EUROPEAN COMMISSION

Thematic Priority:
SIXTH FRAMEWORK PROGRAM

Priority 2.5.3
INFORMATION SOCIETY TECHNOLOGIES
Unit G3 Embedded Systems

Project Acronym:

SOCRADES

Project Full Title:

Service-Oriented Cross-layer infRAstructure for Distributed smart Embedded devices

Proposal/Contract No: EU FP6 IST-5-034116 IP SOCRADES

Deliverable D5.3.2

Generic Services Specification

Status:	**Final**
Version:	**V1.0**[1]
Dissemination Level[2]**:**	**CONFIDENTIAL**
Date:	**08.04.2009**

Organization Name of the Lead Contractor for this Deliverable: **Schneider Electric**

[1] V0.x before peer-review approval, V1.0 at the approval, V1.x minor revisions, V2.0 major revision
[2] See Annex for explanation of Dissemination Levels, as defined in the DoW

Status Description:

Scheduled completion date[3]:	M30 (End of February 2009)	Actual completion date[4]:	08.04.2009
Short document description:	The goal of this document is to specify a set of common and generic services that are available on any device, and being application domain independent.		
Author(s) deliverable:	G. Cândido	**Report/deliverable classification:** [x] Deliverable [ ] Three-Monthly Activity Report [ ] Six-Monthly Activity Report	
Partner Contributions / Peer reviews	[x] [x] Schneider Electric Automation [ ] [ ] ABB [ ] [ ] APS GmbH [ ] [ ] Boliden AB [ ] [ ] FlexLink Automation Oy. [ ] [ ] Institut f. Automation und Kommunikation e.V. Magdeburg [ ] [ ] Kungliga Tekniska Högskolan	[ ] [ ] Loughborough University [ ] [ ] Luleå University of Technology [ ] [ ] Politecnico di Milano [ ] [ ] SAP AG [ ] [ ] Siemens AG [ ] [ ] Tampere University of Technology [ ] [ ] Jaguar Cars Ltd. [ ] [ ] ARM Ltd. [x] [x] Schneider Electric Industries	
Peer review approval :	[x] Approved [ ] Rejected (improve as specified hereunder)	Date:	08.04.2009
Suggested improvements:			

Version History:

Version:	Date:	Comments, Changes, Status:	Person(s)[5]:
V0.1	January 22nd, 2009	Initial Draft	F. Depeisses (SE)
V0.2	February 3rd, 2009	Update/Completion of existing chapters Added Built-in Services, Persistence and Security chapters	Gonçalo Cândido (SE)
V1.0	March 24th, 2009	Update/Completion based on SE team inputs	Gonçalo Cândido (SE)

[3] As defined in the DoW

[4] Scheduled date for approval

[5] A list of company short tags can be found in DoW

Table of Contents:

List of Figures:

List of Tables:

1. Introduction

1.1. Preamble

The goal of this document is to specify a set of **common** and **generic** services that are available on any device, and being application domain independent. DPWS is the common web services middleware and profile for devices, the DPWS specification [1] defines two fundamental elements: the device and its hosted services. The device are the discoverable entity on the network, device can host services (they provide the functional behaviour).

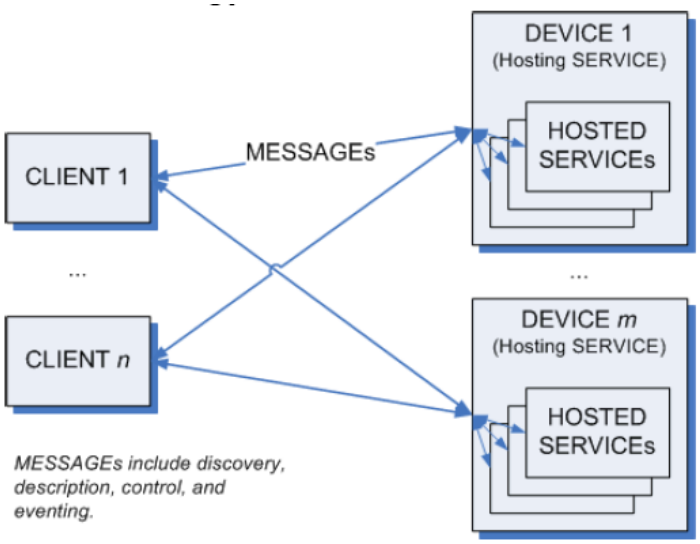


Figure 1: Device and hosted services

We consider those two elements: "device and hosted services" as fundamental resources of the DPWS infrastructure/middleware. We intend to extend the DPWS infrastructure for deploying and managing in a coordinated fashion both resources.

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

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

The integrator has also the ability to manage the complete lifecycle of these resources in accordance with the evolution of production goals.

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

1.2. How to specify these generics services

Two different approaches could be used for realizing such generic services, controlling the lifecycle of devices and services: either use new proprietary Web services, or use the standard WS-Management services with proprietary resources.

It is necessary to introduce first WS-Management. **WS-Management** specification [2] describes a general WS* based protocol for managing systems such as PCs, servers, devices, Web Services, applications, and other manageable entities. To promote interoperability between management applications and managed resources, WS-Management identifies a core set of Web Service specifications and usage requirements that expose a common set of operations central to all systems management. This comprises the abilities to:

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

In each of these areas of scope, the WS-Management specification defines minimal implementation requirements for conformant Web Service implementations. The implementation is free to extend beyond this set of operations, and may also choose not to support one or more areas of functionality listed above if that functionality is not appropriate to the target device or system. The WS-Management specification defines a standard form to access resources, but it doesn't define a resource description model. So, the user is free to define the XML resource description model that best fits its application, and share it so that others can interact with it.

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

As far as these generics services have the objective to manage and control the lifecycle of devices and services, i.e. install, uninstall, start and stop. WS-Management specification fits really this need: the resources being devices and services. For example, the create operation from WS-Management will allow the install phase of the resource, the delete operation, the un-install phase of the resource. **Furthermore, the specification is open to extend this core set of operations (create, delete, get, and put) with custom operations like start, stop, etc**. Using WS-Management standard can open our devices to external tools since they are compliant with an open standard.

