LUND UNIVERSITY

A Reference Architecture for Service-oriented Control Procedures and its Implementation with SysML and Grafchart

Ollinger, Lisa; Zuhlke, Detlef; Theorin, Alfred; Johnsson, Charlotta

Link to publication

A Reference Architecture for Service-oriented Control Procedures and its Implementation with SysML and Grafchart

Lisa Ollinger, Detlef Zühlke
German Research Center for
Artificial Intelligence
Kaiserslautern, Germany
{Lisa.Ollinger, Detlef.Zuehlke}@dfki.de

Alfred Theorin, Charlotta Johnsson
Department of Automatic Control
Lund University, Sweden
{Alfred.Theorin, Charlotta.Johnsson}
@control.lth.se

Abstract

Innovative engineering methods are needed to enhance the adaptability and agility of industrial control procedures and concurrently manage their rising complexity. Service-oriented Architecture (SOA) constitutes a promising paradigm to meet these challenges. To apply the rather abstract SOA principles to industrial automation, a model-driven engineering method is presented in this paper. Therefore, a reference architecture is introduced providing a framework to structure and define services for control tasks in a convenient way. The application of the engineering method is supported by an implementation concept. It uses SysML during the design phase and the process modeling and control language Grafchart for the execution of service-oriented control procedures.

1. Introduction

The complexity of industrial control procedures is steadily increasing. This is mainly due to rising demands on highly flexible and high quality production processes as well as to technologically upgraded production equipment and products. Additionally, production processes are constantly optimized and reconfigured, which implies that the control procedures need to be adaptable and reusable.

Programmable logic controllers (PLCs) are the established control devices in industry used for executing the control procedures. Although, PLCs were technologically improved over the time, their low-level programming style has not changed. This implies that PLC programs are often hardware-dependent, complicated, and monolithic and thus error-prone and difficult to adapt or reuse [1]. Besides old-fashioned programming techniques, there is no well-documented design procedure that is fluently connected to previous engineering domains, in particular the process planning. Thus today, the control system development requires a high effort during the initial engineering as well as during reconfiguration of production plants.

To improve the current situation new methods for efficient development of control software are needed. A promising approach is to apply the paradigm of Service-oriented Architecture (SOA) to design control procedures as cooperating building blocks that are easy to reuse or rearrange. The SOA approached applied within Automation is referred to as SOA-AT [11]. The foundation of SOA is the encapsulation of functionality to services which in turn help to design control procedures in a more abstract way. Thus, SOA can be combined with Model-Driven Development (MDD) and Model-Driven Engineering (MDE) techniques in order to develop control software in the various concretization steps.

The presented work is based on the concept *MDE for SOA-AT* which describes an MDE procedure for the design of service-oriented control architectures (section 2). Then, the concept is detailed regarding a reference architecture for the definition of services for service-oriented control procedures (section 3) and their model-driven design using SysML (section 4). After that, the application of Grafchart as executable service orchestration language and a transformation concept from the SysML design model to Grafchart is presented (section 5). Finally, the application of the overall concept is illustrated by the design and implementation of a service-oriented control architecture at a practical demonstration system (section 6).

2. State of the art

To improve the generation and verification of control procedures, various approaches have emerged that make use of modeling. Most of them focus on the automatic generation of PLC code by means of MDD using different formal and non-formal modeling languages, e.g. UML [2], SysML [3,4], Petri Nets [5], and State Charts [6]. Another approach is the direct execution of the models without additional transformations. Despite lower reliability and real-time guarantees, executing on ordinary computers means a much larger freedom in choosing programming language and execution platform. One particular language designed for both control and

modeling is Grafchart [7]. It is based on the IEC 61131-3 [8] language Sequential Function Charts (SFC). Grafchart has been extended with support for reusability and exception handling and it also has support for formal verification.

Existing applications of SOA in the automation domain focus on the realization of distributed automation system as networked services that are mostly implemented with Web service technologies. In this context the EU projects SIRENA, SOCRADES, and the currently running IMC-AESOP play an important role [9–11].

In previous work the SOA-AT concept is presented, which describes the idea of networked automation systems that work in a service-oriented way [12]. This approach was then combined with the process modeling language Grafchart in order to implement and execute control logics as service orchestrations [13]. For MDE of service-oriented control architectures the method *MDE for SOA-AT* was developed including several concretization steps [14]. The designed control architecture can then be implemented in various ways, for example in a highly distributed way on embedded devices in form of a cyber-physical system [15].

