Devices Profile for Web Services and the REST

Guido Moritz[1], Elmar Zeeb[1], Steffen Prüter[1], Frank Golatowski[2], Dirk Timmermann[1], Regina Stoll[2]

[1]Institute of Applied Microelectronics and Computer Engineering, University of Rostock
[2]Institute of Preventive Medicine, University of Rostock
[1,2]18055 Rostock, Germany
{guido.moritz, elmar.zeeb, steffen.prueter, dirk.timmermann, regina.stoll}@uni-rostock.de

[2]Center for Life Science and Automation, 18119 Rostock, Germany
frank.golatowski@celisca.de

Abstract- **For future application scenarios of resource constrained and low cost smart cooperating objects, miscellaneous technologies for wireless connectivity are existing or upcoming in the near future. But further cross domain middleware and dedicated communication protocols to provide syntactic and semantic interoperability and not only technical interoperability are still missing. Thus, this paper investigates on two major candidates for IP based low power communication, based on known and matured technologies and protocols. Therefore, analyses and pitfalls of RESTful architectures based on HTTP and the Devices Profile for Web Services are presented. Furthermore, the paper discusses differences of DPWS and RESTful design and furthermore proposes an approach for a generic mapping of emerging Bluetooth Low Energy technology with RESTful device architectures for seamless and transparent connectivity.**

I. Introduction

Decades of research and devolvement in the domains of wireless sensor networks (WSN) and ad-hoc device communication have brought miscellaneous application scenarios like habitat monitoring [5], industrial motor monitoring [6], structural monitoring [7], bridge monitoring [8], volcano monitoring [9] and forest fire prediction. Newest developments extend scenarios to home healthcare applications.

Former networking embedded device infrastructure is extended to low power, low cost and highly constraint smart cooperating objects. These developments raise urgent need for platform independent interoperability between devices and also with higher valued services for example in the internet. Specific domains have developed middleware dedicated and used widely only in single domains. While UPnP, DLNA and related technologies are established in networked home and small office environments, the Devices Profile for Web Services (DPWS) is widely used in the automation industry at device level [10] and it has been shown that they are also applicable for Enterprise integration [11], [12]. To overcome problems of interoperability between technologies and protocols, generic gateway architectures and device abstraction layers are developed [13], [14]. This allows integration of incompatible technologies like ZigBee and Bluetooth in complex application scenarios and data access via IP based interfaces. With the rise of IP based communication directly in WSN [4], new device and communication architectures are possible without need for generic gateways and data caching intermediates.

Thus, main scope of this paper is examination of two major candidates to be applied for IP based wireless smart cooperating objects: Service-oriented Architectures (SOA) and Resource-oriented Architectures (ROA) like the Representational State Transfer (REST) style. Because Web services (WS) are widely used as realization of SOAs, this paper provides in detailed information about DPWS section III, which are not published along with the DPWS specifications. This information requires deep knowledge about DPWS that we got by implementing DPWS[1] and during standardization process within OASIS WS-DD[2]. Furthermore, the paper reveals necessary functionalities in RESTful device architectures in section IV, which require further research efforts. Section V discusses differences of DPWS and RESTful HTTP designs. To not exclude non IP based technologies, this paper proposes in section VI a new approach for direct and generic mapping of emerging Bluetooth Low Energy (BTLE) protocols into HTTP protocol, widely used in RESTful applications, without the need of caching intermediates.

II. Related Work

Beside several proprietary solutions or solutions by huge industry consortia, IP based technologies and infrastructures for device communication have been developed. While the Wi-Fi Alliance announced a new wireless networking specification Wi-Fi Direct, to allow direct peer-to-peer communication between devices without the need for management devices, IEEE 802.11 is too expensive in terms of energy consumption for future smart cooperating objects. Other solutions focus on low power, low data rates and low cost solutions to meet the resource and price constraints.

A. ZigBee

Low power, low cost, and low data rate wireless communication are the main scope of the IEEE 802.15 WPAN Task Group 4, which has brought forth the IEEE 802.15.4 specifications. Newest amendments will include TDMA and channel hopping to improve robustness.

[1] http://www.ws4d.org (2010)
[2] http://www.oasis-open.org/committees/ws-dd/ (2010)

Based on 802.15.4 on link layer, the ZigBee alliance has developed further network and application layer protocols. This combines power saving capabilities of 802.15.4 with required network and application protocols, while energy saving power states of radio is transparent to upper layers and are under control of link layer. Furthermore, the ZigBee alliance has developed the ZigBee application profiles, which are composed of optional and mandatory ZigBee cluster libraries.

