Service-oriented device communications using the Devices Profile for Web Services

Harm Smit – Schneider Electric, Corporate Research
harm.smit@ieee.org

AINA-07 – Niagara Falls, Canada, May 22, 2007
Agenda

- Service-oriented architectures (SOA) in general
  - Key concepts & architectural tenets
  - SOA based on Web Services

- SOA in the industrial sector
  - Challenges of future manufacturing systems
  - Promises of using SOA in the industrial world

- SOA for devices
  - Characteristics of the device space
  - Devices Profile for Web Services
  - Practical experience: SIRENA, SODA, SOCRADES projects
SOA in general, what & what not – outline

- SOA defined
- Key concepts of the service metaphor
- Architectural tenets
- Business-process abstraction level
- Services, components and objects
- SOA anti-patterns & caveats
- SOA implementation platforms
- Why is SOA important?
A Service Oriented Architecture (SOA) is a form of distributed systems architecture that is typically characterized by the following properties:

- **Logical view:** The service is an abstracted, logical view of actual programs, databases, business processes, etc., defined in terms of what it does, typically carrying out a business-level operation.

- **Message orientation:** The service is formally defined in terms of the messages exchanged between provider agents and requester agents, and not the properties of the agents themselves.

- **Description orientation:** A service is described by machine-processable metadata.

- **Granularity:** Services tend to use a small number of operations with relatively large and complex messages.

- **Network orientation:** Services tend to be oriented toward use over a network, though this is not an absolute requirement.

- **Platform neutral:** Messages are sent in a platform-neutral, standardized format delivered through the interfaces.

*WC3 Web Services Architecture Working Group (W3C Working Group Note 11, February 2004)*

Quite right, yet not quite complete!
Key concepts of the service metaphor

- SOA is a set of architectural principles allowing **autonomous** entities to **interoperate**
  - Autonomous entities:
    - are created, deployed, versioned, secured independently of each other
    - operate independently of their environment
    - provide self-contained functionality (useful even if used on their own)
  - Interoperability is a key objective
    - prerequisite for reuse
    - seamless connections
    - simple structure for simple tasks, complexity added as needed
    - mandatory use of international open standards
- **Separation of concerns**
  - functional interface vs. implementation details/platform specifics
- **Encapsulation of complexity**
  - entities handle their own problems internally
Interactions between communicating entities are typically:

- **Loosely-coupled**
  - Changes/replacements on one end should not affect the other end

- **Coarse-grained**, document-oriented
  - Necessary for scalability
  - Reduces network bandwidth and reliability requirements

- **Asynchronous**, message-based
  - Necessary
    - in the face of network delays and unpredictable response times (recall "the eight fallacies of distributed computing")
    - to support a variety of concurrency schemes

- These tenets are inter-related
  - Richer content enables looser coupling
  - Loose coupling enables asynchronous communication
Architectural tenets (2)

- **Service description through metadata**
  - **Contracts** describing message exchange patterns
    - behavior-oriented
  - **Schemas** describing message interchange formats
    - structure-oriented
  - **Policies** describing service capabilities and requirements
    - compatibility-oriented

- **Service aggregation/composition**
  - Composing elementary services into a higher-level service, higher-level services into business processes
  - Recursive composition, Russian dolls, holonic structuring
  - Composition facilitated by loose coupling & coarse-grained interfaces
Business-task abstraction level

- Services are aligned with business tasks, roles and rules
- Outside-in design approach
  - Focus on client's perspective
  - Interface designed on "need-to-know" basis
- Build to last & build to change
  - Individual services are expected to have a long lifetime
  - The system they belong to will change during that time
  - New services will appear well after deployment of the initial services
  - Therefore, service availability and stability over time are crucial
- Service autonomy is congruent with
  - Re-use of services across business processes
  - Explicit management of trust and failures at service boundaries
  - Mastering complexity
Services, components and objects

- **Objects**
  - programming level concept
  - usually live in the same process
  - underlie the implementation of components

- **Components**
  - compile, link and package programs into deployable units
  - usually live in different processes
  - underlie the implementation of services

- **Services**
  - provide abstract functionality across a network
  - usually live in different locations and environments
  - underlie the implementation of business processes
SOA anti-patterns

- SOA is neither a set of technologies nor a platform
  - SOA is merely a set of architectural principles
- SOA is not distributed objects by another name
  - SOA is a new paradigm
  - SOA builds on the lessons learnt with distributed objects
- SOA is not based on Remote Procedure Calls
  - SOA is based on message and document exchanges
  - "The message is the message"
- SOA is not wrapping services around objects
  - Many early "SOA tools" did just this (getters, setters…), resulting in tightly-coupled, brittle designs
SOA caveats

- SOA does not fall into place by magic
  - Moving to SOA requires a shift in mindset
    - Question: What should technology developers understand about SOA that they might not currently understand?
    