3. Services for process control

The following section investigates how the SOA paradigm can help to achieve an efficient engineering of control procedures. Furthermore, a reference architecture for defining services to generate industrial control procedures is presented.

3.1. Service-oriented process control

Service-oriented architecture (SOA) describes a concept for distributed computing that originally stems from the business process domain with the objective to realize agile and flexible enterprise IT systems [16]. The basic idea of SOA is the encapsulation of functions to reusable services that serve as building blocks for the generation of new functionality [17]. SOA is a wide-spread concept to enable the realization of adaptable business processes by using standardized software components provided by services.

These principles can be transferred to the control system development where control procedures for production processes have to be generated. In the context of industrial automation, a service-oriented control procedure comprises automation functions that are exposed as reusable services [12]. To execute the desired production process the services are coordinated in a certain order according to the desired process flow.

The major benefit of SOA is provided by the encapsulation principle. It permits to reduce the complexity of control procedures by splitting up the control software in reusable functional units and hiding implementation details. Thus, control procedures can be

developed first in an abstract way so that the overall software structure can be defined independently of implementation and technological details. Furthermore, the development of control software can take place on a higher abstraction level so that the gap between the control system development and the previous process planning phase can be closed. Altogether, SOA is a promising concept to achieve high adaptability and reusability, comprehensible and clear software structure, and seamless connection to other engineering domains.

3.2. Service levels and types

For introducing SOA in a special application domain the definition of a reference structure is recommended that specifies conventions for the determination of services. The complexity of the overall application can be reduced by classifying services according to the application domain. By defining vertical layers the overall application can be split up in composition levels [17]. Furthermore, service types can be defined for grouping services with similar characteristics [16].

Today control procedures, typically implemented as PLC programs, are usually programmed directly in a bottom-up way without first designing the software structure. The programs comprise several aspects: Logics are needed to implement the field device functions and to establish the connection to the respective input/outputs and communication protocols of the controlled devices. Furthermore, control logic defining the process flow and additional control routines for initialization, safety, monitoring, scheduling, etc. of the process are needed [18]. How these aspects are structured in Function Blocks (FBs) and how these are named varies a lot depending on for example the programmer, the company, and the application domain. Derived from this, two service layers can be identified to structure service-oriented control procedures: the application-independent *equipment layer* and the superior *process layer* (Figure 1).
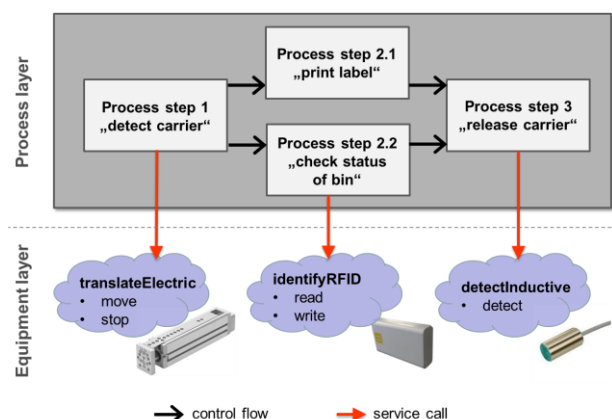

Figure 1. Service layers of service-oriented control procedures.

The services of the *equipment layer* represent the functions of the production equipment that execute the technological process (*Equipment layer* in Figure 1). To achieve the desired behavior, the equipment services have to be executed in a certain order defined by the process logic (*Process layer* in Figure 1). This procedure of arranging services in a certain way to achieve new functionality is called service orchestration. Within the equipment layer the services are classified as *basic services* and *composed services*. The basic services constitute the elementary mechatronic functions of devices and machines that are encapsulated to services. Thus, basic services are the interface between the electromechanical hardware and the IT-based automation system. These basic services can be aggregated to create equipment services with coarser granularity for production modules, units, etc., to *composed services*.

In the *process layer* the executable control procedure depending on the required production process and the available equipment services is generated. In order to achieve the desired process flow *control services* define the process logic and the required functions to execute the desired technological process. In order make the control service executable the binding of the required functions to available equipment services has to be established. Since these control services have to consider the physical structure of the production system (production components, material flow, etc.) as well as the current production tasks (product types and variants) they are highly specialized for the respective application. In order to provide reuse and clarity, parts of the overall control can be outsourced in separate services.

