Service-orientation and Flexible Service Binding in Distributed Automation and Control Systems

Andre Pohl and Heiko Krumm

University of Dortmund, Germany
(pohl, krumm)@ls4.cs.uni-dortmund.de

Felix Holland, Ingo Lück
and Franz-Josef Stewing
Materna Information & Communications
(felix.holland, ingo.lueck,
franz-josef.stewing)@materna.de

Abstract

An experimental study shows the feasibility of service-oriented architectures for industrial automation and control systems even with respect to lower, real-time dependent control functions. For that purpose, general SOA-guidelines were refined in order to cover the distribution of control functions between services and the lay-out and management of device-based sensor, actor and control services. Particular emphasis was placed on the dynamic lease-based binding of services which on the one hand provides flexible and loose coupling of system components but on the other hand has to ensure reliable communication and cooperation. The guidelines were applied to the experimental implementation of a manufacturing cell control system using a real-time version of the Java Runtime Environment. The Device Profile for Web Services (DPWS) was used as basic infrastructure technology. Test and evaluation were performed under distributed simulation of technical processes and devices.

We shortly describe DPWS, present the architecture guidelines, outline the experimental control system implementation, and report on its evaluation.

1. Introduction

Today, many modern business applications adhere to the paradigms of service orientation and service oriented architectures in order to create loosely coupled, modular software systems, easy to maintain and to extend. In the field of automation and control systems, SOA-based flexibility is of even more interest, because it contributes to substantial reductions of installation and setup costs [1]. These costs are of particular importance since manufacturing plants again and again have to be adapted to new products resulting in changes of the technical equipment and the process flows performed. Additional reconfigurations are applied occasionally in the course of repair measures in order to bypass defect equipment and to avoid expensive production downtimes.

Despite the desired flexibility, however, there is a needs for stable and reliable operation phases since the efficiency of the production equipment usually depends on steady operational conditions. For a certain manufacturing operation usually an ensemble of suitable devices, machines and transport equipment is necessary. The members of the ensemble must initially be configured in harmony with each other and thereafter be available for a certain minimal period of operation time, which may only be aborted due to exceptional circumstances. The members of the ensemble have to be allocated before configuration, some of them because they can only be used exclusively, others may be sharable but have to allow for the additional load.

In the service-oriented setting this means, that a client – which may be either a control application or a compound service – must be able to search, find and allocate a suitable ensemble of used services. Since a used service may already have other obligations, it may not be disposable and deny a current allocation request. Then, one member of the planned ensemble fails, and the ensemble as a whole is currently not useful. Therefore, the client shall be able to withdraw the other allocation requests and look for alternative ensembles. In order to fulfill these functional requirements of temporary and atomic ensemble allocation we extended the approach of lease-based allocation [2] by introducing an explicit reservation phase in a way that reservation and allocation perform a two-phase commitment.

Moreover we transposed the architecture of hierarchical control systems to the field of service systems using the platform the Device Profile for Web Services (DPWS) as basic infrastructure technology

supporting the communication between devices via service interfaces as well as the exploration and binding of services. The application of the resulting architecture guidelines and the usage of the lease-based allocation were exemplified by means of a production cell scenario using a real-time Java Runtime Environment.
In the sequel, we outline DPWS and its application to service-oriented industrial applications. Referring to the general structure of automation and control systems, the architectural principles of service-oriented control systems are described. Sect. 5 introduces the flexible, but reliable lease-based service binding. Sect. 6 presents the application scenario and reports on its implementation. Sect. 7 shortly enters into validation aspects before concluding remarks close the contribution.

2. Service Oriented Architectures

In SOA, interoperability of different platforms is established through the definition of common communication protocol and message exchange standards. But not only in enterprise domain software service-orientation is a feasible way of creating flexible software systems, as through the growth of computing power of embedded devices these paradigms are also applicable to embedded software solutions. *Universal Plug'n'Play (UPnP)* [3] was the first specification of a service oriented infrastructure to be used in embedded application scenarios, using SOAP and HTTP as a basic communication layer and providing mechanisms for service discovery, action invocation and event based communication schemes. Its successor, the *Devices Profile for Web Services (DPWS)* [4], is completely based on standardized Web service specifications and defines a profile (a subset) for the use of Web service technology in the embedded domain.