    Answer: "The core thing to understand is that "It's the message, stupid." The fundamental building block of global computing is the message and message exchange. Message-based computing is a different model that many enterprise developers are not too familiar with. It will take some time for them to learn how to effectively design and implement message-based services."

    Mark Hapner, Sun Microsystems, 2006

- Proper service design is key
  - Encompass the entire delivery lifecycle, not just the development phase
  - Concentrate on business logic
  - Respect the autonomy principle
    "Things should be made as simple as possible, but no simpler."

    Albert Einstein
SOA implementation platforms (1)

- Theoretically, SOA may be implemented in various ways
- Practically, **Web Services** constitute by far the most prominent technology
  - The introduction of the term SOA came about as a generalization of the concepts underlying Web Services
  - Technology-neutral Web Services are supported by all major vendors
  - Neutral, widely accepted standards for communication and data formats
  - Unifying power
- **Web Services** promote protocol-based integration
  - Coupling between applications is restricted to wire protocols
    - No programming language APIs
  - Protocols are composable
    - Protocols used either independently or in combination
    - Security and reliability can be added on
The core Web Services standards were jointly defined by major players like IBM, Microsoft and BEA

All major software vendors have weaved Web Services into their platforms, e.g.,

- IBM WebSphere
- Microsoft .NET and Windows Communication Framework (formerly Indigo)
- BEA AquaLogic
- Oracle Fusion Middleware
- SAP NetWeaver
- and many more
Gartner's Magic Quadrant for Web Services platforms, 2005

As of July 2005

source: Gartner Research
Why is SOA important? (1)

- Increased agility (flexibility, adaptation to change)
  - Alignment of resources with business tasks
  - Integration of heterogeneous technologies
  - Reduced vendor dependency
  - Elimination of technology islands
  - Wrap-and-reuse rather than rip-and-replace
  - Faster and more reliable design, reduced time-to-market
  - Incremental deployment
  - Improved decision-making capabilities
  - Scaling over time
  - Easier maintenance
Why is SOA important? (2)

- **Large-scale re-use**
  - Libraries of business-meaningful functionality, not tied to particular usage scenarios, composable to meet new business requirements
  - Accelerated delivery of new solutions/upgrading of existing solutions
  - Improved reliability

- **Improved comprehensibility and value visibility**
  - Alignment of services with business tasks: bridges the gap between business people and technicians
  - Services are business-meaningful both at design time and at run time, hence their value is easier to demonstrate

- **Overall: substantial cost benefits**
How important is SOA? (1)

- "If you don't yet know about SOA and how it will change your enterprise's IT architecture, you are placing yourself at a competitive disadvantage. By 2008, SOA will be a prevailing software engineering practice, ending the 40-year domination of monolithic software architecture (0.7 probability)."
  
  Yefim Natis, Gartner, 2003

- "We can probably characterize 2005 as the year SOA went from being some foggy notion inside a crystal ball to being a design pattern that gained wide acceptance."
  
  Dana Gardner, Interarbor Solutions, 2005

- "Everybody I'm now talking to is working on SOA. People have at least an inkling of what it means and they're starting to plan it."
  
  Anne Thomas Manes, Burton Group, 2005

- "If you're not doing SOA, you're in serious danger. Every sizable software vendor has stated its future roadmap is going to be SOA related. If you don't adopt SOA, you could be cutting yourself off and not be able to upgrade your current applications."

  Ron Schmelzer, ZapThink, 2005
How important is SOA? (2)

- "SOA is heading toward broad implementation in only a matter of a few years in the United States, regardless of organization size or vertical industry. 2006 will be the year of initial SOA project completion on a broad basis: not on a hit-or-miss trend, but through a rising tide of broad and deep adoption of SOA across the market."
  
  *Tom Dwyer, Yankee Group, 2005*

- "The world's largest companies can save up to $53 billion in information technology spending over the next five years by implementing service-oriented architectures."
  
  *William Mougayar, Aberdeen Group, 2006*

- "In the United States, SOA-related services will reach $8.6 billion in 2006, and grow to $34 billion by 2010."
  
  *Marianne Hedin, IDC, 2006*

- "Nine of every 10 companies are adopting or have adopted service-oriented architectures and will exit 2006 with SOA planning, design, and programming experience."
  
