Service-Oriented Device Communications using the *Devices Profile for Web Services*

François Jammes, Antoine Mensch, Harm Smit
Schneider Electric, Odonata, Schneider Electric
francois.jammes@ieee.org, antoine.mensch@odonata.fr, harm.smit@ieee.org

Abstract

This paper outlines the benefits of adopting service-oriented architectures at the level of communications between resource-constrained embedded devices, in particular for industrial device networks. It focuses on the usage of the Devices Profile for Web Services *as the underpinning of such "smart device" architectures and discusses an early implementation thereof. It further illustrates how "dumb" or "legacy" devices can be integrated using a gatewaying approach.*

1. Overview

Ch. 2 outlines the challenges faced by the manufacturing community and the opportunities resulting from increasing miniaturization and usage of standard communication protocols. Ch. 3 describes the rationale for the adoption of a service-oriented architecture (SOA) and the benefits of extending this paradigm to the realm of communications between "smart" devices of all kinds. Ch. 4 presents the major characteristics of the Devices Profile for Web Services (DPWS), a Web Services based device communication framework. Ch. 5 describes an experimental DPWS implementation together with some performance aspects and future extensions. Ch. 6 illustrates how the reach of this approach can be extended to encompass "dumb" or "legacy" devices.

2. Introduction

Shortcomings of current approaches to automation

For the sake of their competitiveness in today's global economy, manufacturing companies urgently need production systems that can provide both the flexibility to support product variety dynamically and the reconfigurability to be adapted economically to manufacture new generations of a given product.

Currently, about one third of the total cost of a manufacturing plant over its lifetime is spent on installation and set-up. Maintenance downtime accounts for another substantial portion of the operating costs. If a plant has to be adapted to new products by changing its process flow and introducing new equipment or replacing obsolete or non-competitive equipment provided by different makers, the downtime and installation costs rise considerably since reuse and reconfigurability are poorly supported. Today, virtually every new piece of automation has its own unique control system. As a result, it is estimated that 80% of the engineering effort is devoted to re-implementing the control and related electrical systems each time a new machine is implemented on a new project. This effort could be significantly reduced, and reconfigurability and reuse improved, if a component-based approach were adopted, i.e., if proven pre-assembled mechatronic components were used as the common building blocks to compose manufacturing automation systems.

In the automotive industry, it was determined that an engine assembly machine is composed of a relatively small number of common control elements. Thus, 80% of the controls and equipment for power-train assembly are standard, and tremendous effort could be saved if control functionality could be better encapsulated at device level [1]. As in other manufacturing sectors, the automotive industry now has to develop, deploy and support automated systems on a global basis in ever shorter timeframes. Furthermore, the lifecycle engineering of such production machines requires complex and timely interaction between geographically distributed members of project engineering teams comprising automation suppliers, control system suppliers, machine tool builders and end-user product, process and control engineers.

Readily configurable information exchange is required, not only between geographically dispersed business partners, but also with the embedded devices – in particular, to enable rapid reconfiguration and reuse of the system components to account for the introduction of new product models, as well as the ability to dynamically provide new value-added services and efficient diagnosis and maintenance.

Furthermore, current industrial information and control systems are often fragmented, hard to manage, difficult to change or extend and isolated from higher level business systems. Customers want to be able to connect all their sub-systems and equipment into the same, easily configurable, information system – hence the need for standard protocols and real-time information flow at the lowest system levels, i.e., embedded device. A distributed system is needed capable of integrating a variety of heterogeneous devices into an interoperable network of resources. Reliability and security are also key requirements.

SOA for collaborative automation

Attempts to address the need for more configurable production systems better able to meet the requirements of agile manufacturing have led to a growing interest in automation paradigms that model and implement production systems as sets of production units/agents/actors collaborating in a complex manner in order to achieve a common goal [2]. This approach is characterized by the adoption of decentralized distributed automation systems, with manufacturing resources composed of intelligent modules that can be easily reconfigured to suit evolving application needs. This collective functionality distributed across many mechatronic devices and machine controls replaces the programming of manufacturing sequences and supervisory functions in traditional production systems.