B. Bluetooth Low Energy

In addition to the existing Bluetooth specifications, the emerging Bluetooth Low Energy (BTLE) technology was developed. Low energy link layer are defined, working under the existing L2CAP (logical link control and adaptation protocol) layer. This allows dual mode architectures, consisting of parallel running classic Bluetooth and BTLE stacks in one circuit.

BTLE revised drawbacks of classic Bluetooth like piconet architecture and thus limited subnet size. Additionally, a broadcast mode is described, which leads to new application scenarios because of the absence of required direct pairing. In contrast to classic Bluetooth application protocols and profiles, BTLE is capable of lightweight attribute protocol and attribute profiles. Payload for attributes is limited to maximum of 27 octets and represents sensor and actor states of, e.g., sensed temperature, time, heart rate, etc. For attributes and for configuration and management purposes, five methods are announced to be supported by BTLE attribute clients and attribute servers: PUSH, PULL, SET, BROADCAST, and GET. While the PULL method is used by clients to retrieve attributes from a server, PUSH is used by a server to avoid bandwidth and power consuming polling. The attributes to be pushed are configured by the SET method, designed for these operations. Furthermore, the SET method can be used to change attribute states on the server, or more general of actors. The BROADCAST method is used by servers to send data to every listening device without need of further pairing or configuration like for the PUSH method. The GET method provides functionalities for finding all or specific attributes of a device and thus provides basic discovery features.

For different application scenarios, specific attribute profiles are already specified for BTLE. Therefore, attributes also include metadata information like human readable descriptions of the attributes.

C. IP for smart cooperating objects

Both ZigBee and Bluetooth Low Energy are chosen by Continua Health Alliance[3] to provide wireless connectivity. Nevertheless, neither ZigBee nor Bluetooth Low Energy is able to communicate directly with higher valued services in other IP based networks without intermediate devices. They require application layer gateways to map payload data in IP based network protocols. Changes in payload require high efforts for maintenance of protocol converters. This limits application scenarios significantly. Other existing and emerging technologies and architectures are developed and extended to be applied in networking device infrastructures and on future smart cooperating objects.

[3] http://www.continuaalliance.org (2010)

Based on uIP [3], proving feasibility of TCP/IP implementation for 8-bit microcontrollers, and in accordance to the IPv6 specification, the Internet Engineering Task Force (IETF) has established the 6LoWPAN working group [1]. The focus of 6LoWPAN is to compress IPv6 headers to be sent on top of 802.15-based technologies, especially 802.15.4 [2]. 6LoWPAN establishes the basis for TCP and UDP data transmissions for smart cooperating objects. The main advantage of 6LoWPAN is the compliance to a regular computer network protocol (in this case IPv6). On top of 6LoWPAN protocols, further application layer protocols, architectures, and concepts can be applied. This extends applicability of existing technologies for a new class of devices.

Based on the 6LoWPAN specifications, in [24] an approach of ZigBee application profiles on top of UDP instead of the ZigBee transport and networking layers is described. While this approach has a considerable advantage concerning interoperability on lower layers, for seamless connectivity with higher valued services still gateway or proxy concepts are required to map the ZigBee application profiles in existing technologies and protocols.

III. DPWS

Service-oriented Architectures are often used to improve flexibility and reusability of components in complex distributed applications. This is achieved by modeling functional blocks as independent services. DPWS can be used to realize a SOA that fits into device centric applications and thus enables the application of SOA in the area of networked devices. The Devices Profile for Web Services (DPWS) was developed to enable secure Web service (WS) capabilities on resource-constraint devices. DPWS is a base technology for device communication that can be easily composed with and extended by other specifications and technologies. DPWS has an architectural concept that is similar but different to the Web Service Architecture (WSA) to fit better into device scenarios. The main difference is the multicast service discovery with WS-Discovery that does not require any central service registry such as UDDI. But the service usage of services on devices is similar to the service usage in WSA, whereby DPWS devices can be directly integrated into WSA based enterprise systems.

A. Common Misunderstandings of DPWS

DPWS specifications have a high learning curve and readers often come to different conclusions than the authors of the specifications. This leads often to a wrong depiction of DPWS.

Extensibility. One major problem of the specification is the extensible nature that leads to a specification where it is hard to figure out the required baseline functionality. So there are often misunderstandings in what is mandatory and what not. Web services are mostly defining only mechanisms. How to use these mechanisms in a deployment is up to the application designer. Hence DPWS is a base technology and provides only basic features. These features can be extended by application specific protocols on a higher level, tailored for each application scenario. In this application specific extension protocols, the protocol designer decides which and how the features of DPWS fit into requirements of the scenario. These decisions are essential for properties such as performance, reliability, scalability, and extendibility of the resulting application. So DPWS leaves some space for these designers to best fit DPWS in a specific scenario. This basic principle is not clearly stated in the specifications and leads to misunderstandings.