If various product types are produced with the same production plant, the definition of *product services* is useful. They specify the sequence of production steps to produce the respective product type. Thereby, the required equipment services can already be bind concretely or be still abstract so that the binding has to be done in the control service (Figure 2). Furthermore, other parts of the control program according to [18] can be implemented in separate services, here summarized as *supporting services*. Altogether, on *process level* the *control service* works as a central orchestrator that uses the *product* and *supporting services* as building blocks and establishes the missing bindings to the available *equipment services*.

Figure 2 illustrates the defined service types and how they interact. For example, the composed service "c2" uses the basic service "b1". Within the control service "cr1" a supporting service "s1" and the *product service* "p1" are called. To define high-level functionality independently of the current equipment layer, the service allocation can be done first in an abstract way so that the respective services are bound in the control service.
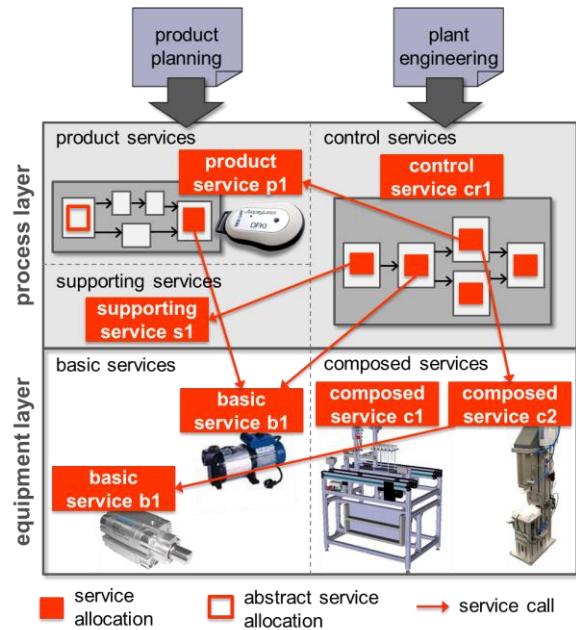

Figure 2. Interaction between the service types.

3.3. Service definition and naming scheme

As Figure 2 depicts, the allocation of services is very important to build new functionality in a flexible way. Therefore, a mapping between the required functionality and suitable services has to be made. How fluently and uncomplicated this mapping procedure is depends on the level of detail of the control services and the granularity of the equipment services. The granularity influences the abstraction level where the mapping between process and equipment services takes place (Figure 3).

As a rule, the coarser the granularity of the equipment services, the less detailed the specification of the control service can be. Since the deep technological details of the production process are hidden then, the service orchestration of the control service is rather simple. However, coarse-grained services lead to a lower adaptability because they hide most of the details which can only be influenced via service parameters from the outside. Furthermore, their reusability is rather low because their implementation relies strongly on the current production system. Consequently, a higher flexibility and reusability of the whole SOA system can only be achieved with more fine-grained equipment services.

To avoid this trade-off the basic services are defined on a low technological level as the functions of the individual field devices. They offer the possibility to specify and influence the technological process in detail. By defining composed services, built upon the basic services, coarse-grained services can be created that help to increase the abstraction level of the mapping procedure.
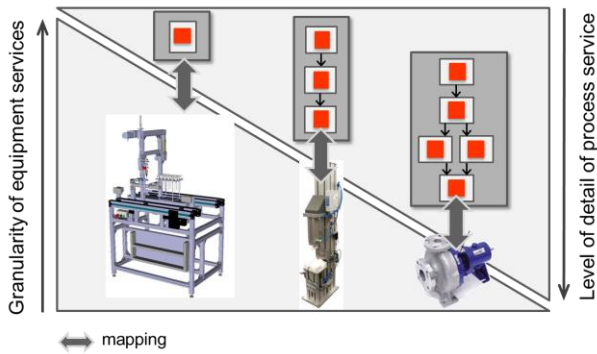
**Figure 3. Mapping between process and equipment services.**

Thus, a crucial issue during the SOA design is to determine the equipment services by considering their granularity and scope in the best way. Since the functionality of the service is encapsulated the *service interface* exposes the service to the outside. Therefore, the name of the service and its related operations have to be defined. First of all, the names of the equipment services have to express the mechatronic function of the production equipment independently of its respective use. For example, a pneumatic cylinder receives the service "translate" because it executes a translational motion and an inductive sensor provides the service "detect".

