

# SOA in Reconfigurable Supply Chains: a Research Roadmap

Gonçalo Cândido<sup>1</sup>, José Barata<sup>1</sup>, Armando W. Colombo<sup>2</sup>, François Jammes<sup>3</sup>

<sup>1</sup> UNINOVA - New University of Lisbon, Portugal

<sup>2</sup> Schneider Electric GmbH P&T / HUB-PMO, Germany

<sup>3</sup> Schneider Electric, Corporate R&D, 38TEC Grenoble, France

**Abstract.** Originally coming from business world, Service-oriented Architecture (SOA) paradigm is expanding its range of application into several different environments. Industrial automation is increasingly interested on adopting it as a unifying approach with several advantages over traditional automation. In particular the paradigm is well indicated to support agile and reconfigurable supply chains due to its dynamic nature. In this domain the main goals are short time-to-market, fast application (re)configurability, more intelligent devices with lifecycle support, technology openness, seamless IT integration, et al. The current research challenges associated to the application of SOA into Reconfigurable Supply Chains are enumerated and detailed with the aim of providing a roadmap into a major adoption of SOA to support agile reconfigurable supply chains.

**Keywords:** Service-Oriented Architecture, Reconfigurable supply chains, SOA challenges, roadmap, service granularity, Industrial Automation, DPWS

## 1 Introduction

In the last years, the manufacturing world has deeply felt the effects of globalization on all its different levels. The consumers demand for highly customized products with high quality at low cost, with a min-

imum possible time-to-market. However, current manufacturing systems still can not entirely cope with it. The main issues are:

- Long time for system design and installation;
- Complex and time-consuming reconfiguration to face product variations;
- Extremely centralized/hierarchical implementations;
- Scalability involves exponential complexity;
- No fault-tolerance/redundancy;
- Incompatibility between different vendors' equipment and legacy systems;
- Lack of widely accepted standards;
- Shop floor systems are still commonly isolated from higher level business environments, although Manufacturing Execution Systems (MES) are starting to become increasingly available in industry.

In reality, the world of real-time embedded computing is characterized by a high degree of diversity in device functionality, form factor, network protocols, input/output features, etc, as well as the presence of many hardware and software platforms. In areas with a large base of installed devices like industrial automation, this has often resulted in a patchwork of technology islands with poor scalability. Several standards are already available to face these integration issues such as OPC (OPC 2008), ISA-88 (ISA-95 2008), ISA-95 (ISA-95 2008), NAMUR (NAMUR 2008), as well as the work currently in progress by ISO committees such as TC 184 – Automation Systems and Integration (ISO 2008).

SOA is an approach that promises to bring an important input to the actual industrial automation environment. SOA establishes an architectural model that aims to enhance the efficiency, agility, and productivity of an enterprise by positioning services as the primary means through which solution logic is represented in support of the realization of strategic goals associated with service-oriented computing (Erl 2006; Erl 2007).

The concept of agility in this context implies being more than flexible or lean. Flexibility refers to a company that can easily adapt itself to produce a range of products (mostly predetermined), while lean essentially means producing without waste. On the other hand, agility corresponds to operating efficiently in a competitive environment dominated by change and uncertainty (Goldman, Nagel et al. 1995). Agility is a fundamental requirement for modern reconfigurable supply-chain companies in order to face challenges provoked by the globalization, environmental and working conditions regulations, improved standards for quality and fast technological mutation (Ching-Torng Lin 2006). However, global company agility is always limited by its least agile building block – all levels of the Computer Integrated Manufacturing (CIM) pyramid, from ERP to shop floor level, need to be agile and interact in a seamless and synchronized manner.

The SOA paradigm is currently expanding from original business IT high level to lower levels and to several other domains of applications. SOA at the device level is just now starting to be adopted into industrial automation applications, mainly due to several collaborative initiatives (see Table 1).

Name	Summary
IST SOCRADES (SOCRADES 2008)	Made up of 15 partners from 6 European countries, led by Schneider Electric and including the major European players in the industrial automation sector. Its primary objective is to develop a design, execution and management platform for next-generation industrial automation systems, exploiting the SOA paradigm both at the device and at the application level.
ITEA SODA (SODA 2007)	The project main goal is to implement a complete ecosystem for designing, building, deploying and running device-based applications leveraging the service-oriented breakthrough provided by the SIRENA framework, within different domains of application including industrial automation.
OPC-UA (OPC 2008)	THE OPC Foundation has embarked on the creation of the Unified Architecture (OPC-UA), a completely new system design, written to be platform agnostic. OPC-UA will replace, modernize and enhance all the functionality of the existing OPC-defined interfaces. Web Services being the only high-level application-to-application communications technology to be embraced by virtually all platform providers including Microsoft, IBM, Sun, and Linux, is the expected technology to implement it.
STREP InLife (InLife 2007)	Focus on ambient intelligence and knowledge management technologies application to assure a sustainable and safe use of manufacturing lines and their infrastructure over their life-cycle.
RI-MACS (RI-MACS 2007)	The scientific and technological objectives of RI-MACS projects are the definition of a radically innovative manufacturing control open architecture based on state-of-the-art information and communication technologies and modular mechatronics. Intelligent distributed control, open architectures, wireless technology, virtual engi-

	neering tools and design methodologies are the main technological focus of this project.
STREP CoBIs (COBIS 2006)	Led by SAP, this European project aims a new SOA approach to business processes involving physical (goods, tools, etc.) in enterprise environments. The intention is to apply advances in networked systems to embed business logic in the physical entities to build automated item identification and tracking systems.
SIRENA (SIRENA 2006)	With the fast growing use of SOA for building web-based applications, the “ITEA Achievement Award 2006” SIRENA project proved that this popular approach can be applied successfully to low-cost micro devices. Technological advances in SIRENA enabled integration of greater intelligence in small devices, while embedded devices become ubiquitous and IP networking reaches the lowest levels of device hierarchy.

**Table 1.** SOA in Industrial Automation - collaborative initiatives

Even though SOA applications are already becoming a common approach in several enterprises at business and management level, at the shop floor level there are still some important points to be addressed. The so much called SOA conception needs to be adapted and mapped to this new domain of application.

Currently within this area of SOA-based control there are mainly 2 approaches: orchestration and choreography (or a possible mixture of both). However these approaches were developed within the high level business context, so there is a need to check both for consistency and effectiveness when applied to industrial domain, to see which one adapts better to which cases. The industrial automation requirements extracted from several use cases are of the most importance while validating the adoption of these approaches to this new framework.

To promote system agility, the development tools should straightforwardly and intuitively allow the integrator to build his application easily and faster than older approaches with, at least, the same level of robustness and performance – this is the key aspect to a wider adoption of SOA in this domain of application (Bloomberg and Schmelzer 2006).

