Generic Management Services for DPWS-enabled devices

G. Cândido [a], F. Jammes [b], J. Barata [a], Armando W. Colombo [c]
[a] UNINOVA – New University of Lisbon, Portugal
[b] Schneider Electric – Corporate R&D, 38TEC – Grenoble, France
[c] Schneider Electric Automation GmbH, BU Automation, SysCo, Germany
gmc@uninova.pt, francois2.jammes@schneider-electric.com, jab@uninova.pt, armando.colombo@schneider-electric.com

***Abstract*-The crescent ubiquity of smart devices across several domains of application can raise important management issues. The complete ecosystem must be capable to handle this new reality. From initial configuration and deployment to lifecycle monitoring and diagnosis, each device needs to be taken in account and easily reachable.**

The application of Service-oriented architectures based on web standards can significantly enhance the interoperability and openness of those devices. Standards as DPWS allied to WS-Management can provide basic building blocks to accomplish this objective. This document specifies a set of common and generic management services built over these standards. Although the focus of this article is mainly over industrial automation domain, the approach can be easily replicated to several other domains.

I. INTRODUCTION

Service-oriented Architecture is currently an important focus of interest from device level to high level IT, being commonly recognized as the silver bullet for all IT in the last few years [1] [2] [3] [4]. SOA promises to lead to near-perfect applications in which every function is implemented and exposed as a service, while it can still invoke other services to implement a required functionality.

The continuous convergence between computing and networking areas, enabled by the advances in semiconductor and transmission technology, allows new approaches to communication between systems and devices, in particular, embedded devices. At crescent rhythm, Internet technology is emerging as the basic carrier for interconnecting electronic devices in widely diverse domains of application. This tendency is the result of several converging evolutions [5]:

- Availability of low cost, high-performance, low-power electronic components allows embedding unprecedented features into ever tinier components.
- Ethernet networks are becoming widely accepted as the medium of choice for device interconnectivity. On top of these networks, Internet protocols of the TCP/IP family are becoming the standard vehicle for ex-changing information between connected devices.
- The emergence of data interchange mechanisms based on XML allows the developing of high-level interaction standards at the device level.
- The increasing exploitation of the Web Services paradigm for interconnecting heterogeneous applications on the basis of a lightweight communications infrastructure opens a perspective of universal, platform- and language-neutral connectivity.

Nowadays, there is very little doubt that the SOA approach will have a major impact in many branches of technology, not exclusively in original ICT sector, but also in other areas where these methodologies can be adapted to [6]. One of the most promising approaches is its application at device level where the usage of high level service-based communications infrastructure allows completely innovative advances. In the industrial automation domain, the development of OPC-UA specification [7] compatible with web services technology is one of many cases.

The goal of this document is to specify a set of management generic services that are available on any device, and being application domain independent. Devices Profile for Web Services is a common web services middleware and profile for devices [8], being currently ongoing a standardization process by the OASIS Web Services Discovery and Web Services Devices Profile Technical Committee [9] to be published in June 2009, together with WS-Discovery [10] and SOAP-over-UDP [11] specifications. The DPWS specification [12] defines two fundamental elements: the device and its hosted services, as present in Fig. 1. The device are the discoverable entity on the network, device can host services i.e. provide the functional behaviour.

1. Traditional Web Services vs. WS-Management

Two different approaches could be used for realizing such generic services: either use a proprietary Web services specification, or use the standard WS-Management services.

It is necessary to first introduce WS-Management [13].

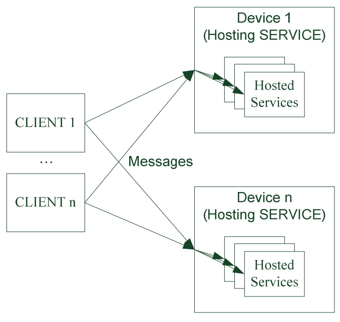


Fig. 1. Devices and hosted services

WS-Management specification describes a general WS* based protocol for managing systems such as PCs, servers, devices, Web Services, applications, and other manageable entities. To promote interoperability between management applications and managed resources, WS-Management identifies a core set of Web Service specifications and usage requirements that expose a common set of operations central to all systems management. This comprises the abilities to:

- Get, put (update), create and delete individual resources, such as setting parameters and dynamic data values.
- Enumerate the contents of containers and collections, such as large tables and logs.
- Subscribe to events emitted by managed resources.
- Invoke specific management methods with strongly typed input and output parameters.

