Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

SOA-PLC – Dynamic Generation and Deployment of Web Services on a Programmable Logic Controller

Lisa Ollinger*. Alexander Abdo.
Detlef Zühlke*. Henning Heutger*****

* German Research Center for Artificial Intelligence (DFKI), 67663 Kaiserslautern, Germany
(e-mail: lisa.ollinger@dfki.de, detlef.zuehlke@dfki.de).
** SmartFactory^{KL}, 67663 Kaiserslautern, Germany
(e-mail: abdo@smartfactory.de)
*** Phoenix Contact Electronics GmbH, Bad Pyrmont, Germany
(e-mail: HHeutger@phoenixcontact.com)

Abstract: The static and inflexible design of control software with current Programmable Logic Controllers (PLCs) and its integration in other automation systems do not meet the demands of agile and dynamic manufacturing systems with their rising complexity. Service-oriented Architecture (SOA) is a promising approach to overcome this shortcoming. Thus, a concept of a SOA-PLC is presented in this paper. The suggested concept allows a fully automated generation and deployment of field device functions as services implemented with the Devices Profile for Web Services (DPWS). Moreover, the concept is demonstrated in a case study using a conventional Phoenix Contact PLC and the DPWS orchestration tool JGrafchart.

Keywords: Manufacturing control, Control engineering, Distributed control, Process models, Flexible manufacturing systems.

1. INTRODUCTION

Manufacturing companies can gain an important competitive advantage by adapting their production systems dynamically to new conditions, like changing or new product variants, process optimizations, or the current order situation. The enabler for this desired agility is a highly integrated and consistent information flow between the business process and technological process [Gerber et al. 2013]. Additionally, the control software should be easy to reuse and adapt in order to save efforts during reconfiguration tasks [Westkämper and Zahn 2009].

Production automation systems can be typically separated in two sections concerning the type of IT systems that are employed. Due to different requirements concerning real-time, reliability, safety, etc. each section uses its specific hardware and communication technologies and standards [Zuehlke 2012]. The tasks related to business processes like production planning and scheduling are realized with IT systems based on common PC and internet standards (IT levels). In contrast, the control and process monitoring of the production process—which is strongly related to the technological process itself—is typically implemented in specialized and proprietary systems like Programmable Logic Controllers (PLCs) and field devices (AT levels). The in- and outputs (I/Os) of the field devices and controllers are connected via field bus systems or even old-fashioned wiring. The deficits of today's situation are a lack of integrated information exchange between the business and the technical level and monolithic and hardware-dependent control software due to the low implementation level and vendor-specific programming methods [Karnouskous et al. 2010, Zoitl and Vyatkin 2009].

To close the existing communication gap a standardized middleware that defines the communication technology is needed. Furthermore, to improve the information flow between different automation levels and the engineering of automation software the abstraction degree of the communication needs to be risen. Instead of exchanging raw bits and bytes the communication has to take place via encapsulated functions whose interfaces are described formally.

In the business process domain (ERP systems) such open and flexible software systems are already established today by means of Service-oriented Architecture (SOA). SOA is an abstract software paradigm which enables a high degree of reusability, flexibility, and interoperability of software components presented as services [Bieberstein 2005]. The most widely used technology for implementing SOA are Web Services.

The application of SOA is a promising approach to encounter the mentioned deficits of automation systems and has already been subject to several research activities. Thereby, a central question is how SOA can be realized on the lower automation levels without the need for revolutionary changes. Since the PLC is not only today's prevailing system for manufacturing control but also the connection to high-level scheduling and

planning systems, a sudden redesign of the automation pyramid without the PLC is simply not practical. Thus, the idea is to enhance the functionality of the PLC so that it can act as a mediator between the IT and AT section and thereby improve the software engineering for manufacturing control (see Figure 1).

