FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

UPnP and DPWS

Differences and Similarities

Seminar

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik

eingereicht von

Stefan Reichhard
Matrikelnummer 0208213

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao.Univ.-Prof. Dr. Wolfgang Kastner
Mitwirkung:

Wien, 04.11.2013

(Unterschrift Verfasser) (Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Stefan Reichhard
Neustiftgasse 15/11, 1070 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser)

Contents

CHAPTER 1

Introduction UPnP

1.1 General Information

UPnP is an acronym for *Universal Plug and Play*. UPnP is one of the first attempts to define an architecture which can fulfil a zero-configuration scenario for a different set of devices. In its early days, it was a technology developed by Microsoft and embedded in its operating system Windows. In 1999, a forum was established, nowadays consisting of more than 900 companies who are industry leaders in a great variation of different sectors (e.g. consumer electronics, computing, home automation, mobile devices) to define UPnP.

UPnP is more than just a simple extension of the Plug and Play peripheral model. A UPnP enabled device can dynamically join a network (and obtain all necessary information like an IP address), announce itself and its capabilities to the (UPnP) network, and learn about the presence and capabilities of other devices currently connected to the same network. It does so with truly zero configuration, the only thing a user has to do is to plug in the device into the network. Devices can subsequently communicate with each other directly, thereby further enabling peer to peer networking.

Universal Plug and Play utilizes a well-known protocol stack, which is tested for many years. The protocols included are IP, TCP, UDP, HTTP and XML. Because of this it is quite easy and stable to expand an existing network with the capabilities of UPnP. Relying on protocols instead of *Application Programming Interfaces* (API) makes the UPnP architecture also independent from any given operating system and also from a specific programming language [1].

1.2 Use Case

Within this thesis, use cases will be combined to demonstrate the power of a UPnP enabled household.

Imagine your UPnP alarm clock or even your stereo or TV wakes you up one hour earlier than ordinary, just because you have added an important meeting in your personal UPnP enabled

calendar the day before. This meeting record has created a UPnP notification. The alarm clock was listening to it and adjusted the wake up time accordingly. Also, the HVAC system was interested in this notification and uses the new alarm time to adjust the set-point temperature accordingly, so you will not freeze when you getting up. When you leave the house you push the "away"-switch resulting in turning off all lights, TVs, stereos and switching the HVAC system to "away"-mode.

At work the meeting went well and your boss gives you some days off. On the web-browser you set the vacation flag for the next day.

Later the day, when you are back home you decide to watch a movie. On a control device you select "theatre"-mode, and all involved devices will respond immediately (the lights will dim down, the window blinds lower, the receiver will turn on and select the corresponding input, and last but not least, the TV will be turned on as well). You are able to enjoy the cinema feeling without the hassle of setup this scenario manually.

The next day you leave town with a couple of friends. You already set the vacation flag the day before via Internet, so during your absence your house is not quiet at all; it is acting autonomously for different reasons that may include:

- periodically turning on and off the lights on a random pattern to simulate you are still at home (security)
- lowering the heat or turning it off (energy saving)
- recording your favorite shows, so you can watch it later (comfort)

All this can be done with UPnP enabled devices and some scripts to automate different scenarios like "theatre"-, "vacation"- or "away"-mode.

1.3 Components of a UPnP Network

A UPnP network consists of 3 major components. These are devices, services and control points. In Figure 1.1, a quick overview is given on how the structure of a UPnP network looks like.

UPnP Device

A UPnP device acts like a container of services, nested services and control points. For instance, a Personal Video Recorder (PVR) could be such a device, with an integrated tuner service, playback service and a clock service. These services may differ widely depending on the device. Thus, a printer will implement other services as an Internet gateway or the earlier mentioned PVR device. Hence, it is the goal of different working groups to standardize the corresponding set of services that a particular device type will provide. The resulting device description is represented in *Extended Markup Language* (XML).

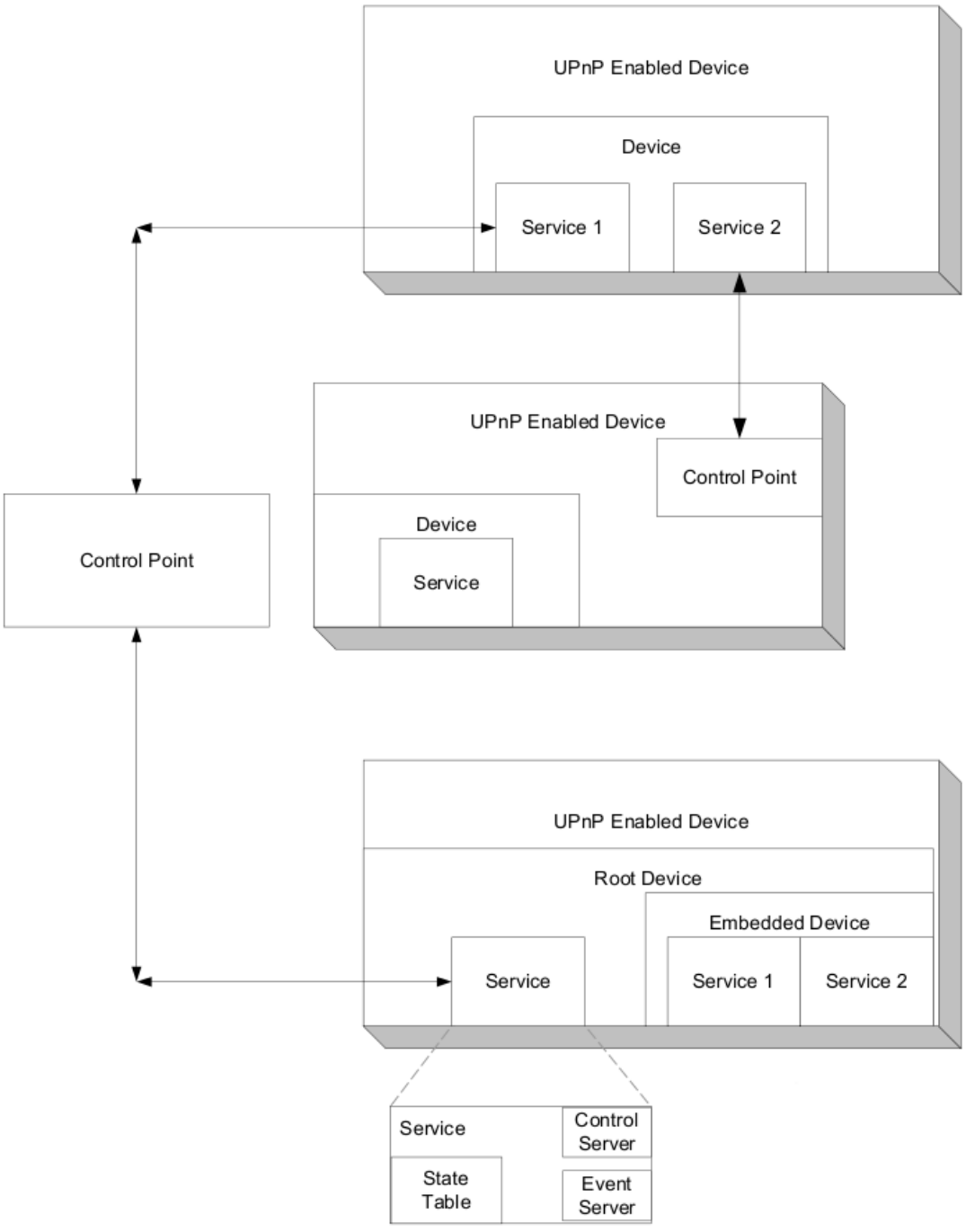

Figure 1.1: UPnP Devices, Services and Control Points [1]

UPnP Service

A service is the elementary part of a UPnP network. The task of a service is to expose its actions, and modelling its state with state variables. You may argue it has similarities with a Java class (it encapsulates actions/methods and holds its public or private state variables). Like the device description, the service description is also represented in XML.

Every service will contain a state table, a control server and an event server. The state table holds the state information of a service and will update its variables when appropriate. The control server executes actions when receiving an action request, updates the state table and returns responses to the caller. The event server will notify any interested subscriber every time the state changes.

UPnP Control Point

Like the name might suggest, the purpose of a control point is to discover and control other devices via their services. The workflow is like following:

- retrieve the device description of devices in a UPnP network and get a list of encapsulated services
- retrieve service descriptions and remember the interesting ones
- invoke actions on the services to control the device
- make a subscription to the event server, so the control point will be notified every time the state changes

1.4 Concept of a UPnP Network

To define how the basic components of a UPnP network work together, different UPnP phases were introduced. They specify a basic pattern, a workflow, each UPnP device utilizes to provide the basic communication between them. These phases are named like following:

- Addressing
- Description
- Discovery
- Control
- Eventing
- Presentation

The phases will be described in more detail shortly. The protocols used for communication in these stages will be listed before.

Overview of Protocols

UPnP utilizes the *Internet Protocol* (IP) and *HyperText Transport Protocol* (HTTP) over *Transmission Control Protocol* (TCP) for basic network connectivity. For specific tasks like discovery of other UPnP participants, it also uses HTTP over *User Datagram Protocol* (UDP). There are two variants, *HTTP over unicast UDP* (HTTPU) because of less overhead, and *HTTP over multicast UDP* (HTTPMU) to reach more than one device simultaneously. Furthermore, UPnP leverages *Simple Object Access Protocol* (SOAP) to control other devices. All these protocols are well-known and, maybe despite HTTPU and HTTPMU, used ubiquitously (see Figure 1.2).

Since HTTP is involved in every UPnP phase except addressing, a short introduction in the HTTP request/response model is given. Both, the HTTP request and the HTTP response, have a similar structure:

- the first line indicates the method or action to be taken
- zero or more header lines define variables and their values (e.g. charset, language)
- a blank line (carriage return, line feed)
- an optional message body, containing the "payload" (e.g. XML document)

One of the shortest examples of an HTTP request/response message pair would be:

- Request: GET http://example.com/index.htm HTML/1.0
- Response: HTTP/1.0 200 OK (where 200 indicates the status code: success)

Addressing

In order to communicate with other devices, each device needs a unique address. Thus, addressing is the first phase in UPnP networking. There are two addressing protocols used by UPnP devices, the *Dynamic Host Configuration Protocol* (DHCP) and the *Dynamic Configuration of IPv4* (Auto-IP). Since both protocols are mandatory, each UPnP device must have a built in DHCP client and needs a mechanism to deal with Auto-IP.

When a UPnP device is connected to the network it tries to get an IP-address from a DHCP server. If the device acquires an address successfully, it may continue with subsequent UPnP Phases.

In case the device fails to receive an address after a second DHCP request, it randomly chooses an IP address in the reserved, non-routeable range of addresses (169.254/16). Once an address is chosen, the device must verify that the address is not already used by another device. It does so by using the *Address Resolution Protocol* (ARP). This procedure (see Figure 1.3) is repeated until a valid, unique address is found [3].

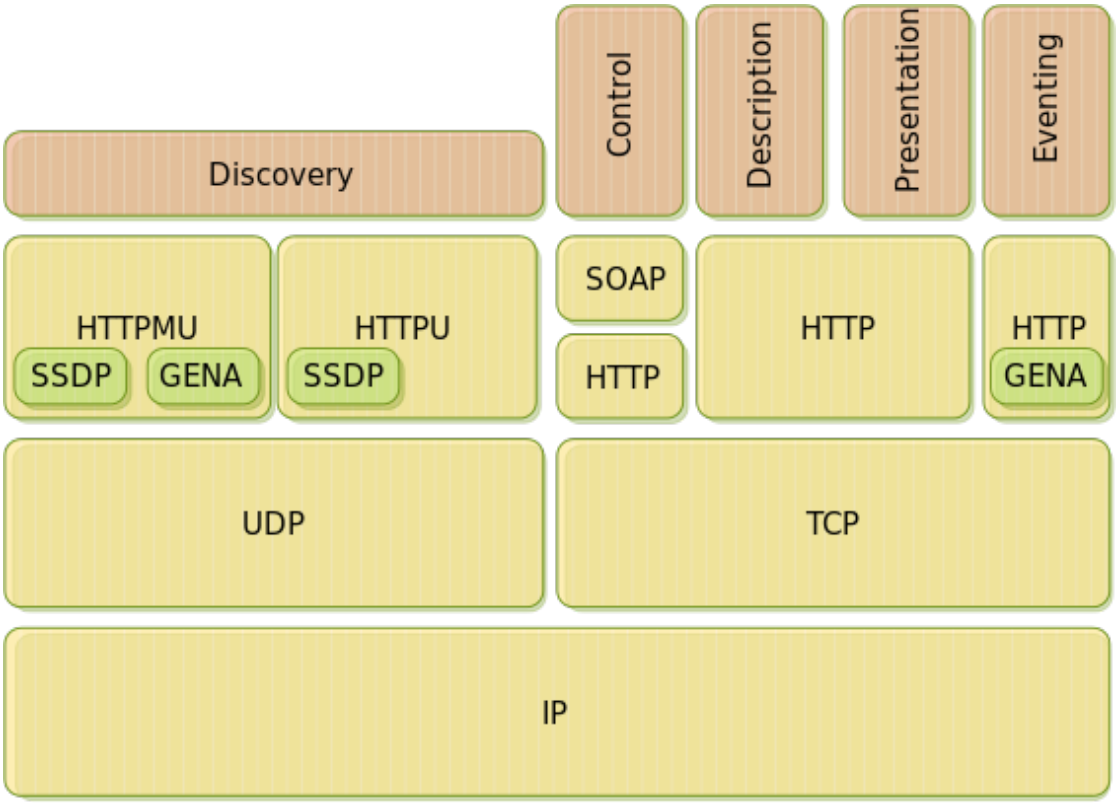

Figure 1.2: UPnP Protocol Stack

Discovery

After acquiring an address, a UPnP device is ready to propagate its capabilities in form of services to control points. Discovery also allows control points to actively search for (distinct) devices and services on the network.