In each of these areas of scope, the WS-Management specification defines minimal implementation requirements for conformant Web Service implementations. The implementation is free to extend beyond this set of operations, and may also choose not to support one or more areas of functionality previously listed if that functionality is not appropriate to the target device or system. The WS-Management specification defines a standard form to access resources, but it doesn't define a resource description model. So, the user is free to define the XML resource description model that best fits its application, and share it so that others can interact with it. For example, it would be possible to apply an OPC-UA Information Model specification [14] to describe a particular physical device.

Regarding traditional Web services, they should provide more complex services which will offer the user a more sophisticated level of information than simply get and set data values. Simple get and set data services must be avoided, since it consist on replicate the functionality already available through WS-Management. The services should provide more intelligent responses (higher level) to the client. For example, instead of simply getting a collection of maintenance parameters, a maintenance service can provide a summary of activity focusing the most common points of interest, still based on those parameters but executing some processing over it before sending it to the invoker.

The first goal here is to extend DPWS infrastructure in order to increase device and services management capabilities. These will also include the dynamic deployment and lifecycle management of devices and services by the end-user.

WS-Management specification fits this need: the resources being devices and services. For example, the create operation from WS-Management will allow the install phase of the resource, the delete operation, the uninstall phase of the resource. Furthermore, the specification is open to extend its core set of operations (create, delete, get, and put) with custom operations like start, stop, etc. Using WS-Management standard, it would be possible to open traditional property-closed devices to external tools since they are compliant with an open standard.

In this approach, the device is to be seen as the main logical entity that abstracts an application element, while its services represent the functionalities that a particular element allows other to exploit to accomplish its own goals. An application can be composed of several devices that interact between them through the services hosted on those with no imposed control architecture envisaged a priori (orchestration or choreography).

By having the ability to easily deploy these devices and its services into a physical device available on the network, the agility of the system is increased. This approach is generic enough to allow the integrator to implement its services with the programming language that best fits its current needs, define the service interfaces and then use it in a standardized manner through DPWS middleware. The application will then be discoverable and interoperable in the network. The integrator has also the ability to manage the complete lifecycle of these resources in accordance with the evolution of production goals.

So, these generics services facilitate incorporating and managing services (and applications) into devices; enabling control, supervision, or management systems to install, to uninstall, to start and stop, "devices" and/or "services". Of course, the lifetime of the service is a subset of the lifetime of its host: the device. These generics services allow devices to evolve and adapt their capabilities through the standard installation of new components.

II. RESOURCES

1. *Introduction*

In this context, a resource can be seen as a software entity that can be managed. This same resource can simply be a logical entity or abstract a physical object, such as a printer, a PLC, or a PC.

Besides the device and service resources already enunciated, the proposed model requires a new resource:

- *The service class resource:* this resource is used to describe the service implementation, which contains both generic information that must be provided by all implementation types, and implementation-specific information, in particular the service implementation code and initialization parameters, when the underlying technology supports dynamic code loading.

2. *Resources lifecycle*

It is important to distinguish two phases in their lifecycle: the deployment phase and the activation phase. The deployment phase is usually performed in two steps:

1) The service implementation code is first uploaded on the device platform, using either a Web Service or some platform-specific mechanism: depending on the technologies used by the platform and the service implementation, the implementation may be usable immediately, e.g. when using interpreted code or some

dynamic code loading and linking mechanism; or may require a reboot, e.g. when the service implementation must be integrated with the platform firmware.

2) The device is configured by registering one or more service instances, based on the previously uploaded service implementations.

The physical device will represent the device itself as a real-world physical entity – such as a PLC, an I/O device, et al. This physical device will already embed some built-in services that allow deploying applications but also other added-value services that allow the integrator to setup, monitor, and diagnose that particular device. These services are deployed by the device builder, being immediately available when taking a new device from the box. They cannot be removed or modified by the end-user – if the end-user wants to add its own services, he can do it through dynamic deployment. These built-in services will be further detailed in a following chapter of this document.

The application will be dynamically deployed in a form of logical devices. These logical devices will represent the logical entities that can be observed from the current application, exposing their hosted services as their capabilities that other can make use of. An application can even compose several of these logical devices into several layers of increasing abstraction, in an orchestration or choreography manner. A logical device can be composed of several other logical devices that interact between them in a particular way to accomplish current performance goals - application construction based on existing building blocks. Both physical and logical devices can be retrieved as normal DPWS devices. These logical devices are created, deleted, started and stopped on the physical device through an implementation based on WS-Management specification.

