Device and Service Templates for the Devices Profile for Web Services

Andreas Bobek, Elmar Zeeb, Frank Golatowski, Dirk Timmermann
Institute of Applied Microelectronics and Computer Engineering
University of Rostock
Email: (andreas.bobek, elmar.zeeb, frank.golatowski, dirk.timmermann)@uni-rostock.de

***Abstract*— The continual improvements in embedded devices and internet technologies increases the leverage of distributed systems in domains like home automation, industry automation, automotive and other application domains. The *Devices Profile for Web Services* (DPWS) is a relative new specification based on Web services that can be used as a cross domain distributed system technology. Although the specification is almost finished and software tools are on the way there are some issues left to be done for pushing DPWS in a competitive direction. DPWS still misses a description mechanism that defines interfaces to the services/devices at development time. On the other hand DPWS provides the concept of device and service types which are used for dynamic discovery without proposing a formal type description. Such mechanism improves interoperability between related devices. This paper addresses this issue and proposes a template approach similar to *Universal Plug and Play* (UPnP).**

I. Introduction

The continual improvements in embedded devices and internet technologies increases the leverage of distributed systems in domains like home automation, industry automation, automotive and other application domains. Because of the complexity of embedded distributed systems there is a need for an intelligent solution to discover, control and manage devices. In the past technologies like *Java Intelligent Network Infrastructure* (Jini) [1], *Universal Plug and Play* (UPnP) [2] and *Home Audio/Video Interoperability* (HAVi) [3] have proven to manage this complexity and fit into one of these domains. However, they have not gained influence as cross domain technology. The *Devices Profile for Web Services* (DPWS) [4] is a relative new specification based on Web services that can be used as a cross domain distributed system technology. DPWS is based on SOAP Web services and is very close to the *Web Services Architecture* (WSA) [5] specified by the *World Wide Web Consortium* (W3C). DPWS allows for the implementation of devices and applications that combine several devices as *Service Oriented Architecture* (SOA). Embedded distributed systems implemented as SOA and the increasing acceptance of Web services provide a promising approach to solve the complexity of distributed embedded applications and propagate DPWS as cross domain technology.

The *Service Oriented Architectures for Devices* (SOA4D) [6] and *Web Services for Devices* (WS4D) [7] initiative deal with the implementation of SOAs with DPWS in cross domain scenarios. Both resulted from the ITEA project *Service Infrastructure for Real-time Embedded Networked Applications* (SIRENA) [8][9] and are carried on within the ITEA projects *Local Mobile Services* (LOMS) [10] and *Service Oriented Device Architectures* (SODA) [11]. WS4D and SODA aim at making the knowledge around DPWS public in the form of open source publications and by forming a community for device-centric SOAs.

Although the specification is almost finished and software tools are on the way there are some issues left to be done for pushing DPWS in a competitive direction. The afore mentioned technologies come with inbuilt description mechanisms which define interfaces to the services at development time. For example Jini uses Java interfaces, Web services use *Web Service Definition Language* (WSDL) and UPnP has a template specification. In DPWS such mechanisms are still missing. This paper addresses this issue and proposes a template approach similar to UPnP.

II. Related work

A. Web services and DPWS

Today WSA and Web services technology are considered to be the prevalent form of SOA implementation. Due to the open specification process and the amount of requirements we are now confrontated with more than 60 specifications and binding protocols [12] (which define collaboration of two or more specific WS protocols). Profiles are means of constraining heavy protocol families for targeting a specific domain.

The Devices Profile for Web Services defines a profile over a specific set of Web services protocols to enable secure Web service capabilities on resource-constraint devices. It allows clients sending and receiving secure messages to and from Web services, dynamically discovering of, describing of, subscribing to, and receiving events from DPWS-enabled devices.

