Abstract—In recent years a movement from distributed systems controlled by users to automatic, autonomous and self-configuring distributed systems is noticeable. Web services is one approach but lacking the secure integration of resource-constrained devices.

This paper describes the Devices Profile for Web Services (DPWS), underlying protocols and a DPWS toolkit implementation based on C and gSOAP and discusses its current state. It has enormous relevance for embedded systems and industrial automation since DPWS targets resource-constraint devices explicitly, and has the potential to shift the industrial landscape which is characterized of heterogeneous devices.

Index Terms—Distributed systems, service oriented architecture, web services, Devices Profile for Web services, embedded systems

I. INTRODUCTION

In recent years a movement from distributed systems controlled by users to automatic, autonomous and self-configuring distributed systems is noticeable. Standardized self-describing interfaces (called services) and advanced separation of interfaces and implementation enhance the abstraction of component-based development and thereby paving the way for non-technical software engineers to develop complex, process-oriented software systems. Service oriented architecture (SOA) [1] is such an open concept supporting plug-and-play capabilities of heterogeneous software and hardware components. The SOA approach addresses issues such as service addressing, announcement, (self) description and discovering services as well as registering in and looking up a central service repository (service registry). The probably most popular implementation of SOA – Web services – is gaining increasing market penetration. The Web service architecture [2] provides a set of modular protocol building blocks that can be composed in varying ways (called profiles) to create protocol sets specific to particular applications. Profiles define which protocols are used, how they are adapted and in which way they are used to achieve a certain aim. Thus, profiles are meaningful instruments for achieving interoperability between software implementations of different vendors.

The Devices Profile for Web Services (DPWS) was developed to enable secure Web service capabilities on resource-constraint devices [3]. DPWS was mainly developed by Microsoft and some printer device manufacturers. DPWS allows sending secure messages to and from Web services, dynamically discovering a Web service, subscribing to, and receiving events from a Web service.

DPWS is not the first SOA that targets device-to-device communication. Technologies such as Open Service Gateway Initiative (OSGi), Home Audio/Video Interoperability (HAVi), Java Intelligent Network Infrastructure (JINI) and Universal Plug and Play (UPnP) are similar approaches.

The OSGi specification defines a service platform that relies on Java [4]. An OSGi service is a simple Java interface but the semantics of a service are not clearly specified. HAVi offers plug-and-play as well as Quality-of-service (QoS) capabilities and is restricted to the home domain [5]. JINI was developed by Sun Microsystems for spontaneous networking of services and resources based on the Java technology [6]. Services/devices carry the code (proxy) needed to use them. UPnP supports ad-hoc networking for devices and services and is easy to develop for [7]. It has a very similar functionality in comparison to DPWS but does not address security issues and is only applicable for small networks (no service registry/proxy).

The big advantage of DPWS compared to all other mentioned SOAs is the reliance on Web service which implies high acceptance among developers and platform as well as programming language independence. Microsoft will include DPWS in the next generation of operating systems.

This paper investigates the specification, implementation and use of DPWS for embedded systems and industrial automation. It was implemented and tested during the pan-European project SIRENA [see [8], [9]].

The paper is organized as follows. In the next section a short introduction to the underlying protocols used and adapted by DPWS are given. Section 3 deals with the DPWS specification. Adaptations and extension of the protocols used for DPWS are discussed. In section 4 our implementation of a toolkit for developing DPWS compliant services, devices and clients with the programming language C will be described. In section 5 our experience including the pitfalls of implementing DPWS will be presented. In section 6 the benefit of using DPWS for resource-constrained devices will be discussed. An overall conclusion and future work is summarized in section 7.

II. THE UNDERLYING PROTOCOLS OF DPWS

DPWS is partially based on the Web Services Archite-

---

ture (WSA) and uses further standards and draft specifications from the Web services protocol family.

**WS-Addressing**

The main objective of WS-Addressing is to provide an addressing mechanism for Web services as well as messages in a transport-neutral matter. By introducing both concepts endpoint references (EPR) and message information headers (MI) WS-Addressing overcomes the lack of SOAP’s independence of underlying protocols and secondly support of asynchronous message exchange. Both limitations are historically caused by the default SOAP to HTTP binding.

**WS-Discovery**

WS-Discovery is a discovery protocol based on IP multicast for enabling services to be discovered automatically. Discovery introduces three different endpoint types: target service, client and discovery proxy. Target services are Web services offering themselves to the network. Clients may search for target services and discover them dynamically. Discovery proxy is an endpoint enabling discovery in spanned networks since simple discovery is limited to a multicast group and hence to local managed networks only.

WS-Discovery defines four operations or messages to discover target services in a network. To explicitly discover target services in a network a client can use the **Probe** operation, send as multicast message. Matching target services will answer with the **Probe Matches** operation send as UDP unicast message to the client. To implicitly discover target services a client can listen for **Hello** and **Bye** messages. A target service announces its availability with these messages send as UDP multicast.