  *Peter Kastner, Aberdeen Group, 2006*

- "SOA will be used in more than 50% of new mission-critical operational applications and business processes designed in 2007, and in more than 80% in 2010."
  
  *Frank Kenney, Gartner, 2007*
SOA in the industrial sector – outline

- Industrial Automation market landscape
- Industrial Automation architectures
  - Problems with current architectures
  - Requirements for next-generation systems
  - Overall objectives
  - At stake: productivity & competitiveness
  - Previous endeavors
- SOA for Industrial Automation
  - Different levels of applying SOA
  - SOA as the unifying paradigm
Industrial Automation market landscape (1)

- Global market size over $200 billion
- Double-digit growth in Asia
- Covers process industries (2/3) & discrete industries (1/3)
  - Boundaries are blurring
- Automotive industry is the trendsetter for R&D innovation
Industrial Automation market landscape (2)

- Growth driven by software and services
- Frequently changing market demands & time-to-market pressure
- Installation & set-up account for 1/3 of total plant costs
- Maintenance down-time can be a showstopper
- Inflexible communications infrastructure using proprietary standards is a roadblock
- Rigid master-slave control structures
- Trend towards componentization and modularization
- Growing need for transparent interfacing between machine control functions and industrial data processing systems (MES, ERP, SCM)
Industrial Automation architectures (1)

- Problems with current architectures
  - Many proprietary or specific communication protocols
  - Islands of non interoperable technologies
  - Vendor dependency
  - Low economy-of-scale due to market fragmentation
  - High development costs and long time-to-market
  - Poor scalability
  - High maintenance costs due to poor diagnostics capabilities
Industrial Automation architectures (2)

- Requirements for next-generation automation systems
  - Dynamic plant reconfigurability
    - Run-time integration of heterogeneous yet interoperable system components
  - Cross-enterprise cooperation
  - Non-disruptive scalability
  - Fault-tolerance

- Overall objectives
  - Business agility from top to bottom
  - "Collaborative automation" across the supply chain

- At stake: efficiency -> productivity -> competitiveness
Industrial Automation architectures (3)

- Previous endeavors
  - Multi-agent systems
    - Emphasis on interactions between autonomous subsystems
  - Holonic systems
    - Recursive composition, Russian dolls
  - Promising directions, but limited adoption due to technological fragmentation

- Leveraging the SOA paradigm in industrial automation settings
  - Allows devices to become fully participatory members of their business environment
  - Entails a need to devise new forms of automation architectures
  - Creates tremendous opportunity for creating new types of services and for increasing efficiency and productivity

- Barriers
  - Conservatism, resistance to change
  - Shift in mindset
  - Large bulk of legacy technology
SOA for devices – outline

- Drivers & enablers
- Service-oriented interaction patterns at the device level
  - Addressing
  - Discovery
  - Description
  - Control
  - Eventing
- Service-oriented protocols for devices
  - Jini
  - UPnP
  - DPWS
SOA for devices – drivers & enablers (1)

- Devices can be made increasingly intelligent
  - Unprecedented horsepower in very tiny, low-cost components
    - Allows to drive intelligence down to the lowest device level
    - Enables self-contained, autonomous devices
- Next wave of Web penetration: device networking
  - 14 billion network-connected devices by 2010 (Forrester Research)
SOA for devices – drivers & enablers (2)

- Wireline & wireless **Ethernet** type networks, replacing fieldbuses
- Pervasive **TCP/IP** networking
- Ubiquitous use of **XML** for platform-neutral data exchange
- Emergence of **Web Services** technology
  - high-level, platform-neutral communications
- Result: enablement of **smart networked devices**
- Business relevance

"The transmission, harvesting and interpretation of device-based information as a basis for strategy and action will make every form of business dramatically more efficient and profitable than ever before"

*Harbor Research, 2003*

"The use of SOA at the device level will explode in 2008 or 2009"

*Gartner, 2006*
Device-level service-oriented interaction patterns (1)

- **Addressing**
  - **IPv4**
    - Managed network
      - DHCP
    - Unmanaged network
      - Auto-IP
  - **IPv6**
    - Managed network
      - DHCPv6
    - Unmanaged network
      - Auto-configuration
Device-level service-oriented interaction patterns (2)

- Discovery
  - Advertisement
  - Probing
Device-level service-oriented interaction patterns (3)

- Description

- Metadata transfer
Device-level service-oriented interaction patterns (4)

- Control
  - Command
  - [Response]
Device-level service-oriented interaction patterns (5)

- Eventing
  - Event subscription
  - Event notification
Service-oriented protocols for devices (1)