Hence, DPWS allows for easy integration of ad-hoc device environments into the Web services world by leveraging the Web services protocol family. This is one of its main benefit. Further DPWS is applied to the latest Microsoft operating system – Windows Vista – where it enables hardware components to plug into the system and therefore is intended to replace special drivers.

Web services and thus DPWS may be applied to heterogenous environments (e.g. operating systems, platforms, programming languages...). They were formed by an open specification process and are maintained by a large community

which give them an enormous potential as the future standard SOA.

DPWS utilizes several protocols such as WS-Discovery, WS-Eventing, WS-Transfer, WS-Policy, SOAP-over-UDP and others.

B. Device and service templates in UPnP

UPnP's original goal was not targeted to come up with a technology stack for spontaneous networked devices and services but to provide a framework for application protocols for distributed device ensembles. Today UPnP's application protocols are their main benefit.

The *Device Control Protocols* (DCP) [13] are building on a template and description system that is part of the *UPnP Device Architecture* (UDA). There are device and service templates. Device templates refer to services they use. Service templates specify optional and mandatory actions as well as state variables they provide. While each vendor may provide his own descriptions the *UPnP Forum* develops DCPs regularly in a standardized process. DCPs target devices/services which are identified to be used by different vendors and hence are expected to become standards.

This kind of process and definition of semantics is essential for competing technologies to ensure interoperability. While Web services have their own description format – WSDL – something similar to UPnP's typing system doesn't exist.

III. Motivation and requirements

Currently there is no formalism available for defining a service type or a device type. This is not quite correct as WSDL has the concept of port types where XML qualified names [14] stand for sets of specific operations. When searching for devices/services with DPWS the WS-Discovery protocol utilizes its own version of *types* which are also XML qualified names, but have no further semantics. We can take advantage of this fact and should define some semantics around these discovery types.

Further we want to shorten the discovery process which may take up to four sequential steps – depending on the environment and dynamic aspects of the system. As devices join and leave the network they are sending *Hello* and *Bye* messages accordingly for announcing their presence/absence. Clients wishing to use specific services have to search (probe) for them. Listing 1 and 2 show two example messages which also demonstrate the usage of DPWS types.

```
<?xml version='1.0' encoding='utf-8'?>
<s12:Envelope
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
      addressing"
  xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/
      discovery">
  <s12:Header>
    <wsd:AppSequence wsd:InstanceId="1193327769"
        wsd:MessageNumber="1" />
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</
        wsa:To>
    <wsa:MessageID>urn:uuid:93F5AF9E3E4FCC8A911193327769653
        </wsa:MessageID>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/
        discovery/Hello</wsa:Action>
  </s12:Header>
  <s12:Body>
    <wsd:Hello>
      <wsa:EndpointReference>
        <wsa:Address>urn:uuid:ce35ec80-f1b4-11dc-bfea-
            dbe7839e6721</wsa:Address>
      </wsa:EndpointReference>
      <wsd:Types xmlns:ns1="http://schemas.xmlsoap.org/ws
          /2006/02/devprof">ns1:Device</wsd:Types>
      <wsd:Scopes></wsd:Scopes>
      <wsd:MetadataVersion>1193327769</wsd:MetadataVersion>
    </wsd:Hello>
  </s12:Body>
</s12:Envelope>
```

Listing 1. Hello message example

```
<?xml version='1.0' encoding='UTF-8' ?>
<s12:Envelope
  xmlns:s12="http://www.w3.org/2003/05/soap-envelope"
  xmlns:my="http://myuri"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/
      addressing"
  xmlns:wsd="http://schemas.xmlsoap.org/ws/2005/04/
      discovery">
  <s12:Header>
    <wsa:MessageID>urn:uuid:ce35ec80-f1b4-11dc-bfea-
        dbe7f0d72218</wsa:MessageID>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2005/04/
        discovery/Probe</wsa:Action>
    <wsa:To>urn:schemas-xmlsoap-org:ws:2005:04:discovery</
        wsa:To>
    <wsa:ReplyTo>
      <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/
          addressing/role/anonymous</wsa:Address>
      </wsa:ReplyTo>
  </s12:Header>
  <s12:Body>
    <wsd:Probe>
      <wsd:Types>my:Device</wsd:Types>
    </wsd:Probe>
  </s12:Body>
</s12:Envelope>
```