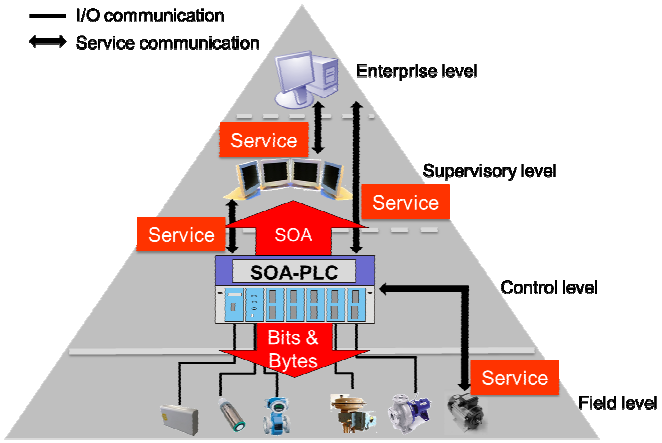

Fig. 1. SOA-PLC in the automation pyramid

2. STATE OF THE ART

SOA has already been applied to industrial automation in several research activities. Concerning the implementation of Web Services on PLCs in [Mathes et al. 2009] the *SOAP4PLC* engine for enabling the implementation of Web Services on PLCs is presented. The European projects SIRENA [Jammes et al. 2005] and SOCARDES [Souza et al. 2008] provided a substantial contribution for realizing services on device level. The follow-up project IMC-AESOP [Karnouskos et al. 2010] addresses a SOA-based monitoring and control applications for batch and continuous production processes. From these projects emerged a SOA technology for devices based on Web Services named the Devices Profile for Web Services (DPWS) [Zeeb et al. 2007]. Two research initiatives, SOA4D and WS4D, provide open source DPWS development toolkits [Mensch 2011, Zeeb 2010].

The authors' previous work investigated the application of SOA for the development of industrial control procedures [Ollinger et al. 2011]. For the realization of service-oriented control procedures an implementation concept has been developed featuring DPWS and the graphical modeling language Grafchart [Theorin et al. 2012]. Moreover, to enable an integrated control system development based on the SOA paradigm a model-driven engineering procedure has been defined [Ollinger and Zühlke 2013, Ollinger et al. 2013]. Additionally, a concept for the automatic code generation for implementing services on embedded devices was investigated by making use of existing MDD methods and tools [Ollinger et al. 2013b]. These research tasks were accompanied by practical applications at industry-related demonstration systems of the *SmartFactory*^{KL} in order to validate the results and to show the practical applicability.

3. CONCEPT OF A SOA-PLC

3.1 Basic concept of a SOA-PLC

A SOA-PLC works like a conventional PLC but has some enhanced features in terms of service-oriented functionality. Its purpose is to provide PLC software components as encapsulated services with a well-defined and openly accessible interface. Thereby, the connection to the field level via field busses or direct wiring of the I/Os and the PLC programming methods according to IEC 61131 are retained unchanged. Additionally to the conventional PLC, innovative devices that have already an own service interface can be accessed as well (see Figure 1).

The core tasks of a PLC are the implementation and execution of field device functions and process logics. The PLC program is structured with Program Organization Units, whereby Function Block (FB) is the main POE type [Tiegelkamp and John 2009]. The capability of a SOA-PLC allows representing any desired set FBs as a service (see Figure 2). A service generated by the SOA-PLC can either be used outside or even inside the SOA-PLC. By providing services to the outside the SOA-PLC acts as a mediator between AT and IT levels to vertically connect different automation systems and applications. This helps to integrate field device functions to automation software, like monitoring, process planning, and scheduling applications.

The internal use of services permits the engineering and execution of control programs in a service-oriented way. The higher abstraction level enable by functional encapsulation to services provides a better reusability, adaptability, and clarity of control software [Ollinger et al. 2013]. Thus, the control logics can be developed in a more abstract way without dealing with the implementation details of the underlying functions. The arrangement of services to a certain process logic is called service orchestration which can be encapsulated to a new service again so that various hierarchical service levels can emerge.

