Service-oriented Process Control with Grafchart and the Devices Profile for Web Services

Theorin, Alfred; Ollinger, Lisa; Johnsson, Charlotta

Published in:
Service Orientation in Holonic and Multi-agent Manufacturing and Robotics: Studies in Computational Intelligence

DOI:
10.1007/978-3-642-35852-4_14

Published: 2013-01-01

Service-oriented Process Control with Grafchart and the Devices Profile for Web Services

Alfred Theorin*, Lisa Ollinger**, Charlotta Johnsson***

*Department of Automatic Control, Lund University, Sweden,
(Tel: +46-46-222 3270; e-mail: alfred.theorin@control.lth.se).
**German Research Center for Artificial Intelligence, Kaiserslautern, Germany
(email: lisa.ollinger@dfki.de)
*** Department of Automatic Control, Lund University, Sweden,
(e-mail: charlotta.johnsson@control.lth.se)

Abstract. To fulfill increasing requirements within the manufacturing sector, highly flexible and adaptable automation systems are needed. It is desirable to have one integrated approach that stretches from the process planning phase, through the implementation phase and all the way to the phase for execution of the process control logics. One promising approach is to use the concepts of service-oriented architectures within automation, here referred to as SOA-AT. As service technology, DPWS has proved to be the most suitable for realizing service based communication on device level. The paper shows how Grafchart, a graphical language aimed for sequential control applications, can support the development of DPWS applications, and how Grafchart can be used for process modeling and execution in the planning and execution phase. This constitutes a unique framework for the development and execution of SOA applications in accordance with the requirements for automatic control tasks. The paper also presents an industry-related experimental setup in which the SOA-AT concepts are demonstrated through the use of Grafchart.

Keywords: Service oriented architecture, Devices Profile for Web Services, Grafchart, Flexible manufacturing systems, Web Services, Graphical languages, Agile manufacturing, Manufacturing control, Process modeling, Production control, Control systems, Automation systems.

1 INTRODUCTION

To fulfill increasing requirements manufacturing companies have to set up and reconfigure their production plants in ever-shorter time intervals and time frames. In parallel, the manufacturing equipment and the control tasks become more complex. To deal with these circumstances highly flexible and adaptable automation systems are needed. Therefore, the automation devices and software should be easy to integrate, configure, extend, and reuse. Today, control architectures comprise several types of automation components that realize different automation tasks. The control of the manufacturing equipment for executing the production process is typically done with

adfa, p. 1, 2011.

a process logic controller (PLC). Usually, the development of PLC processes is based on process diagrams of the planning phase. However, the code is written from scratch because there is no integrated or well-defined information flow between process planning and implementation. Since the process logic and the functionality of field devices are increasing the PLC programs are getting evermore complex. This leads to high efforts for programming, commissioning, and reengineering of control programs.

The increasing demands on automation systems call for advanced automation concepts and technologies that meet todays and future requirements. Component based methods support the handling of complexity and reusability of control programs. To facilitate an integrated information flow the planning phase has to be linked directly to the component based software development. Additionally, technologies are needed to enable a high degree of vertical and horizontal integration of the software components. A promising approach that meets these demands is the paradigm of Service-oriented Architectures (SOA). The potential of applying SOA within the automation domain has already been recognized in several research projects like the SIRENA [1], SOCARDES [2-3], and PABADIS'PROMISE [4] and other publications [5]. However, in practice SOA is still not used for process control applications in factory systems. To make use of the benefits provided theoretically by SOA in real applications, planning methods and technologies for implementing SOA in the automation domain are needed. On this account, the approach SOA-AT (SOA in automation technology) is developed, with the aim to provide methods, models, proceedings, and technologies to support the use of SOA in industrial automation [6]. The work presented in this article is part of this.

In the following, an integrated approach for the planning, implementation, and execution of process control logics by using process models and service-orientation is presented. This allows for the first time to develop and execute control tasks in a service-oriented way with a suitable service technology. First, the conceptual approach SOA-AT is described. After that, the modeling language Grafchart and the tool JGrafchart with the integration of the DPWS technology are introduced. Finally, the concept of combining DPWS and Grafchart for the development and execution of service-oriented control tasks is presented through an industry-related experimental setup.

2 SERVICE-ORIENTED AUTOMATION

2.1 Service-oriented process control

The use of the SOA paradigm for applications in industrial automation has the potential to decrease the engineering effort significantly. The term Service-oriented architectures describes generally a system architecture that represents software functions as encapsulated services in an open and implementation independent way [7]. This enables a high degree of reusability, flexibility, and interoperability of software components. Since the biggest field of application of SOA is enterprise software most definitions and best practices deal with business processes [8-9].