At organization level, managers already noticed that they need to cooperate with other organizations in order to remain competitive (Vemadat 2007; Gunasekaran, Lai et al. 2008; Sarimveis, Patrinos et al. 2008). Although several work has be done on the topics of virtual organizations agility of production and/or collaboration to deal with unexpected demands and volatile markets at level (Camarinha-Matos

and Afsarmanesh 2006) (Camarinha-Matos 2007) (Ding, Benyoucef et al. 2006), this agility can only be achieved if all organization levels are also equally agile. The automation devices level plays a fundamental role on overall supply chain agility, since a device is the last frontier where higher level process workflows are transformed into a structured collection of physical actions to be invoked in a particular sequence – device control and management aspects are here crucial to support above levels agility. SOA-based approaches are now entering this domain of application in a top-down way.

This fresh approach in the domain of field automation has a direct impact on how reconfigurable supply chains will evolve. A complete reconfigurable supply chain implies a smooth integration and alignment between complex supply chain management layers and field level automation behavior. If a lower level environment cannot accomplish the desired agility goals, the overall system will be incapable of delivering a good performance – the agility of a complete system is always constrained by the least agile element. By embedding more intelligence in each device, the overall ambient intelligence is enhanced by having more autonomous, intelligent and self-contained devices which are the main building blocks of a production system. By having already some embedded intelligence in small devices, the task of ambient intelligence becomes eased by only having to handle more abstract and complex information instead of devices low level intricacies. Also, devices easier to setup, debug, monitor and diagnose are a key-factor during (re)engineering or down-time phases by saving a considerable amount of integration time that currently significantly reduces production capacity.

The connection between shopfloor activities and the enterprises services (high level) is fundamental to create more sophisticated high level services to support more reliable decision making. This sophistication at decision making helps enterprises joining or creating more complex supply chains (extended enterprises, virtual enterprises, collaborative networks). Very important research work is currently going on in this direction (Stamatis, Colombo et al. 2007; Colombo and Harrison 2008).

The main issue behind this paper is to highlight the research effort that is needed in SOA based approaches at shop-floor level to equip enterprises with agile shop-floors to enable their participation in complex supply chains. Section 2 discusses current automation requirements at shop-floor level to highlight how fundamental is the development of a new generation of control systems based on distributed intelligent devices to maintain sustainability and competitiveness of current enterprises. The SOA based approach is a suitable candidate to support these requirements and therefore several of the properties of SOA systems to support agile manufacturing systems are described and discussed in section 3. At the end of this section the reader is able to understand how SOA systems can help control systems designers to fulfill the requirements of highly reconfigurable, fully integrated, and agile enterprises. Finally, section 4 discusses the most important research challenges still needed to implement successful SOA based systems. The issues of service conception and granularity are particularly addressed.

It should be noted that this paper does not intend to describe a particular SOA based solution. The importance of this paper is thoroughly discussing why SOA is an adequate solution to current manufacturing enterprises and what still needs to be done from a research point of view.

## **2 Industrial Automation Requirements**

Current industrial automation environments are composed by several heterogeneous network environments that need to be effortlessly integrated (SOCRADES 2007). To avoid these issues, end-users frequently tend to pick a complete line of products from a unique OEM. This situation severely restrains system openness to integrate new devices, especially from different OEMs, since each company is still much based on their own standards and tools, although OPC standards are becoming increasingly available in new equipment. The integration effort in these cases is costly and the reengineering phase tends to be much longer than desired. Nevertheless, OEMs are starting to provide equipment in conformance with

several international standards, such as OPC. Regarding legacy devices, a software wrapper can hide devices intricacies and make it compatible with the current system environment.

The main requirements for the automation layer concerning a SOA implementation are defined in the following sub-sections.

## **2.1 Application (Re)engineering**

The ultimate desire of a manufacturing company is to provide the client with the product he needs in the time-frame he wants at competitive price. The reengineering phase is crucial while trying to accomplish this, so the industrial automation integrator must be capable of reconfigure the system easily, fast and keeping a good performance (update devices' application, configuration, physical and network topology, range of interaction, et al.). At any time, a new device can be added, removed, reconfigured and the existing software architecture must support it, along with it done at a fast rate. The device hardware plays also an important role on system flexibility, e.g. a device with a hardware built-in application impossible of being reconfigured cannot adapt to a new context not predicted during design phase.

The automation application needs to be easily deployed into the devices, either for a centralized or distributed approach, although the last one implies a more complex process (SOCRADES 2007). This is a particular difficult task due to the wide range of incompatible tools and deployment methodologies. Note that before restarting the complete production system or part of it, there is a need to debug and validate the changes made, in addition to possible troubleshooting needs to restore a faulty process.

Other hot topic is the independence of the software application from the target device (hardware). The application code should be portable so that it can be deployed into several heterogeneous devices, with none or just a few variations on the configuration. This implies a flexible code deployment tool capable of discovering and adapting to the target device.

## **2.2 Components Composition**

In order to provide added value over the different elements that constitute the environment, different devices capabilities (here named as services) need to be aggregated, at different levels of abstraction, from low level control devices (sensors and actuators) to higher level SCADA, MES and ERP applications. The so called atomic services, hiding implementation details, need to be straightforwardly assembled into more complex ones and so on, as the Russian doll concept (Jammes and Smith 2005), rising the service abstraction. By rising service abstraction, high level process descriptions become available to be used by the application without the need to regard low level intricacies. Instead of having implicit addressing of a function, e.g. the change of a vague I/O variable value, a service should be discovered based on the action it provides and exposes (self-describing).

The main objective is to turn assembly of services automatic or at least more autonomous from human input. Although this vision is mostly directed to application design phase, this dynamicity can be even applied to run-time phase making a device capable of discovering, choose and interact with the service that best fits its current needs; e.g. during an interconnected device down time (maintenance or failure).

## **2.3 Self-\* Capabilities**

The so called devices self-\* capabilities are increasingly a hot topic allied to the promises of autonomous and still interoperable intelligent devices running on future manufacturing plants, and on many other domains. Since computer-based approaches were envisaged as a major input to manufacturing domain (H. B. Voelcker 1988), and while processors are becoming increasingly powerful, smaller, and cheaper, tiny intelligent devices ubiquity tend to become a reality in other domains of application, such as home and building automation, power distribution, multimedia, etc. Since extensively manage and control an envi-

ronment full of devices will become an oversized task, these devices will need to be more autonomous, by having local decision control, preventive maintenance, diagnosis and even self-healing capabilities.

The service is considered autonomous since it is created and operates independently of its environment providing self-contained functionality, i.e., this functionality would be useful even if it is not associated with any higher-level system. Although its autonomy, a service must maintain its interoperability so that others can exploit its potential.