2.1. Devices Profile for Web Services

The Devices Profile for Web Services defines a common subset of web service based communication patterns for use in embedded devices. The protocol stack utilizes standardized internet protocols, namely TCP/IP and UDP (Single- and Multicast). For basic messaging HTTP and SOAP respectively SOAP-over-UDP are employed. On top Web service protocols are arranged that deal with service and device description, discovery, eventing and security. A DPWS device may host several services, which can be discovered and used by DPWS clients. The DPWS protocol stack is depicted in Figure 1.

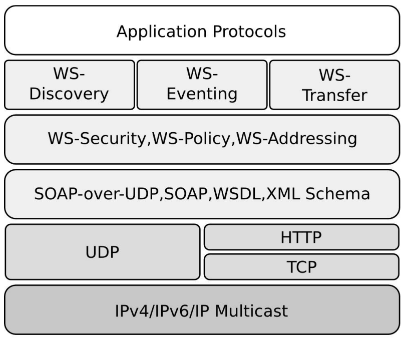

Figure 1. DPWS protocol stack

2.2. SOA in Industrial Automation

The emergence of powerful but less power consuming, affordable, and embedded computing components facilitates the employment of SOA paradigms even in the world of industrial automation. Currently a lot of proprietary standards in device control and communication protocols often prevent the interoperable use of components from different vendors. Thus upgrades or extensions of the manufacturing automation system tend to be costly and time consuming [1]. The usage of SOA in industrial automation provides a common ground for interoperability of all devices in a device network. Moreover an integration of low-level devices and high-level enterprise applications (e.g. an ERP system) is possible. In the European ITEA SIRENA [5] project the applicability of DPWS in an industrial automation scenario was demonstrated for the first time.

3. Automation and Control

An industrial control system commonly has a structure as depicted in Figure 2. This architecture could be divided into three main layers: sensors and actors, control and management.
The actual technical process is located at the bottom of the control hierarchy and subsumes all technical low-level components involved in the production process like motors, pushers or drilling machines. The process is monitored by sensors, collecting data from the involved resources including e.g. temperature, rpm or the position of work pieces (indicated by a light-barrier state change). This information is send via a specialized communication infrastructure to the process control level and is repeatedly evaluated by the control algorithm. Based on the sensor information the control algorithm computes control signals which are in turn send to the actuators connected to the technical process. Moreover status information from the process control level is sent to the process management level. This may include forwarded sensor values, progress

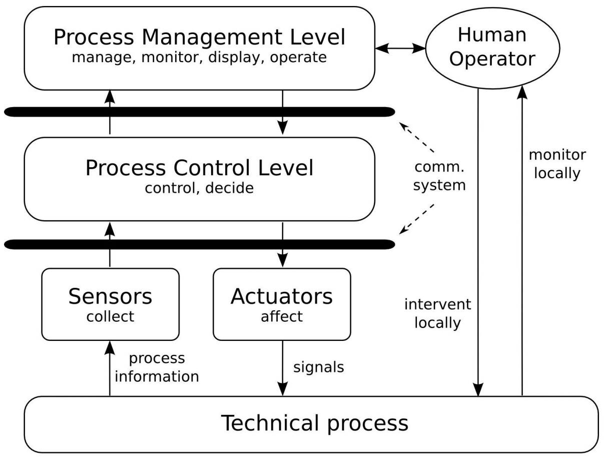

Figure 2. Control system

information and fault messages. At process management level a human operator monitors the overall process behavior, adjusts particular parameters and sends configuration commands to the process control system. Besides the remote high-level controlling and monitoring of the technical process, in some occasions (e.g. a severe fault that requires local intervention and repair) the operator may be forced to directly intervene with the low-level hardware components via the attached control panel.