2. Resources

2.1. Introduction

Before describing the resources, we could distinguish two phases in their lifecycle: the deployment phase and the activation phase. The deployment phase is usually performed in two steps:

- The service implementation code is first uploaded on the device platform, using either a Web Service or some platform-specific mechanism: depending on the technologies used by the platform and the service implementation, the implementation may be usable immediately (e.g. when using interpreted code or some dynamic code loading and linking mechanism) or may require a reboot (e.g. when the service implementation must be integrated with the platform firmware).

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

This proposed model requires a new resource:

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

In the figure below, the relationship between the physical device, logical device, hosted service and service class resources are presented in a UML diagram.

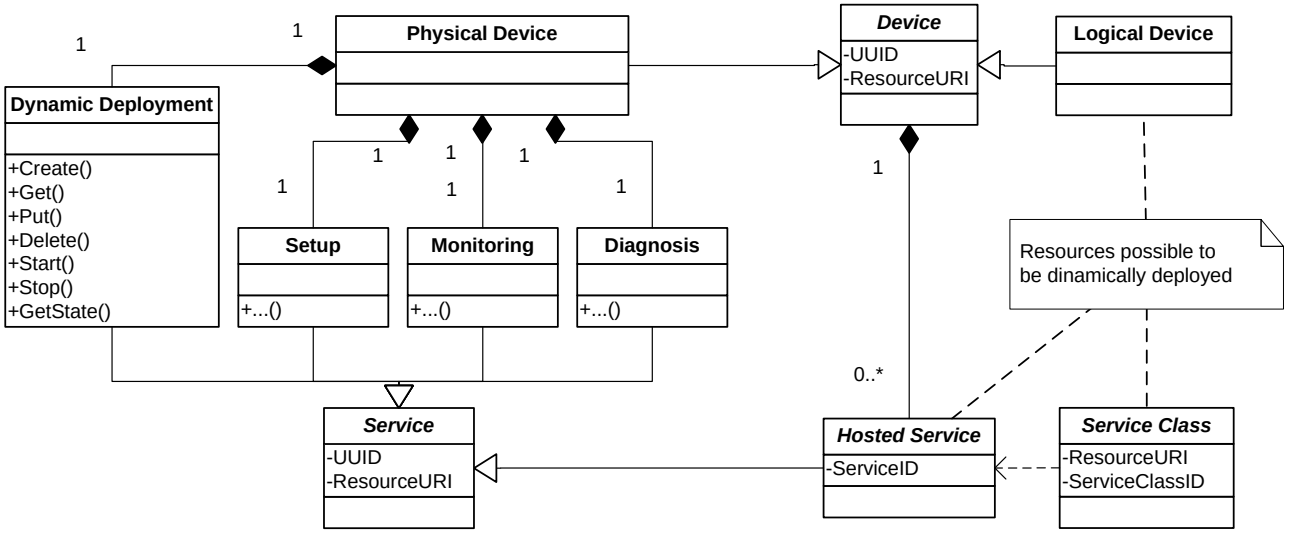


Figure 2: Relationship between physical device, hosted logical device, hosted service and service class

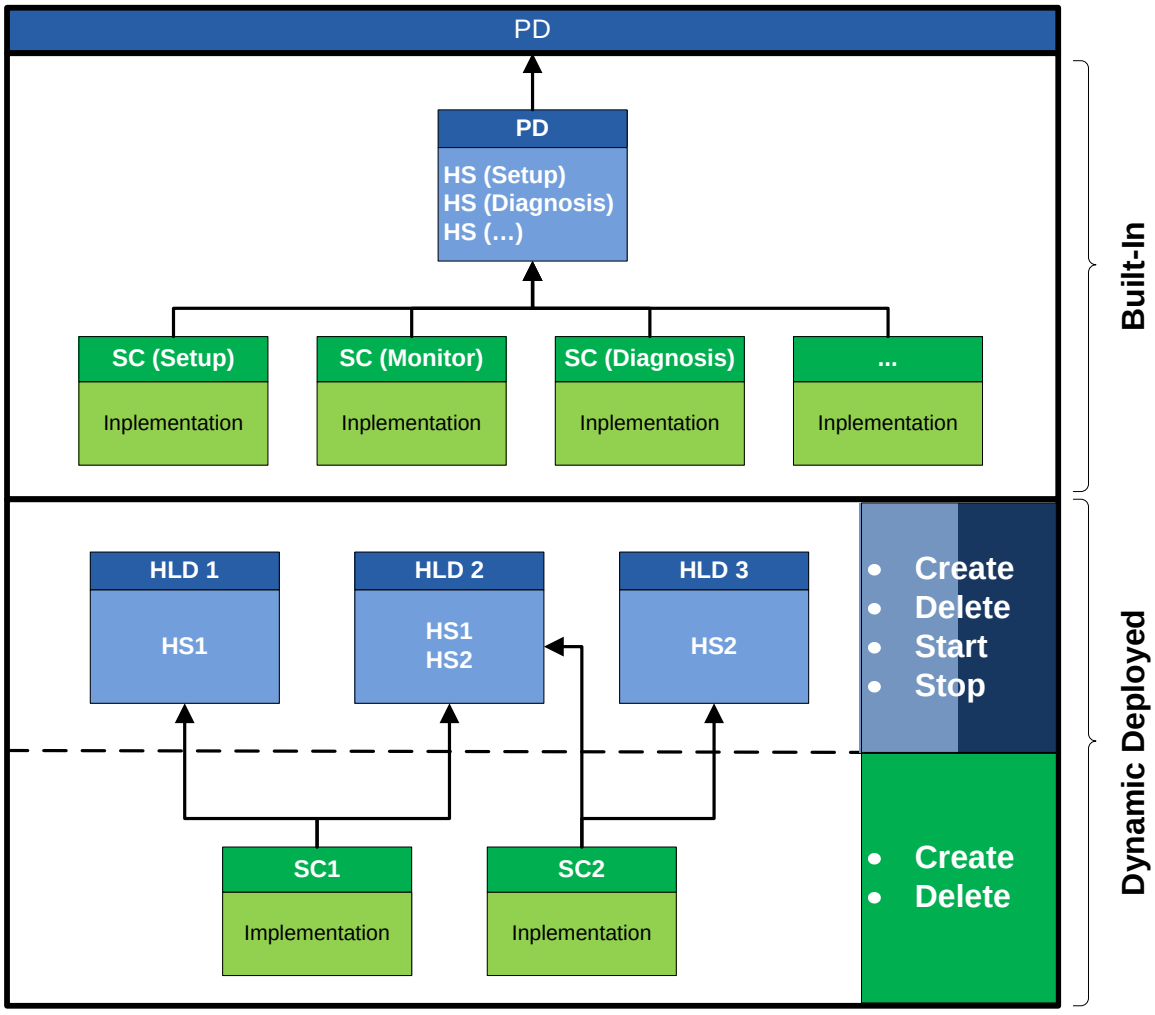
The proposed architecture defines several elements with different objectives and functionalities.