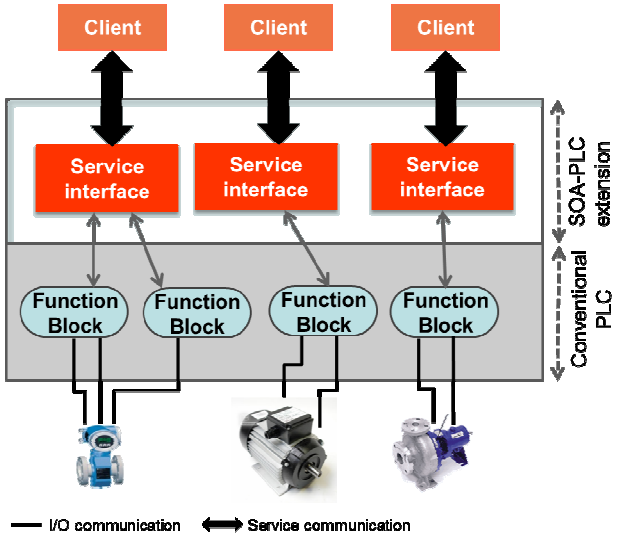

Fig. 2. Concept of a SOA-PLC

Altogether, to design and develop a SOA-PLC the following requirements have to be met:

- The traditional field level remains the same, which means no modifications on field devices themselves or their connection to the PLC via direct I/O wiring or bus systems are necessary.
- The encapsulation of selected FBs to a service is done automatically.
- The services are exposed by a standardized communication technology (e.g. Web Services, DPWS, OPC-UA).
- The service descriptions can be retrieved from outside in order to receive the information which services are available and how to access them.

3.2 Engineering concept of a SOA-PLC

A service can be roughly described as a software construct comprising various functions called operations that are accessible via a well-defined service interface. In terms of a service-oriented automation system the lowest service level represents the field devices so that the individual functions of a field device are exposed as service operations [Ollinger and Zühlke 2013].

To develop a SOA-PLC service representing field device functions the following steps are defined:

1. Functional encapsulation

First, every function of a field device that should be represented as a service operation has to be implemented in an own FB. To gain maximum flexibility during later service orchestrations those FBs should represent all basic functions of the field device and not only consist of aggregated functions. Each FB defines its in- and output parameters that constitute later the in- and output parameters of the service operation.

2. Grouping function blocks to a service

Since every physically present field device has its own service, FBs of the same type can be instantiated several times. Thus, all FB instances belonging to a field device are added to a group representing the device. Consequently, different field devices of the same type can later be used and differentiated easily.

3. Exposing the service to the environment

Finally, such a group of FBs representing a device is exposed to the surrounding environment as a service. Therefore, a description of the service interface has to be made public and the service itself has to be launched. Every FB belonging to the service is then accessible via a single service operation.

Besides field device functions any other desired PLC function can be provided as a service so that the described engineering steps just have to be adapted according to the respective case.

4. IMPLEMENTATION

After defining the basic concept of a SOA-PLC the question arises how the additional SOA features can be realized. Therefore, an implementation concept is defined with the objective to realize a SOA-PLC prototype on a conventional PLC with minimum modifications.

4.1 Integration Concept

There are different ways to realize the behavior of services on a PLC. The most efficient way would be to implement the services as separate tasks so that each service can act as a individual program. Thus, each task will only be started on demand saving resources and optimizing runtime. Moreover, a set of parameters can be handed over to the dynamically started task which is also able to hand back return values, e.g. indicating success or failure of the desired operation. However, today's PLC runtime environments have usually a limited amount of tasks so that the just a few services can be realized with one PLC.

Another possibility is to implement and execute the service behavior of all services with one PLC task in various FBs. An obstacle constitutes the fact that traditional PLC programming methods do not comprise a software concept that groups various functions to a service. However, a possibility to indicate a group of related FBs is a naming scheme. Therefore, naming conventions have to be defined so that the SOA-PLC can automatically identify which FBs belong to which service. To allow such an automated identification of different FBs belonging to a single service, a unique prefix is used for every service only changing the suffix according to the operation name, e.g. *ServiceName_OperationName1*.