However, this definition might be too general because the plant engineer needs to know how the functionality is executed technologically. The addition of the type of energy for actors and the operating principle of sensors can help to reduce the risk of technological incompatibilities. By naming the service of a pneumatic cylinder "translatePneumatic" the engineer knows that additional devices are needed to supply compressed air. The inductive sensor has the service "detectInductive" which clarifies that the service is suitable for a metallic work piece but not for a wooden one.

This naming scheme can be applied independently of the respective realization by concrete devices. This helps to support the mapping procedure by specifying the required functions in the process services as *abstract services* that can be bound later to *concrete services* of various field devices. Ideally the interfaces of the abstract and concrete services are exactly the same. However, the exact functional scope and how the functions are controlled can vary for different devices, especially for more complex ones. In this case, another concretization level is needed where the name of the vendor or a type identifier is added to indicate that the service is tailored to a special device, for example "translatePneumaticSF101" (Figure 4).

How the functionality of the service can be accessed is defined by the operations of the service. During the concretization steps the operations and/or their parameters have to be adjusted or added respectively

(Figure 4). How abstract or specific the service interfaces should be depends on the characteristics of the devices and the application. As a rule, the service definition should be as general as possible and as concrete as necessary.
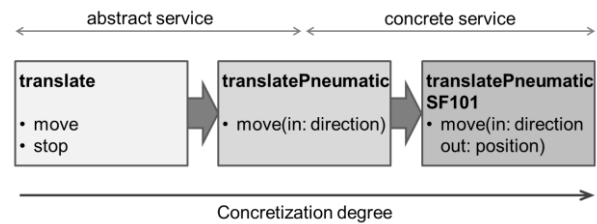

Figure 4. Concretization steps during the definition of equipment services.

Besides the degree of concretization the standardization of service and operations names are essential to enable an automatic detection and binding of suitable services and the exchangeability between devices of different vendors. However, experiences from business process SOAs have shown that the organization of company-wide services constitutes a highly complex task. A first approach for a standardized naming of equipment services makes use of existing automation standards. Promising candidates are field device profiles like the PLCopen Motion Control Specification that comprises the standardized definition of motion control functions [19]. Additionally, the international eCl@ss product classification system which, among other things, comprises a comprehensive library of all types of production equipment [20]. eCl@ss comprises four hierarchical levels where the third and fourth level can help to derive a service name for a field device. For example, a flowmeter is assigned to the group "27-20-04 Meas. istr. flow, volume" where the general function "flowmeasure" can be deduced. The fourth level comprises several types of flowmeters, e.g. "27-20-04-01 Flowmeter (magn.-induct.)" so that the operating principle can be selected. The name of the service is then determined to be "flowmeasureInductive".

4. Model-driven design with SysML

So far, *MDE for SOA-AT* has been defined in a general way [14, 15]. It describes how a fluent information flow can be achieved from process planning to executable control procedures by using service-orientation and modeling. Therefore, the planning model comprises several parts: the Hardware Structure, the Service Structure, and the Service Orchestration (Figure 5). To apply the method the general modeling instructions have to be transferred to an existing modeling language. Therefore, SysML is chosen because it is a wide-spread modeling language for systems engineering [21].
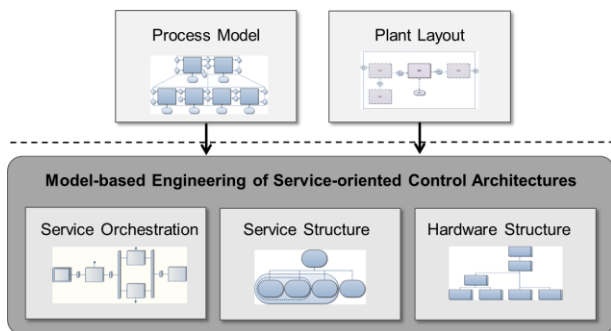
Figure 5. Overview of the *MDE for SOA-AT* method [14].