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

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

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

The next figure shows an example of a physical device with its own built-in and dynamically deployed resources. The figure also focuses over the operations available to the different types of resources (physical device, logical device, hosted service and service class).



Legend:
PD – Physical Device
HLD – Hosted Logical Device
HS – Hosted service
SC – Service Class

Figure 3: Resources layers, relations and available operations

Each one of these element will be further detailed on the next sub-chapters.

2.2. Service class resource description

The service class model is used to describe service implementations. It is derived from the SCA implementation model. A service implementation is characterized by the set of service interfaces (portTypes described in WSDL files) it provides, the set of references to services that it may require and configurable properties that control its behaviour. The service class model contains an additional element (Implementation) used as a placeholder for technology-specific implementation data: it could be a class name and optionally a jar file in Java, an entry point and a DLL on Windows or Linux, or a program and some initialisation data for interpreters. Each technology will define the actual element structure that is required to hold the implementation data.

The outline of the ServiceClass element is:

```
<dd:ServiceClass classId="xs:anyURI" ...>
  <dd:Interface name="xs:NCName"? type="xs:QName"/>*
  <dd:Reference name="xs:NCName" type="xs:QName" mustSupply="xs:boolean"?/>*
  <dd:Property name="xs:NCName" type="xs:QName"
               mustSupply="xs:boolean"? multiple="xs:boolean"?>*
    default-property-value?
  </dd:Property>
  <dd:WSDLInfo targetNamespace="xs:anyURI" location="xs:anyURI"/>*
  <dd:Implementation/>
</dd:ServiceClass>
```

The following describes additional constraints on the above outline:

/dd:ServiceClass/@classId

The class id must be a unique URI used to identify univocally a service class within a device.

/dd:ServiceClass/dd:Interface/@name

The name must be unique within the service class.

/dd:ServiceClass/dd:Interface/@type

The type must be a QName referring to a portType defined in a WSDL file. The QName is composed of the WSDL targetNamespace (or more precisely its associated prefix) and the portType name.

/dd:ServiceClass/dd:Reference/@name

The name must be unique within the service class.

/dd:ServiceClass/dd:Reference/@type

The type must be a QName referring to a portType defined in a WSDL file. The QName is composed of the WSDL targetNamespace (or more precisely its associated prefix) and the portType name.

/dd:ServiceClass/dd:Reference/@mustSupply

mustSupply specifies whether the reference must be bound at deployment time. Its default value is true.

/dd:ServiceClass/dd:Property/@name

The name must be unique within the service class.

/dd:ServiceClass/dd:Property/@type

The type must be a QName referring to a XML Schema type (either predefined or user-defined). The QName is composed of the XML Schema targetNamespace (or more precisely its associated prefix) and the local type name. *In a first version, only predefined simple types will be supported.*

/dd:ServiceClass/dd:Property/@mustSupply

mustSupply specifies whether the property must be bound at deployment time. Its default value is true. When it is false, a default property value must be provided.

/dd:ServiceClass/dd:Property/@multiple

multiple specifies whether the property can receive multiple values at deployment time. Its default value is false.

/dd:ServiceClass/dd:Property/*default-property-value*

It must be a simple or complex type content, valid with respect to the associated property type. It must only be provided when the mustSupply attribute is set to false.

2.3. Device resource description

The device model is used to describe DPWS devices. It is derived from the information useful during the discovery and metadata exchange process. It also contains subsections describing hosted services, which are instances of the service classes described above. This device model will be used to describe both physical and hosted logical devices. From a DPWS specification point of view they are the same, however its purposes are entirely different: a physical device will represent the physical device itself, while a logical device represents a part of the deployed application that exists in that physical device.

The outline of the Device element is:

```
<dd:Device ...>
  <dd:Address>xs:anyURI</dd:Address>?
  <dd:Types>list of xs:QName</dd:Types>?
  <dd:Scopes>list of xs:anyURI</dd:Scopes>?
  <dp:ThisModel ...>
    <dp:Manufacturer ...>xs:string</dp:Manufacturer>+
    <dp:ManufacturerUrl>xs:anyURI</dp:ManufacturerUrl>?
    <dp:ModelName ...>xs:string</dp:ModelName>+
    <dp:ModelNumber>xs:string</dp:ModelNumber>?
    <dp:ModelUrl>xs:anyURI</dp:ModelUrl>?
    <dp:PresentationUrl>xs:anyURI</dp:PresentationUrl>?
    ...
  </dp:ThisModel>
  <dp:ThisDevice ...>
    <dp:FriendlyName ...>xs:string</dp:FriendlyName>+
    <dp:FirmwareVersion>xs:string</dp:FirmwareVersion>?
    <dp:SerialNumber>xs:string</dp:SerialNumber>?
    ...
  </dp:ThisDevice>
  <dd:Service serviceId="xs:anyURI"?>*
    <dd:ServiceClass classId="xs:anyURI"/>
    <dd:ServicePort>*
      <wsa:Address>xs:anyURI</wsa:Address>
      <wsa:ReferenceParameters>...</wsa:ReferenceParameters>?
      <wsa:Metadata>...</wsa:Metadata>?
    </dd:ServicePort>
    <dd:Reference name="xs:NCName">*
    [
      <wsa:EndpointReference>
        <wsa:Address>xs:anyURI</wsa:Address>
        <wsa:ReferenceParameters>...</wsa:ReferenceParameters>?
      </wsa:EndpointReference>
      |
```

```
      <dd:DiscoveryHints onMultipleMatches="pickOne|fail"
                         bindingTime="deployment|runtime"
                         onReferenceLost="retry|ignore|fail"
                         serviceId="xs:anyURI"?>
        <dd:Hint>+
          <dd:Types>list of xs:QName</dd:Types>?
          <dd:Scopes>list of xs:anyURI</dd:Scopes>?
        </dd:Hint>
      </dd:DiscoveryHints>
    ]
    </dd:Reference>
    <dd:PropertyValue name="xs:NCName">*
      ...
    </dd:PropertyValue>
  </dd:Service>
</dd:Device>
```

The following describes additional constraints on the above outline:

/dd:Device/dd:Address

UUID for the device. When omitted, will be generated for a given device. This value is immutable.