#### **2.4 Lifecycle Support**

Allied to the self-\* capabilities, lifecycle support features are also welcome to be applied from field to higher level scale, regarding both application and device scopes. While at application level the parameters to manage are normally too specific to that particular case, the device itself can already embed some generic services that allow the end-user to manage and control it in a standard manner from the first instant it is taken from the box. These intelligent devices should provide built-in services for setup, control, monitoring, maintenance and diagnosis. The way to use these same services must be an open standard widely accepted by the major manufacturers. One of the major clients complains when dealing with a new device is exactly how to interact with it at the start just to define simple parameters (e.g. IP address) or check the current device status and configuration.

#### **2.5 Device Capabilities**

The wide collection of available devices with diverse capabilities and different domain of application makes the choice of the most suitable device some times tricky. Consequently, the platform should be also compatible with low resources devices (such as a 32 bits processor with 512Kbytes Flash and 96

Kbytes of RAM). These low price devices are well suitable to perform as autonomous entities in a wireless Sensor/Actuator network.

New automation approaches must provide, at least, at its smallest set of features, capabilities already available in commonly deployed technologies. Reliable and real-time messaging, in addition to “fieldbus” capabilities are mandatory requirements in automation environments. A bridge between wireless and wired networks is also an added-value feature, although quality control on wireless networks is still not entirely employed.

## **2.6 IT integration**

The integration of industrial automation devices within the enterprise IT platform is becoming a key factor to endorse the overall management and control over a complete business environment, most of the times distributed over different levels of organization and even geographical areas.

Seamless and effective IT integration is still a goal. The effortless access and management of distributed data is still a more complex subject.

The IT platform, as well as the production system, should automatically detect the introduction or the removal of a device in the network. This device should also provide a location awareness feature, so that it would be possible to find it on the environment and, most important, be sure that it is the one to be addressed.

These are the main requirements for an SOA implementation adapted for the automation layer.

### 3 SOA approach

SOA is starting to be a focus of interest due to its possible gains over current production systems issues, from device level to high level IT, being commonly recognized as the silver bullet for all IT in the last few years (Bloomberg and Schmelzer 2006) (Erl 2006) (Michael Rosen 2008) (Bell 2008). 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.

SOA approach defines that the logic required to solve a large problem can be better constructed, carried out, and managed if it decomposed into a collection of smaller, related pieces, although the key is the manner in which it achieves separation (Erl 2006). As society, individual companies are service-oriented and collectively, their businesses comprise a community. By letting business to self-govern their individual services, they evolve relatively independently from each other, avoiding tight connections. Still, some baseline conventions need to be followed. By standardizing key aspects to the benefit of the consumer, it can choose what services best suits its needs and exploit them in an open and uniform way.

The basic SOA application consists of services that provide service descriptions and communicate via messages. These components (services, descriptions and messages) form the core of any SOA implementation. The key aspects of SOA principles are (Erl 2006):

- Loose Coupling: relationship that minimizes dependency and only requires that services retain an awareness of each other.
- Service contract: communications agreement, as described in service description.
- Autonomy: local control over the logic a service encapsulates.
- Abstraction: hides logic from outside world.
- Reusability: logic divided into different composable services.
- Composability: services can be coordinated and assembled to form composite services.

- Statelessness: services don't retain information specific to a particular activity.
- Discoverability: services should be outwardly descriptive to be found and accessed through discovery mechanisms.

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 areas. This tendency is the result of several converging evolutions (Jammes and Smith 2005):

- Availability of low cost, high-performance, low-power electronic components allows embedding unprecedented horsepower into ever tinier components.
- Ethernet networks are becoming widely accepted as the medium of choice for device networking. On top of these networks, Internet protocols of the TCP/IP family are becoming the standard vehicle for exchanging information between connected devices.
- The emergence of data interchange mechanisms based on XML data formatting paves the way for developing high-level interaction standards at the device level.
- Advent 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. 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.

Currently, web services technology is the most adopted to implement SOA application. The traditional view about web services simply refers to it as a static software system designed to support inter-

operable machine-to-machine interaction over a network, being usually just APIs that can be accessed over a network and executed on a remote system hosting the requested services (Haas and Brown 2004). The application of web services in several distinct areas promoted the creation of more specific specifications covering different subjects such as discovery, security, addressing, reliable messaging, etc. These specifications are often combined to form profiles that adapt to a particular domain of application.

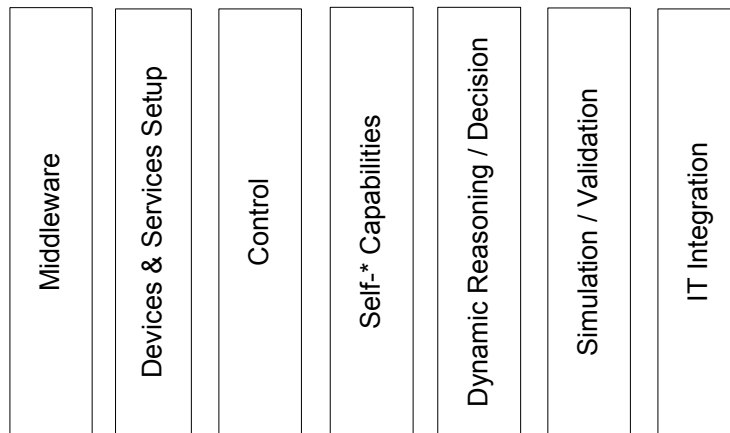
Being a standard technology, Web Services eases the interoperability, integration and reuse of the application components (i.e. services). Using IT-based standard protocols in the different production system levels, it is possible to provide direct and seamless communication between devices and IT level. An immediate example would consist on a device that emits an event upwards every time one of its parts needs an intervention or a replacement. Due to its embedded intelligence, the device will directly provide IT with all necessary data (possibly faulty part details), physical placement, number of previous identical faults, etc. Services described through WSDL (W3C 2001) files are easy to compose and are open for external or generic tools. Independence between the provided services and their implementation enables autonomous development and re-use of components.

Regarding industrial and field devices constraints and requirements, the most important SOA principles to focus are (Jammes and Smith 2005):

- The service design follows an outside-in approach, i.e. the interface of the service is defined by focusing on how that service can fit as an atomic task in a larger process;
- Services can be composed into higher-level services;
- Communications are loosely coupled and of an asynchronous nature;
- Services abstract heterogeneous hardware and software platforms from different providers. As each service encapsulates its own complexity, scalability, manageability and maintenance become built-in features.

- Interaction entities collaborate by sharing info and resources, in a peer-to-peer manner, in any configuration layout (from totally distributed to traditional hierarchy);
- Capability to negotiate their properties, such as Quality of Service (QoS), security level, performance constrains, etc.
- It enables genericity and reusability.

The research topics involved on the adoption of SOA concepts and methodology into industrial automation device level can be exposed as several layers of capabilities that will be available in future intelligent devices (see Figure 1).



**Figure 1. SOA inputs into different industrial automation layers**

The topics referred to each layer of Figure 1 are exposed in the next sub-chapters.