4. Service-oriented Control Architecture

The process control architecture shown in the last paragraph is the structural basis for the service-oriented architecture presented in this paper. The service-orientation of the devices involved in the technical process and the attached sensors suggests the use of service-orientation also on the control and management levels. The sensors and actuators export their functionality through defined interfaces which can be used by higher level control services. Control services may also be layered and arranged in a service hierarchy. Figure 3 illustrates this architecture: the application process interacts with the technical process using the supplied control services. The control services themselves are acting both as a service consumer (client role) and service provider (server role) and thus enable control service layering.
For example, a rotary disk consists of a rotation motor and a motor for moving the conveyer belt on top of the disk. Additionally the disk is supplied with sensors, detecting the location of the work piece currently transported on the conveyer belt and a sensor to measure the position of the rotary disk itself. Both, the rotary part and the transportation part are each controlled by their own control service. For the control of the overall process of moving a work piece on the

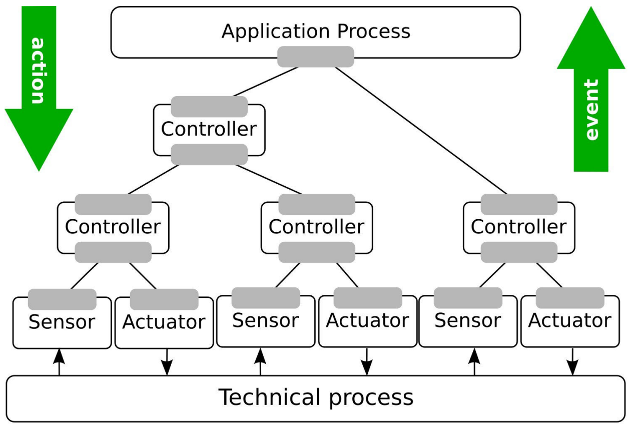

Figure 3. Service hierarchy

disk, stopping the conveyer, turning the disk to its new position and finally transporting the work piece away from the rotary table, an additional control service is provided that uses the control services of the particular parts of the rotary disk. Therefore the control services themselves offer service functionality to higher level control or management services. However, the stacking of control services is constrained by the real-time requirements of the process, as each new layer of control implies additional, time consuming communication between the services.
The services (e.g. sensor or controller) offer different interfaces which can be categorized using the following three classes:

- functional purpose
- discovery and description
- service binding

The *functional* interface offers the functionality of the service, e.g. a *getVariable* method for sensor or a *setVariable* method for actuator services. The functional service interface of control services offers high-level methods like *drillHole*.
The control services comply with the notion of so called *function building blocks* (IEC 61499). Each building block comprises input and output variables plus local status variables. The functionality of a particular function block is defined by the algorithm that is used to compute the outputs by using the inputs and the local variables.
The *discovery* interface contains the necessary methods for services to be able to answer to search requests and to provide data concerning device type, location and binding address. Finally, the *binding* interface subsumes the features for lease based service binding and reservation.

5. Flexible Service Binding

One of the key features of service-orientation is the use of loosely coupled components. As all devices, sensors and actuators provide a service interface the coupling of components can correspond to the flexible binding of services.

This flexible binding of services demands for

- service description, discovery and selection, and
- service association and linking mechanisms.

The service *description* subsumes three basic parts:

- Type and interface definition,
- Binding and communication information,
- Functional properties.

The *type and interface definition* of a service specifies the methods and parameters associated with a specific service type. All services that comply with a specific service type offer the same interface. The *binding and communication* information contains information about the actual communication endpoints and the basic communication mechanisms, such as IP addresses and ports, and application protocol regulations. At last, the *functional* properties complete the information on devices in the automation system. They e.g. include, which sensor is attached to which conveyer and what is the exact position.