/dd:Device/dd:Types

Set of types for the device. The dp:Device type is implicitly added to this set.

/dd:Device/dd:Scopes

Set of scopes for the device. When empty or absent, the default value is a set containing "http://schemas.xmlsoap.org/ws/2005/04/discovery/adhoc".

/dd:Device/dp:ThisModel

The DPWS ThisModel metadata information. See specification for details.

/dd:Device/dp:ThisDevice

The DPWS ThisDevice metadata information. See specification for details.

/dd:Device/dd:Service/@serviceId

URI for the service. When omitted, implied value is the classId of the service class.

/dd:Device/dd:Service/dd:ServiceClass/@classId

Reference to the implementation class for the service.

/dd:Device/dd:Service/dd:ServicePort

When omitted, the service will use a generated physical network address.

/dd:Device/dd:Service/dd:ServicePort/wsa:Address

Absolute or relative URI corresponding to a physical network address of the service. When relative, the service will use the device network address.

/dd:Device/dd:Service/dd:ServicePort/wsa:ReferenceParameters

Not used in first version.

/dd:Device/dd:Service/dd:ServicePort/wsa:Metadata

Not used in first version.

/dd:Device/dd:Service/dd:Reference/@name

Name of the reference that is being set. Should match one of the references defined in the service class.

/dd:Device/dd:Service/dd:Reference/wsa:EndpointReference

Used when the reference can be bound statically to a target.

/dd:Device/dd:Service/dd:Reference/wsa:EndpointReference/wsa:Address

Network address for the reference target.

/dd:Device/dd:Service/dd:Reference/wsa:EndpointReference/wsa:ReferenceParameters

Not used in first version.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints

Used when the reference will be bound using the WS-Discovery mechanisms. Once the device EPR is resolved the type defined in the service class reference should be used to select the referenceEPR.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/@onMultipleMatches

Specifies the behaviour when more than one reference is found. Defaults to *pickOne*.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/@bindingTime

Specifies whether the reference must be bound at deployment or run time. Defaults to *deployment*.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/@onReferenceLost

Specifies the expected behaviour when a bound reference is not available any more. Defaults to *retry*.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/@serviceId

If present, should be used to select the reference EPR.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/dd:Hint

Lookup parameters to be used for the discovery lookup request. More than one set of parameters can be provided, and will be considered as a union.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/dd:Hint/dd:Types

Set of types to be used for the discovery lookup request.

/dd:Device/dd:Service/dd:Reference/dd:DiscoveryHints/dd:Hint/dd:Scopes

Set of scopes to be used for the discovery lookup request.

/dd:Device/dd:Service/PropertyValue

The value of the property that is being set. Must be compliant with the property type of the corresponding property declared in the service class.

/dd:Device/dd:Service/PropertyValue/@name

The name of the property that is being set.

2.4. Service resource

In this context, the service resource will simply refer to the service class that was registered to one device – hosted service. From that point, the device will make it available as one of its services.

3. Resources Life Cycle

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

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

3.1. Resources identifiers

The resource identification leverages the default addressing model of WS-Management [2], it uses:
wsa:To (required): the transport address of the service
wsman:ResourceURI (required): the URI of the resource class representation or instance representation
wsman:SelectorSet (optional): identifies or selects the resource instance to be accessed if more than one instance of a resource class exists.

Service class identifiers:

- wsman:ResourceURI: http://www.soa4d.org/dpwscore/2007/08/dyndepl/resources/ServiceClass
- wsman:SelectorSet: dp:ServiceClassId (identifier of the service class)

Device identifiers:

- wsman:ResourceURI: http://www.soa4d.org/dpwscore/2007/08/dyndepl/resources/Device
- wsman:SelectorSet: dd:Address (uuid of the device)

Service identifiers:

- wsman:ResourceURI: http://www.soa4d.org/dpwscore/2007/08/dyndepl/resources/Service
- wsman:SelectorSet: dd:Address (uuid of the device), dp:ServiceId (identifier of the service)

3.2. Device and Service Resources state

Device and Service can be in one of the following states:

INSTALLED – The resource has been successfully installed.

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

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

ON – The resource has successfully started and is running.

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

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

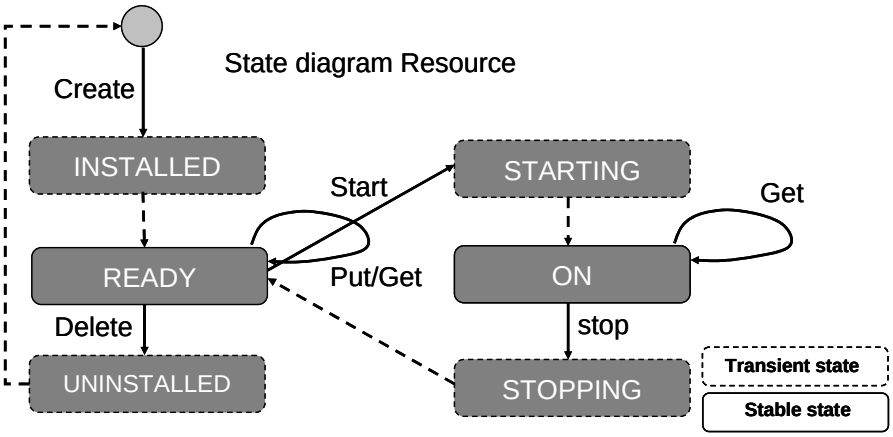


Figure 4: State Diagram Resource

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

3.3. Service class resource state

Service class can be in one of the following states:

INSTALLED – The resource has been successfully installed.