Discovery. The dynamic discovery feature of DPWS is based on WS-Discovery and SOAP-over-UDP specifications. These parts of the specification contain the most obvious shortcomings of DPWS. The worst case scenario of the device and service discovery has a lot of message round trips and thus a huge latency. The following mechanisms here presented in a chronological order as they would take place in the worst case scenario belong to the discovery feature:

- Probe (explicit search for devices) / Hello (implicit device announcement)
- Resolve (resolve network independent device address)
- DNS query (resolve network independent device name)
- RealtionsShip Metadata Exchange (retrieve the meta data about available services)
- WSDL Metadata Exchange (retrieve the service description meta data)

But the discovery depends heavily on the application scenario and how the application level protocol designers apply the available mechanisms. As most of these mechanisms are based on two way message exchange, this worst case scenario can take several seconds. But this worst case is not required in all scenarios. The protocol designer should consider that a client can access device metadata by other means and still conforms to DPWS. A client can use data about the device that is available in advance (at development time) or can use mechanisms like caching of metadata. Depending on the scenario, these message exchanges can and should be kept at a minimum amount.

Binding. DPWS requires an SOAP 1.2 over HTTP binding at least, though messages must be transported by HTTP POST as described in the binding and the DPWS specifications. But this implies not that a service cannot offer more bindings and be used with other transport mechanisms. A "Web Service Description Language"-Document (WSDL) can contain several bindings. For one possible candidate, the SOAP-over-UDP binding, the binding specification is still missing. This might be fixed in the next version of SOAP-over-UDP.

Eventing. DPWS offers a mechanism to subscribe for state changes in a device. These state changes are delivered as events to a client. DPWS defines the concepts of delivery mode and eventing filter. The first one defines how an event is delivered to a client and the latter one defines which events are delivered to the client. It is up to the protocol designer to define new event filters or delivery types. The PUSH delivery mode defined in DPWS uses for event delivery separated TCP connections to each subscriber. This requires high resources in scenarios with many subscribers for one or many events, but may be sufficient for simple scenarios. But advanced delivery modes can be defined that make for example use of the HTTP keep alive feature or avoid TCP connections at all and use a UDP and IP multicast based transport protocol.

Encoding. Like Web services in general, DPWS uses XML in its human readable UTF-8 encoding. There are advantages like platform independence to use XML. But these advantages imply much overhead. The common misunderstanding concerning encoding is the obligation to encode everything in UTF-8 XML. If there is no need for the flexibility given by XML, binary encodings for data can be used. DPWS offers several mechanisms to use binary data. On the one hand, there is the attachment mechanism that enables the attachment of arbitrary data. This attachments use the SOAP Message Transmission Optimization Mechanism (MTOM) that leaves further space for optimizations. On the other hand, it is possible to use an alternative encoding of the XML data. SOAP is based on the XML Infoset specification, which defines an abstract data set that can be used to represent the information in well-formed XML documents. This specification enables the encoding of information of an XML document in other representations than UTF-8. It is used by the Efficient XML Interchange Working Group to define a standard named Efficient XML Interchange (EXI) to encode XML in a very compact binary format. This technology can be combined with DPWS as well [22]. At the time of writing this paper, EXI is in the stage of a specification candidate recommendation at the W3C and there are not many implementations, especially for embedded systems. But this technology will open new research areas and may also minimize the overhead of XML data.

Composability. In general DPWS defines a minimal set of features for baseline interoperability to enable secure Web service messaging on devices. It does not cover all application scenarios. But DPWS is part of the Web service framework and is designed in an extendable way to be combined with existing WS specifications, even if there is still research needed in how to combine certain WS specifications with DPWS. For deployment this is a question of tool support. Besides extending DPWS it is also possible to further restrict the specification as long it is interoperable with DPWS [23].

Web services are always Service-oriented. W3C Web services are often mentioned in the same breath with SOAs. But SOAs do not depend on these SOAP Web services and can also be implemented by using other specifications and standards. Furthermore W3C SOAP Web services are not restricted to model a service-oriented application style. The Web Services Resource Access Working Group (WSRA)[4] at the W3C is heading towards specifications for resource oriented Web services, represented in XML and manipulated by SOAP based mechanisms. DPWS for example uses the WS-Transfer Get mechanisms for metadata exchange of the device. Hence the metadata is modeled as a resource and requested with a lightweight well defined method. A complete application design using DPWS might be modeled in a simple CRUD (Create, Read, Update, Delete) style also but is not restricted to.