- Jini
  - Rooted in Java technology, despite later generalizations
  - Based on synchronous RPC paradigm
  - Relies on mobile code
    - requires resource-hungry JVM
  - Very limited adoption rate
Service-oriented protocols for devices (2)

- **UPnP**
  - First truly platform-neutral SOA framework for devices
  - Leverages Internet and Web technology
    - IP, HTTP, XML, SOAP
  - Predates advent of Web Services
  - Custom XML-based language for description
    - UPnP Template Language
  - Specific protocol for discovery
    - Simple Service Discovery Protocol (SSDP)
    - Rather chatty
  - Specific protocol for eventing
    - General Event Notification Architecture (GENA)
  - Originally designed for home networking
    - Not scalable beyond local networks
Service-oriented protocols for devices (3)

- Devices Profile for Web Services (DPWS)
  - First published in May 2004
  - Not yet submitted to a standards body
  - Overall concept similar to that of UPnP, but
    - Fully aligned with Web Services protocols
    - Takes into account lessons learned with UPnP
    - Designed for enterprise-wide integration scenarios
  - Strong unifying power
  - Natively integrated in Microsoft Vista/Windows Server 2008 (formerly “Longhorn”)
    - Will gradually become the preferred vehicle for connecting devices, instead of device drivers
Device-level SOA using DPWS – outline

- DPWS vision & principles
- DPWS protocol stack
- XML Schema
- SOAP messaging
- WSDL-based description
- WS-Addressing
- WS-Discovery
- WS-MetadataExchange & WS-Transfer
- WS-Eventing
- WS-Security
- WS-Policy
- WS-ReliableMessaging
- WS-Management
DPWS vision

- Web Services have become a simple, economical, widely available means for interaction between information systems
- Electronic devices are increasingly connected to standard networks
- DPWS defines extensions and restrictions required for using Web Services in embedded electronic devices
  - Taking into account their specific constraints: footprint, performance, ...
  - Fulfilling the most common needs: security, plug-and-play, asynchronous and event-driven exchanges
- Goals
  - Support the assembly of complex and adaptive systems based on autonomous and interoperable components
  - Allow such systems to be fully integrated with Web Services based applications operating in higher-level system contexts
DPWS principles

- DPWS promotes Web Services-based, peer-to-peer discovery and interactions between
  - devices, which act as service providers
  - service consumers, which can be other devices or clients running on a variety of platforms

- Services are provided by two types of endpoints:
  - Devices, which can host other services, play an important role during the discovery phase
  - Hosted services, which are deployed and discovered on devices, provide the functional behavior

- DPWS specifies a list of predefined services:
  - Discovery services
  - Metadata exchange services
  - Eventing services
A specification, led by Microsoft, based on:

**SOAP 1.2 and WSDL 1.1**

**WS-Addressing**: a SOAP extension for handling message addressing and routing in a transport-independent way

**WS-Discovery**: a Web Services-based discovery protocol

**WS-MetadataExchange & WS-Transfer**: metadata transfer

**WS-Eventing**: a Web Services-based publish/subscribe protocol

**WS-Security**: a SOAP extension for securing message exchanges

<table>
<thead>
<tr>
<th></th>
<th>WS-Discovery</th>
<th>WS-Eventing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WS-Addressing</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WS-MetadataExchange &amp; WS-Transfer</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WS-Security</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>WS-Policy</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>SOAP 1.2</th>
<th>WSDL 1.1, XML Schema</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UDP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HTTP 1.1</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>IPv4/IPv6</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
XML Schema – objectives

- Support typing of character data
  - Simple elements (character data with no nested elements)
  - Attributes
- Support specification of additional constraints
  - Value space (bounds, enumerations, regular expressions, size…) of typed contents
  - Flexible integrity constraints
- Fit within XML standards family
  - Uses XML syntax
  - Supports and uses namespaces
  - Refers to the XML Infoset (abstract XML representation)
- Support modularity and reuse
  - Support for partial validation
XML Schema – main principles

- Standardized by W3C in May 2001
- A schema defines, for a given target namespace:
  - Element names available in the namespace
  - Attributes associated to those elements
  - Element and attribute types, i.e. their textual or structured contents
  - Integrity constraints
- The schema is described using a specific XML vocabulary
  - Namespace: [http://www.w3.org/2001/XMLSchema](http://www.w3.org/2001/XMLSchema)
XML Schema – main components of a schema

- Two-stage declarations
  - **Type** (content model) definitions
    - Simple type for character data
    - Complex types for structured contents
  - Declarations of **elements** and **attributes** referring to those types
    - An element or attribute name is associated to a type
    - Attributes must be associated to simple types
    - Elements can be associated to simple or complex types
    - Elements and attributes can be declared either globally (at the schema top level) or locally within a complex type
    - Elements and attributes can also define their type in-line, instead of referring to a globally defined type

