Entwicklung standardisierter Geräte mittels Geräte- und Dienstvorlagen für das Devices Profile for Web Services

Andreas Bobek[*], Elmar Zeeb[*], Frank Golatowski[*], Dirk Timmermann[*]

[*] Universität Rostock, Fachbereich Elektrotechnik und Informationstechnik, Institut f. Angewandte Mikroelektronik und Datentechnik Richard-Wagner Str. 31, 18119 Rostock-Warnemuende, {andreas.bobek, elmar.zeeb, frank.golatowski, dirk.timmermann}@uni-rostock.de

1. Einleitung

Mit der stetigen Entwicklung eingebetteter Systeme auf der einen und der steigenden Verfügbarkeit von immer schneller werdenden (u.a. auch mobilen) Internetzugängen auf der anderen Seite kann der Bedarf an leistungsfähigen, vernetzten Geräten in den unterschiedlichsten Domänen (Haus, Industrie, Auto) zumindest aus Hardware-Sicht zunehmend realisiert werden.

Aus Software-Sicht bedarf es darüber hinaus jedoch intelligenter Lösungen, die die Entdeckung, Steuerung und Überwachung solcher Geräte und Gerätegemeinschaften ermöglichen. Hier haben sich in der Vergangenheit Technologien wie Java Intelligent Network Infrastructure (JINI), Universal Plug and Play (UPnP) und Home Audio/Video Interoperability (HAVi) teilweise bewährt, wobei bis heute keiner dieser Standards Domänen-übergreifend eingesetzt wird bzw. eingesetzt werden kann. Das Devices Profile for Web Services (DPWS) ist ein relativ junger Standard, welcher auf Web Services basiert, Geräte als Dienste im Sinne Dienst-orientierter Architekturen abbildet und einen viel versprechenden Ansatz darstellt, die genannten Ziele zu lösen, sowie aufgrund der Nähe zum Web Consortium (W3C) eine breite Zustimmung erhofft.

Für DPWS fehlt bisher ein Mechanismus, um Geräte zu standardisieren. Im vorliegenden Artikel beschreiben wir eine Lösung, die sich nahtlos in die verwendeten Technologien einfügt und zeigen die Vorteile, die sich bei ihrer Anwendung ergeben.

2. Dienst-orientierte Architekturen, Web Services und DPWS

Dienst-orientierte Architekturen (engl.: SOA) (Erl 2005) sind verteilte Softwarearchitekturen, deren Komponenten Dienste und Dienstnutzer sind und deren Interaktionen auf Nachrichten basieren (Abb. 1a). Optional können sich

Dienste bei einem Verzeichnisdienst registrieren, welches von Dienstnutzern durchsucht werden kann, wodurch Dienst und Dienstnutzer entkoppelt sind und erst zur Laufzeit dynamisch gebunden werden (engl.: „late binding“).

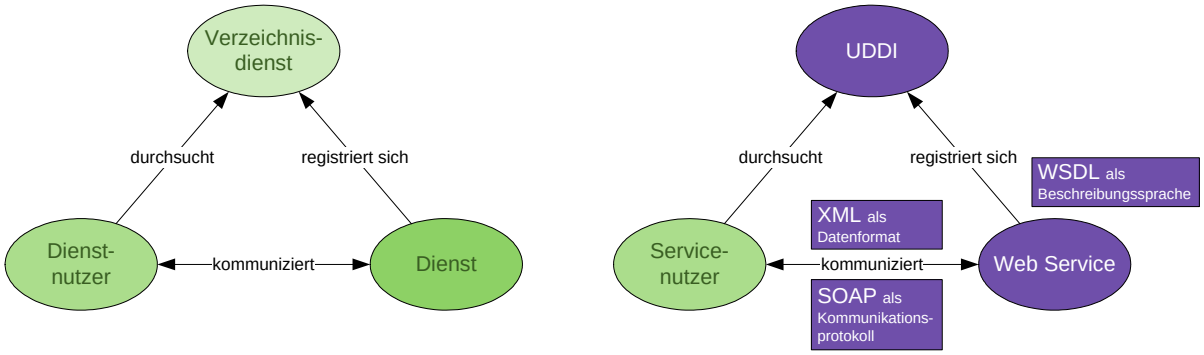

Abb. 1: (a) Komponenten der SOA Architektur und (b) Umsetzung mit dem Web Services Framework