### **3.1 Middleware**

Middleware plays a crucial role in device architecture since it is the support base on which all the other building blocks will be assembled. Concerning SOA at device level, one of the most promising specifica-

tions is the “Devices Profile for Web Services” (DPWS) specification (Microsoft, Intel et al. 2006), developed by Microsoft and other partners, and already available in Microsoft Windows Vista and Windows Embedded CE 6.0 R2 platforms, used at device level for the first time during the SIRENA project. The DPWS adds protocols to the standard core protocols used by web services (IP, TCP, UDP, HTTP, SOAP, WML, et. al.), such as WS-Discovery, WS-Eventing, WS-Addressing, WS-Security, WS-ReliableMessaging, and others, some of them currently being developed (Jammes, Mensch et al. 2005). WS-ReliableMessaging, in particular, would have a major input in order to guarantee service reactivity in a environment where communication is loosely coupled and asynchronous.

Using DPWS, scalability is favoured by the fact that event-driven communications are substantially more efficient, in terms of bandwidth usage and processing constrains, than polling-based communications.

After Schneider-Electric and other European projects partners such as ABB, SAP and Siemens strong push to put DPWS middleware at device level, other automation companies, as Beckhoff (Beckhoff 2007), are already adopting the specification both for industrial and home automation scopes. Besides DPWS-based “Web Services on Devices” (WSDL-S) implementation by Microsoft (Microsoft 2008), there is also open-source implementations available online, such as SOA4D (SOA4D) and WS4D (WS4D).

### **3.2 Devices & Services Setup**

One of the current issues while dealing with field devices is the setup phase, both for devices and services. Each manufacturer has its own tools and methodologies to perform this task, being a complex mission when dealing with large heterogeneous environments with devices coming from several incompatible vendors. The time dispended by an integrator to adapt to every tool and system takes an important frac-

tion of the integration phase. These include simple and yet fundamental tasks as IP configuration, although IPv6-capable devices promise to solve most of IP addressing issues. Concerning generic devices (which can be (re)programmed by the end-user) other current setup issues are related with device description, I/O configuration and services deployment.

Regarding services setup, it is possible to distinguish two types of services embedded on the device: built-in and deployed. Built-in services are embedded in device by its vendor or by an OEM in order to provide not only setup, but also diagnostic, control, monitoring and maintenance capabilities in an open-standard manner. These will focus on the setup issues presented for the device level. These services are easier to integrate, reuse and compose, and are open for external or generic tools, due to the independence between the provided services and their implementation

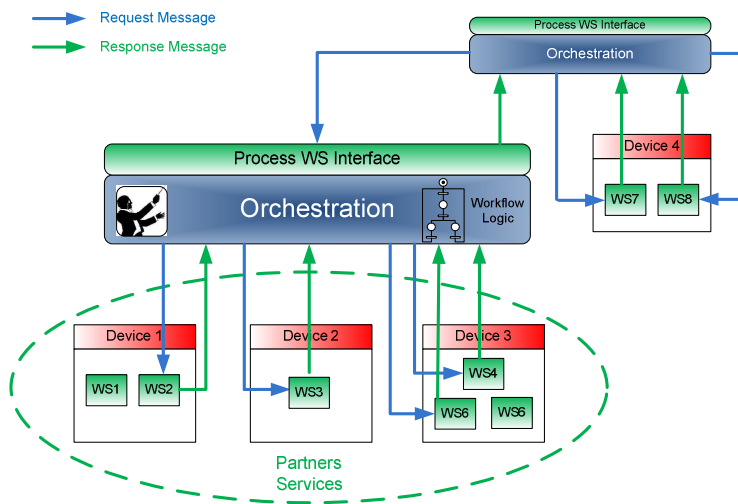
Deployed services correspond to the application itself and its developer has the responsibility to add or not setup features within its code. In this subject, some research is needed to provide guidelines and possibly some templates for different types of applications.

### **3.3 Control**

The aspect of control within SOA approaches is mostly related to the process of making several services work together to create added value in the form of a more complex process. The interactions between services are mostly defined as a sequence of procedures, most of the time owned by different distributed entities, which needs to be followed to accomplish a common goal (Erl 2006). The adopted methodologies to structure these interactions within the SOA framework mostly fall into orchestration or choreography (or even mixed) approaches. These approaches are defined in terms of global behaviour (how the different services are organized to work together), being possible to implement it using different models and/or languages.

### 3.3.1 Orchestration

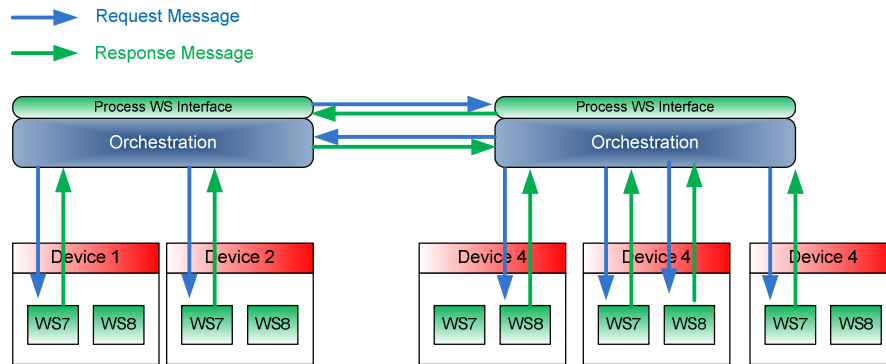
Within the orchestration approach, a centrally controlled set of workflow logic facilitates the interoperability between two or more different applications. With orchestration, being itself exposed as a service, the individual processes don't need to redevelop the solutions that originally automated those. The workflow logic involved in orchestration consists in several rules, conditions and events, i.e. it specifies how different partners should interoperate with the central node in order to carry out a task. Here, the process logic is centralized yet still extensible and composable, being at the same time a way to abstract a process into a single service. A heterarchical approach is also possible by having several orchestrators at different levels of composition (i.e. one of the partners of the orchestration can be itself another orchestrator that encapsulates and orchestrates other partners in a hierarchical way) (see Figure 2).



**Figure 2.** Orchestration example

Since each orchestrator has its own process logic, it's possible to imagine an orchestrator that some time during its process execution needs to invoke the service provided by other orchestrator (at same composi-

tion level or other), and vice-versa. In this case, the orchestration approach tends to present a choreography-like behaviour (see Figure 3). This particular case is not totally in line with the traditional orchestration vision.



**Figure 3.** Orchestration/Choreography similarities

Web Services Business Process Execution Language (WS-BPEL, also know as BPEL4WS or simply just BPEL) (OASIS 2007) is the primary specification related to services orchestration. BPEL structures the workflow logic with predefined primitive activities. Basic activities (receive, invoke, reply, throw, and wait) correspond to fundamental workflow action that can be assembled using the structured activities (sequence, switch, while, flow, pick).