- Fosters reuse
  - Several elements or attributes may share the same type
  - New types can be derived from existing ones
The SOAP 1.2 specification

- W3C recommendation (24 June 2003)
  - Standard status
  - Note: SOAP is not an acronym anymore
    - SOAP wasn’t really about objects
- Two companion recommendations
  - WSDL 2.0: in Last Call stage of W3C standardization process
  - SOAP Message Transmission Optimization Mechanism (MTOM): W3C Recommendation since 25 January 2005
- Extends et clarifies SOAP 1.1 processing model and extensibility principles
  - Processing model
  - Message Exchange Patterns
  - Binding Framework: transport protocol bindings
- Discontinues controversial features of SOAP 1.1
  - RPC usage discouraged
  - Favors use of plain XML documents over data encoding
SOAP message content

- A SOAP message must conform to a restricted XML Infoset
- 'Envelope' root element
  - Container for the Header and Body elements
  - Usually holds declarations of namespaces used in the SOAP message
- 'Header' element
  - Optional, when present must be the first child of Envelope
  - Contains one or several child elements (header blocks), used to transmit "control" information
    - Session or transaction id, authentication, routing…
  - Provides for application-specific extensions of the protocol
- 'Body' element
  - Must appear as a child of Envelope
  - Contains zero or more elements specified by the application
    - These elements should be namespace-qualified
  - Contains a unique predefined 'Fault' element for error conditions
SOAP message structure

Protocol Headers

SOAP Envelope

SOAP Header
- Header block 1
- Header block N

SOAP Body

Protocol headers (HTTP, SMTP, ...)

<Envelope>

<Header>

<Body>
SOAP attachment support

- SOAP 1.2 uses MTOM (Message Transmission Optimization Mechanism)
  - MTOM considers that all binary data are part of the SOAP message Infoset
    - Different from previous approaches, in which attachments are external
  - Instead of being encoded using the base64 approach (standard XML serialization for binary data), the binary parts are extracted from the message and encoded as MIME multipart/related
    - Significant reduction in message size
  - Despite its incompatibility with previous approaches, MTOM is widely supported by the software vendors
  - Deprecates previous approaches, such as DIME
WSDL specifications

- **WSDL: Web Services Description Language**
- **WSDL 1.1: W3C Submission in 2000**
  - Never reached the recommendation status
  - Result of merging specifications from IBM and Microsoft
  - Used to describe
    - The abstract structure of messages supported by a service
    - Their concrete wire representation for a given protocol
    - The physical addresses of the service
- **WSDL 2.0: to become a W3C Recommendation in June 2007**
  - A significant and not backward compatible revision of version 1.1
  - More strictly separates the description of the abstract functionality of a service and the details of its concrete implementation
  - Still work in progress, as most Web Services tools currently only support WSDL 1.1
WSDL 1.1 document structure

- **XML Document**
  - **definitions**: root element

- **Main concepts**
  - **types**: data types definitions, based on XML Schema
  - **message**: abstract definition of a complex structure, based on types
  - **portType**: set of abstract operations
    - operation: abstract description of an action provided by the service, associated to input and/or output messages

- The above elements constitute abstract service interface descriptions; the below elements are related to concrete service implementations
  - **binding**: concrete protocol and encoding specification for a portType
    - operation: concrete description of corresponding abstract operation
  - **service**: set of ports
    - port: binding and address association
WS-Addressing specification

- W3C Recommendation since May 2006
- Initially, SOAP messages used protocol-specific means to specify addressing information
  - e.g., HTTP headers
- This created problems for
  - Interoperability across several protocols
  - Message integrity verification
- WS-Addressing defines
  - Endpoint references that convey the information needed to address Web Services endpoints
  - Message addressing properties (MAPs) that standardize the information found in transport protocols
- WS-Addressing turns SOAP messages into autonomous units of communication
WS-Addressing – Endpoint References (EPRs)

- EPRs provide a mechanism to support use cases not well covered by WSDL 1.1
  - Dynamic creation of endpoints for services
  - Identification of service instances created as a result of stateful interactions (instance = endpoint + session id)
- Endpoint reference properties
  - **Address** (URI): a logical or network address for the service
  - **Reference parameters** (any): zero or more parameters provided by the endpoint issuer, named by a qualified name and opaque to consumers
  - **Metadata** (any): optional information such as portType, service, port… as defined in the WSDL document for the service
Abstract properties added to a message:

- **Destination** (URI): address of the receiver of the message (mandatory)
- **Action** (URI): unique identifier for the semantics of the message (mandatory)
- **Reply endpoint** (EPR): reference of the receiver of replies. Mandatory when a reply is expected.
- **Message id** (URI): a unique identifier for a message. Mandatory when a reply is expected.
- **Relationship** (QName,URI): a pair of values that indicates how the message relates to another. The predefined relationship type "Reply" is used to refer in a reply to the message id property of the request.
- **Source endpoint** (EPR): Reference of the sender endpoint (optional)
- **Fault endpoint** (EPR): Reference of the receiver for faults related to the message (optional)

- ‘Destination’ and ‘Action’ are used to dispatch the message
- When an endpoint address is not resolvable, the following URI must be used to request protocol-specific handling:
WS-Addressing header representation in SOAP

```
<wsa:To>destination URI</wsa:To>
<wsa:From>source EPR</wsa:From>
<wsa:ReplyTo>reply EPR</wsa:ReplyTo>
<wsa:FaultTo>fault EPR</wsa:FaultTo>
<wsa:Action>action URI</wsa:Action>
<wsa:MessageID>message id</wsa:MessageID>
<wsa:RelatesTo RelationshipType="wsa:Reply">
    request message id
</wsa:RelatesTo>
```

wsa:Action is made to correspond to input, output and fault messages of portType operations
WS-Discovery – Plug-and-play discovery protocol

- Specification initially proposed in February 2004
  - Latest version April 2005
  - Not yet proposed for standardization
- Multicast discovery protocol to locate services/devices
  - Based on SOAP-over-UDP and use of multicasting
  - Defines announcement & probing protocols to reduce the need for polling
  - Expressed in terms of "target services"; with DPWS, a device is a target service
  - Defines a Discovery Proxy mechanism, to scale to a large number of endpoints by suppressing the multicast behavior
- Discovery messages
  - Hello and Bye: "one-way" multicast announcement messages produced by devices joining and leaving the network
  - Probe and ProbeMatch: multicast request and unicast reply exchanged between a client and a device matching the probe
  - Resolve and ResolveMatch: address resolution protocol based on a multicast request and unicast reply exchanged between a client and a device matching the logical address
- Message size constraints
  - Multicast messages (Hello, Probe, Resolve) should not be larger than the MTU (typically 1500 bytes)
WS-Discovery message contents

- **Device metadata included in** **Hello**, **ProbeMatch** and **ResolveMatch** **messages**
  - **Types** (QNames): Abstract or functional types (WSDL portTypes) describing the messages that can be handled by the device.
  - **Scopes** (URIs): Additional, application-defined information used to identify the device (e.g. location).
  - **Addresses**: Physical network addresses for the device.
  - **Version**: Integer which is increased each time metadata information for the device changes. Useful for cache management.

- **Probe** **message contents**
  - **Types** (QNames): List of types that must be supported by a matching device.
  - **Scopes** (URIs): List of scopes that must be matched by the scopes of a matching device.

- **Resolve** **message contents**
  - **Endpoint reference** (EPR) containing the device's logical address.
WS-Discovery message exchanges

1. A Target Service sends a multicast Hello when it joins the network. Clients listen to the multicast Hello.
2. Likewise, a Target Service may receive a multicast Probe from a Client at any time.
3. The Target Service sends a unicast Probe Match (PM) to the Client if the Target Service matches a Probe.
4. Similarly, a Target Service may receive a multicast Resolve from a Client at any time.
5. The Target Service sends a unicast Resolve Match (RM) to the Client if it is the target of a Resolve.
6. When a Target Service leaves the network, it makes an effort to send a multicast Bye. Clients listen to the multicast Bye.
WS-Discovery – Discovery Proxy

- Extends network reach beyond single subnet
- Enables multicast traffic suppression
- Principle of operation:
  - The DP registers all Hello and Bye messages from Target Services (devices)
  - When a Client senses the presence of a DP (receiving a unicast Hello in response to a multicast Probe/Resolve)
    - For subsequent searches, the Client sends unicast Probe/Resolve messages to the DP
    - The DP responds with ProbeMatch/ResolveMatch messages, on behalf of the Target Services (devices) it has detected
  - In addition, the DP and the Client(s) may exchange higher-order discovery protocol messages
Metadata transfer