Das Web Services Framework mit seinen Basisprotokollen SOAP, WSDL und UDDI setzt eine SOA konkret um (Abb. 1b). Die Kommunikation zwischen Web Services und Dienstnutzern basiert auf SOAP, einer XML Grammatik (Gudgin 2007). Durch die Verwendung von SOAP können Web Services Plattform-übergreifend, Transport- und Programmiersprachen-unabhängig genutzt werden. Mit der Web Service Definition Language (WSDL), ebenfalls einer XML Grammatik, werden Web Services beschrieben (Christensen 2001). UDDI ist ein zentraler Verzeichnisdienst, bei welchem ein Web Service seine Beschreibung (WSDL) zusammen mit einigen anderen Attributen registrieren kann. Dienstnutzer können UDDI kontaktieren und nach Web Services suchen. Die Möglichkeit des Einsatzes in heterogenen Umgebungen, die freie Verfügbarkeit der Spezifikationen und der gemeinsame offene Spezifikationsprozess, der es ermöglicht die Anforderungen aller Partner zu berücksichtigen, sind die größten Vorteile des Web Services Frameworks.

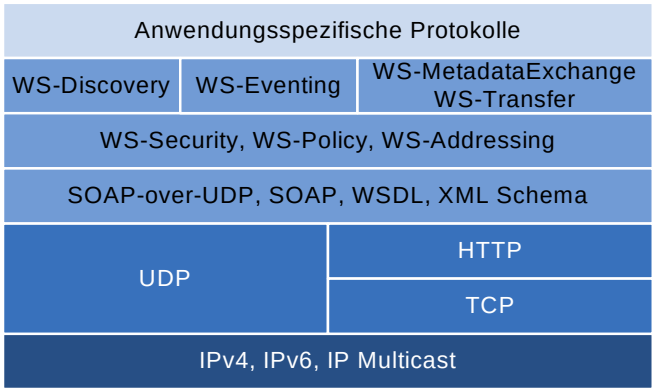

Abb. 2: Der Device Profile for Web Services (DPWS) Stack

Das aktuelle Web Services Framework umfasst neben den genannten viele weitere Protokolle, welche als Bausteine für profilierte Spezifikationen benutzt werden können. Das Devices Profile for Web Services (DPWS) ist ein solches

Profil, welches für kleine eingebettete und ressourcenarme Geräte geeignet ist (Shannon 2006). Dazu verwendet DPWS (teilweise mit Einschränkungen) eine Untermenge der Web Services Protokollfamilie, u.a. WS-Discovery (als Ersatz für eine zentrale UDDI Komponente), WS-Eventing, WS-Transfer, WS-Security, SOAP-over-UDP und WS-Policy (Abb. 2). DPWS befähigt damit Geräte sich dynamisch zu entdecken, sich zu beschreiben, Nachrichten auszutauschen, sich für Ereignisse zu registrieren, sie zu versenden und zu empfangen. Weiterhin definiert DPWS einige Sicherheitsaspekte, die eingehalten werden müssen.
In DPWS sind nur die Geräte Target Services, es werden also nur Geräte nicht jedoch dessen Dienste gefunden. Ein vollständiger Discoveryprozess bis hin zum Dienst umfasst daher mehrere Schritte: (1) Suchen nach Geräten mittels Probe Nachrichten und optionale Angabe von Typen, die das Gerät unterstützen muss. Ein potenzielles Gerät antwortet mit einer ProbeMatch Nachricht. (2) Auflösen der logischen Geräteadresse in eine physikalische Adresse mit einer Resolve Nachricht. Geräte (Target Services) arbeiten mit logischen Adressen (URIs), die für die Geräte konstant bleiben müssen, auch dann wenn das Gerät in einem anderen Netzwerk eingebunden wird. Das passende Gerät antwortet mit einer ResolveMatch Nachricht. (3) Abfrage der Metadaten des Gerätes. In diesen befinden sich Verweise (URL) auf die Dienste, die das Gerät anbietet. (4) Abfrage der Metadaten der Dienste. Diese enthalten die Dienstbeschreibungen in Form von WSDL und die Policy-Daten.