Logical devices will then host services according to the functionalities that each of them exposes. Hosted services are instantiated from the service classes previously deployed on the physical device. In summary, service classes consist of descriptions of service implementations that embed the implementation itself – this way it is possible to be independent from the technology used to implement the services. A logical device just needs to instantiate which service(s) it wants to host.

The relationship between physical device, logical device, hosted service and service class resources is synthesized in form of an UML diagram in Fig. 2.

III. Resources State Model

Although, these three resources: device, service and service class have their lifecycle and own state, their lifetimes are strongly linked. In some cases, deleting resources will be prevented due to locking conditions.

All these resources have a common parameter that stores its current state. This feature allows the integrator to retrieve the current resource state at any time, being, at the same time, the major condition parameter in the resource lifecycle state-machine in order to detect some lock situations over interdependent resources.

1. Device and Service Resources states

The resource state model chosen was based on OSGi [15] bundle lifecycle model. The original model was simplified and adapted to WS-Management scope by taking in account the use of the basic operations already provided by this specification, plus a few extensions.

Device and Service resources can be in one of the following states (see Fig. 3):

INSTALLED – The resource has been successfully installed.

READY – All references that the resource needs at deployment time are available. This state indicates that the resource is either ready to be started or has stopped.

STARTING – The resource is being started, the start operation has been called, and the start operation is not yet completed.

ON – The resource has successfully started and is running.

STOPPING – The resource is being stopped. The stop operation has been called but the stop operation is not yet completed.

UNINSTALLED – The resource has been uninstalled. It cannot move into another state.

When a device, a service or a Service Class are created, first they are stored in the persistent storage of the device platform (and service container), and then the registry of the DPWS middleware is initiated accordingly. They remain there until they are explicitly deleted.

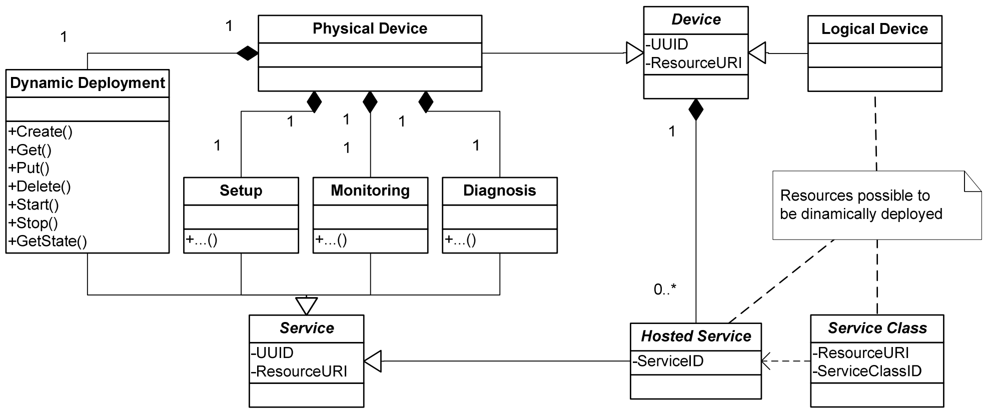


Fig. 2. Relationship between physical device, hosted logical device, hosted service and service class

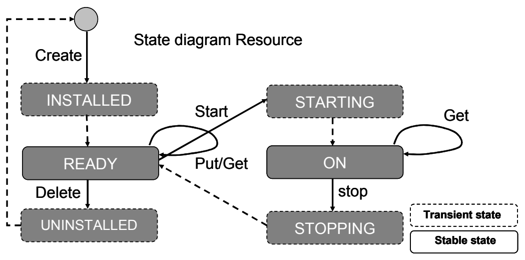


Fig. 3. Device and Service state diagram

2. *Service class resource states*

Service class resources can be in one of the following states (see Fig. 4):

INSTALLED – The resource has been successfully installed.

READY – All references that the resource needs at deployment time are available. This state indicates that the resource is either ready to be started or has stopped.

UNINSTALLED – The resource has been uninstalled. It cannot move into another state.

3. *WS-Management based operations*

WS-Management basic operations extended with a few add-on operations envisaged to control the execution of device and services resources are employed to manage the life-cycle of the resources available on the physical device.