The *Simple Service Discovery Protocol* (SSDP), a discovery solution for HTTP-based resources on the local area network, is used for this purpose. It is designed to be simple and doesn't require any configuration, management or administration by the user.

SSDP leverages a subset of ordinary HTTP communication in conjunction with some special headers, required for the discovery process.

To save network bandwidth, the SSDP protocol uses a combination of actively searching resources on the network (SSDP request/response) as well as an automatic presence announcement mechanism. Whenever a device gets online and after obtaining an address, it sends a multicast message (a presence announcement) to introduce itself on the network. A control point sends a SSDP requests to know what devices are available, respectively.

This combination of SSDP request/response and presence announcement is very effective in a distributed network without central intelligence. All devices have the same knowledge about the UPnP members without the necessity for a control point to search the network for new

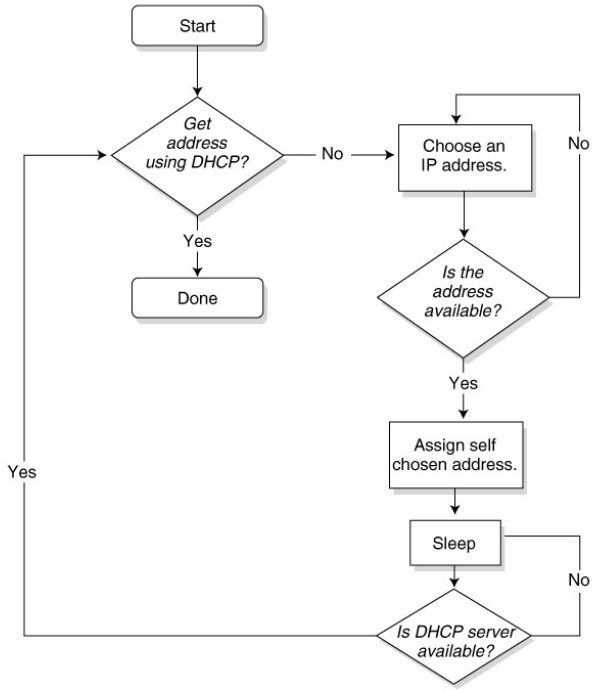

Figure 1.3: Addressing Procedure [1]

devices on a regular basis (devices will announce themselves on availability), nor the device is obliged to introduce itself periodically on the network (a control point will search actively for devices it can control).

Therefore, it is possible to break up the SSDP protocol into 3 parts: Discovery Request, Discovery Response and Presence Announcement:

Discovery Request The SSDP discovery request introduces M-SEARCH, an HTTP request method, and a special header for the search target (ST). To specify the host for which the request is intended for, the Host header is used. Since the search request is directed to all network participants (HTTPMU), the Host header value is set to the multicast address 239.255.255.250:1900, which resides in a reserved multicast address range. Furthermore, a request message requires the mandatory header (Man) and the MX header.

A sample SSDP discovery request may look like shown in Listing 1.1.

Type of Search	Syntax	Description
All devices	ssdp:all	Search for all UPnP devices.
Root devices	upnp:rootdevice	Only root devices will respond; embedded devices will not.
Specific device	uuid:*device-uuid*	Search for a particular device by the device's unique ID. The unique ID is supplied by the device vendor.
Devices of a specific type	urn:schemas-upnp-org: device:*deviceType*:*version*	The device type is defined by a working committee of the UPnP Forum. This kind of search will locate all devices of a given type.
Services of a specific type	urn:schemas-upnp-org: service:*serviceType*:*version*	Like the device type search, this search will find all services of a given type.

Table 1.1: UPnP search types [3]

```
1 M-SEARCH {*} HTTP/1.1
2 Host: 239.255.255.250:1900
3 Man: ssdp:discover
4 MX: 3
5 ST: upnp:rootdevice
```

Listing 1.1: SSDP Request

The mandatory header "ssdp:discover" ensures the HTTP server is capable of the SSDP protocol. The MX header indicates how many seconds the device waits for an answer to this request. In the example above, all devices have to answer to the request within a random time between 0 and 3 seconds. With this mechanism, the responses do not arrive all at the same time at the request initiating device and shifting the network load at the same time. The possible values for the ST header are listed in Table 1.1.

Discovery Response Once a device or service notices a control point is searching for it, it must respond to the sender with a unicast message (HTTPU). The most important HTTP response headers are the ST header, a unique service name header and the location header. All others, except Date, are required as well and are listed in Table 1.2.

Header Name	Description
Cache-Control	Time in seconds how long the response is valid. Must be greater than 1800.
Date	This header is optional and holds the time when the response was generated
Ext	Conforms it is able to serve the Man header (ssdp:discover) in the request
Location	This value points to the URL of the description document of the device
Server	A string consisting of "OS name/OS version, UPnP/1.0, product name/product version"
ST	Search Target: holds a copy from the request message (see Table 1.1)
USN	Unique Service Name: basically the same as the ST header but with a unique ID (uuid) in front of it

Table 1.2: SSDP Response Headers

Only devices or services with a device- or service-type matching the ST header of the request are allowed to respond.

The Cache-Control header indicates how long a SSDP client should keep information about the service in its cache. Once the entry has expired, it must be removed from the SSDP client's cache.

A sample response message looks like shown in Listing 1.2.

```
1 HTTP/1.1 200 OK
2 CACHE-CONTROL: max-age=1800
3 EXT:
4 LOCATION: http://example.com/description.xml
5 SERVER: Linux/2.0.33, UPnP/1.0, dev/0.1
6 ST: upnp:rootdevice
7 USN: uuid:8790c1b0-1dd2-11b2-99b7-d935b74fec25::upnp:rootdevice
```

Listing 1.2: SSDP Response

Presence Announcement As mentioned before a device will broadcast a presence announcement to all other devices whenever it gets online. This announcement is a multicast message with HTTP headers similar to the SSDP Response (see Table 1.2). They share the cache-control, location, server and USN Header. The search target (ST) header becomes the notification type (NT) Header, and the only difference except its name is the "ssdp:all" value which is not available in the NT Header (see Table 1.1). Additionally, a presence announcement includes a NTS Header which value is "ssdp:alive" and the Host header already discussed during the presentation of the SSDP request. The HTTP method is NOTIFY * to send the message to all other devices.

In Listing 1.3 there is an example presence announcement.

```
1 NOTIFY {*} HTTP/1.1
2 HOST: 239.255.255.250:1900
3 CACHE-CONTROL: max-age=1800
4 LOCATION: http://example.com/description.xml
5 SERVER: Linux/2.0.33, UPnP/1.0, dev/0.1
6 NT: upnp:rootdevice
7 NTS: ssdp:alive
8 USN: uuid:8790c1b0-1dd2-11b2-99b7-d935b74fec25::upnp:rootdevice
```

Listing 1.3: Presence Announcement

One important thing to mention is that a device sends multiple presence announcements to advertise all its devices, whether embedded or not, and all of its services and nested services. There are 3 announcements for the root device, two advertisements for each embedded device and one advertisement for each service.

Prior a device leaves the network, it has to notify control points that it will go away by sending a bye-bye message to each advertisement it has previously sent. A bye-bye message looks like a presence advertisement but lacks the cache-control, location and server header. The NTS header has the value “ssdp:bye-bye”.

Description

After a control point discovers a device, it still knows very little about the device. This information includes the device-type, the universally unique ID (uuid) and a URL pointing to the description document. In order to get more detailed information about the device, the control point retrieves the description document. Figure 1.4 illustrates this workflow in a graphical way. There are two different flavors of the description documents: the device description document and the service description document. These documents have to be compliant to the UPnP Template Language, the XML syntax defined by the UPnP Forum for creating device and service descriptions.

Retrieving a device description document from a UPnP device is done by sending an HTTP GET request to the URL specified in the Location Header of the Presence Announcement. To get the service description, an HTTP GET request to the URL specified in the <SCPDURL> Tag (see Listing 1.4) is sent, respectively. Thereafter the device has 30 seconds to respond to the HTTP request by delivering the XML encoded description document within the body of the HTTP response.

Device Description Document A device description document includes vendor-specific information like model name, model number, serial number, manufacturer name as well as a service list. For each service hosted by the device, the device description lists the service type, the service name, a URL to the service description, a URL for control and a URL for eventing. A URL to a Presentation Page is also included in this document. Its structure is usually derived from standard UPnP Device Templates and written by the device manufacturer. The UPnP Device

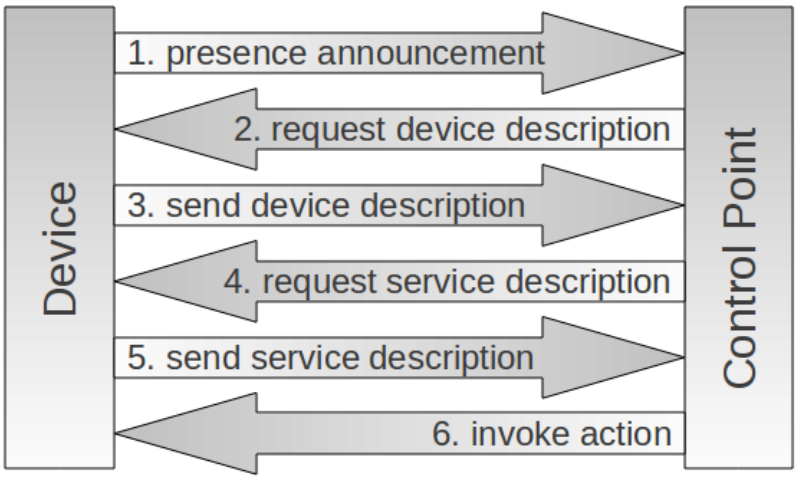

Figure 1.4: UPnP Description Phase

Templates are produced by a UPnP working committee. The Device Architecture Document [2] defines how a document has to look like in detail, but to get a quick overview how it might look like, Listing 1.4 provides a sample device description.

```xml
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">

<specVersion>
  <major>1</major>
  <minor>0</minor>
</specVersion>

<URLBase>base URL for all relative URLs</URLBase>

<device>
  <deviceType>urn:schemas-upnp-org:device:deviceType:1</deviceType>
  <friendlyName>short name for the device</friendlyName>
  <manufacturer>manufacturer name</manufacturer>
  <modelName>model name</modelName>
  <modelNumber>model number</modelNumber>
  <serialNumber>serial # for the device</serialNumber>
  <UDN>uuid:UUID</UDN>
  <iconList>information about icons</iconList>
  <serviceList>
    <!-- description of services provided by the device -->
    <service>
      <serviceType>urn:schemas-upnp-org:service:serviceType:1</serviceType>
            <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
      <SCPDURL>URL to service description</SCPDURL>
      <controlURL>URL for control</controlURL>
      <eventSubURL>URL for eventing</eventSubURL>
    </service>
  </serviceList>
  <deviceList>
    <!-- description of embedded services contained in this device -->
  </deviceList>
  <presentationURL>URL for presentation</presentationURL>
</device>
</root>
```

Listing 1.4: Device Description Document

Service Description Document A device description document contains detailed information about a device. Likewise a service description document contains detailed information about a service. This includes the <actionList> block, listing all actions and associated parameters for a service, as well as a <serviceStateTable> block, containing all variables a service provides. Like the device description, the service description document is also based on a template defined by

a UPnP working committee. Listing 1.5 shows a schematic service description file.