5. Geräte- und Dienstvorlagen

Wie beschrieben verläuft unter DPWS ein vollständiger Discoveryprozess über mehrere Stufen ab. Ein Hauptziel der Gerätevorlagen soll deshalb die Abkürzung dieses Prozesses zur Laufzeit sein, um den Netzwerkverkehr zu senken. Zur Entwicklungszeit dagegen sollen die Gerätevorlagen sowohl auf Dienstnutzer- als auch auf Geräteseite für die Codegenerierung verwendet werden. Um eine Weiterentwicklung zu gewährleisten sollen die Vorlagen durch Anwender erweiterbar sein und zudem Versionierung unterstützen.
WS-Discovery bietet die Möglichkeit innerhalb des `wsd:Types` Elementes nach Diensttypen in Form von qualifizierten Namen (Bray 2006) zu suchen (`Probe` Nachrichten). Wofür ein Diensttyp jedoch steht wird völlig offen gelassen. Er steht vor allem nicht, wie man zunächst annehmen könnte, für den WSDL PortType, den der Dienst implementiert. Es ist lediglich das Verhalten geregelt, wann ein Dienst eine Anfrage erfüllt. Diese fehlende Semantik können wir für unser Vorlagensystem zunutze machen. Ebenso wie ein WSDL PortType auf eine Menge von abstrakten Operationen verweist, schaffen wir eine Abbildung von einem Diensttyp auf PortTypes und andere Parameter.
Als Datenformat benutzen wir XML (das Vorlagensystem ist als XML Schema definiert), da es sich nahtlos in die Web Services Umgebung einpasst, und da die verwendeten Werkzeuge zur Codegenerierung und zur Laufzeit ohnehin mit XML (WSDL, SOAP, DPWS Metadaten) arbeiten können müssen.

Wir verwenden einen Teil der Elemente aus dem DPWS Schema für uns wieder, indem wir uns deren lokale Namen, nicht jedoch den Namensraum ausborgen. So wird beispielsweise aus dem qualifizierten Namen im DPWS Schema {`http://schemas.xmlsoap.org/ws/2006/02/devprof`, `Relationship`} der qualifizierte Name im Vorlagen Schema {`http://www.ws4d.org/templates` , `Relationship`}. Die Semantik der Elemente bleibt jedoch erhalten. Im Folgenden wird der ws4d-Namensraum durch das Präfix `ws4d` substituiert. Die Elementtypen sind mit dem XML Schema Konstrukten `any` und `anyAttribute` erweiterbar gehalten, so dass durch Anwender zusätzliche Werkzeug-spezifische Elemente nachgetragen werden können.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:Hosted 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:URLTemplate>http://{ip}:{port}/services/power</t:URLTemplate>
      <t:Service>
            <t:Reference>
                  http://www.ws4d.org/templates/power/Power2.0.wsdl
            </t:Reference>
      </t:Service>
      <t:ServiceId>http://www.ws4d.org/power/powerservice2</t:ServiceId>
</t:Hosted>
```

Abb. 3: Beispiel für eine Dienstvorlage

Vorlagen für Dienste und Vorlagen für Geräte sind voneinander getrennt; Gerätevorlagen referenzieren Dienstvorlagen. Damit ist es möglich einmal spezifizierte Dienste auch in anderen Gerätevorlagen wieder zu verwenden.
Abbildung 3 zeigt exemplarisch eine Dienstvorlage für einen „Power“ Dienst, welcher Operationen zum Ein- und Ausschalten eines Gerätes sowie das Setzen in und das Zurückholen aus einem Schlafmodus anbietet. Die Operationen befinden sich in der referenzierten WSDL, hier jedoch nicht dargestellt. Die Dienstvorlage bildet den Diensttyp `ws4d:Power2.0` auf die referenzierte WSDL, die angegebene Service ID und auf das URL Template ab. Dienstvorlagen werden durch das `Hosted` Element eingeleitet.
Das optionale URL Template (Gregorio 2006) spezifiziert die Netzwerkadresse, an welcher der Dienst zu finden sein muss. Dazu wurden zwei Platzhalter definiert: `{ip}` steht für die IP Adresse oder einen Domainnamen und `{port}` für den TCP Port. Die Platzhalter werden beim Geräte-Discovery (Schritt 2) durch die IP Adresse und Port des Gerätes ersetzt. Der Port kann wahlweise auch schon in der Vorlage fest angegeben werden. Durch Angabe eines URL Templa-

tes kann der Discoveryprozess wesentlich verkürzt werden. Unter Umständen kann er sogar entfallen, wenn beispielsweise die IP beim Starten eines Dienstnutzers bekannt ist.
Innerhalb des `Service` Elementes wird entweder die WSDL referenziert, die die Dienstbeschreibung enthält oder sie wird direkt mittels `wsdl:definitions` eingebettet. Das optionale `PortTypes` Element fungiert als Filter und enthält eine Komma separierte Liste der PortTypes, die in der Deklaration mit einbezogen werden sollen. Als Voreinstellung kann dieses Element weggelassen werden, wodurch sich die Dienstvorlage auf alle PortTypes in der WSDL bezieht.