The basic idea of the conceptual approach, SOA-AT, is to apply the principles of service-oriented architectures to the domain of industrial automation [10]. This implies that all automation functions within an industrial automation system are represented as services. Since the focus here is the execution of a production process, the implementation of the process logic should be done in a service-oriented way. The interface between the physical process and the automation system constitute the field devices. Thus, their mechatronic functions are the basic services of SOA-AT. To implement a certain control task these basic services have to be arranged with logical operations. This procedure is also called service orchestration. The encapsulation of low-level functions to services enables a higher degree of abstraction for the implementation of the control logic. This allows a hardware-independent development of the process logic and a simpler programming and reuse of control programs.

2.2 Process modeling and execution

The general procedure for developing an executable process within a SOA is based on a process description. This process description models the planned process in an abstract way. For example, a common modeling language for business applications is the "Business Process Modeling Notation". To generate an executable process the abstract process model has to be transferred to a service orchestration. Therefore, a process logic has to be developed based on the process model and the abstract process steps have to be assigned with existing services, see Fig. 1. An example of an orchestration language is the "Business Process Execution Language" for specifying executable web service orchestrations. How fluently the procedure for generating an executable process is depends on many factors, e.g. how detailed the process description already is, how good the services and the process steps match, how well the data integration works.

The goal is to define an integrated procedure from an abstract process modeling to the generation of an executable service orchestration for the process control of production equipment. The prerequisites are suitable modeling languages and a tool chain without any media breaks.

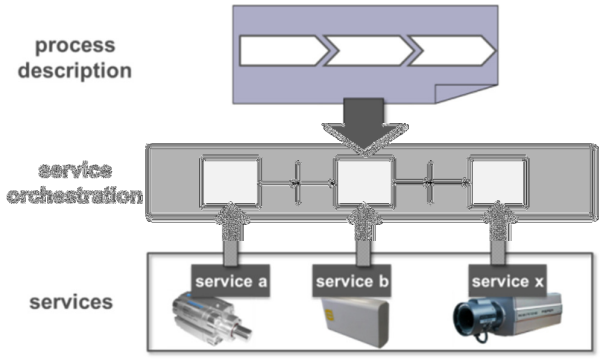

Fig. 1. Generation of a service orchestration.

2.3 Realization aspects

Most of the existing SOA standards, technologies, and tools are tailored to business applications. However, automation and business applications differ in many ways [10]. Thus, an investigation is needed analyzing which existing service technology, process modeling, and service orchestration language are suitable for process control modeling and execution.

Previous work has already evaluated several service technologies [6]. It turns out that the Devices Profile for Web Services (DPWS) proves to meet the requirements best [11]. This is due to the fact that DPWS was developed for realizing web service on resource-constraint devices [12]. Thus, the DPWS defines a profile specifically targeted for SOA at the device level using existing WS-* specifications [1], [12]. Especially the support of eventing and discovery mechanisms makes the DPWS technology attractive. Hence, DPWS is chosen as the service technology. The hierarchy of DPWS services is shown in Fig. 2. On the highest level a DPWS "device" has to be defined. Each device can be discovered by means of the WS-Discovery standard. Furthermore, the devices must provide some metadata to describe themselves more detailed. The underlying DPWS levels with "services", "portTypes", and "operations" corresponds exactly to the structure of common web services. The abstract services that represent the functionality of the field devices are encapsulated to "operations".

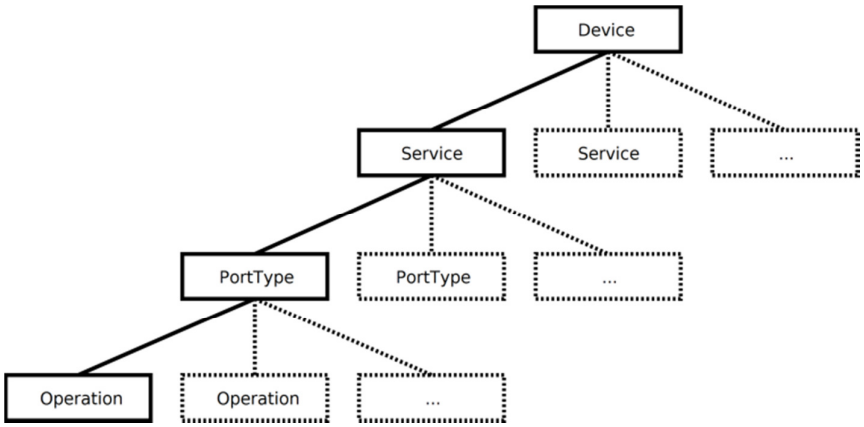

Fig. 2. Hierarchy of DPWS service elements.

A drawback of DPWS is that no standardized modeling or execution languages exist for orchestrating DPWS services. Thus, a framework for the generation and execution of DPWS processes is needed. One of the most important requirements is that the framework should have a high potential of being well-accepted by its future users, i.e. people in the automation domain. Additionally, even high complex logical structures should be presented clearly by means of a graphical representation. A promising candidate to meet these demands is Grafchart [13].