A. *Available operations*

All resources support the following operations:

- *Create*: this operation creates the resource.
- *Get*: this operation retrieves the resource representations.
- *Put*: this operation updates the resource representations.
- *Delete*: this operation deletes resource instances.

Device and Service resources support these add-on operations:

- *Start*: this operation starts the resource
- *Stop*: this operation stops the resource
- *GetState*: this operation gets the state of the resource

All previous operations follow request-request message exchange pattern.

B. *Resources Interdependencies*

The logical interdependencies between the different types of resources will sometimes constrain the execution of the previous operations.

The deployment of resources is always done using *Create* operation which will return a unique resource identifier in case of successful execution. This operation will create a new element in the XML device configuration file (see Fig. 5).

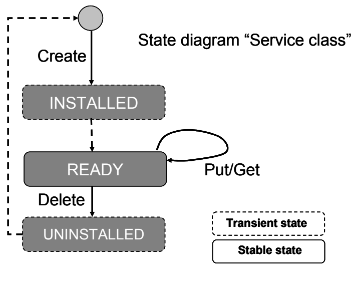


Fig. 4. "Service Class" state diagram

The deployment of a service will simply consist on the registry of a service class to a particular device element – that service will then be available from that device. This operation will be also constrained by the available resources of the current physical device. It is not advisable to deploy identical devices – it will become difficult to distinguish between two or more identical devices during discovery phase.

The *Start* operation can only be applied to device and service resources, since a Service Class resource only consists of a service representation and implementation details. A *Start* of a service from a device not currently in ON state, will imply that the service remains in STARTING state until the device turns ON. The service can only be discovered and invoked while being on ON state, as well as the device that exposes that service. A *Start* of a device will imply the *Start* of all the services it hosts – a DPWS Hello message is broadcasted. A single service class can be registered to several different devices, either hosted on the same or on two or more devices. Still, in the last case, an identical service class must be also deployed on those devices.

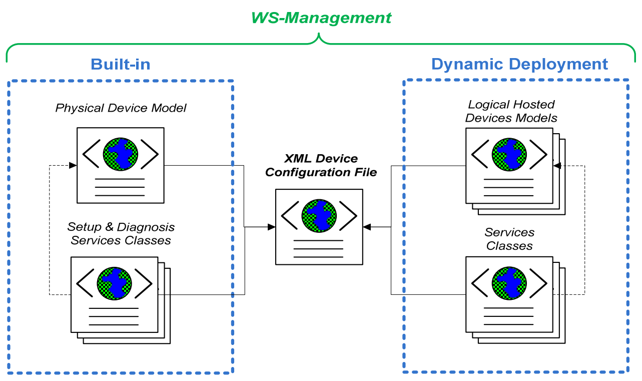
The *Stop* operation, for the same reasons as for *Start* operation, cannot be applied to Service Class resources. When a hosted service is stopped, it remains registered in its device, but it will not be available to be discovered or invoked – return to READY state. Still, to permanently suppress a hosted service from a device, the operation *Delete* must be used instead. The update of the services collection of a device will imply sending a new DPWS Hello message. As *Start* operation, a *Stop* operation over a device will imply the *Stop* of all services it hosts – a DPWS Bye message is sent.

Contrary to *Stop* operation, the *Delete* operation will definitely suppress the resource from the device platform, being necessary to re-deploy them, if needed, later. Again, the relations between different resources are sometimes critical and should be taken in account when executing the different actions.

A *Delete* of a Service Class is constrained to the hosted services that are still exploiting it. The operation will only successful if the Service Class is not registered to any of the devices as hosted service; otherwise the service will return the list of devices that are still registered to this Service Class.

The result of a *Delete* operation over a service registered to a device, or hosted service, will imply that the device that previously hosted that service will no longer expose it, even after a restart of device, except if a new deployment is done.

As previous operations, a *Delete* operation over a device



WS-Management
Built-in
Dynamic Deployment
Physical Device Model
Setup & Diagnosis Services Classes
XML Device Configuration File
Logical Hosted Devices Models
Services Classes

Fig. 5. Built-in vs. dynamically deployed elements

will imply the *Delete* of all its hosted services. Still, according Service Class resources will remain untouched, since other devices might be using them as hosted services implementations.

The *GetState* operation is used to retrieve the current state of a particular resource. The references retrieved during a *Create* operation can now be used as resource selectors.