In the industrial automation area, there is some work in progress in order to use Petri Net-based orchestration (SOCRADES 2007) (Bepperling, Mendes et al. 2006). This graphical and logical approach would allow an easier adoption among the traditional automation community that is mostly used to IEC61131 languages. A summary of orchestration advantages and drawbacks is present in Table 2.

Advantages	Drawbacks
<ul style="list-style-type: none"> <li>• Process workflow logic is encapsulated at a single point, which makes it easy to modify without impacting</li> </ul>	<ul style="list-style-type: none"> <li>• no horizontal interaction - by definition, it is a totally hierarchical approach, where from the bottom to top</li> </ul>

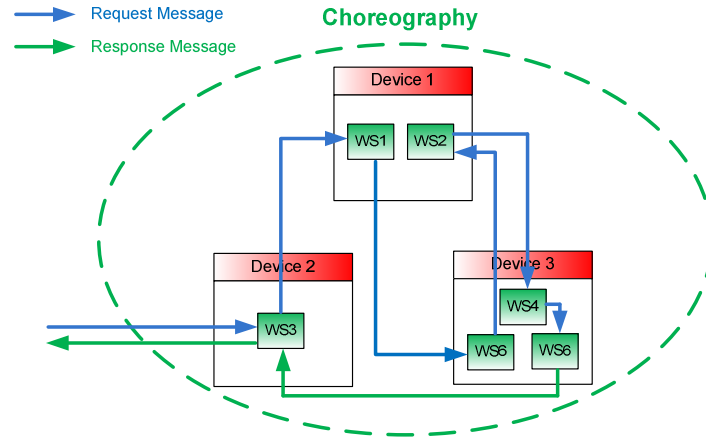
<p>the process partners;</p> <ul style="list-style-type: none"> <li>• Can be applied recursively ("Russian dolls" composition paradigm);</li> <li>• Abstracts the interactions between the orchestration node and the individual process partners, exposing itself as a single service;</li> <li>• Preserves the autonomy of each of the process partners, which should not even be aware of each other's existence;</li> <li>• Supports the evolution into a diversely federated approach.</li> </ul>	<p>there is always a node that abstracts the ones below it;</p> <ul style="list-style-type: none"> <li>• Pushes the process control decision out of the devices – here a device is a simple "slave".</li> <li>• No particular research challenge in its majority.</li> </ul>
--	--

**Table 2.** Orchestration advantages vs. drawbacks

### 3.3.2 Choreography

In summary, choreography enables collaboration between distributed participants. The goal is to set up an organized collaboration between different distributed services, without any other entity controlling the collaboration logic (OASIS 2007). Choreographies allow the definition of patterns to execute a particular task.

A choreography schema assumes that there is no owner of the global collaboration logic, contrary to orchestration where the last one is executed and controlled by a single unity in a centralized manner (see Figure 4).



**Figure 4.** Choreography behaviour example

In order to expose certain choreography, a particular service must know its own role within a process; i.e. what the service supports and how it reacts in a particular context. These services can also be referred to as participants.

Each possible contact between two roles in a choreography is identified as a relationship. Multiple participants can assume different roles and have different relationships. The moulds of this same relationship, i.e. the message exchanges pattern between two roles, are expressed by channels. Channel information can also be passed through a message between services, so that a service can determine how it can interact with a particular service. Furthermore, the message exchange logic is encapsulated within an interaction – the fundamental choreographies building block. These are related to work units, which impose rules and constraints to make that a successful interaction.

Choreography, in the same way as orchestration, can have different levels of abstraction. A choreography can be composed by other choreographies in a block-based composition in order to create a more complex collaboration behaviours.

Natively supporting composability, reusability and extensibility, the choreography approach increases system agility and discover capabilities. Since it is possible to pass channel information, participants can share it with a possible new participant so that it can also join the collaboration.

The Web Services Choreography Description Language (WS-CDL) is a XML-based language that describes peer-to-peer collaborations of participants by defining, from a global view point, their common and complementary observable behaviour; where ordered message exchanges result in accomplishing a common business goal (W3C 2005). A summary of orchestration advantages and drawbacks is present in Table 3.

Advantages	Drawbacks
<ul style="list-style-type: none"> <li>• Allows the definition of a totally distributed control approach</li> <li>• Supports distributed complexity</li> <li>• Services more autonomous (embedded intelligent control decision) and self-* capable.</li> <li>• Possible extension to SOA vision to standardize cross-devices interaction for some simple cases.</li> </ul>	<ul style="list-style-type: none"> <li>• Need to distribute the work flow logic to all involved devices, although less complex</li> <li>• Still no consensus about possible solution, particularly within industrial automation scope</li> <li>• Possible network traffic boost when a large number of services are connected and active.</li> <li>• Difficult to scale to large and complex applications</li> </ul>

**Table 3.** Choreography advantages vs. drawbacks

An orchestration can be regarded as specific application of choreography. In fact, WS-CDL and WS-BPEL specifications mutually overlap in some features due to its different organizations origins, respectively W3C and OASIS.

### 3.3.3 Application to Industrial Automation

Both previous approaches for SOA-based control have different goals depending on the application in study. While choreography denotes a more self-organization approach, orchestration still defends a more

hierarchical approach in line with traditional industrial automation control approaches, although flexible to support also heterarchical topologies.

For simple use cases, choreography would increase autonomy and embedded intelligence of application components (services) and enhancing loose-couple interactions between them. An intelligent system of conveyors could adopt this approach. By embedding intelligence to each module (conveyors, transfers, lifts, etc.), it can automatically detect and recognize its “neighbors” and, together, define which is the best path for a particular pallet to be conveyed, avoiding faulty equipment and congestions, in a similar way, as packet transport algorithms used in telecommunications domain. Here there is no central unit to coordinate the complete system. The global system adapts to the current situation based on local information available to each module and the interactions between them – self-organization.

A traditional example of an orchestration application would consist of a machine composed by several layers of abstraction. Each layer is composed by elements that abstract that level capabilities to the upper level, and so on, from input/output layer to complete machine/workstation abstraction, or even higher. This way, the above layer does not need to deal with previous layers intricacies, being a simple consumer of the clean and self-describing services provided by the level below. Here, orchestration provides enhancements on composability and hierarchical coordination, assuring each building block autonomy and interoperability features.

In a complex environment with several complex machines internally controlled by an orchestration approach, choreography can be used to manage the interactions between them. This would consist of a mixed approach, by having orchestration controlling machine building blocks and exposing the complete machine as a service and then having this same services autonomously interacting to achieve a goal in choreography-like manner.

As a final note, it is not possible to say that any of this WS-based control approaches is the final solution, since it mainly depends on the application in study. The research question here is how to find the right balance between the two.