3 GRAFCHART

3.1 Introduction to Grafchart

Grafchart is the name of a graphical language aimed for sequential control applications. Grafchart has been developed at Lund University [14-15]. Graphical programming is popular in the automation community, e.g. three of the five proposed programming languages of the PLC standard IEC 61131-3 are graphical. The advantages of graphical programming languages are simplicity and declarativeness. They often allow programming in a style that closely mimics the style that people model problems. An added benefit is the possibility to use color and animation to provide feedback as the program executes.

Grafchart is based on the graphical syntax of Grafcet/SFC, one of the graphical languages of IEC 61131-3. The syntax of Grafcet/SFC is well-accepted in industrial application today, however the language itself is a rather low-level graphical language. By adding ideas from high-level Petri Nets [16], Statecharts [17], and object-oriented programming, Grafchart is transformed into a high-level, object-oriented graphical language with support for formal analysis [15].

3.2 Syntax of Grafchart

The primary building blocks of Grafchart are steps, representing states, and transitions, representing the change of states. A step and a transition are connected by an arc. Grafchart also supports alternative and parallel branches. An active step is indicated by the presence of a token in the step. An example of a Grafchart application is depicted in Fig. 3. Associated with the steps are actions that are executed at certain occasions, e.g. when the step is activated (S action) or deactivated (X action). To each transition a boolean condition is associated. The transition is enabled when all preceding steps are active. An enabled transition fires if its condition is true, meaning that the preceding steps are deactivated and the succeeding steps are activated.

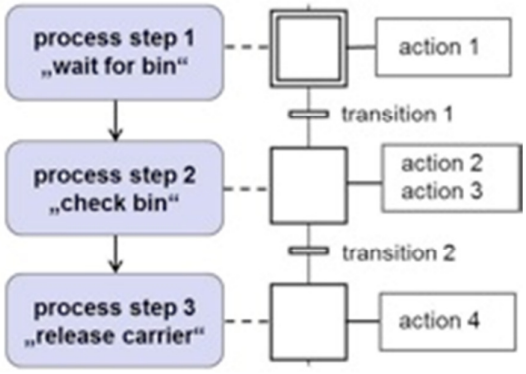

Fig. 3. Process modeling with Grafchart

It is possible to express alternative paths and parallel paths in Grafchart as shown in Fig. 4. Alternative paths means that only one of the possible paths is executed. Parallel paths means that several paths are executed at the same time. The execution is split up and joined with parallel splits and parallel joins respectively.

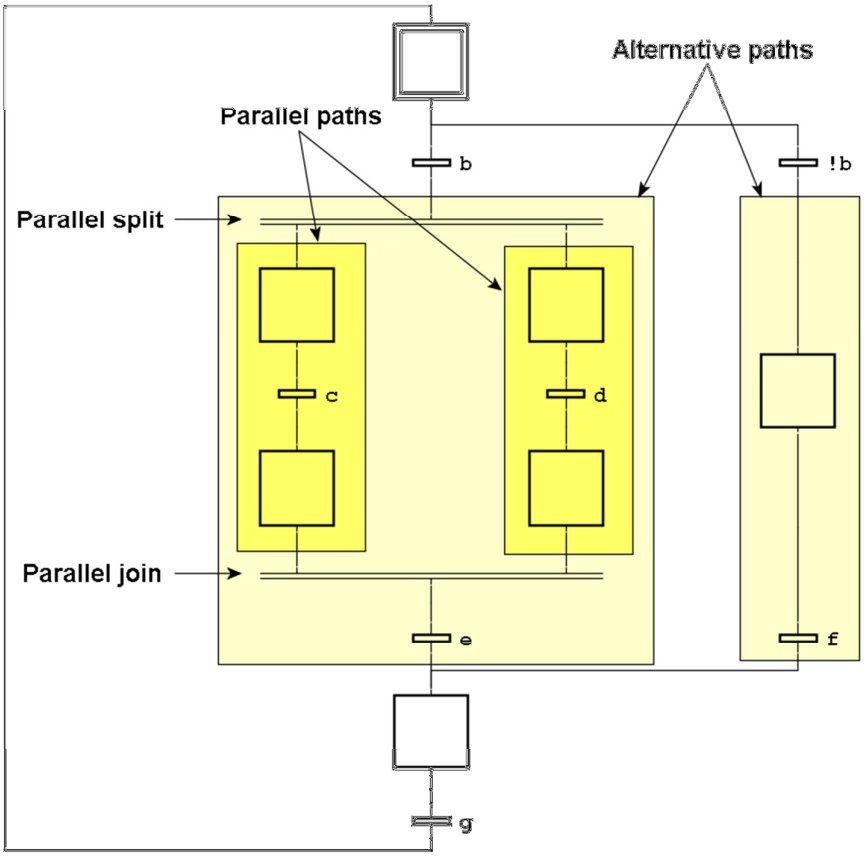

Fig. 4. In Grafchart it is possible to express parallel and alternative paths.