IV. BUILT-IN SERVICES

All the information related to devices and according services available on a physical device are stated in a XML file – XML Device Configuration File that can be accessed through a WS-Management client application. This file contains the elements that describe service classes, devices and hosted services in each device – the complete collection of elements available on that particular physical device. This file will be updated every time a service class or device is deployed or when a service class is registered to a particular device becoming a hosted service (see Fig. 5). Although defined in the same file as deployed resources, the built-in services are protected from external modifications by adding a read-only tag to built-in resources.

1. Dynamic deployment as a built-in service

The dynamic deployment service is itself a built-in service, developed over WS-Management protocol, which allows the integrator to deploy its own resources i.e. logical devices and its services. Since only the Service Class resources comprise implementation details, the integrator can use the best language regarding current application goals, while devices and hosted services interfaces remain abstract.

As proof-of-concept experiment, it was possible to use IEC61131 languages to describe and control machine behavior. These control behaviours were then abstracted as logical devices possible to be discovered and as any other DPWS device – the invoker only cares about service functionality, not how the service is implemented. The IEC61131 code was translated to PLCOpen format and added as an element of according Service Class resource. The implementation code embedded in this Service Class resource will be interpreted or run by device OS or firmware, being available then in the network as any other DPWS device possible to be discovered and its services invoked (see Fig. 6). This implementation was firstly shown in SODA project industrial demonstrator [16], and it is envisaged to be further exploited in SOCRADES project industrial pilot application.

2. Built-in physical device management services

Besides the dynamic deployment feature, the device builder can embed other management services which will increase device added-value. These proprietary management services will be available on a new physical device by the time it is retrieved from the box and it is connected to the network. It will allow the end-user to configure, monitor and diagnose it. Although the machine builder might want to develop its own templates, the support technology remains open and standard which will increase device interoperability with other devices

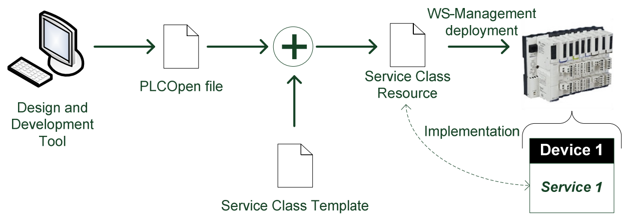


Fig. 6. Dynamic Deployment use-case example

and management tools. These templates can also be shared in order to let the end-user freely choose which tools to use.

A. Setup Services

For TCP/IP Ethernet-capable devices, the setup phase has a major importance since it is the first step to allow the communication with a new device. One of the most common problems is not knowing what is the default IP address of a new device or the need to change it to be in accordance with actual network parameters. By allowing retrieving and setting the IP address of the device in a standardized way, the integration phase is simplified and open to standard tools. The same approach can be applied to other configuration parameters. Almost all these services can be considered bidirectional. Bidirectional in this context means that it is both possible to retrieve or set the configuration data – the operation parameters are the same, but the action to perform is either to retrieve or to set those parameters.

B. Monitoring Services

Similar to built-in setup services approach, but here all services are normally used to retrieve information and not to set it. The monitoring information at physical device level can comprise information related to network statistics, current status, fault historic data, etc. Basically, this sub-set of services is dealing with Feature-based Monitoring indexes, i.e., monitoring information generated by the hardware or software associated to the device. When this kind of device exposing monitoring services is networked in a SOA based infrastructure, the interaction with a new SOA component has to be considered, it is the orchestrator, which could be embedded in one of the devices or placed in an external hardware. In both cases, the orchestration service exposed by this component is working on the basis of resource-, device-, system-models [17], which will mainly be exposing Model-based Monitoring indexes. Further specific application monitoring, combining both kind of indexes can be envisaged for OEM entities that want to increase the added-value of its products.

C. Diagnosis Services

As a complement to monitoring services, diagnosis services can also be event sources with the objective to notify the devices that subscribed to it about the possible reasons / sources of errors, faults, or detected abnormal behaviours. Every time the device detects such a situation it sends an event containing the relevant information about it (monitoring and diagnosis services as major functionalities of an intelligent SOA-based supervisory control system [18]).

The device status can be also modified. This functionality can be useful after a maintenance activity – update the status from an ERROR to an OK state, for example.

Due to the use of DPWS and WS-Management at device level, the classical pyramidal architecture can be easily modified. This way, devices have the ability to push data automatically to the aggregation systems, while the network load in that critical part of the network can be reduced. The diagnostics for all those devices can be aggregated, and at a higher level, this aggregated status can be polled by the management system and used to support the orchestration.