- Discovery metadata must be completed by another mechanism
  - Discovery messages are synthetic, due to message size constraints
  - They describe device metadata, but not hosted services metadata
- Additional device-specific metadata defined by DPWS
  - Model and device characteristics
    - Model: Manufacturer, model name…
    - Device: friendly name, serial number…
  - List of hosted services
    - Each hosted service is described by its EPR and its types (usually WSDL portTypes)
  - Optionally, WSDL descriptions or URLs
- DPWS uses
  - WS-MetadataExchange to describe these metadata
  - WS-Transfer (Get) to retrieve these metadata
WS-Eventing specification

- Specification submitted to W3C in March 2006
  - Initial version published in January 2004
- Generic publish/subscribe mechanism to support application-defined events
  - To reduce polling needs in device networks
    (especially important for low-power wireless network nodes)
  - To promote loose coupling between event producers and consumers
- WS-Eventing extends the description and semantics of WSDL portTypes
  - Addition of a wse:EventSource attribute in the wsdl:portType element
  - Event source portTypes automatically feature a Subscribe operation
  - Notification and Solicit-Response operations are exposed by the portType as event types
  - Each event is identified by its corresponding action, conveyed through WS-Addressing
WS-Eventing – Subscriptions and notifications

- Subscription to an event is initiated by a client sending a Subscribe message to an endpoint supporting an event source portType.

- An event subscription is specified by:
  - The event source endpoint, which receives the Subscribe request.
  - The subscriber endpoint.
  - The subscription duration.
  - Optional event filters, based on the action associated to each event in DPWS.

- Subscriptions are managed by a Subscription Manager:
  - Clients are provided with an EPR (Subscription Manager endpoint + subscription id) in the Subscribe response.
  - This EPR supports the Renew, GetStatus and Unsubscribe operations.
WS-Eventing – Extensions & evolutions (1)

- **WS-Eventing is a base standard**
  - Open to extensions in various directions
- **Events delivery to subscribers uses a simple point-to-point push mechanism**
  - This doesn't scale well for large numbers of subscribers
  - Event brokers may be used to improve robustness & scalability
  - They may further:
    - aggregate and redistribute notifications, e.g. according to "topics" corresponding to application categories
    - act as filters
  - In local environments, multicast notifications may also be considered
WS-Eventing – Extensions & evolutions (2)

- Competing specification WS-Notification, related to WS-Resource Framework (WSRF)
- Conciliation being elaborated by a working group with members from HP, IBM, Intel and Microsoft (set up in March 2006)
- Resulting standard: WS-EventNotification
  - no timeline set as yet
WS-Security specifications

- WS-Security is a family of specifications in support of end-to-end message integrity, confidentiality and authentication.
- Provides corresponding enhancements to SOAP messaging
  - Differs from traditional security mechanisms based on encrypted communication between two endpoints (SSL, HTTPS)
  - Supports partial message encryption
  - Supports confidentiality across several intermediaries
- Specifies how to associate security tokens with messages
- Uses
  - XML Signature (W3C) for message integrity
  - XML Encryption (W3C) for message confidentiality
- Further comprises
  - WS-SecurityPolicy: policy assertion for expressing security requirements
  - WS-Trust: management of security tokens
  - WS-SecureConversation (management of a security context between communicating entities using shared secrets)
WS-Security use in DPWS

- DPWS prescribes a limited, optional use of WS-Security
  - Limited to discovery mechanisms, for integrity and authentication
  - Using a customized format, called compact signature, to reduce the size of a signed Hello message
- Once a device and its client have identified themselves, they can establish a secure communication channel
  - Using TLS/SSL and HTTPS
- May evolve as device computing resources become more powerful
WS-Policy

- Specification submitted to W3C in April 2006
- Complements WSDL
- Allows to express policies associated to a Web Service
  - in the form of "policy assertions"
  - reflecting properties, capabilities, limitations, behavioral characteristics, preferences or requirements, e.g. QoS requirements or characteristics
- Currently only used by DPWS to identify the device profile as such
- May be used in the future for many other purposes
WS-ReliableMessaging

- Submitted to OASIS in June 2005, currently being balloted
- Will oust competing WS-Reliability (also OASIS)
- Addresses the need for predictable, guaranteed, once-only, confirmed, validated delivery of SOAP messages
- Not actually included in DPWS, but may be composed with it
  - in particular to guarantee delivery of vital messages, such as security alerts
  - indispensable in enterprise environments
WS-Management (1)

- Standardized by DMTF in April 2006
- Complementary to DPWS
- Lightweight standard for managing services
- Functionality similar to that of SNMP
- Relies on WS-Eventing for event notifications
- Foundation for enabling devices to provide a high-level management interface, e.g. for management of
  - Fault diagnosis
  - Configuration
  - Performance measurements
Competing WSDM (Web Services Distributed Management) standards, related to WS-Resource Framework
- Management using Web Services (MUWS)
- Management of Web Services (MOWS)
Conciliation being elaborated by a working group with members from HP, IBM, Intel and Microsoft (set up in March 2006)
- sometimes referred to as WS-Unified Management
- no timeline set as yet
**Vision**