In the manufacturing community, multi-agent and holonic systems, also sometimes referred to as collaborative automation systems, have been the subject of great attention. Pilot applications of collaborative automation have indicated that the approach has the potential to reduce significantly the total amount of time for production system engineering [3]. However despite their promise, such systems have not as yet made significant inroads in manufacturing plants. In addition to the lack of widely accepted standards, one of the reasons for this situation seems to be that their implementations only cover part of the manufacturing landscape, whilst other areas remain subjected to the reign of proprietary standards, methods and mechanisms, resulting in a rigid patchwork of technology islands with poor scalability.

The principal inhibitors to the more widespread realization of collaborative automation systems are the inflexible communication infrastructure among current manufacturing process components, and the difficulty of porting existing application software to new configurations. An open, flexible and agile environment with plug-and-play connectivity is therefore desperately needed. Despite several proposals put forward by a variety of consortia and standards bodies to adopt open solutions for manufacturing plants, proprietary communication and control standards still severely impede the progress towards flexibility and agility.

SOA offers the potential to provide the necessary system-wide visibility and device interoperability in complex collaborative automation systems subject to frequent changes. In just a few years, the concept of SOA has gained substantial traction in business system environments, but SOA also holds the promise of meeting the technical and business level requirements for future automation systems.

In a nutshell, SOA is an architectural paradigm for building systems from autonomous yet interoperable components. A service only exposes its interface ("contract"), which fully encapsulates the complexity of its implementation. Services can be published and discovered dynamically. SOA is characterized by coarse-grained service interfaces, loose coupling between service providers and service consumers, and message-based, asynchronous communication. The use of open standards, in particular those of the Web Services family, allows for implementing SOA in a technology-neutral fashion. These features make SOA particularly applicable for a global multi-vendor environment where interoperability is essential.

As outlined in [4], the approach of expanding the use of SOA to low-level real-time embedded devices paves the way for fulfilling the reconfigurability and flexibility requirements of collaborative automation, thus allowing to reshape the automation landscape.

3. Service-oriented device networking

Opportunities and challenges

Internet technology is on its way to underpin a pervasively networked world interconnecting billions of people and trillions of devices – used in industrial automation, automotive electronics, telemetry, telecommunications equipment, building controls, home automation, medical instrumentation, etc. – much in the same way as the Internet came to the desktop before and is coming today to all sorts of personal information appliances. This tendency is the result of several converging evolutions:

- The availability of low-cost, high-performance, low-power electronic components allows embedding unprecedented horsepower into very tiny parts. Leveraging this technology to build advanced functionality into embedded devices, enables new distributed application paradigms based on interconnected "smart devices" with a high level of autonomy.
- Owing to their low cost, both wireline and wireless networks of the Ethernet type 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 communication between networked devices.

- The universal acceptance of information interchange based on Extensible Markup Language (XML) paves the way for developing high-level communication standards for devices.
- The advent of the Web Services paradigm for interconnecting heterogeneous applications through a lightweight communications infrastructure enables universal, platform-neutral connectivity.
- The ubiquitous presence of Internet technology increasingly allows "invisible" embedded devices and user-facing devices as well as higher-level information systems to coexist on the same network and, hence, to communicate.

As a consequence, the device networking market and technology are expected to substantially evolve in the forthcoming years. Market research from Forrester Research expects the market for network-connected devices to expand to 14 billion units by 2010.

Since the real-time embedded computing world is characterized by a high degree of diversity in device functionality, form factor, network protocols, input/output features… as well as the presence of many hardware and software platforms, the adoption of a uniform communication paradigm will greatly facilitate the elimination of the existing technology islands.

Service-oriented interaction patterns for devices

Devices are categorized as either *controlling devices* or *controlled devices*, but a given device may play both roles, thus enabling peer-to-peer interactions. The interaction patterns of a device-level SOA can be categorized according to five levels of functionality:

Addressing. This is the foundation for device networking. In the case of IP-based networking, the addressing capacity is provided by the IP protocol, either IPv4 or IPv6.

Discovery. Once addressing is established, devices need to discover each other. When a controlled device is added to the network, a discovery protocol enables it to advertise its services on the network. Similarly, when a controlling device enters the network it sends out a search request and then the devices that match the request send a corresponding reply.