Remark: This kind of services, as it is also the case of e.g., monitoring, maintenance services, are also known as to be “Infrastructure Services” linking the IT with function performed in the physical world [19].

V. Conclusion

The application of WS-Management at device level promises to enhance the agility and openness of the traditionally morose and complex task of managing a wide range of network devices. The WS-Management specification turned out to be the fittest approach, in comparison with traditional Web Services, to manage devices due to its inherent nature and extensibility easiness. There is no need to replicate the same functionality already provided by WS-Management – traditional Web Services approach should only be applied if more sophisticated and complex management features are needed. The approach presented here is to be seen as a first step on how to apply this SOA-based technology to industrial automation domain devices management. The methodology to deploy and manage resources embedded in a physical device, as well as allowing manufacturers to embed their own build-in management services in their devices, lies over an open standard possible to be employed to different levels of the complete system architectures – unified management approach. By using an open standard, interoperability between devices from different vendors is also improved.

Starting from this point, further developments can be envisaged, such as possible application to other domains, increase security aspects, new functionalities, compatibility check with OPC-UA specifications, etc.

Acknowledgment

The work present here was developed within the scope of the European IST FP6 project “Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded deviceS” (see http://www.socrades.eu)

References

[1] J. Bloomberg and R. Schmelzer, *Service Orient or Be Doomed!: How service orientation will change your business*: Wiley, 2006.

[2] T. Erl, *Service Oriented Architecture – Concepts, Technology, and Design*, 6th ed.: Prentice Hall, 2006.

[3] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: Service-Oriented Architecture and Design Strategies*: Wiley, 2008.

[4] M. Bell, *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*: John Wiley & Sons, 2008.

[5] F. Jammes and H. Smith, "Service-oriented Paradigms in Industrial Automation," *IEEE Transactions on Industrial Informatics,* vol. 1, 2005.

[6] G. Cândido, J. Barata, A. W. Colombo, and F. Jammes, "SOA in reconfigurable supply chains: A research roadmap," *Engineering Applications of Artificial Intelligence,* 2008.

[7] OPC, "OPC Unified Architecture (OPC-UA) Specifications," http://www.opcfoundation.org/UA, 2008.

[8] F. Jammes, A. Mensch, and H. Smith, "Service-Oriented Device Communications Using the Devices Profile for Web Services," in *ACM Int. Conference Proceeding Series*, 2005, pp. 1-8.

[9] OASIS, "Web Services Discovery and Web Services Devices Profile (WS-DD) TC website," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-dd, 2009.

[10] OASIS, "Web Services Dynamic Discovery (WS-Discovery) Version 1.1 Specification," http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html, 2009.

[11] OASIS, "SOAP-over-UDP Version 1.1 Specification," http://docs.oasis-open.org/ws-dd/soapoverudp/1.1/wsdd-soapoverudp-1.1-spec.html, 2009.

[12] OASIS, "Devices Profile for Web Services Version 1.1 Specification," http://docs.oasis-open.org/ws-dd/dpws/1.1/pr-01/wsdd-dpws-1.1-spec-pr-01.html, 2009.

[13] DMTF, "Web Services for Management (WS-Management) Specification," http://www.dmtf.org/standards/published_documents/DSP0226_1.0.0.pdf, 2008.

[14] OPC, "OPC UA Part 5 - Information Model 1.01 Specification," http://www.opcfoundation.org/UA/Part5, 2009.

[15] OSGi, "OSGi Alliance - The Dynamic Module System for Java," http://www.osgi.org, 2009.

[16] SODA, "Industrial Demonstrator Presentation," http://www.soda-itea.org/Demonstrators/Industrial/default.html, 2008.

[17] A. W. Colombo and S. Karnouskos, "Towards the Factory of the Future: A Service-Oriented Cross-Layer Infrastructure," in *European Telecommunications Standards Institute (ETSI) Book on: "ICT Shaping The World - A Scientific View", chapter 6*: John Wiley and Sons, 2009.

[18] R. Du, M. Elbestawi, and S. Wu, "Automated Monitoring of manufacturing Processes, Part 1: Monitoring Methods," *Journal of Engineering for Industry,* vol. 117, pp. 121-132, 1995.

[19] NEXOF, "NEXOF Reference Architecture website," http://www.nexof-ra.eu/, 2009.