEC 61499. To ensure loose coupling in IEC 61499 programs, function blocks should encapsulate most of the service functionalities, which only require minimum external data. For example, a small airport baggage handling system consists of the conveyor control service, the emergency stop service and the check-in procedure service. In this case, signals that need to be sent between services are minimized. From emergency stop services, only the status of emergency stop zone is required by conveyor control service. Check-in procedure service will only inform the conveyor control service when a new bag is inducted into the baggage handling system.

The characteristics of OOP, CBD and SOA are summarized in the TABLE 1 below. As seen from the table, OOP, CBD and SOA are all invented for improving reusability and programming efficiency by introducing reusable software units. All those programming paradigms provide encapsulation concepts to library elements. Components are easier to integrate compared to objects by introducing the idea of assembly: components can embed their meta-models as component networks. Objects rely on external meta-models such as UML diagrams.

TABLE I. COMPARISON BETWEEN OOP, CBD AND SOA

	OOP	CBD	SOA
Software Unit	Classes (Objects)	Software Component	Software Service
Features	Inheritance, Polymorphism	Encapsulation	Loose Coupling, Discoverability
Interface	Method call	Predefined Interface	Service Contract
Reusable Source	Object Library	Component Library	Service Repository
System Hierarchy	Nested Classes	Nested Components	Service Orchestration, Composition
Meta Model	Class Diagram (UML)	Network of Components	Business Process Execution Language, Service Sequence Diagram

SOA has the best flexibility overall due to the configurable interface defined by separated service contract between software units. In addition, SOA is the only approach, which could be integrated with enterprise service bus by using Business Process Execution Language (BPEL). The enterprise resource planning (ERP) software could impact services directly via service bus. In order to enhance flexibility and interoperability for distributed automation systems, adopting SOA concept on the software design level is a feasible approach.

IV. DESIGN OF HEAT PRODUCTION PROCESS CONTROL APPLICATION USING SOA PARAGIDM

The case study uses a heat production process (HPP) system [20] as shown in the piping and instrumentation diagram (P&ID) in Fig. 1. Firstly, cold water is supplied into the makeup water tank (B400). The water from the makeup tank will be fed to the preheater tank (B100) via control valve Y101 when the water level in the preheater tank is determined too low. There is a heater installed in the preheater tank, which will heat the water close to the boiling point. Hot water is pumped into the feed water tank (B200) by using a control valve Y102 and a pump M100. When the boiler (B300) is ready, water from the feed water tank will be pumped into the boiler via valve Y201 and pump M200. There is a pressure indicator and a temperature sensor installed in the boiler to avoid over heating or over pressure. When any emergency situation happens, the valve Y204 may be opened to lower the pressure and temperature in the boiler. In each tank, there is a set of high (Lx01 – Analog, Lx00 - Digital) and low (Lx02 - Analog) level indicators to measure level and detect abnormal conditions. Pressurized water in the boiler tank will be supplied to the customer via the supply valve Y305. Finally, water can be discharged by opening the valve Y105.

The water heating system will be developed according to the SOA paradigm using IEC 61499 function blocks. In the general computing domain, SOA is presented in a layered architecture of composite services, which align with business processes [19]. A similar concept can also be applied to the industrial automation domain. The layered view of SOA is illustrated in Fig. 2.