In the *Service Structure* the services are defined according to their interface specification so that it gives an overview of all available services in the current system. In SysML a service is represented as *block* and the service operations are added as operations of the block for which input and output parameters can be assigned. Within a *Block Definition Diagram* the individual services can be designed and relations between services can be added to express their dependencies. The "use" relation indicates that a service is using another one for providing its own functionality. To indicate its service type stereotypes are defined that can be assigned to each block. In Figure 6 a section of the Service Structure of the use case (section 6) comprising four services is illustrated. The three basic services "TranslatePneumatic" (of a pneumatic cylinder) and "DetectMagnetic1/2" (of two reed switches) are composed to the service "VerifyingTranslate Pneumatic1" that is used by another composed service "AssemblyPneumatic".
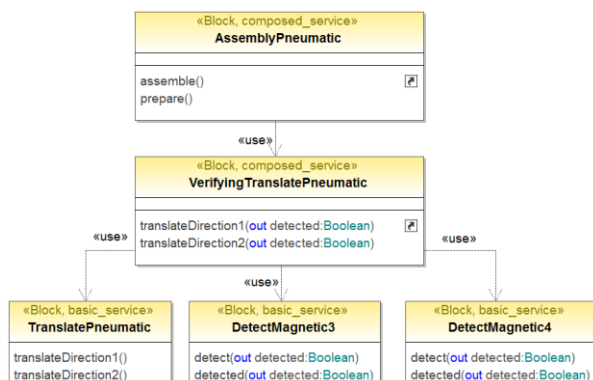
Figure 6. Example of a service structure.

The *Service Orchestrations* that determine the inner behavior of process services are modeled as *activities*. In an *Activity Diagram* the logic of an activity can be designed by using *actions*, representing single process steps, and *control flow* elements. The call of an equipment service can be inserted by a *CallOperationAction* that links to the operation of the block that represents the required service. To insert a

process logic that is outsourced to another service (product/supporting service) a *CallBehaviorAction* is used that links to the respective activity. If the control flow is bound to conditions either *localPre/ Postconditions* of actions or *eventActions* can be used. Figure 7 illustrates how a service orchestration is modeled as an Activity Diagram. The activity "Cr_Production" comprises the action "Supply new product base" that calls the operation "supply" of the service "ControlStorage1". The next action "Transport new product base from storage to assembly module" is not activated until the event "Product base ready" happens. This action calls the service "PickAndPlace1" and uses the input parameter "motionID" to adjust the operation "motion". After the condition "motion finished" is fulfilled the activity "Pr_KeyFinder" is called that represents the product service.
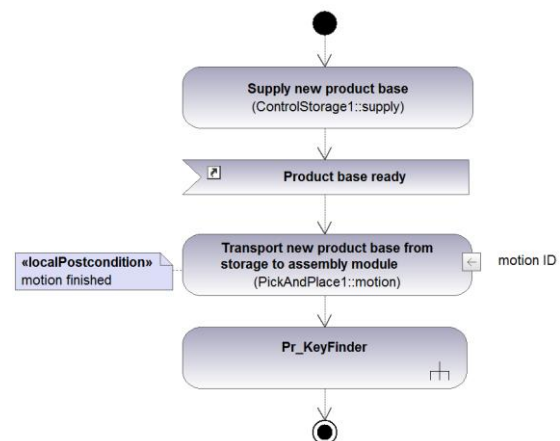
Figure 7. Service orchestration of the service "Cr_Production".

5. Process implementation with Grafchart

The designed control architecture in SysML is completely independent of how the services and the service orchestrations are implemented. In previous work, an implementation concept with Grafchart as a suitable process modeling language for the development and execution of service orchestrations in combination with the Devices Profile for Web Services (DPWS) service technology was introduced [13] which is connected to the SysML modeling in the following.

5.1. Introduction to Grafchart
Graphical programming languages are popular in the automation industry. Some advantages are simplicity and providing a better overview of the applications than textual languages. Visualization of a graphical application is also straightforward and more intuitive compared to textual languages.

Grafchart is a graphical programming language based on Sequential Function Charts (SFC), one of the IEC

61131-3 PLC standard languages, and uses the same graphical syntax with steps and transitions [7]. Steps represent the state of the application and transitions represent the change of state. Associated with the steps are actions which specify what to do at certain occasions. Associated with each transition is a Boolean guard condition. By using additional concepts like several action types, alternative and parallel paths, hierarchical structuring (*macro steps*), reusable functions (*procedures*), exception handling (*exception transitions*), it is convenient to implement large applications that are maintainable and possible to overview.