The SIRENA vision: SOA for devices

- **Sub-system**
  - Device
  - Automatic discovery and start-up
  - Direct interoperable communication between devices
  - Web Service
  - Physical interfaces
  - Common universal access interface

- **IP-based network**

**www.sirena-itea.org**
Main results

- Definition of SIRENA Framework based on DPWS
- Development of DPWS software component for embedded devices
- Development of demonstrators for various application domains, showing:
  - benefits of high-level, platform-neutral communication infrastructure
  - interoperability of heterogeneous devices of different make
  - composition of devices into higher-level devices
  - plug-and-play device connectivity
  - uncoupling of logical and physical device architectures
  - encapsulation of legacy technology
  - ease of integration with enterprise-level processes
  - simplified application development
- Results exploited in newly started large-scale European R&D projects
- Technology being transferred into commercial products

Received the **ITEA Achievement Award 2006** for outstanding contribution to the ITEA programme
Early DPWS implementation

- Software component developed by Schneider Electric
- Ported to Linux, Solaris, Windows CE and XP, VxWorks, ThreadX, Quadros, embOS

- Less than 200 Kbytes on ARM based processors
- The world's first implementation of DPWS for embedded devices!
SIRENA experimental setup (1)

- Dose-maker device
  - Service-oriented approach
SIRENA experimental setup (2)

- Service implementation
  - Formal service description (WSDL, XML Schema)
  - Service orchestration aspects
- Benefits demonstrated
  - Plug-and-play connectivity
  - Device substitution
  - Ease of integration with higher-level systems (MES)
  - Encapsulation of legacy technology
Performance aspects of DPWS usage

- **Memory footprint**
  - 500 KB of Flash ROM + 100 KB of RAM (all included)
  - Various optimizations planned

- **Message processing time**
  - Processing time for sending a request message and handling the response message
    - 10 ms on a 400 MHz Intel PXA255 (XScale) platform running WinCE
    - 39 ms on a 44 MHz ARM7 platform running ThreadX
  - Can be improved by at least an order of magnitude by:
    - using binary instead of textual encoding of SOAP messages
    - using SOAP directly on top of TCP/IP
    - various software optimizations (memory management, string manipulation)

- **Network latency**
  - Negligible on a 100 Mbps industrial Ethernet
  - May be a concern on low-speed wireless networks
SIRENA follow-up

- Implementation of a Java version of the DPWS component
  - Targets JavaME/CDC environments
  - Delivered in Q3 2006

- Open-sourcing of Schneider Electric’s DPWS component
  - Since January 2007
  - Both C and Java versions, LGPL license
  - Provisionally hosted on SODA Web site (www.soda-itea.org)
  - Collaborative, configuration-controlled environment to be opened shortly
  - Components from other partners to be integrated as well

- Ongoing modularization of DPWS component
  - Moving away from gSOAP dependency
  - Component-based architecture
  - Module plug-in mechanism (e.g., for adding security, reliable messaging)
  - Implementation of binary XML encoding being standardized by W3C (Efficient XML)
The SODA project

- Tooling support

- Management framework based on WS-Management
- Support of WS-Security

www.soda-itea.org
The SOCRADES project

- Comprises all major European players in industrial automation: Schneider Electric, ABB, Siemens, SAP,…
- Both discrete and process automation
- Use of wireless sensor/actuator networks in industrial settings
- Integration with enterprise management systems
  - Web Services based ERP, in particular SAP NetWeaver
- Integration with agent-based systems
- Service orchestration engine
- Semantic Web Services support

www.socrades.eu
Gatewaysing low-level devices (1)

- Scenario for demonstrating how fieldbus-connected devices can be made to appear as DPWS-enabled devices:
  - Fieldbus Gateway (FGW) manages field devices:
    - Alarm
    - Heater
    - Valve
  - modeling them as DPWS-enabled devices
  - Seamless migration to real DPWS-enabled devices

- Similar approach can be used to connect:
  - legacy devices
  - other low-level devices that have insufficient resources to be DPWS-enabled, e.g. RFID tags
Gatewaysing low-level devices (2)

- Installer device bootstraps gateway configuration
- Gateway emulates field device announcements
- User device discovers field devices
- Gateway notifies event detected by field device
Questions?
Thanks for your attention