Grafchart supports three hierarchical abstractions mechanisms: macro steps, procedures, and workspace objects, see Fig. 5. Macro steps are used to represent steps that have an internal structure. Sequences that are executed in more than one place in a function chart can be represented as Grafchart procedures. The call to a procedure is represented by a procedure step (procedure call) or process step (separate execution thread). The workspace object is simply a named sub-workspace and is another way to structure large applications. The added features in Grafchart compared with Grafcet/SFC imply that Grafchart is better suited for code reusability, higher abstraction through procedure and process steps, and clarity of processes (Macro steps).

Grafchart also contains constructs for doing more convenient error handling. The exception port on the macro step can connect all the steps in its sub-workspace with a single connection to a special transition, the exception transition, and makes it possible to abort the execution of the macro step. The execution of an aborted macro step can then also be resumed through its history port. The procedure step also has an exception port. There is also the Step Fusion Set which makes it possible to have several steps which are conceptually the same step and thus always are activated and deactivated together.

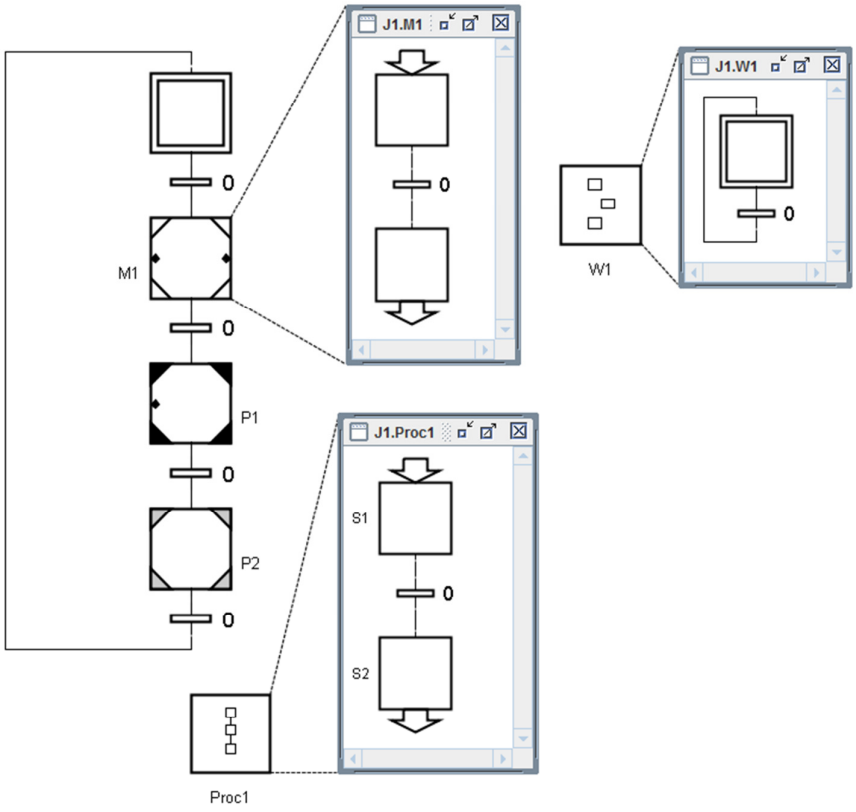

Fig. 5. The ways to do hierarchical structuring in Grafchart: Macro Steps (M1), Procedures (Proc1), and Workspace Objects (W1). Procedures can be used by Procedure Steps (P1) and Process Steps (P2).

3.3 Modeling service orchestrations with Grafchart

Since Grafchart combines the well-known graphical syntax of established process control languages and high-level modeling features it is entirely suited as a formal

language for the mentioned demands. In particular, the possibility to create different abstraction levels by means of encapsulation of sub-processes and object-orientation is an important fact. Due to this service orchestrations can be modeled in various abstraction degrees so that both the process description and the service orchestration can be done with the same language. This enables a top-down engineering procedure without any media breaks.

For generating the process description the process steps can be represented as Grafchart steps, see Fig. 3. During the planning phase the process description can be detailed by decomposing the processes steps by means of the mentioned high-level features of Grafchart. For the development of executable control logic the abstract process description has to be transferred to a service orchestration. Therefore, the process steps have to be enriched with actions and the transitions with conditions. In a last step, the actions and transition conditions have to be implemented by assigning them with existing services. By this procedure the development of control software can be done hardware-independent for the longest time.

3.4 JGrafchart

For realizing this integrated engineering procedure a tool for modeling and executing Grafchart applications with DPWS services is needed. A Java implementation of Grafchart called JGrafchart is developed by the department of Automatic Control at Lund University and is available as freeware [18].

JGrafchart already contains several means of connecting various I/O. For example it is possible to supply completely custom made Java implementations of analog and digital I/O. Using only these it is not possible to make a good DPWS implementation since they only allow boolean and real values to be read and written from a JGrafchart application while DPWS sends XML messages. Something as simple as passing on a returned string from one operation to another operation would be very hard to accomplish.

It is also possible to create more general I/O with sockets. JGrafchart then connects as a client to a TCP/IP server, sends TCP/IP messages to the server each time a Socket output value is changed, and updates the corresponding Socket inputs when a message is received from the server.