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">

<specVersion>
  <major>1</major>
  <minor>0</minor>
</specVersion>

<actionList>
  <action>
    <name>actionName</name>
    <argumentList>
      <argument>
        <name>formalParameterName</name>
        <direction>in xor out</direction>
        <retval />
        <relatedStateVariable>stateVariableName</relatedStateVariable>
      </argument>
      <argument> ... </argument>
    </argumentList>
  </action>
</actionList>

<serviceStateTable>
  <stateVariable sendEvents="yes">
    <name>variableName</name>
    <dataType>variable data type</dataType>
    <defaultValue>default value</defaultValue>
  </stateVariable>
  <stateVariable>  ...  </stateVariable>
</serviceStateTable>

</scpd>
```

Listing 1.5: Service Description Document

Control

After completing the first 3 phases (addressing, discovery, description) of UPnP networking, the control point is finally ready for action. UPnP utilizes *Simple Object Access Protocol* (SOAP) to invoke actions provided by services on other devices. To serve the remote procedure call

functionality, the SOAP protocol leverage's HTTP for transportation and XML for presentation. The structure of the SOAP Encapsulation Model is shown in Figure 1.5.

The skeleton in this concept is a SOAP Message Envelope transported over HTTP. This envelope is declared with a predefined namespace and a SOAP encoding style, and also contains an optional <header> block and a mandatory <body> block. In the UPnP context, the header block is not used for composing SOAP messages. Later on in DPWS the header block is utilized for addressing Web services (see Listing 2.3).

On the transport side, a SOAP-method-request is obligated to specify a host header, a Content-Length and Content-Type header, as well as a header indicating the action name (SOAPAction). The HTTP POST method constructed with these headers need to be sent to the Control URL specified in the <controlURL> Tag of the device description to invoke an action (see Listing 1.6 for a sample SOAP action request).

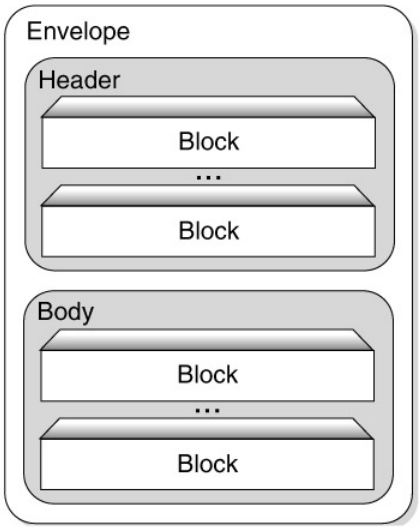

Figure 1.5: SOAP Encapsulation Model [3]

Subsequently a service has to return an action response, also packed in a SOAP envelope, to the originating control point within 30 seconds after invocation. In case the invocation was successful, the SOAP envelope of the response is nearly identical as the requesting one, despite three main differences:

- the SOAPAction Host header is missing
- it concatenates "Response" to the actionName in the <body> element
- and the <argumentName> tags are no longer used for input arguments, but instead for a possible return value

In case of an error, the body element contains an error code and some information regarding the reason of the error. For detailed error response information see [2].

```
POST controlURL HTTP/1.1
Host: hostname:portnumber
Content-Length: length of body in bytes
Content-Type: text/xml; charset="utf-8"
SOAPAction: "urn:schemas-upnp-org:service:serviceType:1#actionName"

<s:Envelope
  xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
  s:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<s:Body>
  <u:actionName xmlns:u="urn:schemas-upnp-org:service:serviceType:1">
    <argumentName>in arg value</argumentName>
    <!-- other in args and their values go here, if any -->
  </u:actionName>
</s:Body>
</s:Envelope>
```

Listing 1.6: Sample SOAP Action Request

Eventing

UPnP utilizes an asynchronous publisher/subscriber model. There are good reasons for that: it is an effective and simple concept. Imagine a control point wants to know if a particular device is powered on. Therefore, it invokes the SOAP action request "GetPowerState" every 5 seconds in order to get the desired information. Instead of permanently polling the device it is much more efficient if the particular device sends a notification on power state change to all interested control points (the ones which have an active subscription on this event).

As for controlling, the pre-requirement for eventing is to complete the first 3 phases of UPnP networking, namely addressing, discovery and description. A control point needs to know all corresponding devices and service descriptions in order to subscribe to evented variables it is interested in.

UPnP uses the GENA Protocol for event notification. GENA is an abbreviation for *General Event Notification Architecture*, and utilizes HTTP for communication. The subscriber may request, renew or cancel a subscription. Once a subscription request is sent to the publisher, it responds with a subscription ID and a time period how long the subscription is valid. If a subscriber wants to renew or cancel a particular subscription it uses the subscription ID as reference.

The HTTP methods to manage event subscriptions with GENA are:

- SUBSCRIBE to register or renew a subscription
- UNSUBSCRIBE to terminate a subscription
- NOTIFY to send event notifications to the subscribers

It is also necessary to introduce a few new HTTP headers for GENA:

- CALLBACK the publisher uses this URL to send event notifications
- NT declares the type of notification
- NTS declares the sub-type of notification
- SID the already mentioned subscription ID

GENA's simple publish/subscribe model maps easily to the UPnP object model: control points are subscribers while devices act as publishers, but there are a few aspects to consider using GENA in a UPnP context.

One obvious thing is, a control point can only subscribe to variables declared with "sendEvents=yes" in the service description. Another convention is the existence of an initial event message. Whenever a control point subscribes to an event, it receives the name and value of all evented variables provided by a service. This initial notification allows a control point to model the state of a service. Also each event notification contains an event key. Each time the subscriber receives a notification, the event key will be incremented by one. This mechanism can be used for simple error detection, because the control point can keep track of the notifications and recognizes if it misses one.

To make a subscription, a control point sends a GENA subscription request (see Listing 1.7) to the subscription URL provided in the <eventSubURL> Tag of the device description document.

```
1 SUBSCRIBE publisher Path HTTP/1.1
2 Host: publisher Host:Port
3 Callback: deliveryURL
4 NT: upnp:event
5 Timeout: requested subscription duration in seconds
```

Listing 1.7: GENA subscription request

If the devices accepts the subscription request it sends back an HTTP response containing a Date Header indicating when the response is sent, a Server Header showing the OS name/product version, an optional Timeout Header and the important SID Header providing the subscription ID.

To renew a subscription, a control point sends a renewal message to the same URL as used for the initial subscription (see Listing 1.7). This renewal message tells the publisher to renew the subscription by providing the SID header only (and without Callback and NT headers).

For unsubscribing from a publisher the HTTP UNSUBSCRIBE method is used in combination with the subscription URL, followed by the Host and SID Header.

If an evented variable is changed, the publisher sends a notification message (see Listing 1.8 for a sample GENA event notification).

```
NOTIFY delivery path HTTP/1.1
Host: delivery host:delivery port
Content-Type: text/xml
Content-Length: length of body in bytes
NT: upnp:event
NTS: upnp:propchange
SID: uuid:subscription-UUID
SEQ: event key

<e:propertyset xmlns:e="urn:schemas-upnp-org:event-1-0">
  <e:property>
    <variableName>new Value</variableName>
    <!--other variable names and values (if any) go here-->
  </e:property>
</e:propertyset>
```

Listing 1.8: GENA event notification

The HTTP header NTS: upnp:propchange indicates a changed variable and provides the new value in the <propertyset> block. The host-specific event key, incremented by one at every event notification, is specified in the SEQ header.

Presentation

In order to be UPnP compliant, the only requirement for a presentation page is to be compatible to HTML 3.0 or higher. Everything else is up to the device manufacturer. It is even possible to provide an empty HTML presentation page. The other end of the spectrum would be a full-featured presentation (multi-)page, allowing to invoke all kinds of actions via e.g. JavaScript representing the status in an ambitious graphical manner.

Since all other communication except presentation is machine-to-machine only, there is no need for localization in the former phases of UPnP networking. But the presentation page introduces direct user interaction, so it may be desirable for this page to appear in different languages, depending on the preference of the user. Luckily this is an easy task, because everything needed is already implemented in HTTP and HTML, and therefore well-known from the ordinary web. The client can include an “Accept-Language” HTTP Header to indicate what language it would like to receive.

To get the presentation page, a control point or a web-browser simply issues an HTTP GET request on the presentation URL given in the device description document.

CHAPTER 2

Introduction DPWS

2.1 General Information

DPWS is an abbreviation for *Device Profile for Web Services*. It is basically a subset of ordinary Web services a device must implement in order to be DPWS compliant. The Device Profile in the name indicates it is intended for (small) devices instead of powerful ones, resulting in minimum requirements upon the device itself, and ensuring the possibility to run DPWS on resource-constrained embedded devices. Another key requirement for DPWS is to find a minimal set of Web service specifications, providing secure messaging, dynamic discovery, description and eventing. Furthermore, this minimal set of Web services should not constrain richer implementations.

DPWS was first introduced by Microsoft in 2004 as a successor of UPnP, designed to be compatible with the popular Web service standard. Nowadays, the DPWS Standard is defined and managed by the OASIS Consortium which represents over 600 organizations.

2.2 Components of a DPWS Network

The Device Profile for Web services aligns to the following Web service standards and protocols:

- SOAP 1.2
- WSDL 1.1 (Web Service Description Language)
- WS-Addressing 1.0
- WS-Discovery
- WS-Eventing
- WS-MetadataExchange

- WS-Policy
- WS-PolicyAttachment
- WS-Transfer
- WS-Security (non-normative)

Since the DPWS Profile is just a collection of Web services, the original Web service definitions are normative unless explicitly superseded in the DPWS specification [5]. Because of that, the W3C Consortium is also responsibly for some normative references (e.g. SOAP, WSDL, WS-Addressing).

DPWS Participants

Figure 2.1 introduces the terminology of participants in a DPWS network and how these are linked together.

A *device* in the DPWS context is a special type of service (a so called Target Service) usually hosting other services (hosted services). A *hosted service* is visible on the network and capable of being addressed separately from its host. Its lifetime is bound by the hosting Service (device). A *client* is a network endpoint exchanging special types of messages with Services. *Messages* are responsible for discovery, description, control and eventing and are always embedded in a SOAP Envelope, along with some Headers for HTTP, TCP and IP.

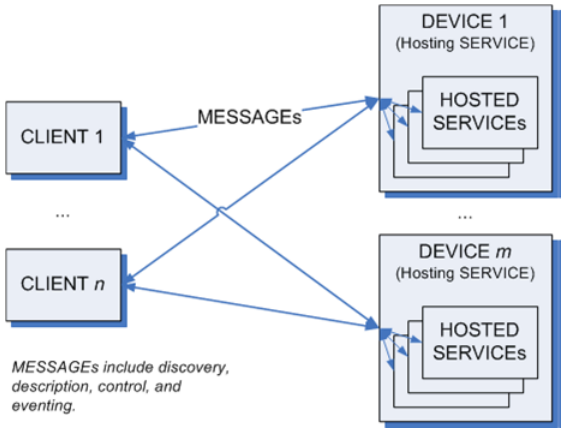

Figure 2.1: DPWS Participants [5]

2.3 Concept of DPWS

In Section 2.2 all components and participants of a DPWS network are introduced. Furthermore the basic interaction between them is outlined. In this chapter, we take a closer look at these components and explain them in detail.

Overview of Protocols

In general, Web services are less specific regarding to the underlying transport mechanism than UPnP. In order to be compliant to the Device Profile, SOAP over UDP must be supported (especially for discovery) as well as a SOAP binding for HTTP has to be present. HTTP and UDP

were already introduced in Chapter 1.4. DPWS participants are free to support other transport mechanisms as well.

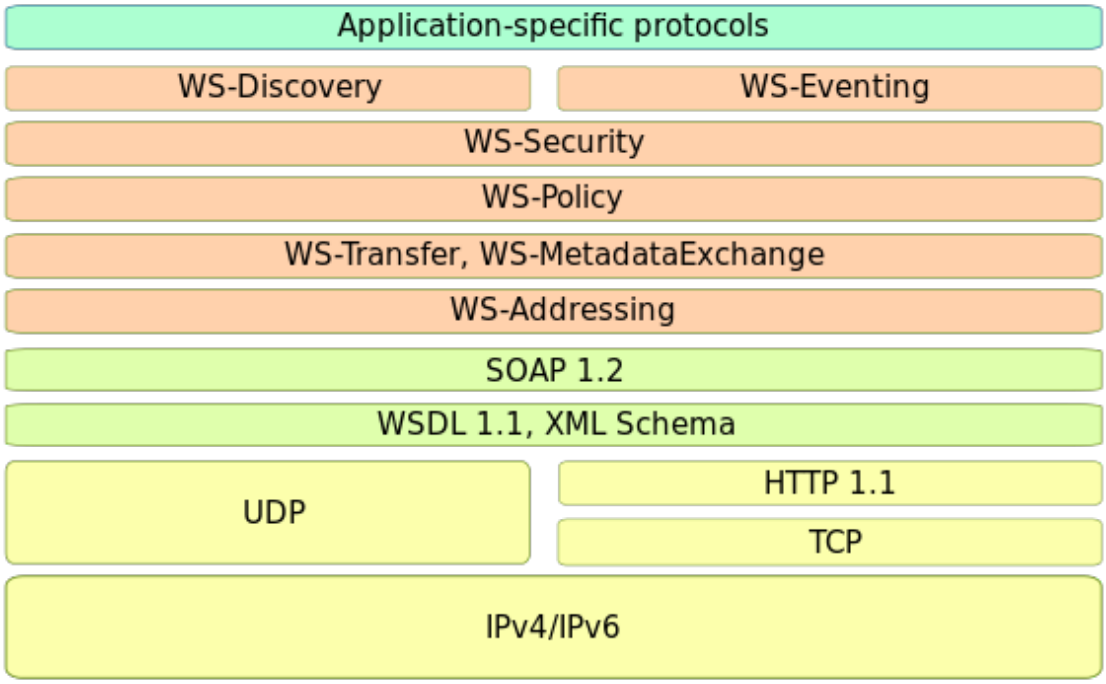

Figure 2.2: DPWS Stack

SOAP 1.2 SOAP Envelopes were already introduced in Chapter 1.4 (see Figure 1.5 for the schematic SOAP Encapsulation Model, and Listing 2.3 for a sample SOAP Envelope, respectively). Unlike UPnP, in DPWS context *Messages* between *Clients* and *Devices* are always encapsulated in a SOAP envelope. These Envelopes act like a Container holding an XML Infoset-based representation of all involved Web services. In order to distinguish one Web service from the other, SOAP uses XML Namespaces (see Table 2.1).

XML Namespace SOAP envelopes use XML Namespaces (e.g. xmlns:Name=anyURI) to declare the scope of XML elements. All Namespaces which can be used in DPWS are listed in Table 2.1. Note however the prefix value is a suggestion only and not normative.

WSDL *Web Service Description Language* (WSDL) is defined by the W3C Consortium in [11]. To get a first vague idea of what it is, one might think of a WSDL document being an abstract Interface definition in a classic Java context. It is using an XML grammar to describe

Prefix	XML Namespace	Specifications
soap	http://www.w3.org/2003/05/soap-envelope	SOAP 1.2
wsa	http://www.w3.org/2005/08/addressing	WS-Addressing 1.0
wsd	http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01	WS-Discovery
dpws	http://docs.oasis-open.org/ws-dd/ns/dpws/2009/01	DPWS
wsdl	http://schemas.xmlsoap.org/wsdl/	WSDL 1.1
wse	http://schemas.xmlsoap.org/ws/2004/08/eventing	WS-Eventing
wsp	http://www.w3.org/ns/ws-policy	WS-Policy, WS-PolicyAttachment
wsx	http://schemas.xmlsoap.org/ws/2004/09/mex	WS-MetadataExchange

Table 2.1: XML Namespace [5]

its “operations” and “messages” and also leverages a concrete binding to a specific protocol for every “operation”.

A WSDL file can be separated in six different parts:

- *Types*– a container for data type definitions using some type system (such as XSD).
- *Message*– an abstract, typed definition of the data being communicated.
- *Operation*– an abstract description of an action supported by the service.
- *Port Type*– an abstract set of operations supported by one or more endpoints.
- *Binding*– a concrete protocol and data format specification for a particular port type.
- *Port*– a single endpoint defined as a combination of a binding and a network address.
- *Service*– a collection of related endpoints. [11]

Addressing

Web Service Addressing is defined by the Web Service Addressing Working Group, and has been reviewed by W3C members, by software developers, and by other W3C groups and interested parties. It represents a W3C recommendation [6].

Since Web services are not necessarily bound to any specific transport system, WS-Addressing defines *endpoint references* and *message addressing properties* to normalize information typically provided by transport protocols and messaging systems.

One restriction of the Device Profile for Web services is the mandatory support for TCP/IP and UDP/IP transport. But, unlike UPnP, obtaining a valid IPv4/IPv6 address is not in the scope of the Device Profile. Normally, the same mechanisms are used to get an address (DHCP or IP autoconfig) that do not conflict with other participants in the network (see also Chapter 1.4).

Endpoint Reference An Endpoint References (EPR) is an XML Infoset-based representation of a Web service endpoint. In Listing 2.1 possible XML elements are shown. Note however that other Web services may introduce new elements to an Endpoint Reference. Furthermore, the Character "?" indicates the XML element is able to be present one or zero times, whereas the "*" Character indicates the XML element can be present zero or more times. This well-known convention is also used in subsequent listings. So the only mandatory element inside an EPR is the Address element.