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

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

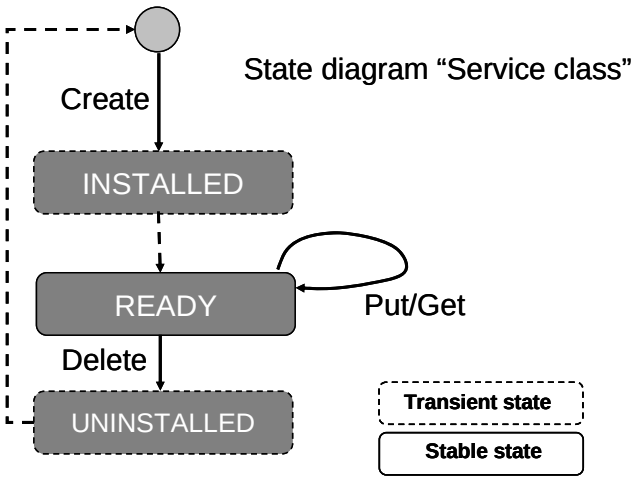


Figure 5: State Diagram “Service Class”

When a service class is created, first it is stored in the persistent storage of the device platform, and then the registry of the DPWS middleware is initiated accordingly. It remains there until it is explicitly deleted.

3.4. Resources operations

All resources support the following operations:

wxf:Create: this operation creates the resource; the wsman:ResourceURI is required at creation; the wsman:SelectorSet (uuid of the device) is also required for the service resource creation; Multiple creation could be supported according to the device platform; if no more resource could be created, the service should return a dd:NoMoreSpace fault.

wxf:Get: this operation retrieves the resource representations; the wsman:ResourceURI and wsman:SelectorSet are required

wxf:Put: this operation updates the resource representations; the wsman:ResourceURI and wsman:SelectorSet are required.

wxf:Delete: this operation deletes resource instances; the wsman:ResourceURI and wsman:SelectorSet are required at the deletion.

Note:

- wsman refers to the namespace “http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd”.
- wxf refers to the namespace "http://schemas.xmlsoap.org/ws/2004/09/transfer".

Device and Service resources support these add-on operations

dd:Start: this operation starts the resource

dd:Stop: this operation stops the resource

dd:GetState: this operation gets the state of the resource

3.5. Deploying resources

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

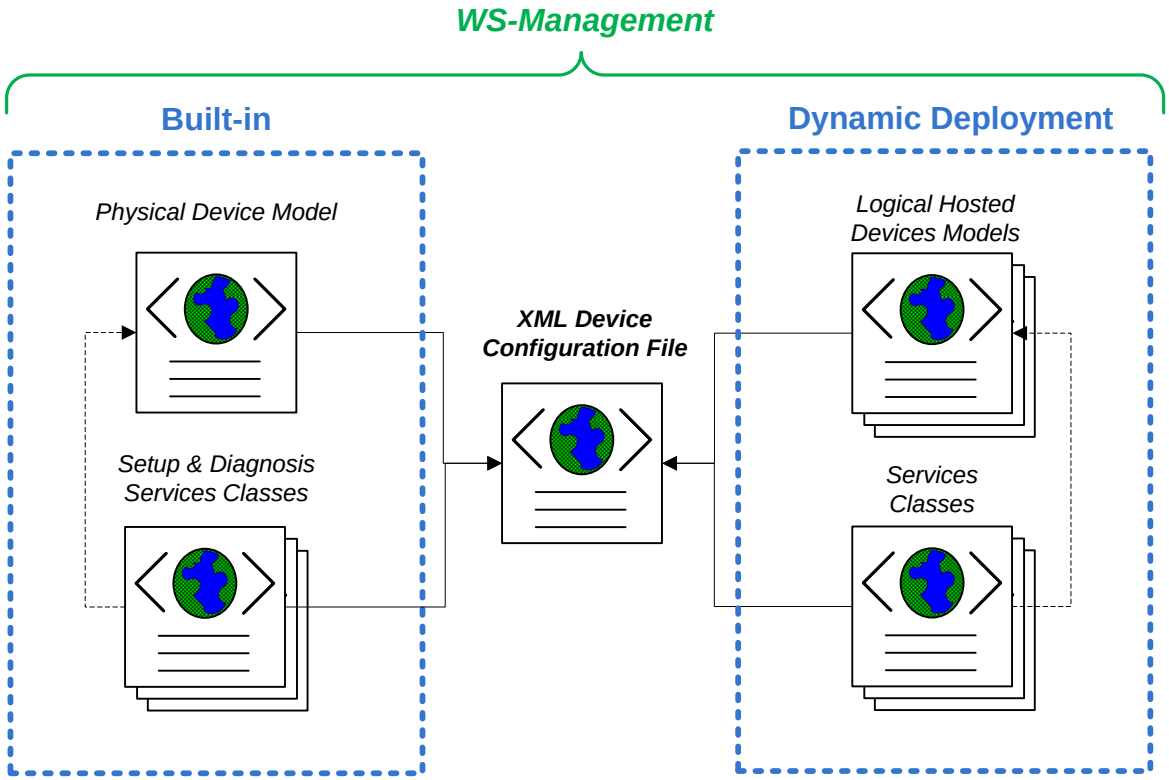


Figure 6: Built-in vs. dynamic deployed elements

Remarks:

- A single service class can be registered to several different devices, either hosted on the same or on two or more devices. Still, in the last case, an identical service class must be also deployed on those devices.
- A device can host several different services, registered from several different service classes.
- It is not advisable to deploy identical devices (same model): it will become difficult to distinguish between two or more identical devices during discovery phase – only possible through *uuid* reference.

3.5.1. Deploying a service class

This operation consists in deploying the service class: the service implementation representation and its implementation. Each service class is uniquely identified by its ServiceClassId. The device platform must assign this unique resource identifier.

Enabling operation

wxf:Create – the ResourceURI is required.

The wxf:CreateResponse must return the resource identifier (ServiceClassId) in case of successful operation, in the other case, the service should return a fault

Pre-State

NA

Pre-conditions

- The resource description must be correct, otherwise the service should return a wxf:InvalidRepresentation fault.

Operation

- The creation of the service class is validated. (NA => *INSTALLED* => READY)

Post conditions

- The creation of the service class is valid

Post-state

INSTALLED or READY

3.5.2. Deploying a device

This operation consists in registering and enabling a device into the device platform. Each device resource is uniquely identified by its uuid. The device platform must assign this unique resource identifier.

Enabling operation

wxf:Create – the ResourceURI is required.

The wxf:CreateResponse must return the resource identifier (uuid) in case of successful operation, in the other case, the service must return a fault.

Pre-State

NA

Pre-conditions

- The resource description must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- The same pre-conditions as those for the deployment of a service – a deployment of a device implies the deployment of its services.

Operation

- The internal references to the services are verified.
- The services are deployed – they will remain in INSTALLED state until the device goes to READY state.
- The device is deployed. (NA => *INSTALLED* => READY)

Post conditions

- The creation of the device and it services are valid.
- The device and its hosted services are ready to be started.