Listing 2. Probe message example

Discovery in DPWS is a four-step process in which firstly devices are discovered using the WS-Discovery protocol – refered as *device discovery* – and afterwards services are discovered by obtaining their WSDL descriptions via WS-Transfer – refered as *functional discovery* (fig. 1). Announcing of and probing for devices in a local network is based on a multicast group predefined in WS-Discovery (step 1). Clients identify a device by its unique logical address (WS-Addressing) which is network independent and statically assigned for the complete lifetime of the device. In step 2 a logical address is resolved into a transport-specific address (e.g. HTTP URL) and in step 3 device metadata (model data, device data...) is transferred to the client using direct unicast transport. Device metadata contains references (addresses) to all hosted services whose description (WSDL) is transferred in step 4. If clients have previous knowledge of their environments some of these steps could be omitted. Moreover, if discovery types (step 1) would already semantically bound to WSDLs (step 4), clients only need to probe before using a special service.

Starting with these motivations we can conclude some requirements which must be fulfilled in order to get a template system that can be practically applied.

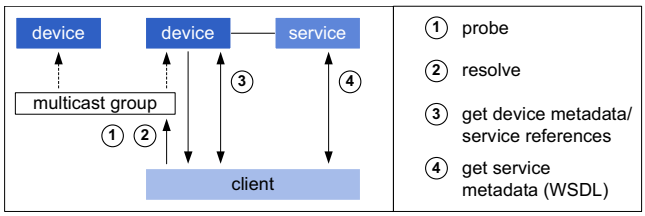

Fig. 1. DPWS discovery steps

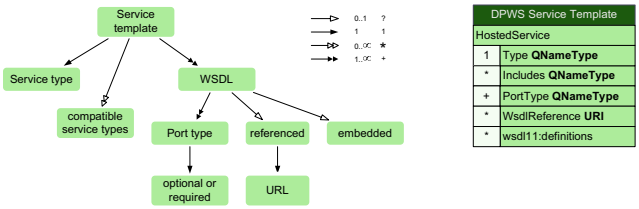

Fig. 2. Service template structure

The system should be easily used at both development time and runtime. At development time code generation should be possible. Generating service and client code based on a WSDL document is a usual and preferred way when implementing or working with Web services (saving of time, decrease of erros etc.). By applying this approach to the templating system we can generate device code, service code and client code (service usage and discovery-related code) at the same time.

As services (devices) and their interfaces evolve over time, we need some kind of version mechanism that also cooperates with the discovery mechanism. There may be additional operations to services or even additional services to device types in the evolution process.

In some cases devices are modeled by using two ore more services of the same type. In such cases the templating system must differentiate between two or more service instances.

Other requirements concern boundary conditions such as extensibility of device and service types, formal expression which allows for automatical processing, and accessibility as open and distributed type definitions are expected.

IV. Design of the DPWS device and service templates

Templates are expressed in XML as this is the dominated markup language in Web services environments. For validity checks we further provide an XML schema [15]. For reusing the semantics given for the XML elements in the WS-Discovery schema some of our elements are named according to their counterparts. This does not mean that these elements or types are equal.

Further we devide template definitions in device and service templates similar to UPnP. Services can then be reused for different devices. For example a *power service* for putting devices in on/off/sleep mode may be reused across device types.