```
<wsa:EndpointReference>
  <wsa:Address>xs:anyURI</wsa:Address>
  <wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?
  <wsa:Metadata>xs:any*</wsa:Metadata>?
</wsa:EndpointReference>
```

Listing 2.1: Endpoint Reference

There exist 2 predefined values for the <wsa:Address> element which are:

- http://www.w3.org/2005/08/addressing/anonymous: If this address is used, the underlying transport protocol is used to determine the address.
- http://www.w3.org/2005/08/addressing/none: If an endpoint reference (EPR) has this address value, no reply message is allowed to be sent.

Message Addressing Properties Message addressing properties are used to define the routing of the SOAP envelopes among other things. Their possible attributes are outlined in Listing 2.2.

```
<wsa:To>xs:anyURI</wsa:To> ?
<wsa:From>wsa:EndpointReferenceType</wsa:From> ?
<wsa:ReplyTo>wsa:EndpointReferenceType</wsa:ReplyTo> ?
<wsa:FaultTo>wsa:EndpointReferenceType</wsa:FaultTo> ?
<wsa:Action>xs:anyURI</wsa:Action>
<wsa:MessageID>xs:anyURI</wsa:MessageID> ?
<wsa:RelatesTo RelationshipType="xs:anyURI"?>xs:anyURI</wsa:RelatesTo> *
<wsa:ReferenceParameters>xs:any*</wsa:ReferenceParameters> ?
```

Listing 2.2: Message Addressing Properties

While the value of the "From", "ReplyTo" and "FaultTo" elements are endpoint references, the optional "RelatesTo" element holds a reference to former MessageIDs in order to keep track of the message sequence.

WS-Addressing Example The 2 constructs of WS-Addressing, the endpoint reference and the message addressing properties, are combined in a sample SOAP Envelope shown in Listing 2.3.

```
1  <S:Envelope
2    xmlns:S="http://www.w3.org/2003/05/soap-envelope"
3    xmlns:wsa="http://www.w3.org/2005/08/addressing">
4  <S:Header>
5    <wsa:MessageID>http://example.com/6B29FC40-CA47-1067-B31D-00DD010662DA</w
6    <wsa:ReplyTo>
7      <wsa:Address>http://example.com/business/client1</wsa:Address>
8    </wsa:ReplyTo>
9    <wsa:To>http://example.com/fabrikam/Purchasing</wsa:To>
10   <wsa:Action>http://example.com/fabrikam/SubmitPO</wsa:Action>
11 </S:Header>
12
13 <S:Body>
14 <!-- other stuff -->
15 </S:Body>
16
17 </S:Envelope>
```

Listing 2.3: WS-Addressing Scheme

Note that the addressing of a SOAP Envelope is always located in the header block. Also the XML Namespace usage can be seen in this example.

Discovery

WS-Discovery is defined by the OASIS Consortium [7] and is used to locate *services* on a network.

Similar to UPnP, WS-Discovery leverages a combination of active search for participants and presence announcement if a new participant joins the network. To further reduce network traffic in the Device Profile, only *devices* are allowed to make themselves available for discovery (the reason a device is also called target service). So not every *service* sends a multicast message to announce its presence to the network, minimizing traffic in a (possibly bandwidth-limited) network.

Multicast messages are always sent to port 3702 and use the address 239.255.255.250 for IPv4 and FF02::C for IPv6, respectively.

Hello A DPWS *device* will announce itself by sending a one-way multicast “Hello” message if it joins a network or if its metadata changed. A Hello message looks like outlined it Listing 2.4.

```
<s:Envelope
 xmlns:wsa="http://www.w3.org/2005/08/addressing"
 xmlns:wsd="http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01"
 xmlns:s="http://www.w3.org/2003/05/soap-envelope"
 xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof" >

<s:Header>
  <wsa:Action>
    http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Hello
  </wsa:Action>
  <wsa:MessageID>
    urn:uuid:73948edc-3204-4455-bae2-7c7d0ff6c37c
  </wsa:MessageID>
  <wsa:To>urn:docs-oasis-open-org:ws-dd:ns:discovery:2009:01</wsa:To>
  <wsd:AppSequence InstanceId="1" MessageNumber="1" />
</s:Header>

<s:Body>
  <wsd:Hello>
    <wsa:EndpointReference>
      <wsa:Address>
        urn:uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
      </wsa:Address>
    </wsa:EndpointReference>
    <wsd:Types>dpws:Device</wsd:Types>
    <wsd:MetadataVersion>1</wsd:MetadataVersion>
  </wsd:Hello>
</s:Body>
</s:Envelope>
```

Listing 2.4: WS-Discovery: Hello Message

Bye When a *Target Service* prepares to leave the network it should send a bye message. The message is similar to the Hello message, but the <wsa:Action> element in the SOAP Header changes to “http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Bye” and the SOAP Body looks like following:

```
<s:Body>
  <wsd:Bye>
    <wsa:EndpointReference>
      <wsa:Address>
        urn:uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
      </wsa:Address>
    </wsa:EndpointReference>
  </wsd:Bye>
</s:Body>
```

Listing 2.5: WS-Discovery: Bye Message

Probe If a *client* is looking for *devices* (*Target Services*) in general or for devices of a special Type or in a distinct Scope, a client will put a multicast probe message on the network. All Target Services matching all of the Types and all of the Scopes requested in the Probe message will return a ProbeMatch message providing some information about them.

The normative outline for Probe is listed in Listing 2.6.

```
<s:Envelope ... >
  <s:Header ... >
    <wsa:Action ... >
      http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Probe
    </wsa:Action>
    <wsa:MessageID ... >xs:anyURI</wsa:MessageID>
   [<wsa:ReplyTo ... >endpoint-reference</wsa:ReplyTo>]?
    <wsa:To ... >xs:anyURI</wsa:To>
    ...
  </s:Header>
  <s:Body ... >
    <wsd:Probe ... >
     [<wsd:Types>list of xs:QName</d:Types>]?
     [<wsd:Scopes [MatchBy="xs:anyURI"]? ... >
        list of xs:anyURI
      </wsd:Scopes>]?
      ...
    </wsd:Probe>
  </s:Body>
</s:Envelope>
```

Listing 2.6: WS-Discovery: Probe Message

A Type T1 will match another Type T2 if the namespace of these two are the same and the

local name of T1 equals T2.

To match a Scope S1 with another Scope S2, the rule used for matching has to be declared by the MatchBy attribute inside the probe message. WS-Discovery knows the following five rules:

- http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/none - according to this rule, a Target Service only matches to a Probe if the Target Service does not have any Scope
- http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/strcmp0 - a case-sensitive string-compare is used for matching
- http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/ldap - used in a *Lightweight Directory Access Protocol* (LDAP) Environment
- http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/uuid - compare the Universally Unique Identifier
- http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/rfc3986 - compare URIs e.g. "http://example.com/abc" matches "http://example.com/abc/def" in a case-insensitive manner because of the same scheme (http) and the same authority (example.com) [8]

A client will receive a ProbeMatch message from every Target Service that matches the formerly described criteria (Types and/or Scopes). Here is the SOAP Body of an example ProbeMatch:

```
<s:Body>
  <wsd:ProbeMatches>
    <wsd:ProbeMatch>
      <wsa:EndpointReference>
        <wsa:Address>
          urn:uuid:98190dc2-0890-4ef8-ac9a-5940995e6119
        </wsa:Address>
      </wsa:EndpointReference>
      <wsd:Types>i:PrintBasic i:PrintAdvanced</wsd:Types>
      <wsd:Scopes>
        ldap:///ou=engineering,o=examplecom,c=us
        ldap:///ou=floor1,ou=b42,ou=anytown,o=examplecom,c=us
        http://itdept/imaging/deployment/2004-12-04
      </wsd:Scopes>
      <wsd:XAddrs>http://prn-example/PRN42/b42-1668-a</d:XAddrs>
      <wsd:MetadataVersion>75965</d:MetadataVersion>
    </wsd:ProbeMatch>
  </wsd:ProbeMatches>
</s:Body>
```

Listing 2.7: An example of a ProbeMatch

Resolve A Resolve message is issued by a client if the ProbeMatch does not include an <wsd:Xaddrs> element, as a ProbeMatch is not obligated to include this element. By sending a multicast Resolve message to the network a client expects a ResolveMatch message, which resolves a known Endpoint Reference Address e.g. “urn:uuid:98190dc2-0890-4ef8-ac9a-5940995e6119” into a Transport Address e.g. “http://prn-example/PRN42/b42-1668-a”.

The structure of the Resolve Message uses a SOAP header <wsa:Action> element with the value of “http://docs.oasis-open.org/ws-dd/ns/discovery/2009/01/Resolve” and a SOAP body in form of:

```
1 <s:Body>
2   <wsd:Resolve ... >
3     <wsa:EndpointReference> ... </wsa:EndpointReference>
4   </wsd:Resolve>
5 </s:Body>
```

Listing 2.8: WS-Discovery: Resolve Message Body

A ResolveMatch looks pretty much the same as a ProbeMatch message, but instead of every occurrence of ProbeMatch it states ResolveMatch, and the <wsd:Xaddrs> element has to be present in the ResolveMatch message. If a ProbeMatch already includes the transport address of a Target Service in the first place, there is no need for a Resolve message.

The Resolve, Probe, Bye and Hello Messages are sent via multicast, while their responses (ProbeMatch and ResolveMatch) are sent via unicast.

Description

After a client located some devices via WS-Discovery, it does only know little about it, like which Types the device supports and in what Scope it is operating. This chapter is dealing with how to get more detailed information (aka metadata) about a device and its hosting services.

Transmission To get general device characteristics the Device Profile leverages *WS-Transfer* [9]. If a client wants specific information for e.g. a hosted service, the Device Profile proposes to use *WS-MetadataExchange* [10].

In order to be compliant with the Device Profile it is sufficient to support the GET Action from WS-Transfer and WS-MetadataExchange to retrieve the description of a device.

A WS-Transfer request is, like all other DPWS messages, embedded in a SOAP envelope. The body block of this envelope is empty. Inside the header there are some message addressing properties from WS-Addressing. The important ones are the <wsa:Action> element with the value “http://schemas.xmlsoap.org/ws/2004/09/transfer/Get”, and the <wsa:To> element, pointing to the Endpoint Reference where to get the desired information from.

The response to this WS-Transfer request is also a SOAP envelope using WS-Addressing in the SOAP header, a <wsa:Action> element with the value “http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse and a SOAP body containing the payload of the requested information.

A WS-MetadataExchange pattern is similar to a WS-Transfer GET request-response message pair. The simplified output uses [action] to indicate the SOAP header message address property <wsa:Action> and [body] to indicate the SOAP body.

In Listing 2.9, a WS-MetadataExchange request is shown, using the convention described above.