4 DPWS INTEGRATION IN JGRAFCHART

4.1 The socket I/O prototype

Calling DPWS operations from JGrafchart was initially prototyped using the already existing Socket I/O. To do this a TCP/IP server was implemented to translate the assignments to socket outputs in JGrafchart to DPWS operation calls, as well as operation responses to socket inputs in JGrafchart, see Fig. 6. Some special socket inputs and outputs as well as some extra code to detect event arrival were also needed for subscriptions and event notifications.

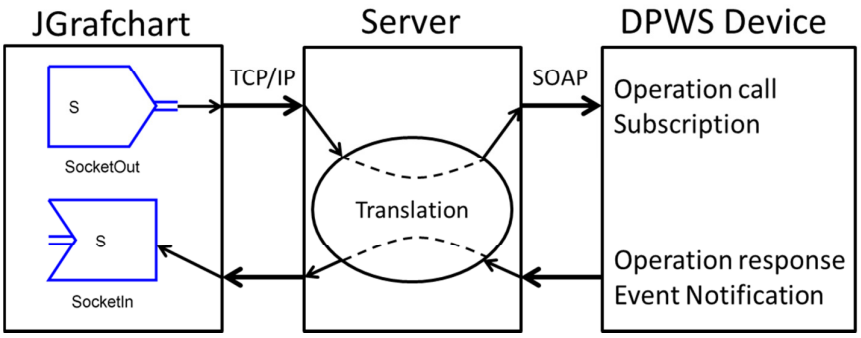

Fig. 6. Overview of solution using the already existing socket I/O.

A big advantage with this prototype is that an almost unmodified version of JGrafchart was sufficient. For each device that you want to use you add a small piece of translation code in the server. The only required modification to JGrafchart was always sending a message to the server upon assignment to a socket output. Previously a message was only sent if the value had changed and since assignments correspond to DPWS operation calls this meant that consecutive calls with the same arguments were not made.

One possible improvement of the prototype would be to make the translation more generic, replacing the need to write specific code for each device by configuring the mapping from DPWS operations to socket inputs/outputs. Together with a library in JGrafchart for event notification this would become rather convenient.

However, a problem with the prototype is that it is hard to make calls to request-response operations synchronous. When JGrafchart has written to a socket output, which is related to invoking of a request-response operation, it does not know that it should wait for the update of a socket input before resuming the execution. Apart from this there is also the aesthetical issue that operation calls look like assignments and returned values are fetched from a separate socket input.

Our conclusion from making this prototype is that it was a surprisingly small effort to get it working, requiring only a tiny modification to JGrafchart. We are quite confident that this approach can also be used for other, similar methods of communication.

4.2 Using DPWS services

In JGrafchart version 2.1.0, a generic DPWS implementation has been integrated directly into JGrafchart using the DPWS4J toolkit [19]. A DPWS service port type is connected to the new DPWS Object in JGrafchart. Using the capabilities of DPWS, existing services as well as service startups and shutdowns are automatically detected and the services are automatically rebound to the corresponding DPWS Objects. Since the services are self-describing it is also possible to check that operation calls are at least well formed at compile time. The WSDL files may also contain documen-

tation that can be, and is, displayed to the user directly in JGrafchart. It is also possible to view the raw WSDL which can be useful for example if the WSDL has insufficient documentation.

WSDL specifies four kinds of operations; one-way, request-response, solicit-response and notification. The one-way and request-response types are initiated by JGrafchart. Request-response calls have a return value and consist of both a message from JGrafchart to the service and a message back from the service to JGrafchart. One-way calls only consist of a message from JGrafchart to the service. Symmetrically notification calls only consist of a message from the service to JGrafchart. This is typically used for eventing. Solicit-response calls work like request-response calls but the other way around. Solicit-response calls are not supported by JGrafchart as they are considered rare in automation.

Calling of DPWS service operations has been designed to look like any other method call in JGrafchart. Fig. 8 contains examples on how the operations defined in Fig. 7 are used. The one-way operation oneWayOp is called on the service port type myPortType bound to myDPWSObj. In this case the operation does not require, nor allow, any parameter, as the corresponding message definition does not specify any parts. The request-response operation reqRespOp on the other hand requires a string as a parameter, in this case "par" is sent, and it will return a boolean which in this case will be assigned to the JGrafchart variable ret. Since the port type specifies that it is an event source it is possible to subscribe to events. On the first line a 10 minute subscription is initiated and on the last two lines a check is done if any eventOp notifications have been received, and if so the oldest event is fetched and stored it in the integer variable ev.

```
...
<message name="oneWay"/>
<message name="req">
      <part name="arg" element="string"/>
</message>
<message name="resp">
      <part name="ret" element="boolean"/>
</message>
<message name="event">
      <part name="id" element="integer"/>
</message>

<portType name="myPortType" EventSource="true">
      <operation name="oneWayOp">
            <input message="oneWay"/>
      </operation>
      <operation name="reqRespOp">
            <input message="req"/>
            <output message="resp"/>
      </operation>
      <operation name="eventOp">
            <output message="event"/>
      </operation>
</portType>
...
```

Fig. 7. A WSDL example where three operations are defined, namely oneWayOp, reqRespOp, and eventOp.

```
S myDPWSObj.oneWayOp();
S ret = myDPWSObj.reqRespOp("par");
S dpwsSubscribe(myDPWSObj, "PT10M");
...
S e = dpwsHasEvent(myDPWSObj, "eventOp");
S ev = e ? dpwsGetEvent(myDPWSob, "eventOp") : 0;
```

Fig. 8. Grafchart actions calling all operations in Fig. 7.