Description. Once a controlling device has discovered a controlled device, to learn more about the latter and its capabilities, the controlling device must retrieve the controlled device's description ("metadata"), including information like manufacturer name, version, serial number, etc. For each service exposed by a device, the device description defines the command messages, or actions, that the service responds to, as well as the associated message formats.

Control. Once it knows a controlled device, a controlling device can exert control over it. To invoke an action on a device's service, a controlling device sends a control message to the network endpoint for that service. Resultantly, the service may or may not return a response message providing any command-specific information.

Eventing. In addition, devices may communicate through asynchronous eventing, usually implemented by a "publish-subscribe" mechanism, through which a service exposes events corresponding to internal state changes, to which controlling devices can subscribe in order to receive event notifications whenever the corresponding internal state change occurs.

Device level service-oriented protocols

Several device-level SOA technologies have been proposed, most notably UPnP (Universal Plug and Play) [5] and Jini [6].

Jini is strongly rooted in Java and therefore lacks platform-neutrality and is ill-adapted to usage in resource-restricted devices.

The UPnP architecture leverages Internet and Web technologies including IP, TCP, UDP, HTTP, SOAP and XML; hence it is truly platform-agnostic. However, it uses specific protocols for device discovery and eventing and a specific XML-based language for device and service description.

A very promising approach is that proposed by the Devices Profile for Web Services (DPWS) [7], described below. It has the same advantages as UPnP, but additionally it is fully aligned with Web Services technology. It is worth noting that Microsoft's Longhorn/Vista platform natively integrates DPWS.

4. Using DPWS for high-level device communications

The DPWS specification defines an architecture in which devices run two types of services: *hosting services* and *hosted services*. Hosting services are directly associated to a device, and play an important part in the device discovery process. Hosted services are mostly functional and depend on their hosting device for discovery. In addition to these hosted services, DPWS specifies a set of built-in services:

- Discovery services: used by a device to advertise itself and/or to discover other devices.
- Metadata exchange services: provide dynamic access to the metadata of a device’s hosted services.
- Eventing services: allowing other devices to subscribe to asynchronous event messages produced by a given service.

The core Web Services standards are documented in [8]. The DPWS protocol stack, depicted in Fig 1, integrates all these core standards, to which it adds Web Services protocols for discovery and eventing.

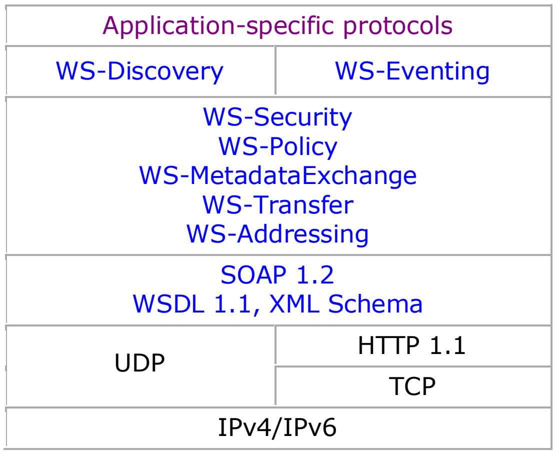

Fig. 1 – DPWS protocol stack

DPWS Messaging

A key aspect of the DPWS protocol stack is that all messaging is based on the use of SOAP and WS-Addressing. Indeed, owing to the extensibility features built into SOAP, the Web Services architecture is highly composable, as the use of SOAP headers allows the various Web Service protocols to be integrated individually and incrementally, without disturbing the rest of the protocol stack, as well as to be improved and versioned in isolation, without affecting the entire stack. Another advantage of using SOAP across the board is that common functionality can be factored among the various higher-level protocols. Thus, the same security mechanisms can be used both for control, discovery and eventing.

The purpose of WS-Addressing is to move all message addressing information into the SOAP header, thereby decoupling the message content from the transport and enabling more complex message exchange patterns than HTTP's request-response model. WS-Addressing provides a well-defined way to do asynchronous one-way messaging, with the ability to correlate messages. Every networked resource is identified by an End-Point Reference (EPR), composed of an Address and Reference Parameters. The Address is a logical address (a URI) that resolves to a physical address through an appropriate binding. The Reference Parameters are pieces of state that the service may use to disambiguate subordinate resources, and are opaque to the message sender. With DPWS, the Address part of any EPR is constrained to be a URI of the UUID type, which uniquely identifies a device.