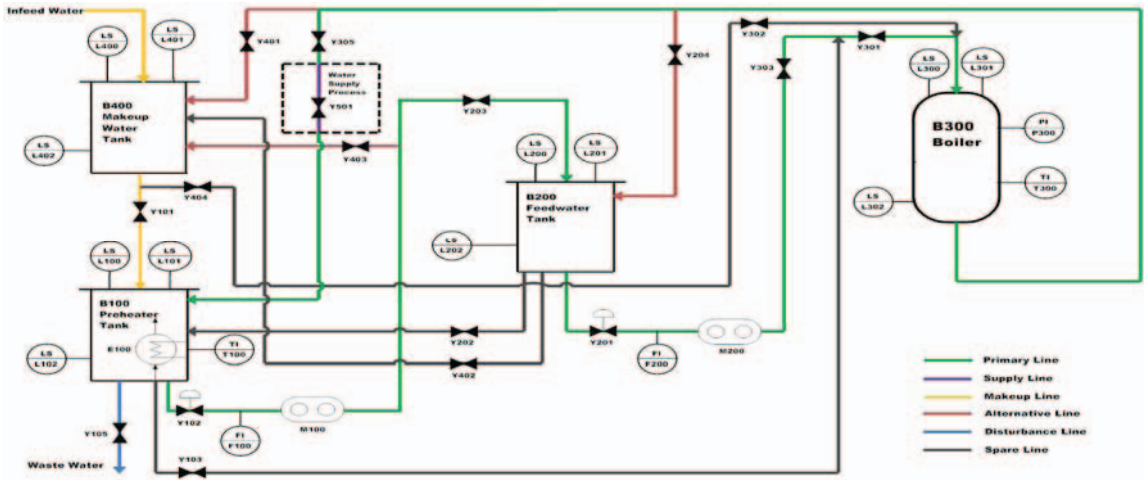

Fig. 1 Heat Production Process Piping and Instrumentation Diagram.

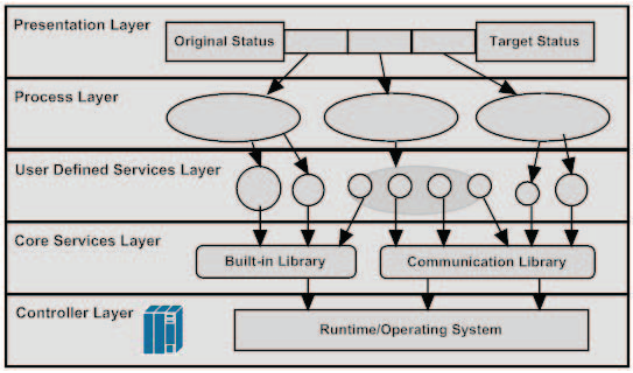

Fig. 2 Multi-Layered Abstract View of SOA in Industrial Automation.

The bottom layer is the controller layer, which consists of operating system and execution environment (runtime) for controllers. On top of the controller layer, services including built-in functions and communication handlers (for external message exchanging and fieldbus access in PLCs) are defined in the core services layer. The next layer is the user defined services layer. Functions or function blocks developed in this layer can act as service consumers invoking services from providers of the core services layer. The second top layer is the process layer, which contains information of individual physical processes controlled by automation functions such as filling water. Finally, the top layer is the presentation layer, which forms individual processes into a complete system by using sequence diagram and flow chart. The presentation layer also contains knowledge of the entire automation system.

Firstly, the design of core services layer function blocks is given in Fig. 3. The built-in libraries, communication to FB in external controllers and fieldbus are usually implemented in Service Interface Function Blocks (SIFB). SIFBs are designed for platform dependent functionalities in the IEC 61499 standard. In this case, built-in functions such as cycle event source FB (*E_CYCLE*), inverter FB (*NOT*), and fieldbus I/O access FB - Analog inputs and outputs (*FB_AI*, *FB_AO*) and Digital inputs and outputs (*FB_DI*, *FB_DO*) are implemented in the core services layer.

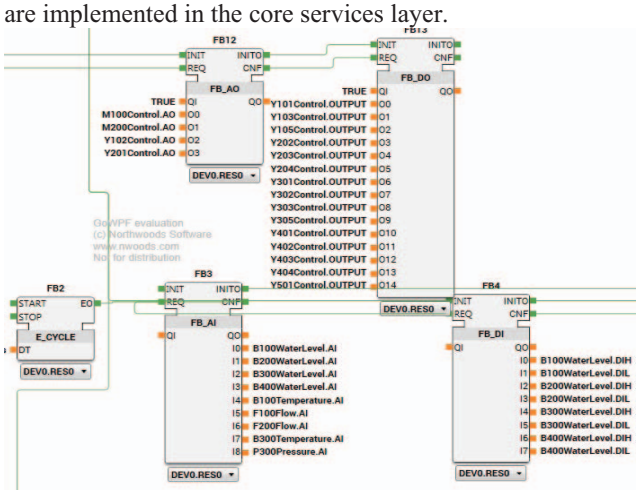

Fig. 3 Core Services Layer Implementation.