The service description is the basis for the discovery and selection of matching services by the automation process and control services. In our system, the discovery and description phase are based on DPWS technology and thus adhere to the WS-Discovery and WS-Transfer (for metadata exchange) standards.

The *association and linking* of matching services with a particular client is handled by our *lease based binding* approach to meet the requirements of a flexible but also stable way for dealing with loosely coupled services in the domain of industrial automation.

The notion of a *lease* was first introduced by [2] and was used to provide an efficient, fault tolerant way for using file caches in distributed environments. Further on leases were used in *Jini* [6] to grant clients access to network services. In the case a client wants to use a particular service, it issues a lease-request which contains a duration for which the client wants the lease to be valid. The service responds with a denial or a grant. A granted lease is valid only for the duration. Thus the client has to request another lease for service

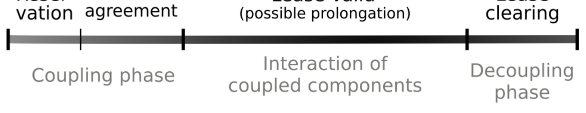

Figure 4. Lease lifecycle

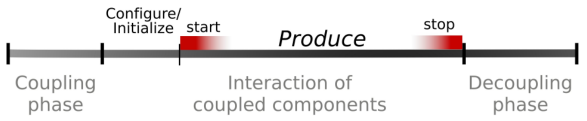

Figure 5. Leases and production

use after the current lease has expired or may prolong it before its valid duration has passed.

In automation systems a client usually uses a set of services (sensors, actuators, and controllers) and has to allocate a suitable ensemble. Therefore we extend the lease model by adding support for the atomic allocation of service ensembles. The atomicity property guarantees that a client either is granted the leases for all requested services or it gets no lease at all. This atomicity is achieved by a 2-phase algorithm, which is similar to the 2-phase-commit protocol. It is a lease granting algorithm with explicit reservations (cf. Figure 4).

During the *coupling phase* the client asks the suitable services for reservations. Reservations are binding for a short duration. If all services positively respond, the client submits lease-requests that yield to valid usage leases. If at least one service cannot satisfy the reservation request, the client cancels all other reservations.

After the coupling phase is completed, the *interaction of coupled components* starts. The client process configures and initializes the services and finally starts production (cf. Figure 5). When the leases are about to expire, the client either issues a prolongation request to extend the production phase or stops the services and performs cleanup operations. The prolongation of existing leases uses the same 2-phase algorithm as used at initial lease creation. In the *decoupling phase* the expired leases are fairly released and deleted.

6. Application Example

The service-oriented control software presented so far was experientially evaluated for an example industrial automation setup. The example system and the tested applications scenarios are presented in this section.

6.1. Example Structure

The structure of our evaluation example is depicted in Figure 6. The work pieces enter the system through conveyer *conv1* and *conv2*. Both conveyers are located next to a *rotary disk*, which is able to collect work pieces from either conv1 or conv2 by rotating the disk and using the conveyer element *conv3* on top of the disk. This conveyer transports the work pieces to conveyer *conv4* which in turn moves them through the *lacquer machine*. After being painted by the lacquer machine, the work pieces are checked by a laser sensor.

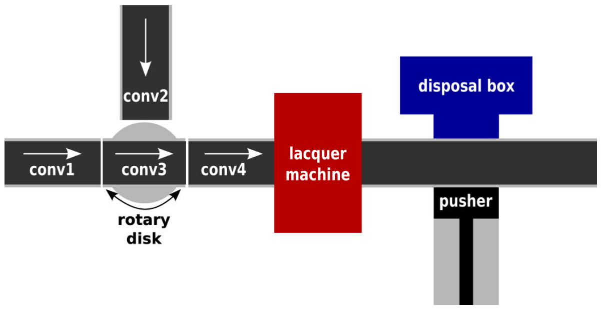

Figure 6. Example system

Inaccurate pieces are pushed into a *disposal box* by a *pusher*. Proper items are moved out of the system to the next work station. The devices and sensors (not depicted) are exporting services as described in section 3. The logical control of the conveyers is implemented using a PID controlling algorithm (similar to [7]), which could be differently parameterized for evaluation purposes.