A. Service templates

Figure 2 shows the conceptual and formal structure of service templates, listing 3 contains an example where we define a power service around the type `ws4dp:Power2.0`. Note that we provide an own type for XML qualified names that distiguish between namespace URI and local name by using according elements. This approach is more suitable for parsers or XSL transformers than the inline notations where qualified names are represented as text nodes used in WS-Discovery and other protocols.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:HostedService xmlns:t="http://www.ws4d.org/templates/">
  <t:Type>
    <t:URI>http://www.ws4d.org/templates/power</t:URI>
    <t:localName>Power2.0</t:localName>
  </t:Type>
  <t:Includes>
    <t:URI>http://www.ws4d.org/templates/power</t:URI>
    <t:localName>Power1.0</t:localName>
  </t:Includes>
  <t:PortType>
    <t:URI>http://www.ws4d.org/templates/power</t:URI>
    <t:localName>PowerPortType</t:localName>
  </t:PortType>
  <t:WsdlReference>
     http://www.ws4d.org/templates/power/Power2.0.wsdl
  </t:WsdlReference>
</t:HostedService>
```

Listing 3. A service template example

With service templates we get service types mapped to specific port types of WSDL descriptions. WSDL service descriptions provide the concept of *port types* where operations are grouped together to interfaces. Service types can be linked to any number of port types. Since port types could be defined at different places we provide both, referencing WSDLs by an URI within `t:WsdlReference` and embedding them as `wsdl11:definitions` elements.

Further we introduced the `t:PortTypes` element which contains a list of mandatory port type qualified names that must be present in an implementation. Other port types defined in the service descriptions are optional. This approach allows for more readable type definitions as you can see the required interfaces (port type); and it opens for extensibility as implementors could add some vendor-specific services without breaking the actual service type definition. As mentioned above a port type can group several operations. Thus, all operations belonging to one of the named port types are required in the implementation.

As services will evolve over time we defined the `t:Includes` element. Such element refers to a service type (qualified name) that is entirely included in the newly defined service type. However, that means the new service has to be completely downwards compatible to each included type. Since there could be any occurences of `t:Includes` elements service types are constructed by multiple inheritance.

Both, the enumeration of required port types and the concept of service type inclusion (multiple inheritance) features the creation of layered service types. For example a service type `S1` requires port type `P1`. A second more specific service type

`S2` could then be created by including `S1` and requiring port types `P1`, `P2` and `P3`. However, both type definitions can be based on the same WSDL description.

B. Device templates

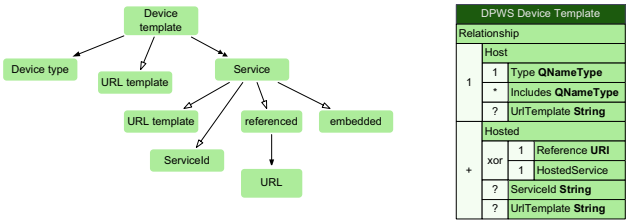

Fig. 3. Device template structure

Figure 3 shows the conceptual and formal structure of device templates, listing 4 contains an example where we define a device type for a webcam.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:Relationship xmlns:t="http://www.ws4d.org/templates/">
  <t:Host>
    <t:Type>
      <t:URI>http://www.ws4d.org/templates/webcam</t:URI>
      <t:localName>Webcam</t:localName>
    </t:Type>
    <t:UrlTemplate>http://{ip}:4672/webcam</t:UrlTemplate>
  </t:Host>
  <t:Hosted>
    <t:Reference>
      http://www.ws4d.org/templates/power/PowerService2.0.
          xml
    </t:Reference>
    <t:ServiceId>
      http://www.ws4d.org/power/powerservice2
    </t:ServiceId>
  </t:Hosted>
  <t:Hosted>
    <t:Reference>
      http://www.ws4d.org/templates/config/
          BasicDeviceConfig2.1.xml
    </t:Reference>
  </t:Hosted>
  <t:Hosted>
    <t:Reference>
      http://www.ws4d.org/templates/webcam/Webcam1.0.xml
    </t:Reference>
    <t:UrlTemplate>http://{ip}:4672/webcam/service</
        t:UrlTemplate>
  </t:Hosted>
</t:Relationship>
```

Listing 4. A device template example

DPWS uses the special concept of *Relationship* for device descriptions. Relationships contain exactly one *host* (the device for that the relationship is defined) and several *hosted* elements (the services the host offers). This concept is applied to our device type definitions.