### **3.4 Self-\* Capabilities**

Due to intrinsic features of SOA, particularly self-contained functionality, devices embedding self-capabilities comply with framework features, and provide a major device added-value in contrast with current automation devices features. Although being autonomous by having some local decision control, self-diagnosis and -healing capabilities, a device keeps its interoperability due to communication standards openness – key factor when interacting with same production process partners and reporting information to upper levels of management.

In (Barata, Ribeiro et al. 2007), a SOA-based approach for diagnosis and condition monitoring capabilities embedded on shop floor devices is presented in a test-case that proves the advantages of SOA when applied to this domain of application.

### **3.5 Dynamic Reasoning / Decision**

In order to promote dynamic, scalable and cost-effective infrastructure for electronic transactions, the Semantic Web Services research community (SWSI; SWSIG 2002) is pushing forward by enriching Web services with machine-processable semantics. Semantic Web services aim at an integrated technology for the next generation of the Web by combining Semantic Web technologies and Web services, thereby turning the Internet from an information Here, an ontology represents the concepts and their relationships that are important in that particular domain of interest, providing a vocabulary for that domain as well as a computerized specification of the meaning of terms used in the vocabulary. Several specifications on

the subject are already available, such as OWL-S (OWL-S 2004), WSMO (WSMO 2005) and WSDL-S (WSDL-S 2005).

OWL-S (Semantic Markup for Web Services) is an ontology of services that also pursues the automation of Web service tasks including automated Web service discovery, execution, interoperation, composition and execution monitoring. OWL-S supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-processable form.

The Web Service Modelling Ontology (WSMO) provides a conceptual framework and a formal language for semantically describing all relevant aspects of Web services in order to ease discovery, composition and invocation of electronic services over the Web. Having the Web Service Modelling Framework (WSMF) (Fensel and Bussler 2002) as support base, this one is refined and extended through a formal ontology and a Web Service Modeling Language (WSML).

The Web Services Semantics WSDL-S specification defines how to add semantic information to WSDL documents. These semantic annotations are used to automate service discovery, composition, mediation, and monitoring. Semantic annotations define the meaning of the inputs, outputs, preconditions and effects of the operations described in a service interface. When comparing WSDL-S to OWL-S and WSMO is possible to extract some important advantages:

- Possibility to describe both the semantics and operation level details in a single WSDL file- a language that the developer community is familiar with.
- It consists on an agnostic approach to ontology representation languages. It allows developers to annotate their Web services with their choice of ontology language (such as UML, OWL or WSML) unlike in OWL-S. This feature eases the reuse of existing ontologies, independently of their modelling language.
- It does not duplicate descriptions already defined in WSDL (ex. input and outputs).

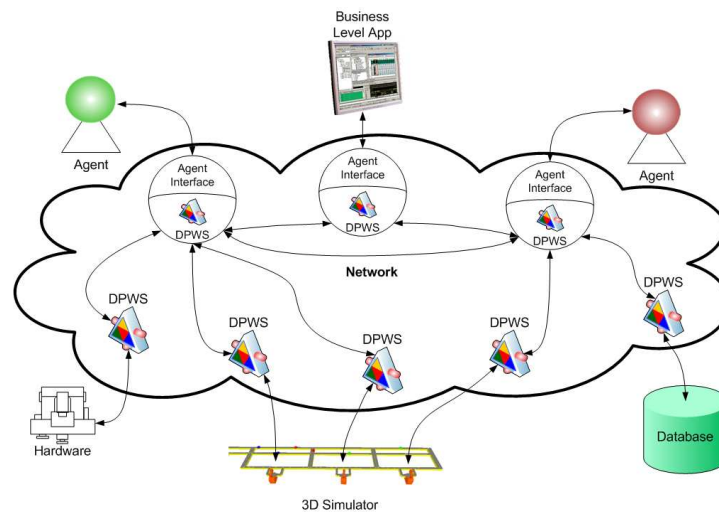
The semantic approach will focus on solving integration problems, trying to enhance assisted or even automatic interaction with newly plugged devices. It will provide input both on design and run-time phase.

During design phase, there is yet different levels of application. In a more hardware analysis, the integrator can be assisted by a semantic reasoner to find the best available device that fits its needs and constraints by checking the semantic tags present in devices description. In a services scale, a similar approach can be foreseen. Abstracting for implementation intricacies, the integrator can search for services (i.e. functionalities) available on the network, being assisted again by a semantic reasoner, and create its application by composing the available services in a perceptive and graphical style. Still during design phase, the reasoner can support the automatic creation of WSDL files that can be after deployed in the device. By extracting information from device description, it would be possible to generate some component of the WSDL file, reducing development effort. This is particularly attractive for management and diagnostic services that can be partly standardized to be reused in future applications.

Even if it is still only a research topic, it is possible to picture the exploitation of semantic tags during system run-time. During a device failure or maintenance action, a service would be capable of discovering and consume a comparable service available on the network. While discovery implies service description and semantic tags comparison, the invocation might need some translation mechanism to adapt to a possible unlike interface – services coming from different vendors might differ in the interface although implement the exact same functionality. Here, research still has several issues to address to really convince industrial automation companies to invest in this approach, such as security, run-time constraints, robustness, etc.

### 3.6 Simulation / Validation

Simulation is another area in which DPWS has been used to validate manufacturing systems. In (Milagaia 2008), a middleware layer was developed to enable software agents to benefit from DPWS functionalities. This middleware layer joins agents, databases, hardware, simulators, human interface applications such as production system management, error correction and maintenance, etc. (see Figure 5).



**Figure 5.** Infrastructure to support DPWS and Agents

The demonstrator consists of a multipath intelligent conveyor system that feeds 4 different workstations. Each of these stations provides a different service that can be used to accomplish a particular product process plan. The system dynamically adapts to avoid bottlenecks or deadlocks in the transport system and allows different product variations being produced simultaneously in the same production system. The demonstrator was created to prove the concept includes a 3D model of a simplified plant, a DPWS Interface per 3D entity, Agents that control the system, a Configuration Tool, a Production Manager Tool, a Communication Log and a Database. Each of these entities communicates via a DPWS Interface. Therefore it was proved that DPWS can also be used to support simulation. Since each simulated and real

(physical) device is modelled by a DPWS interface, replacing one virtual device (simulated) by its physical counterpart is straightforward. Therefore, DPWS can be used to develop simulation systems that support hardware in the loop, which is a major advantage, since it becomes possible to use simultaneously virtual and real devices.

The importance to support agile supply chains can be easily inferred since with DPWS become possible to plug and unplug easily nodes that represent the various companies involved in the extended enterprise. The nodes can represent real IT systems in a certain company or can be just a simulated entity.