6.2. Application Scenarios

The example system was evaluated using different application scenarios. The scenarios use different service hierarchies and thus model different levels of control in the application process.

The first scenario comprises the following process:

1. Work pieces are picked up from *conv1* or *conv2*.
2. The *rotary disk* and *conv3* transport the work pieces to *conv4*.
3. The *lacquer machine* paints the work pieces.
4. The inaccurate work pieces are detected and pushed into the *disposal box*.
5. The acceptable work pieces are moved out of the system.

The service hierarchy for this application process is depicted in Figure 7. The application process uses six different control services (light gray), each responsible for a specific part of the example system. The control services themselves are using a set of sensor and actuator service interfaces to interact with the hardware at technical process level (dark grey). In contrast, the *rotdisk* control service for controlling the rotary disk and *conv3* on top of the disk as a whole uses the

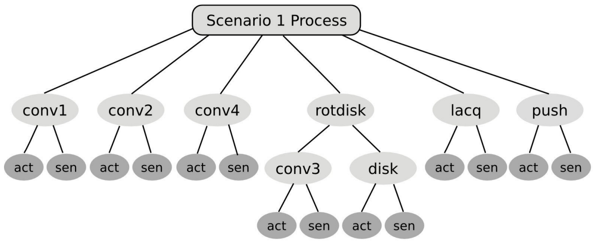

Figure 7. Scenario 1 service hierarchy

control services of the single components. It implements an algorithm for the balanced use of the two attached input conveyers.

The second application scenario uses only the input conveyer *conv1*, thus the usage of the *rotdisk* control service is not necessary:

0. Statically move the rotary disk in *conv1-conv4* position using the *disk* service.
1. *Conv1* transports the work pieces to *conv3*.
2. *Conv3* forwards the work pieces to *conv4*.
3. Conveyer *conv4* moves the pieces through the *lacquer machine*.
4. The *pusher* sorts out erroneous pieces.
5. Acceptable items leave the system.

The service hierarchy used for the second scenario is depicted in Figure 8.

The application process of the second scenario uses seven control services. The sub-component services of the rotary disk now are directly used to initially set up the right direction of the disk and to control the *conv3* at runtime. This change in the process outline does not infer changes in the service implementations of the devices used.

Further scenarios were used to evaluate the applicability of multiple application processes, each controlling a part of the overall process.

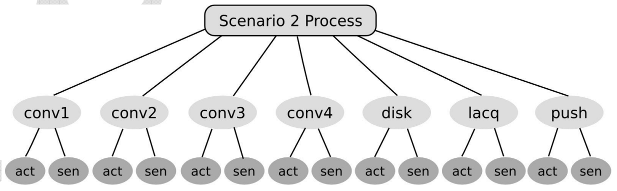

Figure 8. Scenario 2 service hierarchy

7. Evaluation

The evaluation environment comprises three major components: the DPWS stack, the Java Real-time VM and the simulation system.

The *WS4D.org DPWS stack* [8], developed by Dortmund University and Materna, is a Java based implementation of the DPWS protocol stack and provides a service oriented communication infrastructure. It was developed with modularity and extensibility in mind and thus can be adapted to varying application scenarios, ranging from small client-only implementations for mobile phones to multimedia or file-sharing services for embedded set-top boxes.

The Java Real-time System [9] comprises technologies and concepts for correct reasoning about the timing of Java real-time applications. It contains new types of real-time threads, memory handling schemes (e.g.

preventing the garbage collector from influencing the runtime behavior in a nondeterministic way), high precision timers with nanosecond resolution and direct memory access for implementing device drivers purely in Java. Nevertheless, the Java RTS depends on the real-time capabilities of the underlying operating system.