4.3 Example

The DPWS4J toolkit v1.3.0 includes a sample of a lamp controller implemented as a DPWS service. If you download the toolkit and JGrafchart you can try this example out by yourselves.

Consider motion triggered lighting of a room. After any motion, the room shall be lit for two minutes. The lamp controller is implemented using the same interface as the sample and the motion sensor is implemented as a digital I/O.

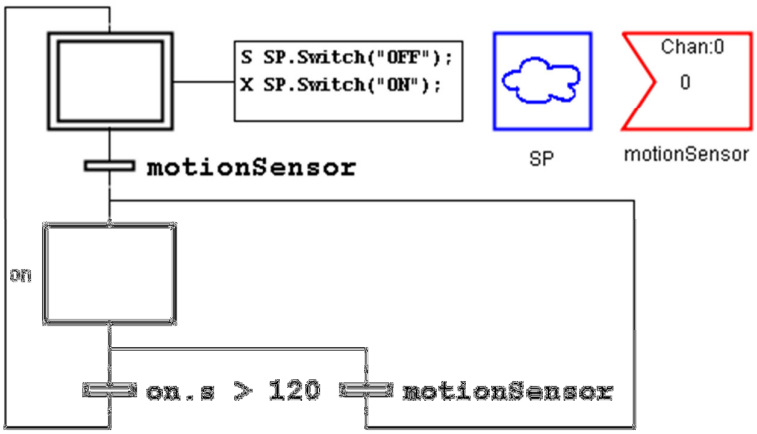

Fig. 9. A Grafchart application for motion triggered lights.

A JGrafchart based implementation could look like in Fig. 9. The lights interface contains a port type named SwitchPower which here is bound to the DPWS Object SP. The operation Switch in the SwitchPower port type accepts the parameter values "ON" and "OFF" which turn on and off the lamp respectively.

The JGrafchart application starts in the initial step where the lamp is initially turned off. When motionSensor is true, the initial step will be deactivated which means that the lamp will be turned on. The construct <stepName>.s returns the number of seconds that the step named "stepName" has been active since the last activation. The bottom right transition makes sure that this counter is reset whenever motionSensor is true.

5 EXAMPLE

To evaluate and illustrate the DPWS implementation, a service orchestration in JGrafchart that controls real production equipment is implemented. The equipment setup represents a small flexible and agile manufacturing system.

5.1 Experimental setup

The experimental setup is part of the demonstration facility of SmartFactoryKL and comprises real industrial devices. It consists of a conveyor belt transporting carriers with bins that shall be filled with a certain number of pills. There are two stations on the demonstrator, one that fills the bins with pills, and one that checks the quality of the filled bins. The latter simply checks if the bins have been filled with the correct number of pills by image recognition. On each bin there is an RFID tag containing life cycle information about the product [20]. Amongst other things it saves some information about the production process, e.g. how many pills that the bin should contain, if it has been filled, and if quality control has been performed.

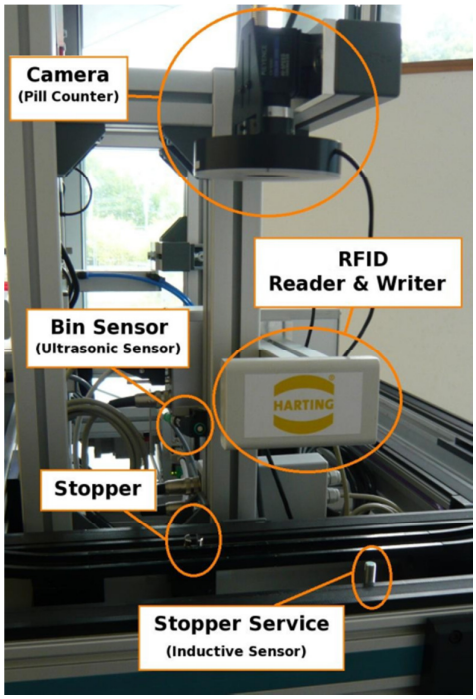

Fig. 10. The quality control station of the demonstrator.

The quality control process in Fig. 10 consists of five devices; an inductive sensor that detects the arrival of a carrier, a stopper that can stop the carriers, an ultrasonic sensor

that can check if there is a bin on the carrier, an RFID reader that can read from and write to the RFID tag on the bin, and a camera that can take top view pictures of the contents of the bin. In previous work the devices of the quality control process have been enhanced with microcontrollers that serve as service gateways. The basic functions of the devices have been encapsulated and implemented as DPWS services [10]. The sequence for coordinating the station can be modeled as in Fig. 11.