To resolve logical addresses introduced with the endpoint structure in WS-Addressing a client can use the **Resolve** operation send as UDP multicast message. The corresponding target service responds with the **Resolve Matches** operation send as UDP unicast to the client.

The discovery proxy does not need any additional operations.

**WS-MetadataExchange / WS-Transfer**

WS-MetadataExchange is a specification that defines data types and operations to retrieve metadata associated with an endpoint. This metadata describes what other endpoints need to know to interact with the described endpoint. WS-MetadataExchange defines the MetadataSection that divides the metadata into separate units of metadata with a dialect specifying its type.

Until the latest version of DPWS only WS-MetadataExchange was used for service and device description and retrieval. In the latest DPWS version of February 2006 WS-Transfer is used to retrieve the metadata. The structure of the metadata is still as specified in WS-MetadataExchange. The main difference is that WS-MetadataExchange defined operations to retrieve all or parts of the metadata of an endpoint, whereas WS-Transfer only can be used to retrieve all metadata of an endpoint.

We expect that WS-Transfer and WS-MetadataExchange will be merged closer in future releases.

**WS-Eventing**

WS-Eventing defines a protocol for managing subscriptions for a Web services based eventing mechanism. This protocol defines three endpoints: subscriber, event source and subscription manager. Subscribers request subscriptions on behalf of event sinks to receive events from event sources. Subscription requests contain an event delivery mode and event filter mechanism to negotiate event delivery mechanisms and event filter mechanism. Subscription managers are responsible of holding subscriptions of event sources.

**III. CONSTRAINTS, LIMITATIONS AND EXTENSIONS SPECIFIED BY DPWS**

In this section the DPWS specific constraints, limitations and extensions for above mentioned Web Service specifications are described. They are divided into five parts covering the topics messaging, discovery, description, eventing and security. The specification uses the notation defined in RFC2219 and defines that for compliance only the “MUST” or “REQUIRED” level must be satisfied.

**Messaging**

The messaging layer of DPWS relies on the SOAP 1.2, SOAP-over-UDP, HTTP/1.1, WS-Addressing, RFC 4122 (UUID) and MTOM specifications. Theses specifications are restricted to limit the functionality needed for resource-constrained device implementations and to meet the packet length limits of the underlying IP network stack. A service on a device must support the HTTP chunked transfer-coding. In general messages exceeding this limit can be sent but then a service on a device may fail to process or reject this message. This may result in incompatibility issues.

For basic interoperability a service on a device must at least support receiving and sending SOAP 1.2 Envelopes over HTTP. It must at least support the Request-Response message exchange pattern and respond to one-way message exchange patterns.

A device must support WS-Addressing by including a relationship field in the message information header of each response or fault message. It must also generate a failure on request messages received over HTTP not including an

A device must at least support the rfc2396 and strcmp0 and Resolve Match messages if types are included. With DPWS and should be included in Hello, Probe Match and Resolve Match messages if types are included.

To simplify a resource-constrained device implementation a device must at least support the rfc2396 and strcmp0 scope matching rules.

**Description**

To describe a devices or services metadata, DPWS relies on XML Schema, WSDL, Basic Profile Version 1.1, WS-MetadataExchange, WS-Policy, WS-PolicyAttachment and WS-Transfer. The DPWS specification further divides the description metadata into characteristics, hosting, WSDL and policy description. All description of a (hosted) service or device (hosting service) can be retrieved with a WSDL and policy description. All description of a (hosted) service or device (hosting service) can be retrieved with a WS-Transfer Get operation.

For highly-constrained circumstances the DPWS specification assumes that a client knows all it needs to use hosted services on a device, so no retrieval of description is needed. For more generic usage, a client must retrieve a device description from its hosting service, which address is announced by discovery. This is the only reliable way for service discovery as the hosting description contains a device’s Relationship MetadataSection. In the Relationship data structure defined by DPWS each hosted service with its address, id and supported types as defined in WS-Discovery.

To further inspect a device a client may retrieve a device characteristics description that is contained in a This-Model MetadataSection and a ThisDevices MetadataSection. Both MetadataSections contain data structures defined by DPWS.

The other description metadata is not only specific to the hosting service but can also be retrieved from the hosted services. The WSDL description contains the WSDL- Documents of a service and the policy description its attached policy.

**Eventing**

The devices profile includes an event mechanism based on WS-Eventing. A service providing events must therefore have the attribute EventSource=true in its WSDL and be able to process subscribe request messages.

To limit the functionality needed for resource-constrained device implementation DPWS defines one event delivery mode and one event filter mechanism that has to be supported at least. The event delivery mode called push delivery mode is already defined in WS-Eventing. This delivery mode must be supported at least for conformance to DPWS. The profile further defines a message filter called action filter letting the subscriber specify which events should be received.

As synchronized clocks will not meet highly-constrained resource requirement, an event source only must support subscriptions with duration type xs:duration but not xs:dateTime.