B. Discussion

After this survey of common misunderstandings of DPWS that lead to wrong assumptions concerning the performance and application of DPWS, a short discussion is about the disadvantages and advantages of DPWS is necessary.

The most obvious disadvantage of DPWS is the overhead because of data representation in XML format and especially the usage of XML namespaces. This compromise is based on the principle “flexibility over optimization” that is common in SOAs and is of course arguable in device centric applications. Additionally, there is the quite high learning curve to understand DPWS and its capabilities. The style WS specifications are written does not enhance the learning curve, because they reference a lot of other WS specifications and are not very verbose. Furthermore, there are yet missing white papers and additional information beside the specifications. This situation will hopefully get better when there are more white papers by the OASIS WS-DD technical committee and DPWS toolkits are more matured and include sufficient documentation.

The major advantages of DPWS are flexibility, abstraction through loose coupling, standards approved by well known organizations and available tool support. The flexibility is provided by the use of XML and all the WS specifications that are optimized for flexibility. This results in a DPWS specification that is extendable in many variations to meet requirements of most imaginable application scenarios. The loose coupling is a feature that is important to applications that integrate devices. It decouples applications from the devices itself and makes them only dependent on the abstract service interfaces. This extends the product life of such applications as integrated devices can be exchanged without adaptations of the application.

The DPWS specification is approved as OASIS standard in Version 1.1. since July 2009 and aligned to the other WS standards at OASIS and W3C. During the standardization process, the interoperability of several DPWS implementations was tested. Thus, several proven open source implementations from the WS4D-Initiative, from SOA4D and implementations from Microsoft part of Windows Vista, Windows 7 and the .NET Micro Framework are available.

[4] http://www.w3.org/2008/11/ws-ra-charter.html (2010)

```
<?xml version="1.0" encoding="utf-8"?>
<application xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xmlns:uws="http://example.de/ping"
   xsi:schemaLocation=
      "http://wadl.dev.java.net/2009/02 wadl.xsd"
   xmlns:tns="urn:example"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://wadl.dev.java.net/2009/02">
   <resources base="http://uservices.de/">
      <resource path="ping">
         <method name="GET" id="ping">
            <request>
               <param name="request"
                  type="xsd:string" style="query"
                  required="true" />
            </request>
            <response>
               <representation
                  mediaType="application/xml"
                  element="uws:PingRespondse" />
            </response>
         </method>
      </resource>
   </resources>
</application>
```

Figure 1. WADL ping example message

IV. REST

With DPWS, SOAs based on Web services can be developed and deployed. However, it still has a big overhead due to many expensive bidirectional message exchanges and data representation in XML. Another approach are Resource-oriented Architectures (ROA). One proper approach for a ROA is the Representational State Transfer (REST), a software architectural style based on the work of Roy Fielding [15]. REST is using similar architectures than the World Wide Web (WWW). Each data is handled as a resource and each resource is an atomic data unit. Only a strong limited number of methods for the manipulation of a resource are specified, and each method works in a fixed functionality. In contrast to DPWS, the methods of REST are restricted to a principle similar to the CRUD (Create, Read, Update, Delete) style. The two most common methods are GET for the request of a resource and POST for the manipulation of a resource or requests to data processing resources requiring input data. Furthermore, REST must be based on a stateless protocol. Each resource can be handled and manipulated over different states and these states must be handled by the service client and not by the server. REST methods GET, PUT, and DELETE should also be implemented idempotent, so that the same method applied on the same resource one or many times have the same result. Therefore, the implementation of a RESTful service can be more simple and lightweight than in other architectures.

A. Service Description Language

Service description languages are always required to ensure interoperable interfaces and to provide possibility of tools support for interface development and implementation. REST services can be developed as easy machine readable service interaction with the Web Application Description Language

(WADL) [16]. WADL is the upcoming interface description language for RESTful architectures. The resource based handling of methods allows a simple and easy scalable service description.

WADL is based on XML and describes applications based on HTTP. It supports the automatic description of RESTful Web services with machine process-able service descriptions. WADL is supported by the java.net community and is currently a submission to the W3C. It can be assumed that WADL will be a W3C standard soon. Figure 1 depicts a possible service that checks the availability of a server. As common for RESTful architectures, all objects are handled as resources. In the first step, a resource is created with the base to the uService infrastructure. The location of a special resource ping is declared and the available method, with request and responds parameters, is shown.

B. *Semantic Web Services*

By deploying RESTful architectures, adding of semantics to service and resource descriptions are a main challenge. Here SA-REST as an open, flexible, and standards-based approach for adding semantics to RESTful services can be used. With SA-REST, most advancements of SAWSDL [16] for semantics in service oriented architectures are used. For example a service element can be linked to an ontological model by the preparation of syntactic descriptions to semantic metamodels using annotations. Usually, REST services can be described in a HTML page, but unlike a WSDL description this can be really human readable pages. Therefore, any website can be used as service and also any HTTP client can be used to access a REST service. Furthermore, so-called microformats can be used in these descriptions to allow an easy machine to machine interaction with semantic services.