Post-state

INSTALLED or READY

3.5.3. Deploying a service

This operation consists in registering a service to a device. Each service is uniquely identified by its ServiceId. The device platform must assign this unique resource identifier.

Enabling operation

wxf:Create – the uuid of the device and its ResourceURI are required

The wxf:CreateResponse must return the resource identifier (ServiceId) in case of successful operation, in the other case, the service must return a fault

Pre-State

NA

Pre-conditions

- The device uuid and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- The resource description must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

The creation of the service is validated:

- Internal references (to the service class) are verified
- The service class is registered to the service. (NA => *INSTALLED* => READY)

Post conditions

- The creation of the service is valid.
- The service will be then registered to a logical device – it will become its hosted service.

Post-state

INSTALLED or READY

3.6. Starting device and service resources

This operation could not be applied to the service class resources since it only consists of a service representation and implementation details.

3.6.1. Starting a service

This operation consists in starting a hosted service on a device.

Enabling operation

dd: Start – the uuid of the device, the ServiceId and its ResourceURI are required.

The dd:StartResponse must return the service state in case of successful operation, in the other case, the service must return a fault

Pre-State

READY

Pre-conditions

- If the service is already on STARTING or ON state, this operation is transparent – the service is already started or in its way to start; any other state will deliver a fault message.
- The selectors uuid, ServiceId and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- The resource description must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- The service will be started, but it remains in the "STARTING" state while the hosting device is not into the ON state. (READY => *STARTING* => ON)

Post conditions

- The service will be started (or ready to start when the device passes to ON state) and it will be possible do be discovered and invoked as a hosted service of a particular device.

Post-state

ON

3.6.2. Starting a device

This operation consists in starting the device and, by consequence, also all its hosted services.

Enabling operation

dd:Start – the uuid of the device and its ResourceURI are required.

The dd:StartResponse must return the device state in case of successful operation, in the other case, the service must return a fault

Pre-State

READY

Pre-conditions

- If the device is already on STARTING or ON state, this operation is transparent – the device is already started or in its way to start; any other state will deliver a fault message.
- The selectors uuid and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- The resource description must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- Same pre-conditions as for starting a service – the start of device implies the start of its hosted services.

Operation

- The device and its hosted services are started – it implies the multicast of a HELLO message at the end of this process. (READY => *STARTING* => ON)

Post conditions

- The device will enter into ON state and it will be now possible to discover it and invoke its hosted service.

Post-state

ON

3.7. Stopping device and service resources

As the start operation, this operation could not be applied to the service class resources since it only consists of service implementation details.

3.7.1. Stopping a service

This operation consists in stopping a hosted service. When a hosted service is stopped, it remains registered in the device, but it will not be available to be discovered or invoked. Still, to permanently suppress a hosted service from a device, the operation delete must be used instead.

Enabling operation

dd:Stop – the uuid of the device, the ServiceId and its ResourceURI are required.

dd:StopResponse must return the service state in case of successful operation, in the other case, the service must return a fault.

Pre-State

ON

Pre-conditions

- If the service is already on READY or STOPPING state, this operation is transparent – the service is already stopped or in its way to stop.
- The selectors uuid, ServiceId and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- The service is stopped – the device will no longer expose it as one of its hosted services. (ON => *STOPPING* => READY)
- The device that hosted that service multicasts the network with a new Hello message – notify peers that it changed its composition.

Post conditions

- The service will return to READY state and it will not possible to discover or invoke this service.

Post-state

READY

3.7.2. Stopping a device

This operation consists in stopping the device and, by consequence, also its hosted service of the device platform.

Enabling operation

dd:Stop – the uuid of the device and its ResourceURI are required.

The dd:StopResponse must return the resource state in case of successful operation, in the other case, the service must return a fault.

Pre-State

ON

Pre-conditions

- If the device is already on READY or STOPPING state, this operation is transparent – the device is already stopped or in its way to stop.
- The selectors uuid and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.
- Same pre-conditions as for stopping a service – the stop of a device implies the stop of its registered services.

Operation:

- The device and its hosted services are stopped.
- The device multicasts the network with a BYE message.

Post conditions

- The device becomes "invisible" on the network, although it is still deployed in a physical device. The device returns to READY state, as well as all the services it hosts.

Post-state

READY

3.8. Deleting resources

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

3.8.1. Deleting a service class resource

This operation consists in deleting a service class: the service implementation representation and its implementation. Since services classes are the base of the application, its suppression is constrained to the devices (hosted services) that are still exploiting it.

Enabling operation

wxf:Delete – the ServiceClassId and its ResourceURI must be provided.

The wxf:DeleteResponse must return an OK code in case of successful delete operation, in the other case, the service should return a fault.

Pre-State

READY

Pre-conditions

- The ServiceClassId and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

- The service class is not registered to any of the devices deployed on that physical device (as hosted services); otherwise the service must return the list of devices that are still registered to this service class (uuids).

Operation

- Service class is deleted. (READY => *UNINSTALLED* => NA)

Post conditions

- The service class is completely suppressed from the device platform

Post-state

NA

3.8.2. Deleting a service resource

This operation consists on deleting a service registered to a device – hosted service.

Enabling operation

dd:Delete – the uuid of the device, the ServiceId and its ResourceURI are required.

The wxf:DeleteResponse must return an OK code in case of successful delete operation, in the other case, the service should return a fault.

Pre-State

READY or ON

Pre-conditions

- The selectors uuid, ServiceId and ResourceURI must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- Service in ON state: The service is stopped – it implies the sending of a multicast HELLO message by the device that previously hosted that service. (ON => *STOPPING* => READY)
- Service in READY state: .Service is deleted (READY => *UNINSTALLED* => NA)

Post conditions

- The device that hosted that service will no longer provide it as one of his hosted services. Even if the device is restarted, that service will not become available again on that device, except if a new deployment is done.

Post-state

NA

3.8.3. Deleting a device resource

This operation consists on deleting a device from the device platform. By deleting a device, all its hosted services will be also deleted – the device element describes which services are registered to it. The services classes linked to these need to be explicitly deleted (if needed), since they can be in use by other devices.

Enabling operation

dd:Delete – the uuid of the device and its ResourceURI are required.

The wxf:DeleteResponse must return an OK code in case of successful delete operation, in the other case, the service should return a fault.