```
1 [action]
2 http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Request
3 [Body]
4 <mex:GetMetadata ...>
5   (<mex:Dialect>xs:anyURI</mex:Dialect>
6   (<mex:Identifier>xs:anyURI</mex:Identifier>)?      )?
7 </mex:GetMetadata>
```

Listing 2.9: WS-MetadataExchange request

It is possible to filter the requested Metadata by so called Dialects. These Dialects for Metadata are used to distinguish the Scope of XML elements and are similar to XML Namespaces. More about Dialects later on in Characteristics.

The corresponding WS-MetadataExchange Response looks like following:

```
1 [action]
2 http://schemas.xmlsoap.org/ws/2004/09/mex/GetMetadata/Response
3 [Body]
4 <mex:Metadata ...>      ...      </mex:Metadata>
```

Listing 2.10: WS-MetadataExchange response

The basics how to transfer a generic description via Web services are shown above. Now the description itself will be discussed in more detail.

Characteristics The Device Profile for Web services defines “ThisModel” metadata to express device characteristics that are typically fixed across all devices of the same model by their manufacturer. An Instance of this Metadata is indicated as a WS-MetadataExchange Dialect of “http://docs.oasis-open.org/ws-dd/ns/dpws/2008/09/ThisModel”.

```
<wsdp:ThisModel ...>
  <wsdp:Manufacturer xml:lang="..."? >xs:string</wsdp:Manufacturer>+
  <wsdp:ManufacturerUrl>xs:anyURI</wsdp:ManufacturerUrl>?
  <wsdp:ModelName xml:lang="..."? >xs:string</wsdp:ModelName>+
  <wsdp:ModelNumber>xs:string</wsdp:ModelNumber>?
  <wsdp:ModelUrl>xs:anyURI</wsdp:ModelUrl>?
  <wsdp:PresentationUrl>xs:anyURI</wsdp:PresentationUrl>?
  ...
</wsdp:ThisModel>
```

Listing 2.11: WS-MetadataExchange Dialect ThisModel

DPWS also defines “ThisDevice” metadata to express characteristics that is different from one device to another like serial-number, a friendly name like “device in kitchen” or firmware version. A WS-MetadataExchange Dialect of “http://docs.oasis-open.org/ws-dd/ns/dpws/2008/09/ThisDevice” indicates an instance of following metadata:

```
<wsdp:ThisDevice ...>
  <wsdp:FriendlyName xml:lang="..."? >xs:string</wsdp:FriendlyName>+
  <wsdp:FirmwareVersion>xs:string</wsdp:FirmwareVersion>?
  <wsdp:SerialNumber>xs:string</wsdp:SerialNumber>?
  ...
</wsdp:ThisDevice>
```

Listing 2.12: WS-MetadataExchange Dialect ThisDevice

An instance of the ThisModel metadata as well as an instance of the This Device metadata must be present in the Metadata if a WS-Transfer GET message is issued to a DPWS device.

Hosting As stated in the beginning of Chapter 2.3, a device also lists the relationship between hosted services and themselves. To express this relationship the following metadata is used:

```
<wsdp:Relationship Type="xs:anyURI" ... >
 (<wsdp:Host>
    <wsa:EndpointReference>endpoint-reference</wsa:EndpointReference>+
    <wsdp:Types>list of xs:QName</wsdp:Types>?
    <wsdp:ServiceId>xs:anyURI</wsdp:ServiceId>
    ...
  </wsdp:Host>)?
 (<wsdp:Hosted>
    <wsa:EndpointReference>endpoint-reference</wsa:EndpointReference>+
    <wsdp:Types>list of xs:QName</wsdp:Types>?
    <wsdp:ServiceId>xs:anyURI</wsdp:ServiceId>
    ...
  </wsdp:Hosted>)*
  ...
</wsdp:Relationship>
```

Listing 2.13: WS-MetadataExchange Dialect Relationship

The <wsdp:Host> and the <wsdp:Hosted> element block defines the Endpoint Reference of a Target Service and a Hosted Service, respectively. Also a optional Type element and a mandatory ServiceID is defined for every Service. The ServiceID must be unique within a Device and has to be persistent across re-initialization. This Relationship metadata equals to a MetadataSection Dialect of “http://docs.oasis-open.org/ws-dd/ns/dpws/2008/09/Relationship”. It is mandatory and has to be included in the <mex:Metadata> block if a client issues a WS-Transfer request to a DPWS device.

Policy WS-Policy and WS-PolicyAttachment is defined by the World Wide Web (W3C) Consortium [12][13]. It provides a general purpose model to describe the policy for Web services. A policy can determine if and how messages have to be secured to meet policy requirements, or simply indicates what profile a device is compliant to (e.g. Basic Profile, Device Profile).

The first specification (WS-Policy Framework) defines the grammar how to express policies. The second specification, the WS-Policy Attachment, is used to apply these formerly expressed polices to different subjects. Polices are very flexible and can be attached to an Endpoint Reference, to messages, or just to an XML block of a SOAP envelope. They can be attached inline, by reference or implicit by using WSDL.

The following example shows a policy expression (a so called policy assertion), saying it supports the Device Profile and Timestamp usage is not required in its messages.

```
<wsp:Policy
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecuri
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy"
xmlns:wsp="http://schemas.xmlsoap.org/ws/2004/09/policy"
xmlns:wsdp="http://docs.oasis-open.org/ws-dd/ns/dpws/2008/09"
wsu:Id="PolicyName" >
 <wsdp:Profile />
 <sp:IncludeTimestamp wsp:Optional="true" />
</wsp:Policy>
```

Listing 2.14: WS-Policy Example

Attaching the former Policy to an arbitrary XML element would look like following (relative anchors works only if the policy assertion and the reference elements are in the same file):

```
<MyElement>
  <wsp:PolicyReference URI="#PolicyName">
...
</MyElement>
```

Listing 2.15: WS-Policy Example 2

Example To show a practical Example of the Description process from a real world scenario, the response of a WS-Transfer GET request directed to a Windows 7 PC is shown in the next lines:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
xmlns:wsx="http://schemas.xmlsoap.org/ws/2004/09/mex"
xmlns:wsdp="http://schemas.xmlsoap.org/ws/2006/02/devprof"
xmlns:un0="http://schemas.microsoft.com/windows/pnpx/2005/10"
xmlns:pub="http://schemas.microsoft.com/windows/pub/2005/07">
<soap:Header>
 <wsa:To>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymou
 <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/transfer/GetResponse
 <wsa:MessageID>urn:uuid:bb28d4dd-e494-4a5e-905a-bf19ef77166c</wsa:Mess
 <wsa:RelatesTo>urn:uuid:924bf530-dbc4-11e1-80fd-7debaa78f337</wsa:Rela
</soap:Header>
<soap:Body>
```

```
16   <wsx:Metadata>
17    <wsx:MetadataSection
18  Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisDevice">
19     <wsdp:ThisDevice>
20      <wsdp:FriendlyName>Microsoft Publication Service Device Host</wsdp:Frien
21      <wsdp:FirmwareVersion>1.0</wsdp:FirmwareVersion>
22      <wsdp:SerialNumber>20050718</wsdp:SerialNumber>
23     </wsdp:ThisDevice>
24    </wsx:MetadataSection>
25    <wsx:MetadataSection
26  Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/ThisModel">
27     <wsdp:ThisModel>
28      <wsdp:Manufacturer>Microsoft Corporation</wsdp:Manufacturer>
29      <wsdp:ManufacturerUrl>http://www.microsoft.com</wsdp:ManufacturerUrl>
30      <wsdp:ModelName>Microsoft Publication Service</wsdp:ModelName>
31      <wsdp:ModelNumber>1</wsdp:ModelNumber>
32      <wsdp:ModelUrl>http://www.microsoft.com</wsdp:ModelUrl>
33      <wsdp:PresentationUrl>http://www.microsoft.com</wsdp:PresentationUrl>
34      <un0:DeviceCategory>Computers</un0:DeviceCategory>
35     </wsdp:ThisModel>
36    </wsx:MetadataSection>
37    <wsx:MetadataSection
38  Dialect="http://schemas.xmlsoap.org/ws/2006/02/devprof/Relationship">
39     <wsdp:Relationship Type="http://schemas.xmlsoap.org/ws/2006/02/devprof/ho
40      <wsdp:Host>
41       <wsa:EndpointReference>
42        <wsa:Address>urn:uuid:1a5b1299-d39e-4e71-8fec-5cc30bad1eae</wsa:Addres
43       </wsa:EndpointReference>
44       <wsdp:Types>pub:Computer</wsdp:Types>
45       <wsdp:ServiceId>urn:uuid:1a5b1299-d39e-4e71-8fec-5cc30bad1eae</wsdp:Ser
46       <pub:Computer>MADMAX/Workgroup:WORKGROUP</pub:Computer>
47      </wsdp:Host>
48      <wsdp:Hosted>
49       <wsa:EndpointReference>
50        <wsa:Address>urn:uuid:1a5b1299-d39e-4e71-8fec-5cc30bad1eae</wsa:Addres
51       </wsa:EndpointReference>
52       <wsdp:Types>pub:HomeGroup_Invitation</wsdp:Types>
53       <wsdp:ServiceId>fdid:Provider%5CMicr<--snip-->ion_ID</wsdp:ServiceId>
54       <pub:Resource>\l?xml\sversion="1.0"<--snip-->RECORD&gt;</pub:Resource>
55      </wsdp:Hosted>
56     </wsdp:Relationship>
57    </wsx:MetadataSection>
58   </wsx:Metadata>
```

```
59 </soap:Body>
60 </soap:Envelope>
```

Control

Before a DPWS client is able invoke an action, it needs to know some information about it. This information can be found in the WSDL file, which was retrieved earlier in the description phase.

Very similar to the UPnP control message, the DPWS control message is packed inside a SOAP envelope. It basically consists of an operation name and one or more input argument(s) embedded in the SOAP body block. The operation name is derived from the *port type* section of the WSDL file. The input argument name and its data type are defined in the *types* section of the WSDL file using an XML Schema Definition (XSD) grammar.

The envelope constructed this way (see Listing 2.16 for an example), is then sent to the address location of the *service* section from the WSDL file to execute the formerly mentioned operation.

```
1  POST /WeatherWS/Weather.asmx HTTP/1.1
2  Host: domain.com
3  Content-Type: application/soap+xml;charset=UTF-8;
4    action="http://domain.com/WeatherWS/GetCityWeatherByZIP"
5  Content-Length: length of body in bytes
6
7  <soap:Envelope
8  xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
9  xmlns:weat="http://domain.com/WeatherWS/">
10    <soap:Header/>
11    <soap:Body>
12       <weat:GetCityWeatherByZIP>     <!-- operation name -->
13          <weat:ZIP>10001</weat:ZIP>  <!-- input message -->
14       </weat:GetCityWeatherByZIP>
15    </soap:Body>
16 </soap:Envelope>
```

Listing 2.16: Web service control invocation

Eventing

WS-Eventing is defined by the World Wide Web Consortium [14]. Normally WS-Eventing utilizes an asynchronous publisher/subscriber model like UPnP (see also Chapter 1.4). On the same time, WS-Eventing is much more flexible and extensible. For example, it is possible to define more delivery modes (other than the standard asynchronous Push Mode).

In order a Web service (subscriber) receives notifications about another Web service (event source), it has to announce its interest by a subscription request. The event source then grants

the subscription or denies it. To improve robustness, a subscription is normally only valid for a certain amount of time and has to be renewed if there is still interest on notifications from the event source.

The SOAP Envelope for a subscription request has to look like following (schematic output):

```
<s:Envelope ...>
  <s:Header ...>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/08/eventing/Subscribe</wsa:Ac
    ...
  </s:Header>
  <s:Body ...>
    <wse:Subscribe ...>
      <wse:EndTo>endpoint-reference</wse:EndTo> ?
      <wse:Delivery Mode="xs:anyURI"? >xs:any</wse:Delivery>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires> ?
      <wse:Filter Dialect="xs:anyURI"? > xs:any </wse:Filter> ?
      ...
    </wse:Subscribe>
  </s:Body>
</s:Envelope>
```

Listing 2.17: WS-Eventing Subscription request

As indicated by the <wse:Filter> element, a subscription can be filtered. This means it is possible to subscribe to one or more notifications exposed by a service. To achieve this, the Device Profile defines a filter dialect designated “http://docs.oasis-open.org/ws-dd/ns/dpws/2008/09/Action”. Inside the <wse:Filter> element with the previously presented dialect there is a white space-delimited list of URIs indicating the [action] property of desired Notifications.