WS-Addressing defines a set of standard SOAP message addressing properties: 'To', 'Action', 'ReplyTo', 'FaultTo', 'MessageId', 'From' and 'RelatesTo'. The 'To' URI specifies the message destination, while the 'Action' URI corresponds to a WSDL port type and identifies the message semantics. A 'ReplyTo' endpoint must be specified only when a response is expected, but it can be used to route that response to any valid endpoint. In that case, a 'MessageId' must also be specified, which the destination endpoint will return in the 'RelatesTo' header of the response. Optionally, 'From' can be used to identify the message originator. The optional 'FaultTo' header allows SOAP fault messages to be routed to the specified EPR. Together, these addressing constructs allow for completely asynchronous message exchanges.

Finally, SOAP messages may carry attachments following the SOAP Message Transmission Optimization Mechanism (MTOM).

DPWS Description & Discovery

Like any Web Service, DPWS-based services are described using XML Schema, WSDL and WS-Policy.

DPWS uses the WS-Discovery protocol for plug-and-play device discovery. WS-Discovery defines a multicast discovery protocol to search for and locate network-connected resources. The primary mode of discovery is a client searching for one or more so-called "target services". In the context of DPWS, a target service is a device. Hosted services do not participate in the discovery process, but can be individually addressed (through their respective EPRs) once the hosting device has been discovered. The search can either specify the *type* of the device or a *scope* in which the device resides or both; it materializes as a Probe message sent to a multicast group; devices that match the probe send a ProbeMatch response directly to the client (in unicast mode). Similarly, to locate a device by name, a client sends a Resolve message to the same multicast group and the device that matches sends a ResolveMatch response directly to the client.

WS-Discovery leverages the SOAP-UDP binding in order to minimize network traffic overhead. When a device joins the network, it announces itself by sending a multicast Hello message. Thus, clients can detect newly available devices without repeated probing. When leaving the network in an orderly manner, a device announces this through a Bye message.

Multicast-based discovery is limited to local subnets. In order for discovery to be scalable to enterprise-wide scenarios, WS-Discovery introduces the notion of Discovery Proxy (DP). A DP has two functions: multicast suppression (to reduce network traffic) and extending the discovery protocol's reach beyond the local subnet. When a DP detects a multicast Probe or Resolve request, it sends a Hello for itself. By listening for these announcements, clients detect DPs and switch to use a DP-specific protocol. However, when a DP is unresponsive, clients revert to use the ordinary discovery protocol.

During the discovery process, a device exposes the following metadata:

- its EPR, which allows to determine the device's physical network address;
- 'Types': a set of messages the device can send and/or receive; these can be either functional WSDL port-types (e.g. 'turn on', 'turn off') or abstract types grouping several port types and/or hosted services (e.g. 'gripper', 'lighting', 'residential gateway');
- 'Scopes': a set of attributes that may be used to organize devices into logical or hierarchical groups, e.g. according to their location or access rights.

Subsequently, further metadata on a device and/or on its hosted services can be obtained using WS-Transfer Get messages with different "dialects":

- 'ThisModel' metadata provides device type information like manufacturer name, model name, model number, etc.;
- 'ThisDevice' metadata provides information on the device itself such as serial number, firmware version and friendly name;
- 'Relationship' metadata is the list of services hosted by the device, which comprises the EPR and types of each of the hosted services;
- 'WSDL' pertains to the location where to find the definition of the port-types (operations and message structures) implemented by the endpoint addressed; this might be exploited by generic clients that dynamically interpret the WSDL definitions.

DPWS Control & Event Notification

Control and event notification messages are simply SOAP and WS-Addressing based messages, formatted according to the WSDL definitions they relate to.