Pre-State

READY or ON

Pre-conditions

- The uuid and ResourceURI of the device must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- Device in ON state: The device is stopped – it implies the stoppage of its hosted services and the send of a multicast BYE message. (ON => *STOPPING* => READY)
- Device in READY state: The device and its hosted services are deleted. (READY => *UNINSTALLED* => NA)

Post conditions

- The device is permanently suppressed from the device platform (including its hosted services).

Post-state

NA

3.8.4. Delete all resources from physical device

This operation deletes all resources previously deployed by the integrator in that particular physical device (logical devices, hosted services and service classes). Although this operation might seem a bit strong, its application can be envisaged whenever there is a need to reuse that particular physical device to a completely different or new application, for example.

Enabling operation

dd:DeleteAll – the uuid of the physical device and its ResourceURI are required.

The wxf:DeleteAllResponse must return an OK code in case of successful delete operation, in the other case, the service should return a fault.

Pre-State

READY or ON

Pre-conditions

- The uuid and ResourceURI of the physical device must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- The logical devices available on that physical device are sequentially deleted (following the approach described before about how to delete a device resource).
- The services classes are then deleted (following the approach described before about how to delete a service class resource).

Post conditions

- The physical device will be empty regarding dynamically deployed resources.

Post-state

READY

3.9. Retrieve resources state

The objective of this operation is simply to retrieve the current state of a particular resource. The different possible states that a particular resource can show during its lifecycle were already described before. The process of retrieving it is very similar to all different types of resources.

Enabling operation

dd:GetState:

- Device: the uuid of the device and its ResourceURI are required.
- Service: the uuid of the device, the ServiceId and its ResourceURI are required.
- Service class: the ServiceClassId and its ResourceURI are required.

The wxf:GetStateResponse must return the current state of that resource, in the other case, the service should return a fault. In the case where the resource is in a transitional state, an additional parameter is added to response giving details why the resource is still in that state.

Pre-conditions

- The selector parameters (depending on the type of resource) must be correct, otherwise the service must return a wxf:InvalidRepresentation fault.

Operation

- The resource existence is verified.
- The resource state is retrieved from the device platform.

Post conditions

- NA

4. Built-in services

Although in this context WS-Management application is used to allow dynamic deployment of DPWS devices and services, it is also a built-in service. However, the scope of this chapter is to describe the services available on the physical device that allows the end-user to configure, monitor and diagnose it.

4.1. Setup services

For TCP/IP Ethernet-capable devices, the setup phase has a major importance since it is the first step to allow the communication with a device. One of the most common problems is not knowing what is the default IP address of a new device or the need to change it to be in accordance with actual network parameters. By allowing retrieving and setting the IP address of the device in a standardized way, the integration phase is simplified and open to standard tools. The same approach is applied to other configuration parameters.

In the table below, each group represents an operation available from the "Setup" service, while the functions define the structures of data to be passed as input/output parameters in each of these operations.

Almost all these services can be considered bidirectional (except "Password Management" – check "Access" column from the table below). Bidirectional in this context means that it is both possible to retrieve or set the configuration data – the operation parameters are the same, but the action to perform is either to retrieve or to set those parameters.

Group	Function	Data	Type	Access
IP Configuration	Ethernet parameters	Ethernet frame format	Enum	R/W
	IP parameters	DHCP/Automatic/Local	Enum	R/W
		IP address	String	R/W
		Subnet mask	String	R/W
		Default Gateway	String	R/W
Master IP configuration	Master IP address	Address 1	String	R/W
		Address 2	String	R/W
		Address 3	String	R/W
	Parameters	Reservation time	Long	R/W
		Holdup time	Long	R/W
		Link failure mode	Enum	R/W
Password Management	Change password	Login	String	WO
		Old password	String	WO
		New password	String	WO

Table 1: Built-in Setup service parameters

4.2. Monitoring services

Similar to built-in setup services approach, but here all services are used to retrieve information – except "Counters Reset" operation that resets all statistic history.

Group	Function	Data	Type	Access
TCP/IP statistics	TCP/IP parameters	Device name	String	RO
		IP address	String	RO
		Subnet mask	String	RO
		Default Gateway	String	RO
		MAC Address	String	RO
	Ethernet statistics	Frames received	Long	RO
		Frames transmitted	Long	RO
	Counters reset			Command
Ethernet port statistics	Transmit statistics	Frames OK	Long	RO
		Collisions	Long	RO
		Excessive Collisions	Long	RO
		Carrier sense errors	Long	RO
		Internal MAC errors	Long	RO
	Receive statistics	Frames OK	Long	RO
		Alignment errors	Long	RO
		CRC errors	Long	RO
		FCS errors	Long	RO
	Counters reset			Command
Messaging TCP statistics	Statistics table (10 entries)	Remote IP	String	RO
		Remote port	Int	RO
		Local port	Int	RO

		Sent messages	Long	RO
		Received messages	Long	RO
		Sent errors	Long	RO
	Counters reset			Command

Table 2: Built-in Monitoring services parameters

4.3. Diagnosis services

Again, the built-in diagnosis service approach is similar to the previous. The main different is that there is also an event source. This event source has the objective to notify the devices that subscribed to it that an error/fault has been detected.

Group	Function	Data	Type	Access
Identification	Configuration	Device name	String	RO
		Coding wheels	Int	RO
	IP Configuration	IP address	String	RO
		MAC address	String	RO
		URN	String	RO
		Port 2 status	Enum	RO
		… Port N status	Enum	RO
Status	Status	Status	Enum	R/W
Fault (Event)	Fault Information	Code	String	RO
		Type	Enum	RO
		Time	String	RO
		Description	String	RO

Table 3: Built-in Diagnostic service parameters

The identification operation has particular interested when there is a need to discover where the physical device is located, either physically on the area either on the communication network.
The status can be also modified. This functionality can be useful after a maintenance activity – update the status from an ERROR to an OK state, for example.
Every time the device detects an error/fault it sends an event containing the relevant information about it. This information can also be later retrieved by invoking the according operation.
The next figure shows an example of a complex industrial system using this diagnostic service: This is a classical pyramidal architecture. However, thanks to DPWS and WS-Management, such architecture should be easily modified, and the ability for the devices to push data automatically to the aggregation systems, the network load in the critical part of the network can be reduced. The diagnostics for all the devices can be aggregated, and at a higher level, this aggregated status can be polled by the management system.