If the event source accepts the subscription, it must respond with following SOAP message:

```
<s:Envelope ...>
  <s:Header ...>
    <wsa:Action>
      http://schemas.xmlsoap.org/ws/2004/08/eventing/SubscribeResponse
    </wsa:Action>
    ...
  </s:Header>
  <s:Body ...>
    <wse:SubscribeResponse ...>
      <wse:SubscriptionManager>
        wsa:EndpointReferenceType
      </wse:SubscriptionManager>
      <wse:Expires>[xs:dateTime | xs:duration]</wse:Expires>
      ...
    </wse:SubscribeResponse>
  </s:Body>
</s:Envelope>
```

Listing 2.18: WS-Eventing Subscription response

The Device Profile suggests to use duration instead of dateTime for subscription expiration, because a DPWS device is only obligated to have an internal clock, but under circumstances has no knowledge about date.

Security

For a DPWS device it is not necessary to support WS-Security in order to be compliant to the Device Profile. But it is recommended to support security. The Device Profile specification also provides a guideline what security features to use and their workflow. That said some DPWS frameworks did not implemented WS-Security, because of complexity and performance issues.

CHAPTER 3

Comparison between UPnP and DPWS

3.1 General

In Chapter 1 and Chapter 2, UPnP and DPWS and their specifications were introduced. In this chapter these two specifications are compared to each other and their similarities will be highlighted as well as their differences.

One obvious thing is their similar logical concept. Almost for every UPnP phase (see Section 1.4) there is a corresponding Web Service specification (see Section 2.3). The concept elements common in both specifications are Addressing, Discovery, Description, Control and Eventing. But there are also elements which do not have any direct counterparts like Presentation on the UPnP side, as well as WS-Policy and WS-Security on the DPWS side.

An important difference between UPnP and DPWS is the significant different degree of flexibility of these two specifications. In UPnP, almost everything is strictly standardized, from the underlying transport mechanism to eventing there is no space for variation. Only the presentation is completely up to the device manufacturer. Because of the tight specification it is ensured that two UPnP devices can talk to each other and are able to execute actions if available. In the DPWS specification, there is a lot more space for interpretation. It is possible to use different transport mechanisms for example, and especially regarding WS-Security and WS-Policy, it is not ensured that every DPWS devices can exchange messages with every other DPWS device other than announce their existence and possibilities. On the other hand, with DPWS it is possible to tailor the devices to the needs of different scenarios.

3.2 Addressing

Despite the same name, addressing has not much in common between UPnP and DPWS. In the UPnP context, addressing is all about obtaining a valid IP address in a network and uses DHCP and IP-autoconfig for this task to fulfil.

Since Web services are network agnostic, and DPWS is aligned to their specific Web service specifications, addressing in the DPWS context has another signification. In DPWS, every message is embedded in a SOAP message, and these messages can be addressed regardless of their underlying network. So addressing in DPWS is responsible for address routing (e.g. specify sender, receiver and who will get a response) among other things like references to former messages and actions.

3.3 Discovery

The Discovery process in UPnP is similar to DPWS not only by name, but also by its concept and workflow. Both uses a mix of active search of devices and a presence announcement when a new device joins the network. While UPnP is using a special protocol named SSDP (Simple Service Discovery Protocol) for Discovery, DPWS solely relies on information embedded in SOAP messages. In both specifications multicast messages are used to reach all participants in a network with one message. The workflow can be divided into several parts and includes presence announcement, device search and a goodbye message.

Presence Announcement When a UPnP/DPWS device joins a network, it will announce itself with a Unique ID and a type indicating which kind of device it is. A UPnP device also gives a direction where to find detailed information of the device.

A UPnP device and a DPWS device have different approaches not only in their structure but also in the amount of Presence announcements. While UPnP device is reporting every device (root and embedded) and every service to the network, a DPWS client is only broadcasting the target service.

Device Search Control-Points (UPnP) or Clients (DPWS) are able to search for devices and may apply a filter, so only specific types of devices are replying. While these types of devices are predefined in UPnP, in DPWS these types are declared by an XML namespace, which point to a definition of the type. DPWS also offers to filter not only by types, but also by context scope.

While UPnP includes the direction to a more detailed description inside its presence announcement, DPWS may include a direction in its search response (a so called *Probe Match*). If the direction (a transport address) is not included in the probe match, or only the unique ID of a presence announcement is available, the client has to issue a *Resolve Request* to the unique ID of the device to obtain a transport address.

Good Bye Both specifications want their devices to say good bye before leaving the network. This way, the other devices do have a clear picture of active participants on the network without polling.

3.4 Description

The workflow how to obtain detailed description is similar in both specifications but they have their own way how to reach their goals.

After the Discovery phase a UPnP and a DPWS device have similar knowledge of other devices on the network. The information includes a unique ID (UUID) and an address where to get more information. The next step is to retrieve further information. A UPnP device retrieves a *Device Description Document* located at the address specified in the *LOCATION* HTTP Header of a Simple Service Discovery Protocol Response (see also Listing 1.2). A DPWS device issues a WS-Transfer Get message to the address retrieved by a Probe or Resolve Message (see also Listing 2.7). This way a device, whether it speaks UPnP or DPWS, gains information about things like the model name, the serial number as well as the ID and the address of services hosted by the device.

A UPnP device can fetch more information about provided services by issuing a HTTP GET request to the service address (SCPDURL), and will retrieve a *Service Description Document* including detailed information about the service capabilities and how to execute them. A DPWS device issues a WS-MetadataExchange request and will get a Web Service Description Language Document in return. This way, the device knows all the functions a service is offering, and what bindings they use.

It has to be noted that the DPWS approach is more flexible and the workflow is not as rigid as in UPnP. DPWS can use different ways to gain the same knowledge.

3.5 Control

Although the service definition heavily differs from DPWS and UPnP, the control messages itself are very similar. The main difference is that UPnP uses SOAP1.1 binding and DPWS is obligated to use SOAP1.2 binding. Surprisingly, this has more affect on the HTTP headers as on the SOAP envelopes itself, which are almost identical despite their different namespaces.

While it is common sense in DPWS context to deploy your own functions including your own function- and argument-names, this does not necessarily apply in the UPnP context. Normally a UPnP forum working group defines standard services, actions, arguments and state variables. If a UPnP vendor wants to extend a standard service there are some conventions to keep in mind. For example, action names and state variables have to begin with "X_" followed by a vendor domain name and another underscore before the actual action or state variable name. If a UPnP vendor does not extend an existing service, but wants to define a completely new one, it has to define a new serviceType within their own vendor domain name. See also Figure 4.6 for an example.

3.6 Eventing

Like in previous sections, UPnP and DPWS also use similar concepts for Eventing (asynchronous publish/subscribe model). But again, like in previous sections, DPWS is more flexible and extensible than UPnP and can be extended to use other, so called *Delivery Modes* as well.

The ordinary asynchronous, event-based publish/subscriber model can be divided into subscribe, unsubscribe and notify.

Analogue to the discovery phase, where UPnP established a new protocol (SSDP), it also introduces a new protocol for the eventing task by creating new HTTP Headers. The name of the protocol is General Event Notification Architecture (GENA). On the other side, and also analogue to its corresponding discovery phase, DPWS does not alter the HTTP Headers but encapsulates all eventing information inside a SOAP envelope.

CHAPTER 4

Translation between UPnP and DPWS

4.1 Related Work

There are many papers covering the idea of making different home networks compatible to each other. Basically this can be achieved in two different ways:

- translate between different networks directly,
- try to unite different home networks under one roof with the help of an intermediate protocol/middleware.

A very generic approach is the *Automatic Generation of Network Protocol Gateways* [15]. The idea is to describe the protocol in several configuration files and then use the provided compiler to automatically generate the gateway binary.

Service-Oriented Device Communications Using the Devices Profile for Web Services [16] explains the DPWS architecture and did a performance test of an implementation of DPWS running on resource restricted devices. The paper also presents a gateway approach which transforms dumb or legacy devices into DPWS devices.

A bridge between UPnP and the fieldbus KNX (EIB) is presented and implemented by *UPnP Connectivity for Home and Building Automation - A case Study for EIB* [17]. The author created a bridge and a UPnP control point and succeeded in integrating KNX devices into a UPnP network.

Web Services on Universal Networks [18] (WSUN) is a SOA-based framework supporting dynamic service discovery on different network environments like Jini [19], UPnP and DPWS. The services within WSUN also consider service location and service status during discovery.

The paper *Using DPWS to Bridge Isolated OSGi Platforms* [21] presents a solution to open up OSGi [20] platforms and make them DPWS enabled devices. This way, not only different OSGi platforms can communicate to each other, but also native DPWS devices can utilize OSGi services.

An already existing implementation for combining UPnP and DPWS networks is introduced in *Home SOA – Facing Protocol Heterogeneity in Pervasive Application* [22]. This very powerful implementation uses the OSGi platform to combine not only UPnP and DPWS but also fieldbus protocols like X10 or ZigBee. Theoretically it can be extended via so called base drivers to support any other protocol as well.

4.2 Concept

The existence of a black box gateway is assumed. This gateway is able to speak UPnP and DPWS and is capable of translate between these two specifications. What functions this gateway has to provide exactly will be discussed in this chapter.

Translation Idea When the gateway receives a UPnP message, it has to analyse and parse the message and hold the interesting parts in memory. These parts need to be filled into a suitable DPWS stub and then sent to the DPWS network.

If the gateway receives a DPWS message it also needs to extract certain parts and put it into a UPnP stub. This generated message will be sent to the UPnP network. If the DPWS and UPnP devices reside on the same network, it must be ensured the gateway does not translate its own messages (prevent a message-loop).

So the idea is to define what parts of a message have to be parsed or generated, to create a set of message stubs, and also to define where to put the former extracted parts into the stubs in order to fulfil the protocol translation.

4.3 Address Mapping

There are a few requirements for the gateway regarding addressing. One is to support DHCP and Auto-IP. Also a Message ID and sometimes an AppSequence has to be generated and filled into a suiting stub. See also next chapter or Figure 4.1 for more information and examples.

4.4 Discovery Mapping

A first step for a UPnP-DPWS gateway is to introduce all devices available in the network to all other devices, regardless if they speak UPnP or DPWS. So a UPnP Device needs to know the existence of a DPWS client in the network and vice versa. Therefore, it has to translate the “presence announcement” as well as the “device searches” and “good-bye messages” from UPnP to DPWS and from DPWS to UPnP.

Presence Announcement

UPnP to DPWS When the gateway receives a Simple Service Discovery Protocol Multicast Message, it has to parse the USN header containing the UUID of the device, the NT header containing the device type, as well as the LOCATION header. On the USN header value, the

last part "::upnp:rootdevice" needs to be cut off. See also Figure 4.1 to compare the UPnP and DPWS Presence announcement and to look at a proposal of how to translate between them.

The preparation of the DPWS message is divided into a few tasks. The following information has to be placed in a DPWS stub:

- generating a unique Message ID
- generating a Web service Discovery "AppSequenc" (an InstanceID and a MessageNumber)
- placing Address Information parsed from UPnP message inside the <wsa:Address> brackets
- mapping the type information, e.g. for a root device the gateway must map from UPnP type upnp:rootdevice to DPWS type dpws:Device.

DPWS to UPnP Translation from DPWS to UPnP is a similar task like in the previous paragraph, but of course in the opposite direction. See Figure 4.1 for more details.

One thing to consider is the fact, that UPnP is using time-outs for message validation while DPWS marks messages with a MetadataVersion to keep track of valid information. Since they use a fundamental different approach for defining how long a Presence Announcement is allowed to be cached, it is not possible to translate between them and a default value for the CACHE-CONTROL Header have to be assumed. Another thing to consider is that a DPWS message does not necessarily transmit a transport address within its Presence Announcement. Therefore, a Resolve Request (see Listing 2.8) is possibly needed before a UPnP equivalent of a Hello Message can be sent to the network (including the LOCATION header).

Device Search

UPnP to DPWS When the gateway is discovering a UPnP device search, only the device type has to be remembered and mapped to a corresponding DPWS device type. A Message ID also has to be generated and inserted into the DPWS stub.

DPWS to UPnP Like before, a mapping between their types have to be made. If there is a search filter for scopes it gets a little complicated since UPnP does not support scope search. A decision has to be made if such messages do not return anything at all, or just ignore the scope and return all devices within the right type. See Figure 4.2 for a graphical representation of the translation.

Good Bye

The Good-Bye message is very similar to the presence announcement, especially the SOAP Header, but also the SOAP Body can be considered as a simplified twin of a presence announcement.