### **3.7 IT Integration**

Enterprises are moving towards service-oriented infrastructures for a long time and now that the goal is to expand this approach to all levels of the enterprise it would be easier to communicate along the different layers. Today, intelligent manufacturing systems based on distributed embedded devices, integrate system intelligence in a limited amount of monolithic computing resources accompanied by large numbers of resource constrained devices (Karnouskos, Baecker et al. 2007).

The IT level can have two different approaches (or both simultaneously): direct access to individual field devices parameters or access to a summarized information report provided by data concentrator entity responsible for the monitoring of a group of devices or production workstation. By connecting directly to the device, the IT application needs to deal with lower level data intricacies, pushing results interpretation logic to the enterprise IT upper levels and boosting network traffic (and possible “bottleneck”) if a centralized IT application will retrieve data from a large number of devices. Still, this approach allows a more detailed view over the production system current status. By having data concentrators (ex. one for each production workstation), these entities can retrieve local information from the devices, extrapolate relevant information and provide only summary report to upper level IT. Al-

though pushing data interpretation logic to shop-floor level, this way higher level IT will possibly not have access to detailed information about a particular device that it might need in a particular situation. An example would consist of a device that after a fault is detected, it can provide the IT platform further details over it with different levels of granularity in accordance with the current fault type, like a debug mode.

For a complete and robustness implementation, the combination of both approaches would be more valuable. A web service-based direct integration of the production machine with an alert resolution dashboard would give enough time to contact the client in advance and initiate counter-measures (Karnouskos, Baecker et al. 2007). In diagnostic phase, the IT application can be supported by the data concentrator summary report, requiring more specific details, if needed, by connecting directly to the device(s) involved.

#### **4 SOA Design Challenges**

Disregarding possible implementation technology issues, SOA-based application designer also needs to take in account several aspects while modeling a particular application. The integrator needs to reflect over some essential questions, such as:

- What resource/service model best suits it?
- What would be the best granularity to choose?
- What should be considered a service and an operation?
- What would be the service interface?
- What device and service parameters are important to higher level applications e.g. SCADA, ERP, MES, etc.?

## **4.1 SOA Design**

In order to simplify the representation of a SOA-based application in graphical terms, Service Component Architecture (SCA) (BEA, IBM et al. 2005) provides a set of specifications which describe a model for building SOA applications and systems. SCA is a complement of initial approaches to deploy services, and as Web Services, it is built on open standards. SCA supports the representation of logic as autonomous and reusable components that can be easily integrated into any SCA-compliant application. SCA divides up the steps in building a SOA application into two major parts: the implementation of components which provide services and consume other services; and the assembly of sets of components to build composites. The SCA application is then built by wiring the references to the according services (atomic or composites).

SCA decouples service implementation and service assembly from the details of infrastructure capabilities and from the details of the access methods used to invoke services. SCA also supports a range of programming styles, including asynchronous and message-oriented styles, in addition to the synchronous call-and-return style.

## **4.2 Granularity**

In SOA, as well as in most of industrial applications and others, the choice of granularity is one of the most subjective topics to address. This issue is even more significant at design phase when the integrator while modeling the application needs to choose the best granularity for that actual case, e.g. should a sensor be a service or only the device that controls that sensor exposes a higher level service?

The overall quantity of functionality encapsulated by a service determines the service granularity. For instance, a service based on an entity service model will have a functional context associated with one or more related entities. Functionality associated with the chosen entity belongs within the service's

functional boundary. The larger the quantity of related functionality, the coarser the service granularity. Then, services with more narrow or targeted functional contexts will tend to have a finer grained level of service granularity (Erl 2007). Note that the level of service granularity is set by the service's functional context, not by the actual amount of functionality that resides within the physically implemented service. The data granularity is determined by the quantity of data exchanged by a specific service capability. For example, a service that retrieves a complex machine status will have a coarser level of data granularity than a service that simply retrieves the motor status present in that same machine.

Sometimes it is very difficult and time consuming to define the fittest granularity for a service to adapt to a particular application. If it is designed too small, there is a risk to develop a useless service (if considered alone), that must be composed together to have an application meaning – like this there is a need to shift some business logic (that should be encapsulated in services) to composition or orchestration logic. This tends to an application with uncountable very small services with the real application logic totally implemented with composition or orchestration. On the other side, services too big can be less reusable: it is a designer task to decompose these services in a composition of smaller, more reusable services.

To be completely effective to an integrator, service interfaces should clearly expose the operations they perform as well as the required input parameters, possible errors or exceptions, and results. Service interfaces should be easily understood by humans and at the same time possible to be used by reasoning systems to extract its features and “understand” its functionality and added-value.

The granularity should be in adequacy with the considered application and be permissive to support agile reconfiguration. The choice of granularity is especially attached to physical topology of the devices that compose the system and the functionality they provide to the system.

## 5. Conclusions

The adoption of SOA in all different aspects of the production environment can bring major advantages while deploying and managing reconfigurable supply chains. Traditional approaches only refer to the higher level applications to support reconfigurable supply chains and omit the low level details that most of the times will significantly constrain the initial expected results.

Several automation companies are already starting to present their products compliant with the SOA approach, while all big players in business and management applications have already a solid experience on this approach. Current developments are now envisaging putting together a complete, unified and open solution by following SOA guidelines, but many of the requirements and challenges highlighted in this paper still need to be answered. However, the development path clearly points to a full integration of SOA-based approaches between all enterprise levels.

**Acknowledgements.** A part of this work was done in the scope of the European IST FP6 project "Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices" (SOCRADES; <http://www.socrades.eu>), so the authors would like to thank the European Commission and their partners for their support.

## References

- Barata, J., L. Ribeiro, et al. (2007). Diagnosis using Service Oriented Architectures (SOA). International Conference on Industrial Informatics. Vienna, Austria.
- BEA, IBM, et al. (2005). "Building Systems using Service Oriented Architecture - White paper v0.9." from [http://www.iona.com/devcenter/sca/SCA\\_White\\_Paper1\\_09.pdf](http://www.iona.com/devcenter/sca/SCA_White_Paper1_09.pdf).
- Beckhoff. (2007). "WSD: Plug-and-play for building automation." Press release, from [http://www.beckhoff.de/download/press/2007/english/pr332007\\_Beckhoff.pdf](http://www.beckhoff.de/download/press/2007/english/pr332007_Beckhoff.pdf).
- Bell, M. (2008). Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture, John Wiley & Sons.
- Bepperling, A., J. Mendes, et al. (2006). A Framework for Development and Implementation of Web service-enabled Intelligent Autonomous Mechatronics Components. IEEE International Conference on Industrial Informatics.
- Bloomberg, J. and R. Schmelzer (2006). Service Orient or Be Doomed!: How service orientation will change your business, Wiley.
- Camarinha-Matos, L. (2007). ECOLEAD – Achievements in Collaborative Networked Organizations. 8th IFIP Working Conference on Virtual Enterprises. Guimarães, Portugal.
- Camarinha-Matos, L. and H. Afsarmanesh (2006). A modelling framework for Collaborative Networked Organizations. 7th IFIP Working Conference on Virtual Enterprises. Helsinki, Finland.
- Ching-Torng Lin, H. C., Po-Young Chu (2006). "Agility index in the supply chain." International Journal of Production Economics 100(2): 285-299.
- COBIS. (2006). "EC FP6 STREP CoBis Project." from <http://www.cobis-online.de/index.html>.