**IV. IMPLEMENTATION OF DPWS**

In this section the implementations of a toolkit for developing devices and clients almost conforming to the devices profile for Web services will be introduced. This toolkit is not yet fully compliant, as the security section of the profile is not yet followed and several other parts as message attachments are not supported. This toolkit is based on another toolkit to develop Web services in the programming language C/C++ called gSOAP. The toolkit uses gSOAP’s plug-in mechanism and implements several specifications like WS-Addressing, WS-Discovery, Description (WS-Transfer/WS-MetadataExchange) and WS-Eventing as plug-in independent of DPWS. In the following subsection gSOAP, and the implementation of the devices profile will be described.

**DPWS**

The DPWS Toolkit implements the constraints, limitations and extensions defined in DPWS based on gSOAP and plug-ins implementing WS-Adressing, WS-Discovery, WS-MetadataExchange/WS-Transfer and WS-Eventing. gSOAP [10] is a toolkit for building SOAP-based Web services in C/C++.

It supports the three roles that can be switched at compile time: device, client and peer. With the device role an endpoint implements the device side of the specification. To implement a client the client role must be enabled at compile time. In case of a peer endpoint, that means support for device and client role, the peer role must be used.

The toolkit does not yet implement the security require-
V. PITFALLS OF IMPLEMENTING DPWS

When implementing DPWS the first pitfall is to figure out the basic functionality needed for compliance. As DPWS is designed for resource-constraint devices as well as for functionality-richer implementations the specification contains a lot of optional requirements and only a few mandatory requirements, respectively. Since most of these requirements are related to devices, this leads to a well-defined device specification but a vague client specification. Therefore, device implementation is straightforward and client implementation leaves a lot of open questions.

When implementing DPWS based on a Web services toolkit the choice which toolkit to use should be made carefully. A toolkit should support at least transmitting SOAP-over-UDP messages in form of unicast as well as multicast and support one-way, request-response and solicit-response message exchange patterns. Further the underlying IP stack must support IP multicast.

Before using a service on a device a client has to be aware of the service’s description and endpoint. Assuming a high dynamical environment a client needs a maximum of three steps for accessing this description: (1) device discovery by device type and scope, (2) device description transfer, and (3) service description transfer. In case the client knows all it needs in advance to communicate with a service (static case) these steps can be ignored. In all other cases only the needed steps have to be done. So discovery is only used for device discovery. To determine the capabilities and the endpoint of a requested service the description must be acquired with WS-Transfer. For example a service providing events has to include wse:EventSource=true as attribute in its portType definition in its WSDL.

The approach for service discovery that is at the bottom of DPWS can be best seen in the WSD-API from Microsoft [11]. The WSD-API is the programming interface to the client implementation integrated into next versions of Microsoft’s operating system. In this implementation all device and service description metadata is cached and discovery is used to keep this metadata up to date. The actual service discovery (called function discovery) is based on this cache. This approach shows that DPWS assumes resource-constrained devices and resource rich clients.

To classify devices and services, DPWS uses a type system. Until now the semantic of this type system is not defined. It is important to not mistake these types with WSDL portTypes! DPWS only defines the device type dpws:Device that indicates compliance to the profile and support for retrieving descriptions.

Device discovery as defined in DPWS will cause interoperability problems, which may lead to clients that will not discover all requested devices. Length restrictions for message fields or whole messages defined in the messaging section may lead to interoperability issues. The client side could consider the restrictions when sending messages and the device side could reject messages that exceed the restrictions.

There are also technical pitfalls when implementing discovery for DPWS. WS-Discovery uses a special message exchange pattern that is comparable to the request response message exchange pattern. The request is sent by the client by UDP multicast and the response is sent by the device by UDP unicast. The destination of the request is defined by WS-Discovery. The destination of the response is corresponding to the source address of the request. This means a discovery client, sending a request over UDP multicast to a device must listen on the same UDP port, the request was send for receiving the unicast response.

Another technical issue for a device is to obtain its IP address. A device can announce its real address with discovery or must be able to resolve it’s logical to its real address. Implementations relying on DNS to obtain its IP address will only work in a network where DNS is setup correctly.

VI. CONCLUSION AND FUTURE WORK

In this paper we presented an overview of the Devices Profile for Web Services, its underlying Web services technology and its pitfalls when implementing it. Furthermore we briefly introduced our gSOAP-based toolkit.

As we have shown in this paper the devices profile will need some slight changes to become a standard which allows implementing interoperable Web services on resource-constrained devices as well as clients. At the moment a devices description is only available at run time. A schema and/or template system for defining device types would improve the support of code generators towards easy service creation and allow a finer grained device discovery. We will further investigate how device templates can improve DPWS and complete the toolkit. For implementing Web services on devices with even lower resource constraints and for solving problems with too big message sizes XML-binary technologies like Fast Web Services will be investigated in future research.

REFERENCES

[9] F. Jammes and H. Smit, “Service-Oriented Architectures for Devices the SIRENA View,” in In proc. 3rd IEEE Int. Conf. on Industrial Informatics (INDIN), 2005