5.2. Service-oriented control with JGrafchart

The tool JGrafchart (JG) is a free Grafchart implementation that supports numerous graphical elements to make it possible to create rich and interactive operator interfaces. It also supports various customizable IO to make it possible to connect applications to practically any external environment with only a small or moderate effort. Since version 2.1.0 JG also has support for DPWS. JG depends on the DPWS mandatory web service extensions WS-Discovery, WS-MetadataExchange, and WS-Addressing to enable automatic discovery of DPWS devices, retrieving available services and operations, and connecting to devices as a client. It also has generic support for interacting with the devices. Writing a JG application using DPWS devices is done by binding a portType in a discovered DPWS device to a DPWS Object element in JG. No device specific glue code is required to interact with the device. Once the binding has been specified, the operations are called on the DPWS Object, see Figure 8.

In the figure, a device called "Kitchen light" has been automatically discovered and device metadata and available services and operations have been added to the service registry. To use the operations in the "SwitchPower" portType a DPWS Object called "myDPWSObj" has been added to the application and bound to this portType. "myDPWSObj" is then used to call the operations "Switch" and "GetStatus". The function *dpwsSubscribe* initiates a subscription and *dpwsHasEvent* is used to check if an event of type "StatusChanged" has been received.

In recent versions of JG the DPWS feature has been further improved. In version 2.2.0 it is possible to detect if a service call failed using the new functions *dpwsHasFault* and *dpwsGetFault* and handle this in the application. In version 2.3.0 it will also be possible to start the execution without all devices being present and to manually specify a binding for a device that will be available later. This is especially useful in the beginning of the implementation phase since all devices might not yet exist. The possibility to have comments in transition conditions has also been added to make it possible to have text strings like the ones in a SysML activity

diagram as comments specifying the intended transition conditions in natural language.
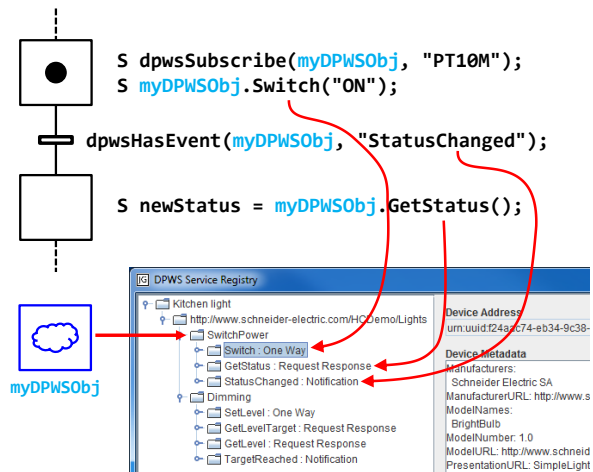

Figure 8. Integration of DPWS in JGrafchart.

5.3. Transformation of SysML to Grafchart

For realizing service-oriented control procedures the chosen implementation concept uses DPWS as service technology for the equipment services and JG to implement the process services. Within JG the individual services in the process layer are not implemented as DPWS services but as independent Grafchart Procedures. To reduce the effort of developing the executable control applications the tool "SysML2JG" automates the transformation of service orchestrations specified as Activity Diagram to JG programs. The most important transformation rules are listed in Table 1.

Table 1. Transformation rules from SysML to JGrafchart

| SysML	Grafchart
Action	Step
Activity	Procedure
Link to service block in CallOperationAction	DPWS object + action with service call in step
link to service block in CallOperationAction	procedure object + procedure step
Input pins	Variable
Event or Pre/Post-condition of an action	Transition condition
InterruptibleActivityRegion with ExceptionHandler	Marco step with exception transition
Initial/Final node	Enter/Exit step
Fork/JoinNode	Parallel Split/Join
Additional information (input parameters, etc.)	Comments in steps and transitions

The SysML model has to be exported to the standardized XMI format which serves as input for the "SysML2JG" tool. The tool analyses the XMI file and automatically generates a JG file that comprises all the information about the process services that was specified in the SysML model. To make the JG file executable it only needs to be detailed with implementation specific details and the service bindings.