A part of user defined services layer function block implementation for the HPP system is shown in Fig. 4. Four types of services are defined in this layer: analog sensor measurement (*Service_AIMeasure*), analog actuator control (*Service_AOControl*), digital sensor measurement (*Service_DIMeasure*) and digital actuator control (*Service_DOControl*). Analog and digital measurement function blocks take readings from water level proximities (*Lx00, Lx01 and Lx02*), temperature sensors (*Tx00*), pressure sensors (*Px00*), and flow measurement sensors (*Fx00*) and generate alarms from each process variable. There are five alarm types generated for each sensor: HH, H, L, LL and F which refer to High Alarm, High Warning, Low Warning, Low Alarm and Fault respectively. Actuator control can be manually overridden by operators for testing purpose or fault recovery. Each actuator service function block takes two inputs: one from the upper level (automatic mode) and one from the Human-Machine Interface (HMI, manual mode).

The next layer above the user-defined services layer is the process layer. In the process layer, services are grouped by functionalities in the process, namely tank control (*FB_TankControl*), PID control (*FB_PIDControl*), heater control (*FB_HeaterControl*), pump control (*FB_PumpControl*) and valve control (*FB_ValveControl*) as given in Fig. 5. The tank control service collects alarm signals from sensor measurement services and generates tank status like is tank ready for feed in and out water, can tank be heated and is tank over pressured. The PID control service reads process value from the flow measurement service and recalculates control value for valve and pump control services. The heater, pump and valve control services check that the control value is in the valid range and produce output command to actuator control services.

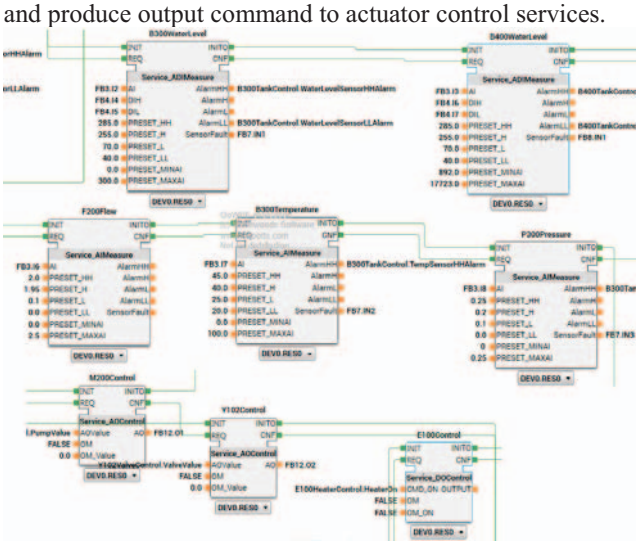

Fig. 4 User-Defined Services Layer Implementation.

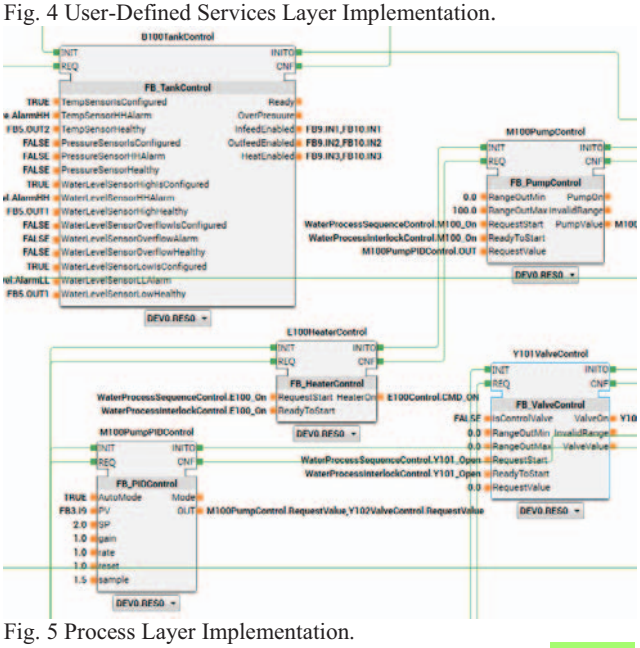

Fig. 5 Process Layer Implementation.

On the top layer, two services are defined: sequence control service and interlock service. The sequence control service controls all valves, heater and pumps based on the feedback collected from tank control services in the process layer. The sequence diagram of the HPP is defined using the execution control chart (ECC) in the sequence control FB. The HPP sequence is defined as six sequential steps including Filling, Wait, Start-up, Supply, Shutdown and Emptying.