Furthermore, the encapsulation of the service implementation to with a SOA interface according to a respective SOA technology needs to be done. This could happen inside of the PLC environment so that the PLC software components can be exposed as service directly. However, today's PLCs do not offer this feature. Thus, the service interface has to be realized outside of the PLC runtime environment. This implies that the PLC variables need to be accessible from the outside which could be realized with "virtual I/Os". Instead of communicating via electrical signals as real I/Os, virtual I/Os interact with services, being able to forward and receive parameters to and from them. Every time a service is called a virtual input variable is set and automatically and forwarded to the specific FB. In turn this FB sets a return value which will be forwarded as a virtual output to the service interface.

4.2 Implementation

The platform for the SOA-PLC prototype is a RFC 470 PN 3TX from Phoenix Contact which uses Windows CE 5.x as the basic operating system. For the implementation of the

services the technology Devices Profile for Web Services (DPWS) is chosen which has already been used for previous applications in the *SmartFactory*^{KL}.

To realize SOA-PLC prototype, a first solution with minimum adjustments of the PLC platform was preferred. Thus, the services are realized within the conventional PLC environment in a single task and the service encapsulation takes place outside of the PLC environment. During every cycle of the PLC runtime all FBs are executed. At the beginning of each FB the program checks whether the code inside the FB shall be executed indicated by a caller variable. If the service is not called the code implementing the service behavior is skipped. A specific piece of software called PLC handler constantly checks the status of the virtual I/Os forwarding the corresponding variables to the PLC environment.

The PLC platform is extended with additional software components to transform the ordinary PLC into a SOA-PLC. Since the platform uses a Win CE operating system a .NET Compact Framework was installed to use a DPWS stack which relies on the framework. The communication between the functions inside the conventional PLC code and the program parts outside the PLC environment are realized via virtual I/Os as described above. Figure 3 shows the system architecture of the SOA-PLC with its different software modules.

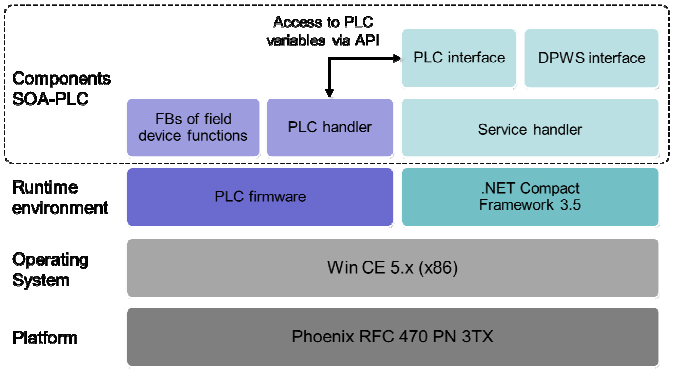

Fig. 3. SOA-PLC system architecture

The functionality of the individual software modules of SOA-PLC are described in the following:

a) Function blocks inside the PLC environment

During the implementation of the function blocks for the field devices in the PLC code a strict naming scheme is used. Thereby an automatic detection of input and output parameter is made possible by the known prefixes *IN_* and *OUT_* respectively. Every function block uses specific *CALL* and *DONE* variables of the bool type thus creating a virtual I/O.

b) PLC handler

The PLC handler is responsible for checking the caller variables to recognize whenever a FB is called. It is realized as a simple *IF ELSE* sequence, constantly detecting whether a *CALL* variable was set to *true* and thereby a function was called or not. Since the PLC handler is located right inside the conventional PLC environment, the PLC handler is simply implemented as a function block as well. Once a *CALL* variable is set the PLC handler starts the corresponding FB and hands over the values of the *IN_* parameters to the FB. After the FB was executed the PLC handler sets the *OUT_* parameters.

c) PLC interface

Outside the PLC environment a special PLC interface communicates with the PLC environment. The PLC interface writes the *CALL* and *IN_* variables into the PLC environment whenever a service is requested and reads the *OUT_* variables back. This is possible by using an API realized as a static library provided by the PLC manufacturer Phoenix Contact. Since the .NET Framework can only handle dynamic link libraries, the static library has been converted into a dll.