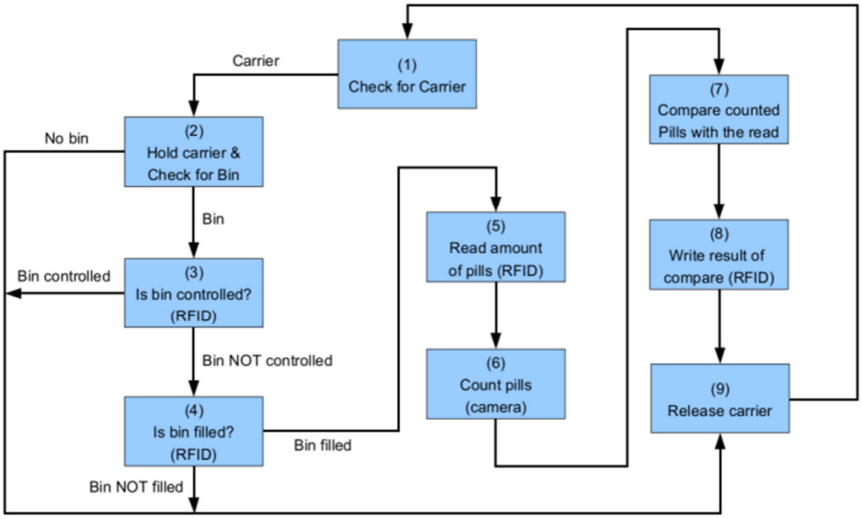

Fig. 11. A model of the coordination sequence for the quality control station, where state (1) is the initial state

5.2 Process execution with JGrafchart

Using the model as a basis, a JGrafchart application for coordinating the quality control station is created, see Fig. 12. As some states in the model have a straight forward flow, they can be implemented in the same Grafchart step. The steps CheckBinRFID and QC in JGrafchart correspond to the model states (3)-(4) and (5)-(8) respectively.

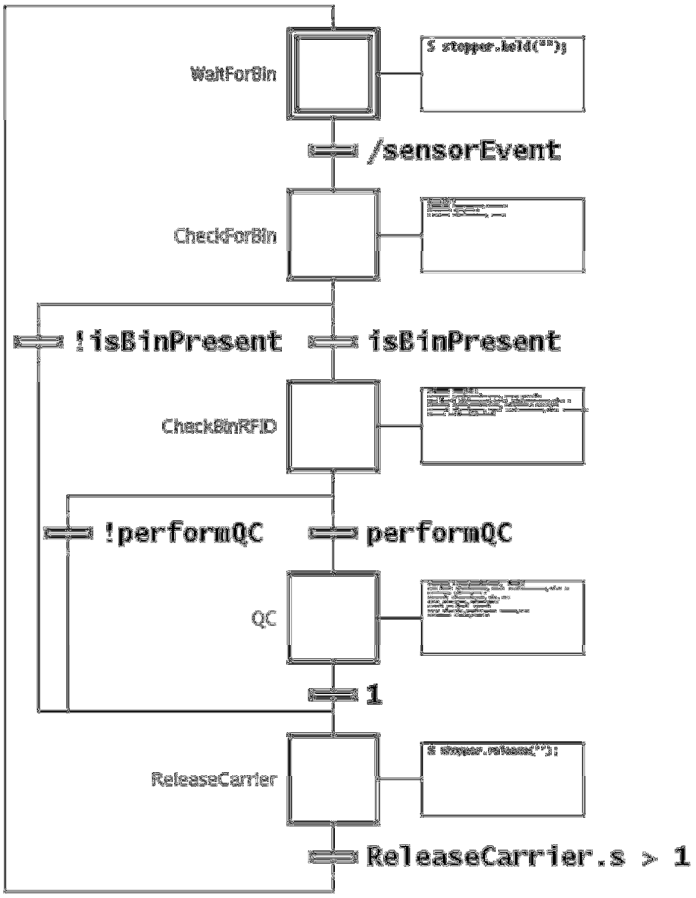

Fig. 12. A JGrafchart implementation of the quality control station. The code is shown to highlight the approximate amount of code that is required; it is not intended to be readable.

XML utility functions are used to simplify the code, e.g. xmlFetch is used to obtain a derived value from an xml string. The camera's count operation returns a sequence of value elements, where each element describes the number of pills of a specific color. The total number of pills is fetched with xmlFetch(resp, "value", "sum"); where resp is the returned string, "value" is an XPath that selects all elements with the tag name value, and "sum" is a built-in handler that calculates the arithmetic sum of the selected elements' texts.

6 CONCLUSIONS

Service-oriented architectures constitute a powerful concept to improve industrial automation systems regarding the flexibility, integration capability, and re-usability of their devices and software. However, the effective use of SOA in automation application depends heavily on how well the concept can be realized with existing tools, technologies, and engineering proceedings. Therefore, an integrated procedure from the process planning to the operation phase is presented. Grafchart is used as the process modeling and service orchestration language and DPWS as the service technology.