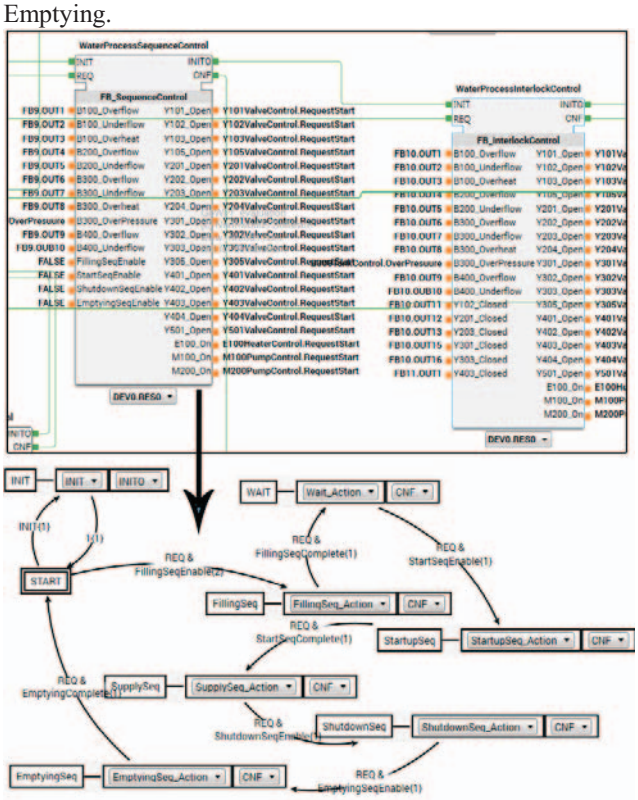

Fig. 6 Presentation Layer and Sequence Control Implementation.

Finally, the overview of the HPP system configuration in IEC 61499 is given in Fig. 7. An individual instance is created for each object due to limitation of the IEC 61499 standard: there is no shared memory so data cannot be stored globally; each data input can only be connected to one data output so a service cannot be shared by multiple instances.

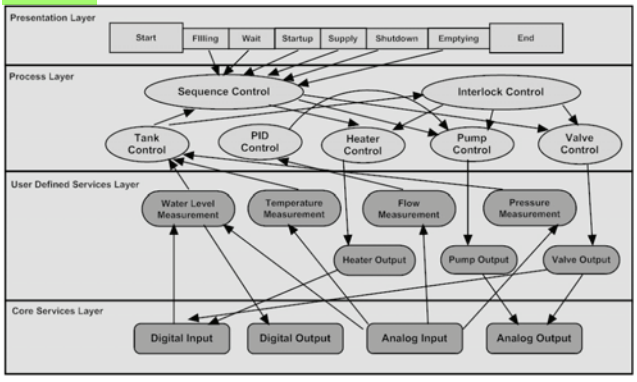

Fig. 7 Heat Production Process System Design Overview.

V. Discussion

The aim for adopting SOA paradigm in the design level of automation programs is to provide better reusability and

flexibility as well as improve software development efficiency and productivity. A guideline on how to design a system according to the SOA paradigm for software engineers is required. A multi-layered approach is proposed and demonstrated in the previous section. In this approach, upper layer services trigger lower layer services by sending request signals and collect response data from lower layer services. The presentation layer in the SOA approach provides a semantic view of system configurations. Instead of just illustrating connections between software units like OOP and CBD, SOA based approach describes the automation solution as a sequence, which specifies the actions the system will perform in a particular process state. The SOA paradigm fills the gap for integrate meta-models within the control logic for automation programs.

In the proposed SOA structure, the core service layer is designed for FBs provided by vendors. Those services are platform-dependent and usually implemented in SIFBs for IEC 61499. The user defined services layers are reusable functions developed in this system configuration which could be utilized in the future. Those services provide an abstract layer for core services and a bridge between processes and base functions. The process layer composes basic functionalities from user-defined services and core services to form simple sequences. Finally, the presentation layer is the "brain" of the control system. Service sequences indicating control flow are implemented in ECC or FB network to provide semantic information by arranging simple sequences in correct order.

Applying SOA paradigm can bring substantial benefits into the software design of industrial automation. The SOA based systems are easy to extend, modify and debug. There is no encapsulation of function blocks needed like components of components in CBD. In the SOA view, encapsulation is fulfilled by using service choreography, which defines the global view on the service interaction. Different from the hiding of complexity concept in component-based design, SOA provides a "flat" structure, which is convenient to identify the target function block for modification during debug process.