DPWS leverages WS-Eventing, which defines a publish-subscribe event handling protocol allowing one Web Service ("event sink") to register interest ("subscription") with another Web Service ("event source") in receiving messages about events ("notifications"). A subscription is leased by an event source to an event sink and expires over time; hence, it must be regularly renewed. WS-Eventing therefore provides three built-in operations: Subscribe, Renew and Unsubscribe. Event notifications themselves are one-way messages (or solicit-response interactions), the content of which may include any data of any type. They are transported in the same way as any other SOAP message. An event source may further support filtering; if it does and a subscribe request contains a filter expression, the event source sends only notifications that match the requested filter. While WS-Eventing allows for filtering based on XPath predicates, DPWS limits the filtering capability to the matching of a list of URIs.

DPWS Security

Depending on the openness of the application context, device communications may be more or less subject to a variety of security attacks and may therefore require that communication be secured using WS-Security mechanisms. WS-Security is used here as a collective name for an extensive set of Web Service specifications related to various aspects of security.

DPWS specifies the protocols and message formats related to:

- authentication of devices,
- integrity of message exchanges between devices,
- confidentiality of message exchanges between devices.

This specification is a minimal set of recommended default mechanisms for interoperable security between devices. It may be enhanced in the future as device processing capabilities evolve. Furthermore, devices are free to support other security mechanisms, specified through policies. These optional security mechanisms are factored in through SOAP.

The default DPWS model for security setup encompasses a mutual authentication phase, as a result of which a secure transport channel is established, over which subsequent communication takes place, using a session key for message encryption.

A client's security requirements, if any, are advertised during the device discovery process; they may include authentication and secured discovery. With secured discovery, the integrity of all multicast and unicast discovery messages is protected using message-level signatures. Discovery messages are not encrypted. According to the security requirements conveyed, the communicating entities negotiate the authentication and key establishment protocols to be used.

The basis for device authentication is a device-level certificate. In simple deployments, a device may have a self-signed certificate. In managed deployments, a device may have a certificate with a root that is trusted by the client. The authentication handshake process encompasses the verification of the credentials of both communicating entities, as well as the establishment of a session key for protecting later message exchanges. The default mechanism recommended by DPWS for device authentication is to set up a TLS session. This channel-based security approach is sufficient as long as devices communicate without intermediaries. The prescribed device certificate format uses the X.509v3 standard. Once the authentication phase is completed, a secure channel is established between the communicating entities, over which the HTTPS protocol is used for exchanging encrypted description, control or eventing messages.

5. Experimental DPWS toolkit for embedded devices

In the context of the ITEA/SIRENA project [10], Schneider Electric realized a proof-of-concept DPWS implementation destined to be integrated into embedded devices for a variety of applications in the industrial automation, home, automotive and telecommunications sectors. It implements all the DPWS protocols except WS-Security.

Applications created using this DPWS Toolkit are directly interoperable with the DPWS implementation included in the Microsoft Vista platform.

The DPWS Toolkit has been ported to many target software platforms (various flavors of Linux, Microsoft's Windows and Windows CE, Sun's Solaris, WindRiver's VxWorks, ExpressLogic's ThreadX and Quadros Systems' Quadros). At present, it runs on various hardware platforms (foreshadowing single-chip implementations), including the following:

- Type A: a processor board comprising a 44-MHz ARM7 TDMI and associated memory (but no cache memory), running ThreadX.
- Type B: a processor board housing a 400-MHz Intel PXA255 (XScale) with on-chip cache and associated memory, running Windows CE.

Principles of operation

Fig. 2 outlines the general architecture of a device built using the DPWS Toolkit. In this figure:

- "App services" and "Events" are user-defined services and events, provided as user-written code and generated code in the DPWS Toolkit.
- "Execution Services", "Eventing Services" and "Discovery Services" are predefined services, including the embedded SOAP 1.2 engine, provided as run-time libraries in the DPWS Toolkit.
- Two network interfaces are shown: the primary interface uses the standard SOAP 1.2 over HTTP binding, while the discovery interface uses SOAP over UDP and a multicast address to broadcast and listen to the discovery messages. These interfaces rely on a commercially available IP stack.

The DPWS Toolkit is written in C and is derived from the gSOAP open-source software package, which has been modified to accommodate asynchronous control flows based on WS-Addressing. It is fed by a WSDL file, from which it generates C structures representing the message contents, together with marshalling/demarshalling code for transforming between C structures and SOAP/XML messages, as well as proxy (client-side) and skeleton (server-side) code (including network interactions and message dispatch mechanisms).