C. *Missing REST functionality*

To apply RESTful architectures also for device communication, further research is required. This subsection discusses most important ones.

Asynchronous messaging. REST is often based on HTTP, which uses synchronous request-response transmissions. The client initiates a connection by opening a socket to the server and holds this socket open until the responds from the server is complete. In device architectures, the data processing and response generation may not happen immediately. This would require long lived connections. Especially in dynamic and mobile device scenarios, asynchronous short duration transmissions are required to solve this problem. DPWS makes use of WS-Addressing to overcome this problem. WS-Addressing includes message IDs in every request to assign responses to the correlated request.

Addressing. For devices in mobile and dynamic scenarios, changing transport addresses (IP) might occur. Usage of WS-Addressing in DPWS assigns unique identification for every device, independent of the transport specific address. RESTful implementations might push this issue to DNS to abstract a human readable URI from IP based transport address. But this requires high efforts for DNS server synchronization or usage of Multicast-DNS (mDNS) to avoid centralized design.

Eventing. Without extension, REST cannot provide eventing functionalities. Here an extension e.g. to the standard HTTP protocol is necessary that allows push message exchange towards the client. REST relies on classic server/client communication, whereby servers are passive and inactive until a client request occurs. Push messaging would require change of these roles and active listening of the origin client. If the client uses NAT or other specific network architectures, further efforts like port forwarding or usage of an intermediate proxy are required. Some interesting ideas can be found but are no standard defined yet. Webhooks [17] are one solution and are used for example by Google Code [18] or PayPal [19]. This approach is using HTTP POST as an indirect message to the client to inform about an event. Furthermore, a current W3C working draft [20] defines a server push or client pull mechanism. But for a comprehensive eventing concept, additional efforts are required to include lease concepts, data distribution design, and filtering mechanisms. DPWS includes WS-Eventing and allows for own extensions to solve these problems.

Service modeling. REST models data and states of devices as resources. SOAs instead are using interacting services to model functional blocks. Thus, for simple data access, DPWS deployments provide services and methods similar to GET methods of HTTP to request data (e.g. via WS-Transfer). Hence by restriction concerning the supported methods of a DPWS device, CRUD like messaging pattern is possible also. Because REST is based on HTTP with well defined methods, this is not required. But the limitations of methods/verbs in RESTful designs make modeling of interactive services difficult. Simple functionalities like commands for an actor (e.g. opening a door) must be modeled as changes on a resource and require rethinking of application designers.

D. *Common Misunderstandings of REST*

Web services and *Web services*. The W3C defines Web services as “programmatic interfaces”. The W3C has brought a complete protocol framework, often referred to as WS-* protocols or in general as SOAP Web services. RESTful web services are proposed as to be more lightweight than W3C WS-* specifications. However, for a clear separation, protocol and application designers should be aware of the difference (i.e. architecture and used protocols).

REST and RPC. The simple usage of ROA frameworks does not save for the creation of RPC-Style interfaces. During the creation of interfaces, developers must follow the rules of ROA. Other well-known applications use some of these advantages and are quite successful. One important is SQL with 4 major methods (SELECT, INSERT, UPDATE, and DELETE) and all objects handled as resources.

REST is/uses HTTP. ROA is an architecture, while REST is an architectural style going back to work of Roy Fielding. REST as concept is completely protocol agnostic. Indeed, Roy Fielding (principle author of the HTTP specification) proposes HTTP to build RESTful designs. But ROA and REST must not use HTTP. HTTP is capable of establishing a resource-oriented architecture. The success of HTTP in ROAs can be ascribed to its common use in many communication frameworks and low learning curve. DPWS for example also realizes device metadata exchange in a RESTful style.

REST is a standard. REST is not a standard yet, but most technologies for communication and data representation that can be used to create a RESTful architecture are standardized. The first steps to create a complete REST-* framework are done by Red Hat, which has founded the REST-* Community [21]. The goal is to create combined REST-* standards like the WS-* approach for SOA based Web services.

V. DPWS AND REST

In section III the pitfall that SOAP based Web services are always modeled in a service-oriented design style is discussed and rejected. The term *service* must be used carefully in this context as a *service* in general describes a collection of functionalities. But the term *service* is used sometimes different in SOAP Web services and RESTful services in contrast are based on resource. Thus RESTful Web services can describe services based on different protocols but are based on the same architectural style. DPWS can be deployed also in a RESTful application design, whereby SOAP Web services are used to access and manipulate the resources. Additionally a DPWS device can host services which can be accessed by miscellaneous different interfaces.