Figure 4.1: UPnP and DPWS Presence Announcement mapping

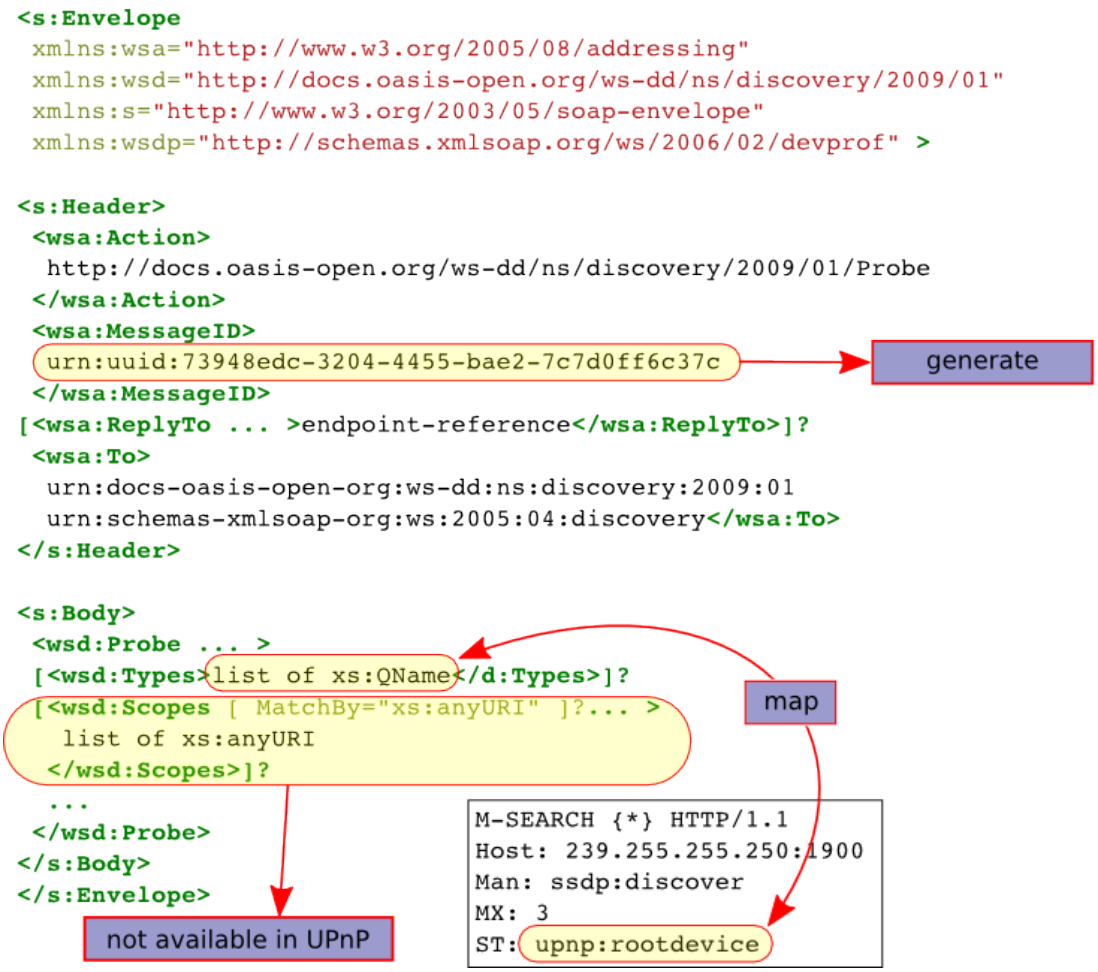

Figure 4.2: UPnP and DPWS Device Search mapping

UPnP to DPWS Parsing a UPnP Good-Bye message is easy since only the USN Header need to be remembered and placed in the DPWS stub. Within the Good-Bye message the namespace and address declarations are the same as in the presence announcement despite a “Hello” becomes a “Bye” in the <wsa:Action> brackets. Only the SOAP Body differs slightly. Therefore and to keep things short Figure 4.3 misses the namespace and address declaration and only show the SOAP Body of the DPWS Good-Bye stub.

DPWS to UPnP The <wsa:Address> value will be inserted in the UPnP stub at the USN Header position and is extended by the string “::upnp:rootdevice”

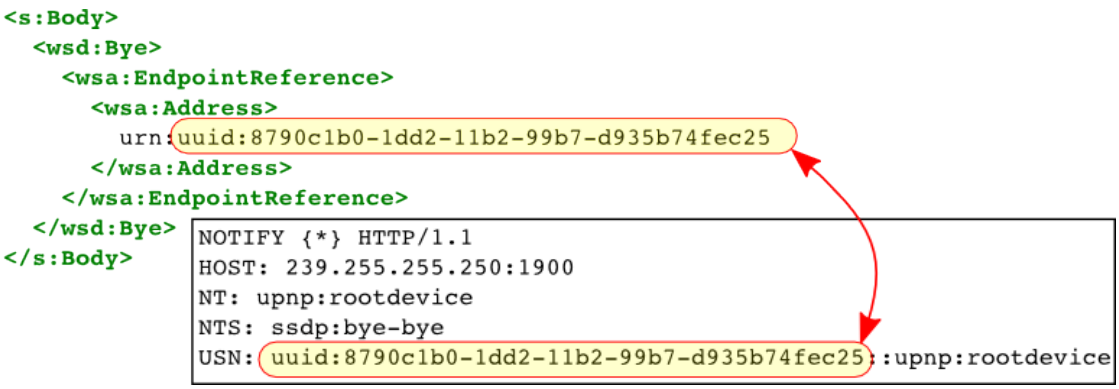

Figure 4.3: UPnP and DPWS Good-Bye mapping

4.5 Description Mapping

Sharing description information between UPnP and DPWS devices is maybe the most challenging task in basic translation.

Device Description

Figure 4.4 proposes a way how to translate device information. Unlike obvious translation mappings like “Manufacturer”, “SerialNumber” and so on, there are also some fields which do not have any direct counterpart. For some of these fields the gateway has to provide some kind of a conversion service.

For example, if a UPnP device wants to know some details about a previously found DPWS device, this conversion service has to generate a Service Description URL (SCPDURL) for every Endpoint Reference Address (UUID) embedded inside the DPWS “Relationship-Metadata-Dialect” section and place it in the UPnP Device Description stub. The gateway needs to store this mapping inside an internal table to keep track of the mapped SCPDURLs and UUIDs.

In order to get the information needed for the <controlURL> brackets, the gateway has to retrieve the WSDL file and extract the information from there. In the <binding> section of the WSDL file, particularly in the <soap:operation> brackets, the desired URL for mapping to the controlURL is found.

This way, the gateway can translate between the Device Description Document and the corresponding Metadata parts inside the DPWS messages.

Service Description

If a UPnP device wants to know more details about services of a DPWS client, it issues an HTTP GET command to a Service Description Document URL (SCPDURL). This SCPDURL was generated by the gateway as described above in the Device Description process. So the UPnP Device acts in the same way as it would in a UPnP only conversation.

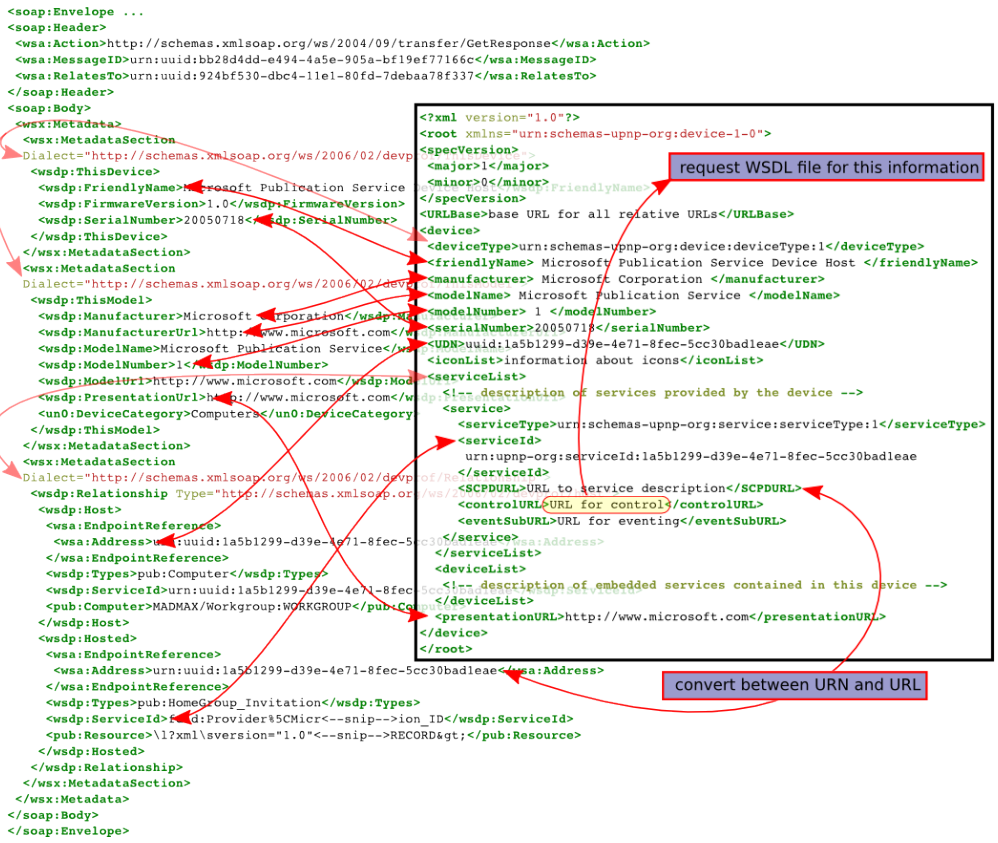

Figure 4.4: UPnP Device Description and DPWS Metadata mapping

In order to fulfil the Device Description request, the gateway already retrieved the WSDL file of the DPWS device of interest. So the gateway already has all information needed to complete the Service Description stub and sends it back to the UPnP Device, which wanted to know about the DPWS client services in the first place. See Figure 4.5 for a proposal of a possible mapping. Basically the *action name* of the <actionList> section corresponds with the WSDL *operation name* of the <portType> section, the *argument name* corresponds with the *message name*, and the related variables from the <actionList> as well as the <serviceStateTable> section correspond with elements inside the <schema> section of the WSDL file.

4.6 Control Mapping

Considering a UPnP control message and its translation into a DPWS one, there are three main things to take care of:

- perform a SOAP1.1 to SOAP1.2 conversation

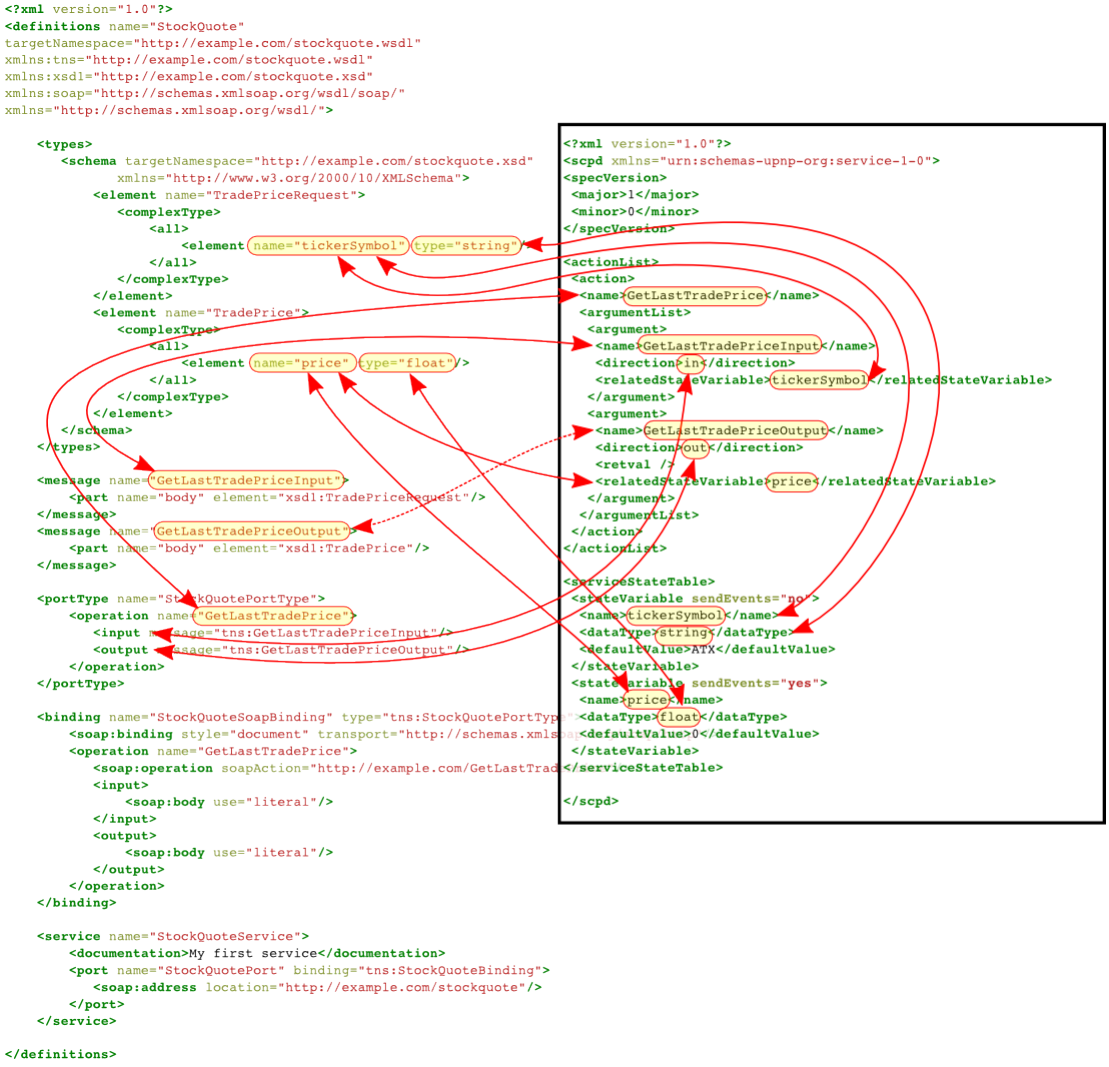

Figure 4.5: UPnP Service Description and DPWS WSDL mapping

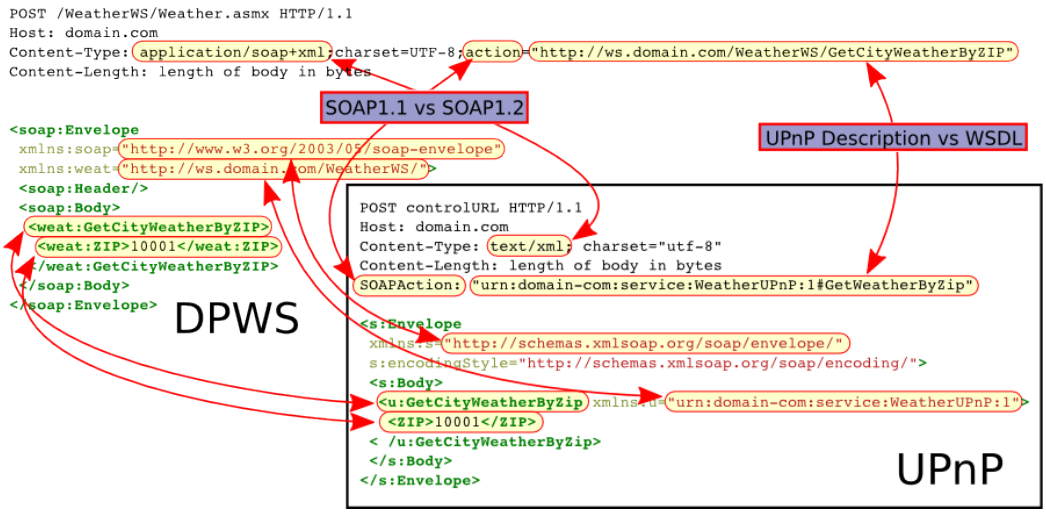

Figure 4.6: UPnP and DPWS control mapping

- translate the SOAPaction value

- adjust the service namespace

In order to do a SOAP1.1 to SOAP1.2 conversation the XML namespace for the SOAP envelope has to be changed from "http://schemas.xmlsoap.org/soap/envelope/" to "http://www.w3.org/2003/05/soap-envelope". Also some HTTP headers requires a modification. The Content-Type header changes from "text/xml" to "application/soap+xml" and the (converted) value from the SOAPaction header is concatenated to the same header. The SOAPaction header itself disappears in the SOAP1.2 HTTP binding.

The task of the SOAPaction value conversation already happened during Device/Service Description to WSDL file translation in the description mapping (see also 4.5). If this translation has been successful the right value can be looked up in the corresponding file(s). The value for the DPWS stub is found in the WSDL binding section (<soap:operation soapAction="value"/>). The information needed for the SOAPaction header inside a UPnP control message, is found in the Device Description (urn:domain-name:service:serviceType:v) and the Service Description (#ActionName).

Also the namespace of the action itself need to be adjusted accordingly (eg. "urn:domain-com:service:WeahterUPnP:1" to "http://domain.com/WeatherWS"). Again this information can be looked up in the description files (Device Description respectively WSDL).

See also Figure 4.6 for an example of how to translate an action invocation.

4.7 Event Mapping

Subscribe

If a device likes to subscribe to an event source, it needs almost the same information in both architectures (UPnP and DPWS). The information includes an address of the event source to which the device wants subscribe to, a callback address where the notifications should go to and the amount of time the subscription should be valid.

But like so many times before, the flexibility of DPWS makes it difficult to achieve a real translation of these messages.

One problematic issue might be the possibility of DPWS to distinguish between the address where the reply to the subscription should be sent, and the address where the actual notification goes to. In other words the possibility of having a subscription manager is absent in UPnP.

Another issue is that the DPWS Endpoint Reference is often extended by <wsa:ReferenceProperties> to carry proprietary information. Often this extension is used to keep track of the subscriptions and therefore holds a subscription ID. UPnP also knows the concept of a subscription ID, but in a UPnP network the event source determines this ID (in UPnP context also referred to SID).

Furthermore, DPWS has the ability to set a filter for a subscription, so it only receives a notification if the filter criteria are met. In UPnP, a device issues a subscription to a service and receives all updates of this particular service.

Another minor issue to take into account is that DPWS may express the expire time in date-time or in seconds whereas UPnP only takes care of seconds. A simple conversion is possibly needed if DPWS uses a datetime notation.

Figure 4.7 compares the subscription message between UPnP and DPWS and outlines what information has to be exchanged.

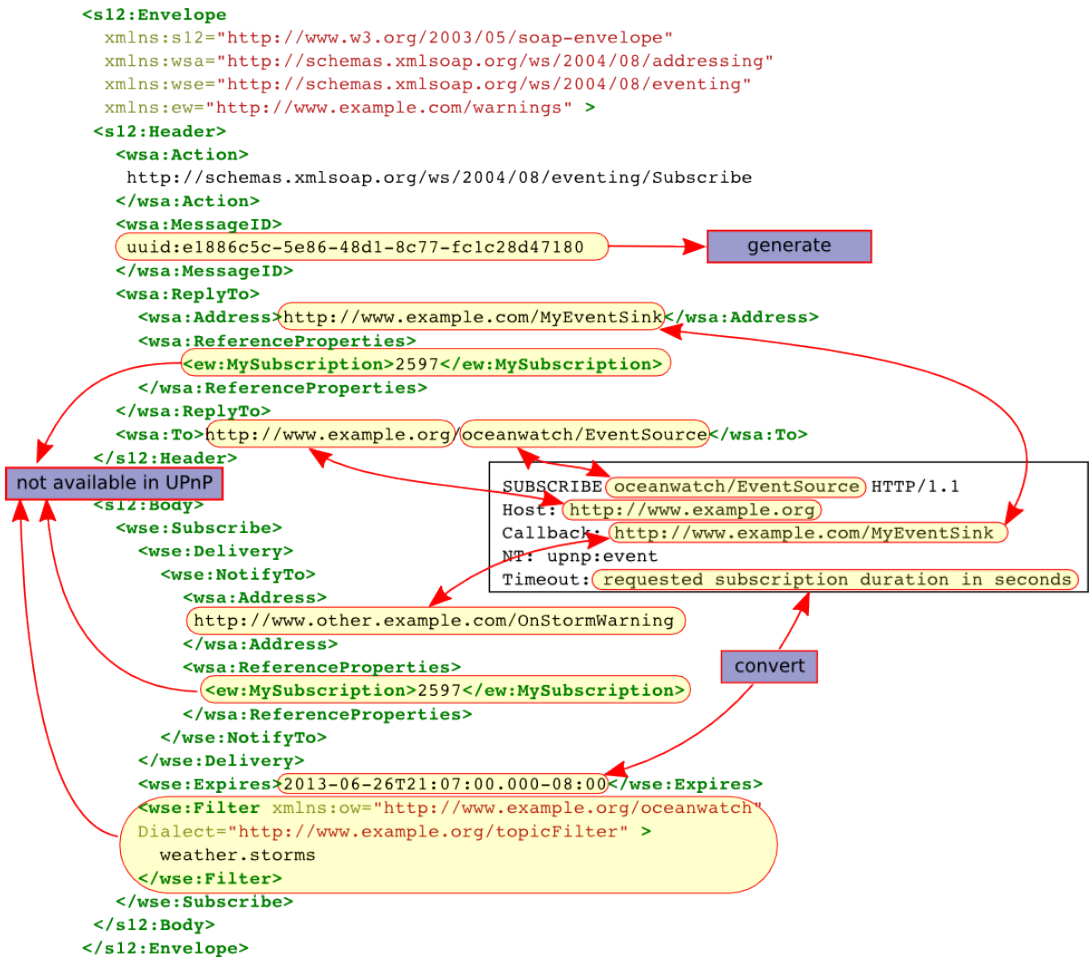

Figure 4.7: UPnP and DPWS Subscription mapping

CHAPTER 5

Conclusion

5.1 General

Due to the great flexibility of DPWS it is difficult to generate a universally valid gateway between UPnP and DPWS. But since structure and workflow are similar, under certain circumstances it should be possible to hand craft a translator between these two specifications.

One key part for translation is to find a way how to map types and functions since they are kind of static in UPnP (predefined by a working committee of the UPnP Forum) but not predefined at all in DPWS. In DPWS, every manufacturer define own types and functions and include their definitions via namespace. Therefore, it is better to know them in advance and do a manual mapping e.g. via a table.

Another thing to keep in mind is even though the communication between these two specifications might be possible, this does not mean all end-user devices are compatible to each other. There are still a lot of pitfalls to come over like application-specific protocols on top of the basic specification (e.g. UPnP-AV). While it might be possible to extend the gateway in a way the application-specific protocols are supported too, there are scenarios where no communication between UPnP and DPWS devices is possible at all, like incompatible policy or security requirements.

CHAPTER 6

Bibliographic Issues

Bibliography

[1] Microsoft Cooperation. Understanding Universal Plug and Play - www.upnp.org/download/UPNP_UnderstandingUPNP.doc

[2] UPnP Forum. UPnP Device Architecture 1.1, http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf, October 2008

[3] Michael Jeronimo, UPnP Design by Example , Jack Weast. 2003 Intel Press

[4] HTTP/1.1, http://datatracker.ietf.org/doc/search/?name=http&activeDrafts=on

[5] Toby Nixon, et al, Devices Profile for Web Services Version 1.1, http://docs.oasis-open.org/ws-dd/dpws/1.1/os/wsdd-dpws-1.1-spec-os.pdf, July 2009

[6] Martin Gudgin, et al, Web Services Addressing 1.0 Core, http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/ , 9 May 2006

[7] Toby Nixon, et al, Web Services Dynamic Discovery (WS-Discovery) Version 1.1, http://docs.oasis-open.org/ws-dd/discovery/1.1/wsdd-discovery-1.1-spec.html, July 2009

[8] T. Berners-Lee, et al, Uniform Resource Identifiers (URI): Generic Syntax , IETF RFC 3986, http://www.ietf.org/rfc/rfc3986.txt, January 2005

[9] Jan Alexander, at el, Web Services Transfer (WS-Transfer), http://www.w3.org/Submission/2006/SUBM-WS-Transfer-20060315/, March 2006

[10] Keith Ballinger, at el, Web Services Metadata Exchange 1.1 (WS-MetadataExchange), http://www.w3.org/Submission/2008/SUBM-WS-MetadataExchange-20080813, August 2008

[11] Erik Christensen, Web Services Description Language (WSDL) 1.1, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, March 2001

[12] Siddharth Bajaj, et al, Web Services Policy 1.2 - Framework (WS-Policy), http://www.w3.org/Submission/2006/SUBM-WS-Policy-20060425, April 2006

[13] Siddharh Bajaj, et al, Web Services Policy 1.2 - Attachment (WS-PolicyAttachment), http://www.w3.org/Submission/2006/SUBM-WS-PolicyAttachment-20060425/, April 2006

[14] Don Box, et al, Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/WS-Eventing/, March 2006

[15] Automatic Generation of Network Protocol Gateways - Yerom-David Bromberg , Laurent Reveillere , Julia L. Lawall , Gilles Muller. 2009 Springer Verlag

[16] Service-Oriented Device Communications Using the Devices Profile for Web Services - François Jammes, Antoine Mensch, Harm Smit. 2005 ACM 1-59593-268-2/05/11

[17] UPnP Connectivity for Home and Building Automation A case Study for EIB; Horst Scheichelbauer. 2003, TU Vienna, Master Thesis

[18] Web Services on Universal Networks - Yun-Young Hwang, Il-Jin Oh, Hyung-Jun Yim, Kyong-Ha Lee, Kangchan Lee, Seungyun Lee, Kyu-Chul Lee. Chungnam National University&Electronic and Telecommunications Research Institute, Korea

[19] Jini Community, http://www.jini.org.

[20] OSGi Alliance, OSGi Service Platform Core Specification and Service Compendium - Release 4, Version 4.2, 2009

[21] Towards the Web of Things: Using DPWS to Bridge Isolated OSGi Platforms - Oliver Dohndorf, Jan Krüger, Heiko Krumm; Christoph Fiehe, Anna Litvina, Ingo Lück, Franz-Josef Stewing. 2008 IEEE 978-0-7695-3367-4/08

[22] Home SOA – Facing Protocol Heterogeneity in Pervasive Application - André Bottaro, Anne Gérodolle. 2008 ACM 978-1-60558-135-4/08/07