```
<?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:HostedRef>
            http://www.ws4d.org/templates/power/PowerService2.0.xml
      </t:HostedRef>
      <t:HostedRef>
            http://www.ws4d.org/templates/config/BasicDeviceConfig2.1.xml
      </t:HostedRef>
      <t:HostedRef>
            http://www.ws4d.org/templates/webcam/Webcam1.0.xml
      </t:HostedRef>
</t:Relationship>
```

Abb. 4: Beispiel für eine Gerätevorlage

Die Versionierung wird mit `Includes` ermöglicht. Dieses Element enthält alle Diensttypen, die durch die Vorlage erweitert werden (Prinzip der Mehrfachvererbung). Im vorliegenden Fall repräsentiert die Vorlage eine Erweiterung des Diensttyps `ws4d:Power1.0`. Dadurch kann `Power2.0` auch überall dort eingesetzt werden, wo ein Dienst vom Typ `Power1.0` verlangt wird. Es können beliebig viele `Includes` angegeben werden. Voraussetzung für dessen Einsatz ist jedoch, dass der neue Dienst vollständig abwärtskompatibel zu den referenzierten Diensten sein muss. Durch den Einsatz von `Includes` kann die Nachrichtengröße beim Suchen nach und Bekanntgeben von Diensten verringert werden. Im Beispiel reicht eine Suche nach `Power1.0` aus, um den Dienst zu finden, die zusätzliche Angabe von `Power2.0` ist nicht notwendig.
Ein Gerät („Hosting Service“) kann mehrere Dienste („Hosted Services“) anbieten. Abbildung 4 zeigt exemplarisch eine entsprechende Gerätevorlage für eine Webcam. Die Gerätevorlage wird durch das `Relationship` Element eingeleitet, das Gerät durch das `Host` Element, welches ähnlich aufgebaut ist wie `Hosted` innerhalb einer Dienstvorlage. Die dargestellte Gerätevorlage bildet

den Gerätetyp `ws4d:Webcam` auf das URL Template und die drei durch URIs referenzierten Dienstvorlagen ab.
Die Menge der Typen, die ein Gerät implementiert, ergibt sich vollständig aus den Vorlagen und umfasst den Host-Typ, die Hosted-Typen sowie die optionalen Host- und Hosted-Include-Typen. Für eine Suche nach Typen ist es aber irrelevant, um welche Art Typ es sich handelt, maßgebend ist allein die Tatsache, dass das Gerät den Typ implementiert. Daher kann ein Gerät alle diese Typen beim Discovery gleich behandeln, was zu einer Verringerung des Codeumfangs führt. Da es sich hierbei um statische Daten handelt, kann ein Codegenerator zur Entwicklungszeit den kompletten Discovery-Code erzeugen.

6. Ausblick

Die beschriebene Lösung wird innerhalb der Web Services for Devices (WS4D) Initiative als ein erster Entwurf behandelt und ist derzeit Gegenstand der unterschiedlichen Stack-Implementierungen. WS4D ist aus dem EU-Projekt Service Infrastructure for Real-time Embedded Networked Applications (SIRENA) hervorgegangen und wird derzeit im Projekt Local Mobile Services (LOMS) weitergeführt. WS4D vereint fünf Stacks für unterschiedliche Anwendungsdomänen und Plattformen. Ebenso wie der laufende Interoperabilitätsprozess, wird auch das Vorlagensystem zwischen den Partnern regelmäßig getestet, praxisnah evaluiert sowie gegebenenfalls angepasst und weiterentwickelt.

Diese Arbeit wurde über das LOMS Projekt vom Bundesministerium für Bildung und Forschung (BMBF) unter der Referenznummer 01|SF11H gefördert.

7. Literatur

Bray T., Hollander D., Layman A., Tobin R. (2006) Namespaces in XML 1.0 (Second Edition). http://www.w3.org/TR/REC-xml-names/#ns-qualnames

Christensen E., Curbera F., Meredith G., Weerawarana S. (2001) Web Services Description Language (WSDL) 1.1. http://www.w3.org/TR/wsdl

Erl, T. (2005) Service-Oriented Architecture: Concepts, Technology, and Design. Prentice Hall PTR

Gregorio, J.C. (2006) URI Template. (Network Working Group, Internet Draft). http://bitworking.org/projects/URI-Templates/draft-gregorio-uritemplate-00.html

Gudgin M., Hadley M., Mendelsohn N., Moreau J. et al (2007) Simple Object Access Protocol (SOAP) 1.2. http://www.w3.org/TR/soap/

Shannon C., Conti D., Kaler C., Kuehnel T. (2006) Devices Profile for Web Services. http://schemas.xmlsoap.org/ws/2006/02/devprof/

WS4D Initiative. http://www.ws4d.org