Often RESTful design is wrongly mixed up with HTTP as discussed above. Because DPWS and RESTful HTTP deployments base on partly the same specifications, a mapping might be possible as discussed in the remainder of this section.

Protocol design. For a comprehensive generic mapping the protocol designs must be analyzed more in detail. RESTful deployments may use HTTP as application layer protocol. The resource representation, addressing and the method to be applied on the resource are encoded in the HTTP header. The further required data for processing input or output are encoded in the HTTP body while the format is not restricted. SOAP based Web services and thus also DPWS are using the HTTP binding as transport mechanism. Hence SOAP based Web services additionally include information for the target service or the client in the SOAP header. Because of this the HTTP header must not be used as extensive as in REST.

Addressing. DPWS uses WS-Addressing that allows identification of devices independent of transport specific addresses (IP), carried in the SOAP header. The transport addresses are used by DPWS to identify the endpoint to send the service invocations to. In REST the address is used to identify the resource to be manipulated. For a mapping each service in DPWS must be restricted to support only a restricted set of methods and a service must be split up in different endpoints, each with a unique address. Based on the addressing discussion, further research is required how to map the hosting/hosted service concept of DPWS in RESTful resources. Services of a DPWS device are mostly independent and often without a common basis, whereby resources in REST are often related to each other.

Payload. A core principle of REST is the payload agnostic design. In a HTTP based RESTful design, the HTTP body is neither restricted to an encoding nor to specific syntax formats. Widely used are XML and JSON based representation formats. But SOAP Web services are based on XML Infoset to provide lowest possible interoperability concerning data representation. For a generic mapping the RESTful deployment must be aware of the XML Infoset syntax. If not the payload must be attached or included by DPWS messages e.g. by using attachment mechanisms like MTOM.

Summarized, a generic DPWS/REST mapping is not necessary, because DPWS can be used to model a RESTful application. A generic mapping of HTTP based RESTful design and DPWS is possible by considerable restrictions of the DPWS protocol design concept.

VI. BLUETOOTH LOW ENERGY / RESTFUL HTTP MAPPING

For complex application scenarios, domain specific and dedicated technologies are required to widen scenarios. Thus, this section proposes a seamless HTTP/BTLE mapping.

As described in section II, BTLE features usage of a lightweight attribute protocol (ATT) and attribute profiles. ATT supports several methods to request and set attributes (states) of attribute servers. Because REST also uses states for modeling information about devices and services (i.e. resources), there are similarities between both. Furthermore, the methods used by the ATT are similar to the methods defined by HTTP also. A direct mapping without need of further caching and protocol transformations between BTLE and IP based solutions allow seamless integration of BTLE technology in networking infrastructures. Especially in ad-hoc networks without management devices, the mapping will lead to new application scenarios.

Attributes are identified by attribute handles, which specifies how to address the attribute. There are no information available how handles are represented and encoded, but a mapping into URIs for resource identification like used in HTTP based RESTful design may be possible. Therefore, the gateway for connecting IP based networks and the BTLE devices have a URI to allow REST based data access. The attribute handles and further BTLE device metadata are used to generate generic URIs for external access (cf. figure 2). The type of the attribute value, the semantic meaning, is defined by a 16 bit UUID and is defined in the profiles. The value of the attribute can be represented by using specific data types like integer, float, string, etc. This information can be embedded in the HTTP message body using standard data representation formats like JSON and XML.

Table 1. HTTP / BTLE ATT method mapping

HTTP 1.1 methods	BTLE ATT methods
OPTIONS - request communication options, i.e. which methods are supported on this resource	GET – services/attributes of device and/or attribute types and handles
GET - request resource representation	PULL
HEAD - same like GET, but server returns no message body	-
POST - send data to data-handling process and/or annotate existing resource	SET
PUT - change resource state	SET – write operations on attributes require specific procedure to get write permissions
DELETE - delete resource	SET
TRACE - HTTP "ping"	-
CONNECT - establish connection via proxy	link layer operation
-	PUSH
-	BROADCAST