VI. Conclusion and Future Works

A multi-layered service-oriented architecture is proposed for software design in industrial automation. In order to find out a proper approach to introduce SOA on the design level, a process control case study of a heat production process is presented and a reference implementation of SOA-based IEC 61499 application is developed. The guideline provided on how to use SOA paradigm in automation application design is also applicable for Internet-of-Things (sensors, actuators and embedded controllers).

This work will be continued with formal definition of the SOA paradigm for industrial automation and formal design rules. Software agents could be introduced on the presentation layer of the SOA paradigm to enable self-management of software services. Finally, the integration between industrial automation and Internet-of-Things could be achieved by using the proposed SOA design approach.

References

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

[2] B. Werner, "Object-Oriented Extensions for IEC 61131-3", *IEEE Industrial Electronics Magazine,* Vol. 3, Issue 4, Page 36 – 39, 2009

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

[4] W. Dai and V. Vyatkin, "A Component-based design pattern for improving reusability of automation programs", *39th Annual Confrence of the IEEE Industrial Electronics Society (IECON),*Page 4328 – 4333, 2013

[5] W. Dai, V. Dubinin and V. Vyatkin, "Migration From PLC to IEC 61499 Using Semantic Web Technologies", *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, Vol 44, Issue 3, page 277 – 291, 2014

[6] S. Karnouskos, A. Colombo, T. Bangemann, K. Manninen, R. Camp, M. Tilly, P. Stluka, F. Jammes, J. Delsing and J. Eliasson, "A SOA-based architecture for empowering future collaborative cloud-based industrial automation", *38th Annual Conference on IEEE Industrial Eletronics Society,* Page 5766 – 5772, 2012

[7] K. W. Young, R. Piggin and P. Rachitrangsan, "An Object-Oriented Approach to an Agile Manufacturing Control System Design", *The International Journal of Advanced Manufacturing Technology,* Vol. 17, No. 11, pp 850 – 859, 2001

[8] B. Werner, "Object-oriented extensions for IEC 61131-3", *IEEE Industrial Electronics Magazine,* Vol. 3, No. 4, pp 36 – 39, 2009

[9] V. Vyatkin, J. Christensen, J. L. Lastra, "An Open, Object-Oriented Knowledge Economy for Intelligent Distributed Automation", *IEEE Transactions on Industrial Informatics*, *1*, *(1)*, pp. 4-17, 2005

[10] 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

[11] A. Zoitl and H. Prahofer, "Guidelines and Patterns for Building Hierarchical Automation Solutions in the IEC 61499 Modeling Language", *IEEE Transactions on Industrial Informatics,* Vol. 9, No. 4, pp 2387 – 2396, 2013

[12] G. Black and V. Vyatkin, "Intelligent Component-Based Automation of Baggage Handling Systems with IEC 61499", *IEEE Transactions on Automation Science and Engineering,* Vol 7, No. 2, pp 337 – 351, 2010

[13] D. Ivanova, G. Frey and I. Batchkova, "Intelligent component based batch control using IEC 61499 and ANSI/ISA S88", *4th International IEEE Conference on Intelligent Systems,* Vol. 1, pp 444 – 449, 2008

[14] G. Cengic, O. Ljungkrantz and K. Akesson, "A Framework for Component Based Distributed Control Software Development Using IEC 61499", *IEEE Conference on Emerging Technologies and Factory Automation,* pp 782 – 789, 2006

[15] V. Vyatkin, "Software Engineering in Factory and Energy Automation: State of the Art Review", *IEEE Transactions on Industrial Informatics*, 9(3), 2013, pp. 1234 - 1249

[16] B. Meyer, "Object-Oriented Software Construction", *Cambridge: Prentise Hall International Series in Computer Science*, p. 23, ISBN 0-13-629049-3, 1988

[17] W. Dai, V. Vyatkin, "Redesign Distributed PLC Control Systems Using IEC 61499 Function Blocks", *IEEE Transactions on Automation Science and Engineering*, Vol. 9, No. 2, pp 390 – 401, 2012

[18] Web Services Glossary [Online, 2004], available from http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

[19] A. Arsanjani, "Service-oriented modeling and architecture" [Online, 2004], available from https://www.ibm.com/developerworks/library/ws-soa-design1/

[20] J. Peltola, S. Sierla, P. Aarnio and K. Koskinen, "Industrial Evaluation of Functional Model-Based Testing for Process Control Applications Using CAEX", *IEEE Int. Conf. on Emerging Technologies in Factory Automation*, 2013, Cagliari, Italy.