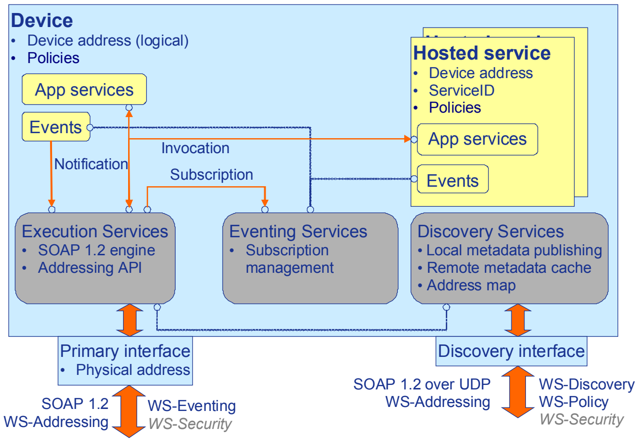

Fig. 2 – DPWS-based device architecture

Performance aspects

Based on the above-mentioned Type A and Type B boards and on the DPWS software component, several experimental devices have been built, connected through 100 Mbps Ethernet networks. This network transfer rate is so high that transmission delays are almost negligible compared to CPU processing times.

Measurements on a Type A platform show the following results:

- The static memory footprint of the device software including the OS, the TCP/IP protocol stack and the DPWS software is less than 500 KB, while the dynamic memory requirements are below 100 KB.
- The total time required for preparing and sending a request message to a device and for receiving and handling its response message is about 39 ms.

On a Type B platform, the aggregate request and response handling time measured is about 10 ms.

There is ample room for improving the performance characteristics of this early DPWS implementation. Directions being explored include:

- software optimizations (better memory management, faster string manipulation),
- using SOAP directly on top of TCP/IP,
- binary instead of textual encoding of SOAP messages, in particular, using the Efficient XML solution retained by W3C [9].

Overall, it is estimated that an improvement of between one and two orders of magnitude over the currently observed processing speed is achievable. Thus, while sub-millisecond message processing times can easily be obtained on PC-class devices today, this remains a challenge on embedded devices but is considered to be attainable.

Future extensions

Besides the performance improvements mentioned above, the DPWS toolkit is being consolidated and extended in various directions, such as:

- Implementation of a Java version of the toolkit; this version has been released by mid-2006.
- Support of the optional WS-Security mechanisms.
- Inclusion of the Discovery Proxy functionality and extension of the description and discovery mechanisms allowing for integration of semantic service aspects.
- Addition of an Event Broker, allowing to improve the scalability of the eventing mechanisms.
- Support of the WS-ReliableMessaging protocol for guaranteed message delivery.
- Implementation of a generic component model in support of dynamic deployment.
- Support of the IPv6 protocol family.

6. Integration of "dumb" or "legacy" devices

If implementing "smart devices" at the lowest level of the device hierarchy is the ultimate perspective of the device-level SOA approach, this is not yet feasible today for all types of devices in a cost-effective manner. Furthermore, there is a strong requirement to be able to integrate existing devices on a "wrap-and-reuse" rather than a "rip-and-replace" basis.

In such cases, a gateway approach can be used to DPWS-enable devices that do not natively implement DPWS. This approach may be illustrated through an example, in which a hypothetical Fieldbus Gateway (FGW) device manages some sort of fieldbus to which three devices, designated as "Valve", "Heater" and "Alarm", are connected. The FGW exposes itself as a DPWS device to various operator-controlled devices, such as a PC or a PDA. Its initial configuration is done by personnel familiar with the fieldbus characteristics. The FGW exposes all fieldbus devices as individual devices, as if each device were a DPWS server (Fig. 3). This approach allows for seamless future migration to a situation where every device natively supports DPWS.