Using the basic concepts of SOA-AT, together with the DPWS service technology and the sequential language Grafchart, three main advantages are achieved; 1) the development and modeling of elaborated processes can be made independently of the implementation of the process control logic which is vendor and hardware dependent, 2) the language used for modeling of elaborated processes can also be used for execution of the same processes, 3) the coupling to the services is made in a simple and straight forward way using the DPWS technology.

The focus of the work has so far been on integrating DPWS in JGrafchart. Version 2.1.0 of JGrafchart can be used for realizing DPWS service orchestrations and is freely available at http://www.control.lth.se/Research/Tools/grafchart/.

Future plans include linking the process implementation in Grafchart to previous factory planning phases. Another future research area is the realization of services for production equipment. Concepts are needed for defining service for the different automation tasks. Furthermore, technological questions have to be answered, e.g. demands on the SOA communication system like real-time, security, and safety issues and how industrial devices can provide the computational power and networking capacity.

7 REFERENCES

1. Jammes, F., Mensch, A. & Smit, H. (2005). Service-oriented device communications using the Devices Profile for Web Services. In *Proceedings of the 3rd international workshop on Middleware for pervasive and ad-hoc computing*. Grenoble, France.
2. Souza L, et. al. (2008). SOCRADES: A Web Service based Shop Floor Integration Infrastructure. *In Proceedings of the Conference Internet of Things 2008*. Zurich, Switzerland.
3. Kirkham, T., et. al. (2008). SOA middleware and automation: Services, applications and architectures. In *Proceedings of the Conference of Industrial Informatics 2008*. Daejon, Korea.
4. PABADIS'PROMISE Consortium (2008). *Structure and Behaviour of a PABADIS'PROMISE System.* White Paper. http://www.uni-magdeburg.de/iaf/cvs/ pabadispromise/dokumente/whitepaper2_v60.pdf. Retrieved November 05, 2011.

5. Mersch, H., Schlutter, M., and Epple, U. (2010). Classifying services for the automation environment. In *Proceedings of the Conference on Emerging Technologies and Factory Automation (ETFA 2010)*, Bilbao, Spain.
6. Ollinger, L., Schlick, J., and Hodek, S. (2011b). Konzeption und praktische Anwendung serviceorientierter Architekturen in der Automatisierungstechnik. *In VDI-Berichte 2143. AUTOMATION 2011*. Baden-Baden, Germany.
7. Melzer, I. (2008). *Service-orientierte Architekturen mit Web Services: Konzepte – Standards – Praxis.* Spektrum Akademischer Verlag, Heidelberg, Germany.
8. Krafzig, D., Banke, K., and Slama, D. (2004). *Enterprise SOA: Service-Oriented Architecture Best Practices*. Prentice Hall, Upper Saddle River, New Jersey, USA.
9. Bieberstein, N., et. al. (2005). *Service-Oriented Architecture Compass: Business Value, Planning, and Enterprise Roadmap.* Prentice Hall PTR. Upper Saddle River, NJ, USA
10. Ollinger, L., Schlick, J., and Hodek, S. (2011a). 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.
11. OASIS Web Services Discovery and Web Services Devices Profile (WS-DD) TC (2009). *OASIS Devices Profile for Web Services (DPWS) Version 1.1. Oasis Standard.* http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf. Retrieved November 05, 2011.
12. Zeeb E., et. al. (2007). Lessons learned from implementing the Devices Profile for Web Services. *In Proceedings of the Conference Digital EcoSystems and Technologies Conference* 2007. Cairns, Australia.
13. Johnsson, C. and Årzen K.-E., (1998): Grafchart Applications. In *Gensym User Society Meeting*. Baltimore, MD, USA, 1998.
14. Årzen, K.-E. (1994): "Grafcet for intelligent supervisory control applications." *Automatica*, 30:10.
15. Johnsson, C. (1999): *A Graphical Language for Batch Control.* PhD thesis ISRN LUTFD2/TFRT--1051--SE. Dept. of Automatic Control, Sweden.
16. Jensen, K. and G. Rozenberg (1991): *High-level Petri Nets.* Springer Verlag.
17. Harel, D. (1987): "Statecharts: A visual formalism for complex systems." *Science of Computer Programming*, No 8, pp. 231–274.
18. Årzen, K.-E. (2002): "Grafchart: Sequence Control and Procedure Handling in Java." In *Reglermötet, 2002*. Linköping, Sweden.
19. DPWS4j toolkit: DPWS4j toolkit webpage https://forge.soa4d.org/projects/dpws4j/ Retrieved November 05, 2011.
20. Stephan, P. et. al. (2010). Product-Mediated Communication through Digital Object Memories in Heterogeneous Value Chains. *In Proceedings of the Conference on Pervasive Computing and Communications (PerCom 2010)*, Mannheim, Germany.