6. Use case

6.1. Demonstration system

The presented concept was applied to a real demonstration system of the SmartFactoryKL. It is a production unit that assembles several product parts to a finished product. It comprises three different assembly modules, a conveyor belt, a Pick&Place unit, two storages, and several other devices with a total of about 50 field devices. In a first version the unit was controlled by a PLC that should be replaced by a distributed control system with service-oriented control.

To employ the production equipment as mechatronic units that expose their functional capabilities directly as services, some enhancements have been made to the hardware. 13 so-called *service gateways* were installed which extend several production modules and some single field devices with computational power and an Ethernet interface. A service gateway is realized with a Digi Connect ME 9210 microcontroller and an I/O board to connect the device interfaces to the microcontroller. Figure 9 depicts the complete demonstration system and its pneumatic assembly module that is extended by a service gateway.
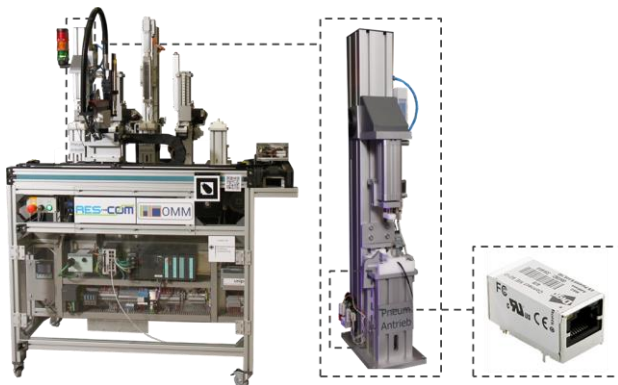
Figure 9. Demonstration system (left), pneumatic assembly module (middle), microcontroller (right)

6.2. Control system development

The service-oriented control architecture was designed according to *MDE for SOA-AT* including the development of a SysML model of the complete demonstration system. The equipment services (40 basic services and 12 composed services) have been implemented as DPWS Web services on the micro controllers by using the DPWS Core toolkit [22, 23]. The implementation of the equipment services constituted the foundation to enable creation of high-level process services.

In addition to the control service, two product services were defined to produce two different product types. The SysML2JG tool was used to transform the SysML model to a JG program. The content of the control service was transferred to the top level of the JG program and the product services were transformed to Grafchart procedures. Then the natural language comments were replaced by actual service calls and transition conditions. Furthermore, some details concerning error handling were added. To receive the production orders, a connection to the ERP system was implemented by using the JG Socket IO capability. In the last step the service bindings were established. The DPWS Objects were bound to the available services in the automation network that could be selected via the JG service registry. The abstract service allocations in the product services were carried out with dynamic references that are allocated dynamically to the respective DPWS Object depending on the current order. This applies for deciding which assembly station to choose.

The implementation and testing of the new control program were performed successfully and the complete production unit now runs on a service-oriented control architecture that replaces the PLC systems completely.

7. Conclusions and outlook

The presented MDE method provides an integrated procedure from design to implementation of service-oriented control procedures. The tool for the automatic transformation from SysML to JG will be further developed and made freely available. The reference architecture constitutes a design guideline for a suitable structuring and definition of services. Here, further collaborating activities are needed to establish standardized and vendor-independent service libraries.

The application of the method has shown that the development of the overall control procedures can be performed very easily and quickly as soon as the equipment services are established. Various extensions and changes to the control logic and enhancements on the equipment were made. Due to the raised clarity of the control structure and the high reusability of the services the control procedures could be adapted conveniently and with low efforts. The advantages and disadvantages compared to ordinary PLC engineering will be examined systematically in the future.

Since the implementation of the equipment services on the embedded systems was the most complicated part there is much room for improvement. In future work the automatic implementation of the equipment services

based on the engineering models will be elaborated. The use of embedded devices as a computational platform selected here constitutes the basis for the development of future cyber-physical production systems. The vision contains the dynamic deployment of services on varying hardware platforms that are distributed in an automation network. Another possible solution to evince a migration path for today's automation systems would be the implementation on ordinary PLCs.

Acknowledgements

References

[1] Zühlke, D. and Ollinger, L., "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, 2012.

[2] Vogel-Heuser, B., Witsch, D., Katzke, U.: "Automatic code generation from a UML model to IEC 61131-3 and system configuration tools", *Proceedings of the International Conference on Control and Automation, ICCA 2005,* pp. 1034 –1039 Vol. 2, 2005.