d) Service handler

The service handler is mainly responsible for the creation of the DPWS services. Consequently, it generates polls to get the existing service groups which exist inside the PLC environment using the already mentioned API from Phoenix Contact. By processing the described naming scheme, e.g. *ServiceName_OperationName1*, all operations belonging to one service are grouped into a single service. Afterwards the service handler creates the DPWS device with its hosted services. Finally the service handler starts the DPWS server making the services accessible from outside the PLC.

e) DPWS interface

To offer the functions as web services a DPWS stack is needed. Since the .NET Compact Framework has no build in DPWS stack, an already ported version [Züg10] from a .NET Micro Framework was used. The DPWS interface is responsible for providing the web services handling all the communication with the environment outside the PLC.

5. CASE STUDY

The concept has been evaluated and demonstrated with a small setup consisting of a PLC, a bus coupler, and two industrial field devices in the *SmartFactory*^{KL}.

The used PLC is the same Phoenix Contact PLC (RFC 470 PN 3TX) as the one used during the general implementation of the SOA concept. An ultrasonic sensor (UB200-12GM-ES-V1) from Pepperl+Fuchs and a signal lamp from Werma were attached to digital I/Os of a Phoenix Contact bus coupler (IL PN BK DI8 DO4 2TX-PAC) which connects to the PLC via PROFINET bus. Additionally, a 24 V DC power supply from Phoenix Contact was used to power the whole setup (see in Figure 4).

The setup was designed in such a way that the signal lamp turns on whenever an object comes into a specific range of the ultrasonic sensor. Once the object moves backwards out of the specific range of the sensor, the lamp turns of again.

To make this a SOA-PLC setup a dedicated computer was connected to the PLCs 2nd LAN port, which has been used for development matters throughout the project. The whole orchestration of the services takes place in the tool JGrafchart running on a dedicated computer outside the PLC communicating with the PLC via web services.

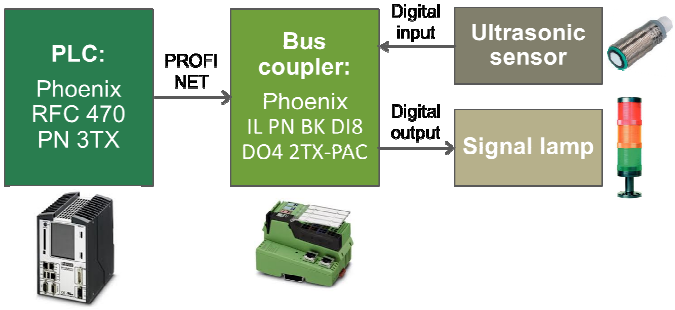

Fig. 4. Test setup in the SmartFactory[KL]

6. CONCLUSIONS

The presented concept of a SOA-PLC overcomes the current limitations of existing PLC systems, meeting the demands of a rapidly changing manufacturing environment, and its challenging requirements on flexibility and reusability.

By grouping function blocks and encapsulating them into web services field device functions are made accessible outside the PLC on higher level of the automation pyramid. Consequently the control logic of a SOA-PLC is realized no longer by an inflexible sequential code but by a dynamic orchestration of services. The orchestration can be done by any DPWS orchestration tool like JGrafchart on any conventional PC or even on another PLC. Since the generation and deployment of the web services are fully automated once the developed software modules are installed on the PLC, no extra effort is need during the implementation of the function blocks except keeping to a specific naming scheme. The described implementation on the RFC 470 PN 3TX Phoenix Contact PLC running Windows CE 5.x gives proof of concept.

The promising results of this work offer multiple options for subsequent research. One of the next logical steps is the implementation of additional functionality so that the orchestration of the services can be done right within the same SOA-PLC which offers the services. Moreover there is no implementation of any IT security in the SOA-PLC concept by now which leafs plenty of work for innovative security concepts like dynamic rights management and suitable endpoint related security fitting the needs of a SOA.

REFERENCES

Bieberstein, N., et. al. (2005). *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap.* Prentice Hall PTR. Upper Saddle River, NJ, USA.