The principle for implementing the DPWS device embedding scheme by the FGW is as follows:

- During the fieldbus device configuration stage, a device configuration document is built up, in which the necessary metadata is added to the description of each of the fieldbus-connected devices.
- During installation of the FGW, the installer downloads this configuration document to the FGW, and the latter builds up an internal database representing the aggregate fieldbus device configuration, including the appropriate metadata.
- Once its configuration is done, the FGW emulates the DPWS discovery process for the devices it manages, using "Hello" messages to advertise each of the devices and supplying metadata on request. This allows the installer to display the entire fieldbus device configuration. Optionally, the installer may then further want to group devices into named groups according to various criteria, e.g. their physical location, which facilitates issuing commands to a whole group of devices at once.
- When the operator brings up his device, the latter discovers the FGW as well as each of the fieldbus devices managed by the FGW – through "Hello" and/or "Probe" and "ProbeMatch" messages.

The operator is then capable of addressing commands to any fieldbus device as if that device were directly visible to his controlling device.

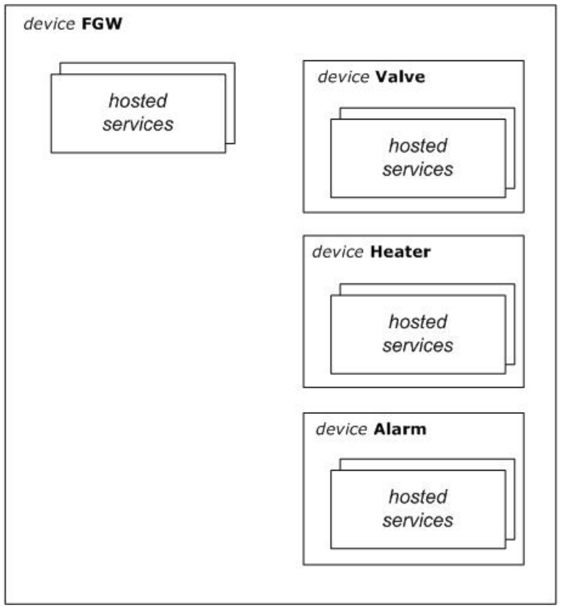

Fig. 3 – Device gateway with embedded virtual devices

The sequence diagram shown in Fig. 4 illustrates these interactions, starting from the FGW installation phase. This figure also shows the generation of an event notification following the detection of an abnormal condition by the Alarm device.

A major advantage of using standard Web Services at the device level is to also use Web Services aggregation and orchestration technology at the device level. In the present example scenario, equipping the FGW with a lightweight orchestration engine would allow to readily create and execute complex customized scripts invoking multiple sequential and/or concurrent device services, e.g. to close all Valve devices, to set the Heater device in energy-saving mode and to activate the Alarm device.