Table 1 proposes a new approach for mapping of methods defined by the HTPP 1.1 specification and BTLE ATT methods. The usage of the simple set of verbs in BTLE ATT is very similar to the usage of the few verbs used in REST, in contrast to many semantically not clear defined verbs in Web services technologies. The ATT GET method is used for basic discovery functionalities. The discovery process bears analogies to DPWS discovery with separation of device and service discovery. In the first step, devices and their provided services are discovered. In the next step, the specific attributes of a service are requested. After devices, services and attribute handles are known, the attributes can be requested by using the ATT PULL method. The response is the attribute value representing the state, which is a resource in REST. Operations on the attributes (i.e. changes of resources and delete of resource) can be mapped in ATT SET operations. If deleting or changing of a specific attributes is possible depends on permissions and security configurations of the attribute. In REST this may also be restricted and allowed methods, which can be requested by using HTTP OPTIONS. For the ATT PUSH method of an attribute server, no direct counterpart exists in HTTP. For PUSH indications, REST server and client roles are swapped, which requires active listening of the origin client. But new concepts are emerging for RESTful architectures like Webhooks and W3C EventSource. DPWS includes WS-Eventing to provide these server initiated message exchange functionalities. ATT PUSH indications also include mechanisms for reliability (acknowledgments) and flow control of indications to avoid flooding. In HTTP, acknowledgements are also used as HTTP response headers and additionally on network (TCP) layer. In DPWS this functionality is provided by 2-way message exchange pattern, which prescribes response of the service provider in contrast to 1-way message exchange pattern. Flow control in REST and DPWS are task of the network layer also. The ATT BROADCAST method has no direct mapping into the HTTP protocol and REST style. HTTP is strictly based on server/client connections. Thus, HTTP relies and point-to-point communication and due to reliability issues on TCP. Thus, DPWS specifies for discovery purposes an own SOAP-over-UDP binding without the need for HTTP. This allows usage of IP multicast instead of unicast messaging. Hence, REST is not capable of such functionalities.

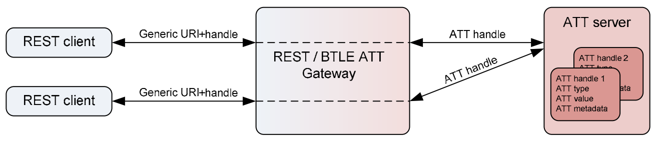

Figure 2. BTLE / REST device architecture

Unfortunately, only few data is available about ATT and no comprehensive definition of a direct mapping between ATT and REST can be defined in this paper.

VII. Conclusion

For future IP based wireless communication of smart cooperating objects, RESTful resource-oriented architectures based on HTTP and SOAs implemented by DPWS are proper candidates. Both provide basic functionalities to meet requirements not only of single application scenarios but to be applied as platform independent cross domain technologies. Nevertheless, the high degree of extensibility and flexibility coupled with missing documentation and partly high learning curve leads to pitfalls for both of them. These pitfalls include eventing, discovery, and encoding concepts as well as extensible and composable nature and are presented in detail in this paper. The underlying architectures are most remarkable difference between both, but while based on partly same protocols and technologies, discrepancies are pre-programmed. For example DPWS must not be used to model a SOA, but can also be used to realize a RESTful application. Hence SOAs and RESTful style are not a contradiction. The restrictions concerning methods and about stateless design of the server in RESTful deployments may lead to more lightweight implementations, independent of the used protocols. But prove of this thesis is still outstanding, especially because of e.g. missing eventing and discovery concepts and mechanisms of used protocols for RESTful applications. Especially eventing and discovery may require sufficient more resources and implementation efforts for the servers also in RESTful deployments.

Often the diversity of atomic operations/methods/verbs in a SOAP Web services based application is described as drawback. But the diversity of the methods or more in general of services and actions must be mapped to resource representations in a ROA to provide similar functionalities on the one hand and still offer flexibility and extensibility on the other hand. Hence RESTful applications often use complex URI addressing schemes to overcome this issue.

Not to exclude non IP based technologies of future scenarios, the emerging Bluetooth Low Energy Attribute Protocol can be mapped into HTTP protocol by using such a generic URI mapping, as presented in this paper also.

ACKNOWLEDGMENT

This work has been achieved in the European ITEA2 projects uSERVICE and OSAmI and has been funded by the German Federal Ministry of Education and Research under contract numbers 01|S0902F and 01|S08003I.

REFERENCES

[1] IETF, *IPv6 over Low power WPAN (6lowpan)*, Technical report, http://tools.ietf.org/wg/6lowpan/, 2008.

[2] IETF Network Working Group, *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*, RFC 4944, http://tools.ietf.org/html/rfc4944, 2008.

[3] Adam Dunkels, “Full TCP/IP for 8-Bit Architectures,” *International Conference On Mobile Systems, Applications And Services (MobiSys 2003)*, San Francisco, California, pp. 85-98, 2003.

[4] Hui, J. W. and Culler, D. E., “IP is dead, long live IP for wireless sensor networks,” *6th ACM Conference on Embedded Network Sensor Systems (SenSys 08)*, New York, NY, pp. 15-28, 2008.