Within `t:Host` the device type is declared. Here again `t:Includes` elements can be enumerated. Devices that implement this type must be downwards compatible to the included types.

For each service type the device type offers there is one `t:Hosted` element. It contains either an URI reference to a service type definition or the actual `t:HostedService` element goes inline. To each service a specific service ID can be applied. Service IDs are used within DPWS to identify service instances unambiguously. Thus, it is poosible for a device type to include two services of the same type but with different IDs.

URL templates are used for establishing fixed URL parts and linking them to a service instance or device type. In the example above the device is bound to the port 4672 and the path is also forced. Therefore, when knowing the IP address the device can be accessed without further discovery processes. Parts of the URL that have to be dynamically resolved are quoted with curly braces as defined in [16].

There is always a trade-off between referencing parts of the specification and embedding them statically. Reference targets could be changed or manipulated whereas embeddings make reusability and readability more difficult. Real-world device and service specifications should always be published as human-readable documents. The provided notations should then be part of this document.

C. Discovery revised

As already shown, a complete discovery process with DPWS contains several steps. One objective of device and service templates is the reduction of steps in the discovery process to shorten its duration and reduce network traffic.

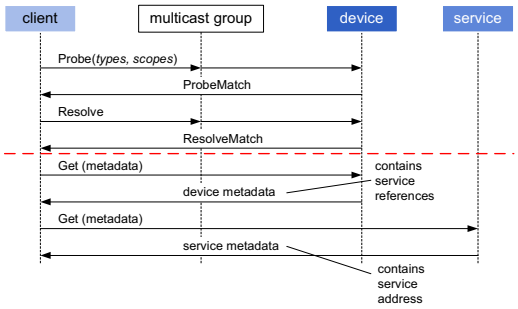

Fig. 4. Shortened discovery

For our device example the network addresses of the webcam device and the webcam service are given if the IP address of the device is known. Therefore discovery can be finished after resolving the device (fig. 4). In a more static scenario discovery could even be omitted. Note that for the other two services no URL templates are specified for which reason at least device metadata has to be requested.

V. Template integration into toolkits, development flow and example

One of the requirements of device and service templates was the better integration of software tools to support the developer at development time. SOAP Web services already offer this benefit by WSDL. Many SOAP Web service toolkits offer code generators for WSDL. These code generators generate stub and skeleton code and often also an XML Schema data binding.

This feature is already utilized in the WS4D DPWS toolkits. But WSDL covers only service level metadata while DPWS has device and service level metadata at run time. Device and service templates alone won't solve this issue. As a device can implement several device or service templates another concept is required.

Therefore the concept of *device instances* is introduced. Device instances are needed to offer code generation for device and service description of DPWS devices at development time. The format of device instances should have at least the same elements as the service and device template format as well as toolkit specific extensions. Instead of template values device instances should contain instance values that are specific for one device or model.

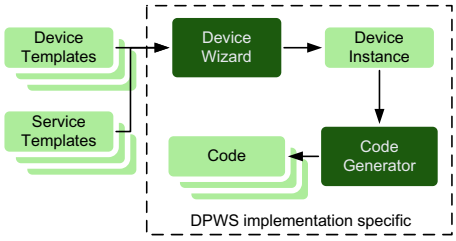

Fig. 5. Code generation with device and service templates

The design flow starts with the device and service templates that are implemented by a device, goes to the corresponding device instance and finally to the generated code as illustrated in figure 5. In this design flow two software tools are involved. The first tool that creates a device instance from several device and service templates may be a command line tool or a wizard that supports the developer. The second tool is the actual code generator that generates code based on a device instance for the device and the client side. On the device side the code covers functions to initialization of device and model description, functions to setup service instances and service types and a function to set up the device. The generated code on the client side offers functions to find devices of a specific type, to find services of a specific type on a device and to find a service with a specific ID on a device.