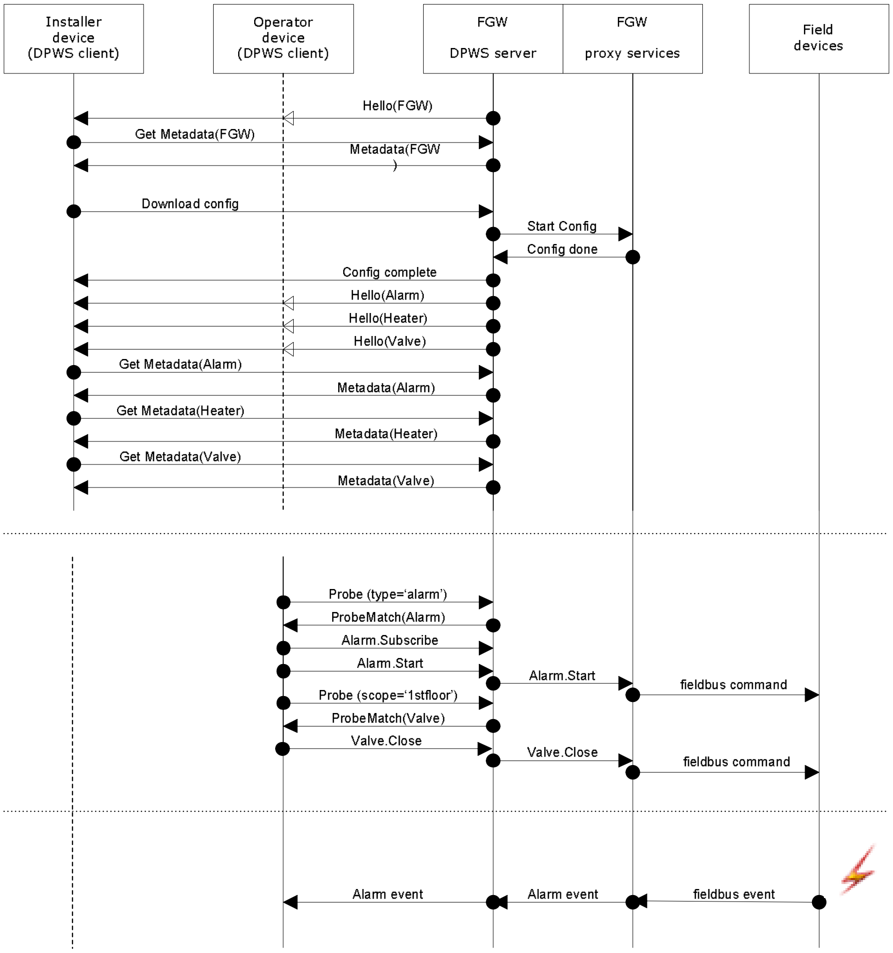

Fig. 4 – Device gatewaying sequence diagram

7. Conclusion

The convergence between computing and networking is revolutionizing the way communications are organized at the level of embedded devices. As the intelligence of computing and communications can thus be driven down to the lowest device levels, higher-level device communication paradigms supported by open Internet protocol standards are emerging. Homing in on this tendency, the *Devices Profile for Web Services* constitutes the most promising avenue in this new device communication space. It leverages the widespread adoption of service-oriented architectures and is fully aligned with Web Services standards, hence totally platform- and language-neutral. This approach enables novel device networking architectures and holds the promise of allowing seamless integration of device-level functionality into higher-level business processes, as well as integration of legacy technology through gateways. With its early implementation of this new device networking paradigm, the ITEA/SIRENA project [10] broke new ground for a wide range of application domains.

8. Acknowledgement

This work was supported by the ITEA/SIRENA project, funded in France by the Ministry of Economics, Finance and Industry. SIRENA was attributed the ITEA Achievement Award 2006. Its results are used by the ITEA/SODA [11] and the IST/SOCRADES [12] projects.

9. References

[1] R. Harrison, A. W. Colombo, A. A. West and S. M. Lee, "Reconfigurable Modular Automation Systems for Automotive Powertrain Manufacture", *2005 CIRP-sponsored 3rd International Conference on Reconfigurable Manufacturing*, Ann Arbor, Michigan, 10-12 May 2005.

[2] R. Harrison, A. W. Colombo, "Collaborative Automation. From Rigid Coupling Towards Dynamic Reconfigurable Production Systems". *Proc. of the IFAC World Control Congress 2005*, Prague, Czech Republic.

[3] A. W. Colombo, R. Schoop and R. Neubert, "Collaborative (Agent-Based) Factory Automation". In *"The Industrial Information Technology Handbook"*. R. Zurawski (Ed), chapter 109, CRC Press, 2004.

[4] F. Jammes, H. Smit, "Service-Oriented Paradigms in Industrial Automation", *IEEE Transactions on Industrial Informatics, Vol. 1(1), pp. 62-70*, February 2005.

[5] The UPnP Forum: http://www.upnp.org

[6] The Community Resource for Jini technology: http://www.jini.org

[7] S. Chan *et al*: *"Devices Profile for Web Services"*, http://specs.xmlsoap.org/ws/2006/02/devprof/devicesprofile.pdf

[8] D. Box, L. F. Cabrera, C. Kurt, *"An Introduction to the Web Services Architecture and its Specifications"*, http://msdn.microsoft.com/webservices/webservices/understanding/advancedwebservices/default.aspx?pull=/library/en-us/dnwebsrv/html/introwsa.asp

[9] The Efficient XML Working Group: http://www.w3c.org/XML/EXI

[10] The SIRENA project: http://www.sirena-itea.org

[11] The SODA project: http://www.soda-itea.org

[12] The SOCRADES project: http://www.socrades.eu