- Colombo, A. W. and R. Harrison (2008). "Modular and collaborative automation: achieving manufacturing flexibility and reconfigurability." International Journal of Manufacturing Technology and Management 14(3/4): 249-265.
- Ding, H., L. Benyoucef, et al. (2006). "A simulation-based multi-objective genetic algorithm approach for networked enterprises optimization." Engineering Applications of Artificial Intelligence 19(6): 609-623.
- Erl, T. (2006). Service Oriented Architecture – Concepts, Technology, and Design, Prentice Hall.
- Erl, T. (2007). SOA Principles of Service Design, Prentice Hall.
- Fensel, D. and C. Bussler (2002). "The Web Services Modelling Framework (WSMF)." Electronic Commerce Research and Applications 1(2).
- Goldman, S. L., R. N. Nagel, et al. (1995). Agile competitors and virtual organizations: strategies for enriching the customer. New York, Van Nostrand Reinhold.
- Gunasekaran, A., K.-h. Lai, et al. (2008). "Responsive supply chain: A competitive strategy in a networked economy." Omega 36(4): 549-564.
- H. B. Voelcker, A. A. G. R., R. W. Conway (1988). "Computer Applications in Manufacturing." Annual Reviews on Computer Science 3: 349-387.
- Haas, H. and A. Brown. (2004). "Web Services Glossary." from <http://www.w3.org/TR/ws-gloss/>.
- InLife. (2007). "EC FP6 STREP InLife Project ", from <http://www.uninova.pt/~inlife/>.
- ISA-95. (2008). "Technology ISA-88." from <http://www.isa-95.com/subpages/technology/isa-88.php>.
- ISA-95. (2008). "Technology ISA-95." from <http://www.isa-95.com/subpages/technology/isa-95.php>.

- ISO. (2008). "TC 184 - Automation systems and integration." from [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_tc\\_browse.htm?commid=54110](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_tc_browse.htm?commid=54110).
- Jammes, F., A. Mensch, et al. (2005). Service-Oriented Device Communications Using the Devices Profile for Web Services. ACM Int. Conference Proceeding Series.
- Jammes, F. and H. Smith (2005). "Service-oriented Paradigms in Industrial Automation." IEEE Transactions on Industrial Informatics 1(1).
- Karnouskos, S., O. Baecker, et al. (2007). Integration of SOA-ready Networked Embedded Devices in Enterprise Systems via a Cross-Layered Web Service Infrastructure. IEEE Conference on Emerging Technologies and Factory Automation. Patras, Greece.
- Michael Rosen, B. L., Kevin T. Smith, Marc J. Balcer (2008). Applied SOA: Service-Oriented Architecture and Design Strategies, Wiley.
- Microsoft. (2008). "Web Services on Devices (WSD)." from <http://www.microsoft.com/whdc/rally/Rallywsd.msp>.
- Microsoft, Intel, et al. (2006). "Devices Profile for Web Services (DPWS) Specification." from <http://schemas.xmlsoap.org/ws/2006/02/devprof/>.
- Milagaia, R. (2008). DPWS Middleware to support Agent-based Manufacturing Control and Simulation. Robotics and Integrated Manufacturing. Monte de Caparica, Universidade Nova de Lisboa. Master.
- NAMUR. (2008). "Standards and draft standards." from <http://www.namur.de/publications-and-news/technical-information/standards/?L=2>.
- OASIS. (2007). "Web Services Business Execution Language Version 2.0." from <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html>.
- OASIS. (2008). "Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.1 Specification." from <http://docs.oasis-open.org/ws-rx/wsrn/v1.1/wsrn.html>.

- OPC-UA. (2008). "OPC-Unified Architecture." from <http://www.opcfoundation.org/UA>.
- OPC. (2008). "OPC Unified Architecture (OPC-UA) Specifications." from <http://www.opcfoundation.org/UA>.
- OWL-S. (2004). "OWL-S: Semantic Markup for Web Services." from <http://www.w3.org/Submission/OWL-S/>.
- RI-MACS. (2007). "IST NMP-1 RI-MACS Project website." from <http://www.rimacs.org>.
- Sarimveis, H., P. Patrinos, et al. (2008). "Dynamic modeling and control of supply chain systems: A review." Computers & Operations Research 35(11): 3465-3478.
- SIRENA. (2006). "ITEA 02014 SIRENA Project ", from <http://www.sirena-itea.org>.
- SOA4D. (2008). "SOA4D - Service-Oriented Architecture for Devices website." from <https://forge.soa4d.org/>.
- SOCRADES (2007). "EU FP6 IST-5-034116 SOCRADES – Deliverable D1.2: Requirements of end users and component vendors/system integrators."
- SOCRADES (2007). EU FP6 IST-5-034116 SOCRADES – Deliverable D7.1: User requirements for the application systems engineering and lifecycle support of distributed smart embedded devices.
- SOCRADES (2007). IST SOCRADES Project - Deliverable D2.1: Framework specification for device-level service platform.
- SOCRADES. (2008). "EU FP6 IST-5-034116 SOCRADES Project." from <http://www.socrades.eu>.
- SODA. (2007). "ITEA 05022 SODA Project." from <http://www.soda-itea.org>.

Stamatis, K., A. W. Colombo, et al. (2007). Towards Service-oriented Smart Items in Industrial Environments. International Newsletter on Micro-Nano Integration - MST News, VDI/VDE-IT: 11-12.

SWSI. "Semantic Web Services Initiative ", from <http://www.swsi.org>.

SWSIG. (2002). "W3C Semantic Web Services Interest Group." from <http://www.w3.org/2002/ws/swsig/>.

Vemadat, F. B. (2007). "Interoperable Enterprise Systems: Principles, Concepts, and Methods." Annual Reviews in Control 31(1): 137-145.

W3C. (2001). "Web Service Description Language (WSDL)." from <http://www.w3.org/TR/wsdl>.

W3C. (2005). "Web Services Choreography Description Language ", from <http://www.w3.org/TR/ws-cdl-10/>.

WS4D. "WS4D - Web Services for Devices website." from <http://www.ws4d.org/>.

WSDL-S. (2005). "Web Service Semantics – WSDL-S." from <http://www.w3.org/Submission/WSDL-S/>.

WSMO. (2005). "Web Service Modelling Ontology (WSMO)." from <http://www.w3.org/Submission/WSMO/>.