[3] Chiron, F., Kouiss, K.: "Design of IEC 61131-3 function blocks using SysML",*Proceedings of the Mediterranean Conference on Control Automation MED 2007*, 2007.

[4] Thramboulidis, K., Frey, G.: "An MDD process for IEC 61131-based industrial automation systems", *Proceedings of Emerging Technologies & Factory Automation (ETFA 2011),* 2011.

[5] S. Klein, G. Frey, M. Minas, "PLC Programming with Signal Interpreted Petri Nets", in: Aalst, W.M.P. van der, Best, E. (Eds.), *Applications and Theory of Petri Nets 2003*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 440–449, 2003.

[6] D. Thapa, C. M. Park, S. C. Park, G. N. Wang, "Auto-generation of IEC standard PLC code using t-MPSG", Int. J. Control Autom. Syst. 7, 165–174, 2009.

[7] C. Johnsson., *A Graphical Language for Batch Control,* Ph.D. thesis, Department of Automatic Control, Lund University, Sweden, March 1999.

[8] IEC-61131-3 ed2.0, "Programmable Controllers, Part 3: Programming languages", Technical report, *International standard,* International Electrotechnical Commission, 2003.

[9] F. Jammes, A. Mensch, H. Smit, "Service-oriented device communications using the Devices Profile for Web Services", *Proceedings of the 3rd International Workshop on Middleware for Pervasive and Ad-hoc Computing (ACM)*, Grenoble, France, pp. 1–8, 2005.

[10] L. Souza, P. Spiess, D. Guinard, M. Köhler, S. Karnouskos, D. Savio, "SOCRADES: A Web Service based Shop Floor Integration Infrastructure", *Proceedings of the Conference Internet of Things 2008*, Zurich, Switzerland, pp. 50–67, 2008.

[11] S. Karnouskos, A. W. Colombo, F. Jammes, J. Delsing, T. Bangemann, "Towards an architecture for service-oriented process monitoring and control", *Proceedings of the 36th Annual Conference on IEEE Industrial Electronics Society. (IECON)*, 2010.

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

[13] A. Theorin, L. Ollinger, C. Johnsson, "Service-oriented Process Control with Grafchart and the Devices Profile for Web Services", *Studies in Computational Intelligence,* Vol 472, pp 213-228, Springer, 2013.

[14] L. Ollinger and D. Zühlke, "An Integrated Engineering Concept for the Model-based Development of Service-oriented Control Procedures", Proceedings of the IFAC Conference on Manufacturing Modelling, Management and Control (MIM 2013) (to be published), Saint Petersburg, Russia, June 2013.

[15] L. Ollinger, M. A. Wehrmeister, C. E. Pereira, D. Zühlke, "An Integrated Concept for the Model-Driven Engineering of Distributed Automation Architectures on Embedded Systems", Proceedings of the 11th IFAC Workshop on Intelligent Manufacturing Systems (MIM 2013) (to be published), São Paulo, Brazil, May 2013.

[16] D. Krafzig, K. Banke, D. Slama, *Enterprise SOA: Service-Oriented Architecture Best Practices,* Prentice Hall, Upper Saddle River, New Jersey, USA, 2005.

[17] T. Erl, *Service-Oriented Architecture: Concepts, Technology, and Design,* Prentice Hall, 2005.

[18] K. Güttel, P. Weber, A. Fay, "Automatic generation of PLC code beyond the nominal sequence", *Proceedings of the IEEE International Conference on Emerging Technologies and Factory Automatio ETFA 2008,* pp. 1277–1284, September 2008.

[19] PLCopen Technical Committee (2011). *Function blocks for motion control,* Version 2.0, Technical Specification.

[20] eCl@ss e.V, *eCl@ass 7.0: Classification and Product Description,* URL: http://www.eclass.de, retrieved March 18, 2013.

[21] OMG, *Systems Modeling Language V1.3 (SysML),* Specification doc., http://www.sysml.org/specs/, 2012.

[22] OASIS, *Devices Profile for Web Services Version 1.1,* OASIS Standard, URL: http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf, 2009.

[23] SOA4D Forge, *Service-Oriented Architecture for Devices Project website,* URL https://forge.soa4d.org/ (accessed January, 3.2013), retrieved March 21, 2013.