Gerber, T., Bosch, H.-C., Johnsson, C. (2013). Vertical Integration of Decision-Relevant Production Information into IT Systems of Manufacturing Companies, *In: Service Orientation in Holonic and Multi Agent Manufacturing and Robotics,* Studies in Computational Intelligence, pp. 263–278. Springer Berlin Heidelberg.

Karnouskos, S., Colombo, A.W., Jammes, F., Delsing, J., Bangemann, T. (2010). Towards an architecture for service-oriented process monitoring and control, *In: Proceedings of the 36th Annual Conference on IEEE Industrial Electronics Society,* IECON 2010. pp. 1385 – 1391, Phoenix, USA.

Mathes, M., Stoidner, C., Heinzl, S., Freisleben, B. (2009). SOAP4PLC: Web Services for Programmable Logic Controllers, *In: Proceedigns of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing,* PDP 2009, pp. 210 –219, Weimar, Germany.

Mensch, A. (2011). *Introduction to SOA4D projects*, URL: http://www-old.itm.uni-luebeck.de/projects/tekomed/workshop3/Mensch%202011.%20Introduction%20to%20SOA4D%20projects.pdf?lang=de, retrieved January 3, 2013

Ollinger, L., Schlick, J., and Hodek, S. (2011). Leveraging the Agility of Manufacturing Chains by Combining Process-Oriented Production Planning and Service-Oriented Manufacturing. *In Proceedings of the 18th IFAC World Congress*. Milan, Italy.

Ollinger, L., Zühlke, D. (2013). An Integrated Engineering Concept for the Model-based Development of Service-oriented Control Procedures. *In: Proceedings of the IFAC Conference on Manufacturing Modelling, Management and Control.* MIM 2013, IFAC PapersOnLine pp. 1441–1446. Saint Petersburg, Russia.

Ollinger, L., Zühlke, D., Theorin, A., Johnsson, C. (2013). A Reference Architecture for Service-oriented Control Procedures and its Implementation with SysML and Grafchart, *In: Proceedings of the 18th IEEE International Conference on Emerging Technologies and Factory Automation.* ETFA 2013, Cagliari, Italy.

Ollinger, L, Wehrmeister, M., Pereira, C., Zühlke, D. (2013b). An Integrated Concept for the Model-Driven Engineering of Distributed Automation Architectures on Embedded Systems, *in: Proceedingss of the IFAC Workshop on Intelligent Manufacturing Systems. Presented at the IMS 2013*, São Paulo, Brazil, 2013.

Tiegelkamp, M., John, K. H.: *SPS-Programmierung mit IEC 61131-3*. Springer, Berlin, 2009.

Theorin, A., Ollinger, L., and Johnsson, C. (2012). Service-oriented Process Control with Grafchart and the Devices Profile for Web Services, *In: Proceedings of the IFAC Symposium on Information Control Problems in Manufacturing (INCOM 2012),* Bucharest, Romania.

Westkämper, E., Zahn, E. (2009). *Wandlungsfähige Produktions-unternehmen: Das Stuttgarter Unternehmensmodell.* Springer.

Zeeb, E., Bobek, A., Bonn, H., Golatowski, F. (2007). Lessons learned from implementing the Devices Profile for Web Services, *In: Proceedings of the Digital EcoSystems and Technologies Conference 2007*, DEST '07. pp. 229–232, Cairns, Australia.

Zeeb, E., Moritz, G., Timmermann, D., Golatowski, F. (2010). WS4D: Toolkits for Networked Embedded Systems Based on the Devices Profile for Web Services, in: Proceedings of the 39th International Conference on Parallel Processing Workshops, ICPPW 2010, pp. 1 –8

Zoitl, A., Vyatkin, V. (2009). IEC 61499 architecture for distributed automation: The "glass half full" view. *IEEE Industrial Electronics Magazine*, Vol. 3, pp. 7–23.

Zühlke, D. and Ollinger, L. (2012). Agile Automation Systems Based on Cyber-Physical Systems and Service-Oriented Architectures. *Advances in Automation and Robotics*, Vol. 122, Part 1, pp. 567–574, Springer Berlin Heidelberg, Germany.