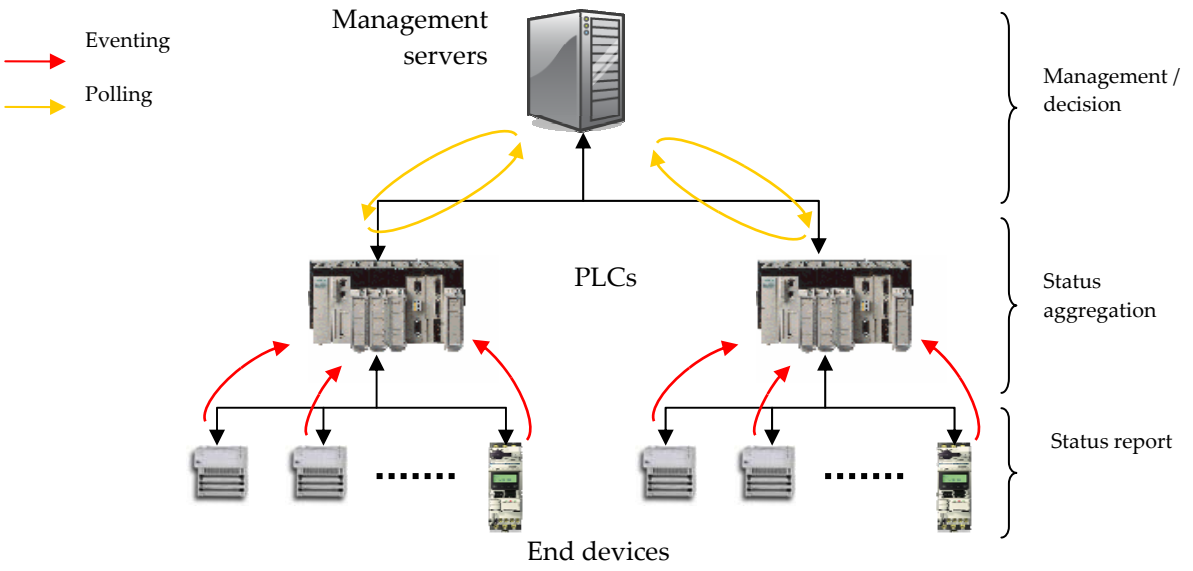


Figure 7: Diagnostic architecture

The next figure shows the direct advantages of using Web services across all the levels of the infrastructure: An operator with a PDA can access all the different levels, from the PLC to search the faulty device, to the device itself to establish a more complete diagnostic, and even to the management infrastructure to order a new one.

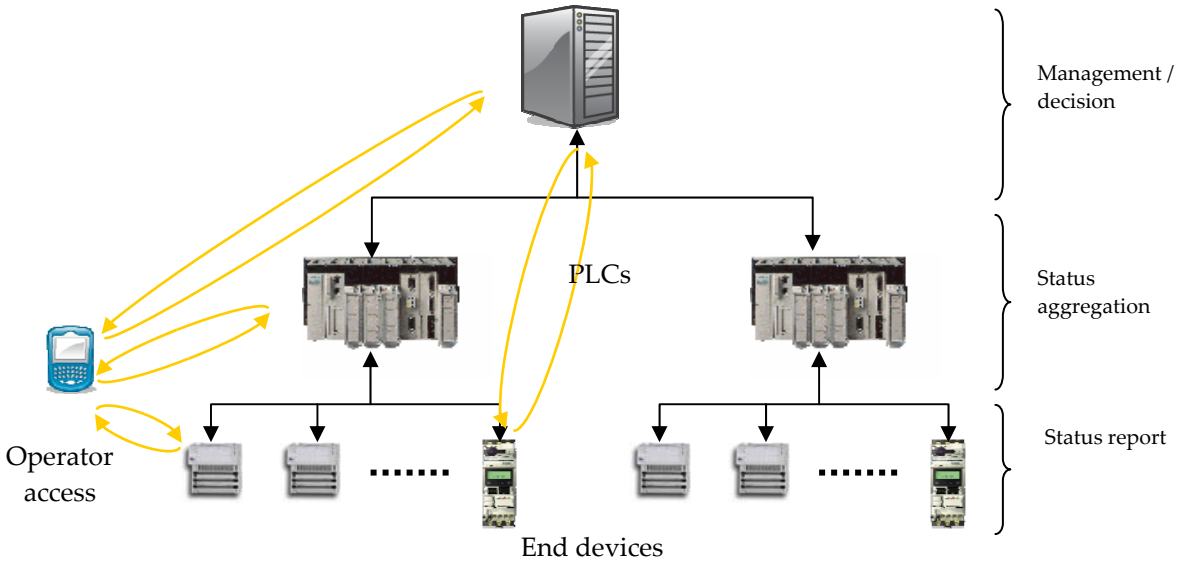


Figure 8: Maintenance operation

5. Persistence

Persistence is fundamental to avoid reconfiguring/redeploying a device each time it is powered off and on again. The middleware must warrant the management of resources states along their complete lifecycle.

Not only the configuration parameters must be kept (network addressing, access parameters, diagnosis/monitoring history, etc.), but also the application elements that were deployed and active within that physical device before the shut down.

Regarding the so-called built-in services, the implementation will remain proprietary and it will depend of each device characteristics. This way, although the interface of these services can remain unchanged, the way to implement persistence features for these kinds of services is out of the scope of this document.

The elements deployed in a physical device are represented in the XML device configuration file that will be permanently stored on the device file system. Any update of configuration/deployment will alter this same file – this way, every time a device is shut down this file will be kept in internal persistent memory.

Every time the physical device is powered on, the XML device configuration file will be interpreted and the previously deployed elements will be started all over again. Of course, those which were deleted in a previous session would not be available unless a new deployment takes place.

6. Security

Although security is recognised to play an important role to avoid unauthorized access and modifications of these resources during its lifecycle, its specification and use-cases are not focused in this document. Still, it remains a subject for future study and application.

References

[1] Devices Profile for Web Services specification, February 2006

[2] Web Services for Management (WS-Management) Specification version 1.0.0

Annex A – Dissemination Levels

PU	Public
PP	Restricted to other programme participants (including the Commission Services)
RE	Restricted to a group specified by the consortium (including the Commission Services)
CO	Confidential, only for members of the consortium (including the Commission Services)

Table 4: Dissemination levels for a document