This design flow was implemented as prototype for the WS4D-gSOAP toolkit. This toolkit is an implementation of DPWS based on gSOAP [17] for the programming languages C and C++. To use the format of the device and service templates as basis for the device instance format seems to be a good way to proceed. The device wizard was implemented with XSLT. It simply merges all the templates into the device instance and tests if there are conflicts like several URI templates for the hosting service that don't match. If the templates can be merged, they are extended with toolkit specific information and written into the device instance file. Toolkit specific extensions are needed as the code generator may require further information to generate code. In the case of the WS4D-gSOAP the code generator needs name attributes for services as the service IDs from the service templates are URIs. So it is easier to specify additional identifiers that can be mapped to C or C++ language identifiers than mapping URIs to C or C++ identifiers. The code generator is also implemented with XSLT. It generates C code to support the device and service setup on the device side and functions to ease the discovery of devices and services on the client side.

VI. Conclusion and Future Work

In this paper a template format for DPWS devices and services similar to UPnP was introduced by a simple webcam device example. This template format clarifies the device and service discovery process in DPWS and makes the integration of code generation tools easier. The integration into the design flow of a DPWS device was shown with a prototype for the WS4D-gSOAP toolkit.

Concerning future work, the template mechanism will be fully integrated into the toolkits of WS4D. With the template mechanism WS4D can start to define a set for basic device and service templates that can be reused by developers to reduce the effort making devices or applications DPWS capable.

Acknowledgment

This work has been funded by German Federal Ministry of Education and Research under reference number 01—SF11H.

References

[1] *Jini Architecture Specification, Version 1.2*, Sun Microsystems, 2001.
[2] *UPnP Device Architecture v.1.0.1*, UPnP Forum, 2003.
[3] J. Teirikangas, *HAVi: Home Audio Video Interoperability*, Helsinki University of Technology, 2001.
[4] *Devices Profile for Web Services*, Microsoft, 2005.
[5] "Web Services Architecture," W3C Working Group, 2004, http://www.w3.org/TR/ws-arch/.
[6] "Service-oriented Architectures for Devices," SOA4D, http://www.soa4d.org.
[7] "Web Services for Devices Initiative," WS4D, http://www.ws4d.org.
[8] H. Bohn, A. Bobek, and F. Golatowski, "SIRENA - Service Infrastructure for Real-time Embedded Networked Devices: A service oriented framework for different domains," in *International Conference on Networking (ICN)*, 2006.
[9] "SIRENA: Service Infrastructure for Real-time Embedded Networked Applications," 2006, http://www.sirena-itea.org.
[10] "LOMS: Local Mobile Services," 2007, http://www.loms-itea.org.
[11] "Service Oriented Device Architectures," SODA, http://www.soda-itea.org.
[12] "Web Services Standards as of Q1 2007," innoQ, 2007, http://www.innoq.com/soa/ws-standards/poster/.
[13] "UPnP Device Control Protocol," UPnP Forum, http://www.upnp.org/standardizeddcps/default.asp.
[14] T. Bray, D. Hollander, A. Layman, and R. Tobin, *Namespaces in XML 1.0 (Second Edition)*, 2006, http://www.w3.org/TR/REC-xml-names/ns-qualnames.
[15] "XML Schema," W3C, http://www.w3.org/XML/Schema.
[16] J. Gregorio, *URI Template*, Network Working Group, Internet Draft, 2006, http://bitworking.org/projects/URI-Templates/draft-gregorio-uritemplate-00.html.
[17] R. A. van Engelen, *gSOAP*, 2007, http://www.cs.fsu.edu/%7Eengelen/soap.html.