[5] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler, “An Analysis of a Large Scale Habitat Monitoring Application,” *2nd international Conference on Embedded Networked Sensor (SenSys 2004),* New York, NY, pp. 214-226, 2004.

[6] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis, “Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea,” *3rd international Conference on Embedded Networked Sensor Systems (SenSys 2005)*, New York, NY, pp. 64-75, 2005.

[7] Ning Xu, Sumit Rangwala, Krishna Kant Chintalapudi, Deepak Ganesan, Alan Broad, Ramesh Govindan, and Deborah Estrin, “A Wireless Sensor Network For Structural Monitoring,” *2nd international Conference on Embedded Networked Sensor Systems (SenSys 2004),* New York, NY, pp. 13-24, 2004.

[8] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fenves, Steven Glaser, and Martin Turon, “Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks,” *6th international Conference on information Processing in Sensor Networks (IPSN 2007*), New York, NY, pp. 254-263, 2007.

[9] Geoff Werner-Allen, Konrad Lorincz, Jeff Johnson, Jonathan Lees, and Matt Welsh, “Fidelity and Yield in a Volcano Monitoring Sensor Network,” *7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7 Operating Systems Design and Implementation.* Berkeley, CA, pp. 27-27, 2006.

[10] H. Bohn, A. Bobek, and F. Golatowski, “SIRENA - Service Infrastructure for Realtime Embedded Networked Devices: A service oriented framework for different domains,” *International Conference on Systems and International Conference on Mobile Communications and Learning Technologies (ICNICONSMCL'06)*, Washington, DC, USA, page 43, 2006.

[11] Stamatis Karnouskos, Oliver Baecker, Luciana Moreira Sá de Souza, Patrik Spieß, “Integration of SOA-ready networked embedded devices in enterprise systems via a cross-layered web service infrastructure,” *IEEE 12th International Conference on Emerging Technologies and Factory Automation (ETFA2007),* Patras, Greece, pp. 293-300, 2007.

[12] Zeeb, E., Prüter, S., Golatowski, F., and Berger, F., “A context aware service-oriented maintenance system for the B2B sector,” *3rd International IEEE Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE 2008)*, Ginowan, Okinawa, Japan, pp. 1381-1386, 2008.

[13] Zeeb, E.; Behnke, R.; Hess, C.; Timmermann, D.; Golatowski, F.; Thurow, K., "Generic sensor network gateway architecture for plug and play data management in smart laboratory environments," *IEEE 14th International Conference on Emerging Technologies and Factory Automation (ETFA 2009)*, pp.1-8, La Palma, Spain, 2009.

[14] Lipprandt, M.; Eichelberg, M.; Thronicke, W.; Kruger, J.; Druke, I.; Willemsen, D.; Busch, C.; Fiehe, C.; Zeeb, E.; Hein, A., "OSAMI-D: An open service platform for healthcare monitoring applications," *2nd Conference on Human System Interactions (HSI 2009),* pp. 139-145, Catania, Italy, 2009.

[15] Fielding, Roy T., *“Architectural Styles and the Design of Network-based Software Architectures,”* University of California, Irvine, Department of Informatics, DISSERTATION, 2000.

[16] Web Application Description Language, java.net project, https://wadl.dev.java.net/, 31.08.2009,

[17] Webhooks, last accessed 23.01.2010,webhooks.org/

[18] PostCommitWebHooks, Google Community, last accessed 23.01.2010, http://code.google.com/p/support/wiki/PostCommitWebHooks

[19] Instant Payment Notification, PayPal, last accessed 23.01.2010, https://www.paypal.com/ipn

[20] HTML5, W3C, last accessed 23.10.2010, http://dev.w3.org/html5/spec/spec.html

[21] REST-*, Red Hat and REST-* Community, last accessed 23.10.2010, http://jboss.org/reststar/

[22] Guido Moritz, Dirk Timmermann, Regina Stoll, Frank Golatowski, “Encoding and Compression for Devices Profile for Web Services,” 5th International Workshop on Service Oriented Architectures in Converging Networked Environments (SOCNE2010), Perth, Australia, 2010.

[23] Guido Moritz, Elmar Zeeb, Steffen Prüter, Frank Golatowski, Dirk Timmermann, Regina Stoll, “Devices Profile for Web Services in Wireless Sensor Networks: Adaptations and Enhancements,” 14th International IEEE Conference on Emerging Technologies and Factory Automation (ETFA2009), La Palma, Mallorca, 2009.

[24] IETF Network Working Group, A UDP/IP Adaptation of the ZigBee Application Protocol, Internet-Draft, http://tools.ietf.org/html/draft-tolle-cap-00, 2008.