For evaluation purposes we developed a testing environment, split into two blocks: a simulation system and the sensor, actuator and control service implementations. The time discrete simulation system is composed of four major components. The *simulation model* component manages a *grid model* for locating devices, sensors and work pieces in the system and a *component model* for preserving the state of the simulated components. The *simulation control* component periodically updates the model information. Changes in the internal state of sensors and actuators are sent to and received from the distributed components via an *UDP based communication protocol*. It was especially designed to consume few network bandwidth. A *graphical user interface* is used to track and control the simulation.

The simulated system comprises sensor, actuator and control service implementations. The sensor and actuator implementations are connected to the simulation system via the UDP based communication protocol (s.a.) to receive and publish state information.

The simulations were run on an Athlon64 X2-3800 machine with two GB of memory and an OpenSolaris installation as basis for the Java RTS.

7.1. Results

A series of experiments focused on the evaluation of the functional behavior of the control system. Particular test sequences checked the feasibility and stability of the lease-based allocation. Atomic allocation and setup of service ensembles were as well tested as atomic lease prolongation and occasional aborts followed by the searching and switching to alternative ensembles.

In the course of additional experiments the service call roundtrip times (using simple input and output parameters) were measured in order to check the current real-time limits of Java VM and DPWS based control system implementations. Table 1 presents the values obtained for local VM-internal (on the OpenSolaris host) and for remote DPWS-based service calls (between the OpenSolaris and the PC host). The configuration was able to support low to medium real-time requirements (e.g. cycle times >50ms). The application for fast control processes is not yet advisable, mainly because of the high roundtrip time variance observed. In the remote case, it stems from the delay variance of Ethernet frames. The variance observed in the local scenario shows that the integration of the real-time Java VM into the operating system has to be improved further.

	remote call (ms)	local call (ms)
maximum	70,02	0,1666
minimum	8,21	0,0069
mean	11,30	0,0127
median	9,46	0,0129

Table 1. Action call roundtrip times

8. Concluding Remarks

We have presented a service-oriented control architecture for automation systems. The architecture forms a service hierarchy ranging from low-level sensor and actuator services, over a number of control service levels up to application processes. Instead of statically associating services for the different client operations, a flexible lease based binding approach is used. This approach follows the loosely coupled nature of components in service-oriented architectures. The algorithm used for the flexible binding approach was tested in different application scenarios. The evaluation results regarding action call roundtrip time exhibit that the Java-based service-oriented approach may not yet be a feasible solution for all applications. However, the applicability can be extended by using e.g. hardware-based message processing and real-time capable network infrastructures (e.g. PROFINET [10]).

9. References

[1] H. Smit, F. Jammes, “Service-Oriented Paradigms in Industrial Automation”, *IEEE Transactions on Industrial Informatics*, Vol. 1, No. 1, pp. 62-70, 2005.

[2] C. Gray, D. Cheriton, “Leases: an efficient fault-tolerant mechanism for distributed file cache consistency”, *ACM SIGOPS Operating Systems Review*, Vol. 23, Issue 5, pp. 202-210, Dec. 1989.

[3] Universal Plug and Play (UPnP), http://www.upnp.org, 1999.

[4] Devices Profile for Web Services (DPWS), http://schemas.xmlsoap.org/ws/2006/02/devprof/, 2006.

[5] Service Infrastructure for Real-time Embedded Networked Applications (SIRENA), http://www.sirena-itea.org, 2006.

[6] Sun Microsystems, Jini, Network Technology, http://www.sun.com/software/jini, 1999.

[7] Kapsers, Küfner, “Messen – Steuern – Regeln: Elemente der Automatisierungstechnik”, Vieweg Verlag, 6th Edition, p. 253, 2006.

[8] WS4D.org Java Multi Edition DPWS Stack, http://www.ws4d.org, 2007.

[9] Sun Java Real-time System 2.0 (Java RTS), http://java.sun.com/javase/technologies/realtime, 2007.

[10] PROFINET, http://www.profibus.com/pn/, 2007.