Web-Konzepte für das Internet der Dinge: Ein Überblick

Samuel Wieland

ETH Zürich, Mai 2008

Zusammenfassung. Kleine eingebettete Systeme und Sensoren drängen in den Alltag. Mit Kommunikationsschnittstellen und Rechenkapazität ausgestattet sind diese „Dinge“ internet- und webtauglich. Anhand diverser Konzepte - REST, SOAP, OWL und URI um ein paar Wenige zu nennen - soll gezeigt werden, wie solche smarten Objekte ins Web eingebunden werden können und welche Hürden noch zu meistern sind.

1 Einleitung

Als Tim Berners-Lee 1989 am CERN [1] den Grundstein für das moderne Web legte, dachte er wohl kaum an ein Netz aus Millionen von Teilnehmern. Durch den steten Fortschritt in der Fertigung von immer kleiner werdenden Hardware-Bausteinen können heute beinahe alle Gebrauchsgegenstände aus unserem alltäglichen Leben mit Kleinst-Computern ausgestattet werden und mit dem Internet verbunden werden. Mit der Massenverbreitung werden diese eingebetteten Systeme und Sensoren zusätzlich günstiger. Beinahe jedes neue Mobiltelefon ist heute internettauglich und mit verschiedensten Sensoren ausgestattet (seien es Kameras, Temperatur-Sensoren usw.).

Die enormen Datenmengen, die durch tausende von Endgeräten und letzten Endes auch durch die Nutzer anfallen, müssen effizient verfügbar gemacht werden. Die Daten sollten überall auf der Welt, unabhängig von der Methode des Zugriffs, abrufbar sein. Im Gegensatz zum Internet der Dinge, wo Fragen der Kommunikation auf physikalischer Ebene geklärt werden, geht es im Web der Dinge vor allem darum, wie Dinge Informationen effizient austauschen können. Der traditionelle Ansatz, dass im Web ein Nutzer mit seinem Webbrowser von Webseite zu Webseite navigiert, weicht schleichend einem neuen Nutzungsbedürfnis: Intelligente oder smarte Dinge nutzen das Web um Informationen auszutauschen und sich gegenseitig zu koordinieren, um so dem Nutzer einen Mehrwert zu generieren. Es müssen also neue Methoden entwickelt werden, wie dies sicher, effizient, langfristig und vor allem herstellerübergreifend umgesetzt werden kann.

Diese Arbeit behandelt verschiedene Methoden zum Datenaustausch resp. der Datenverarbeitung im Web: Zustandslose Kommunikation am Prinzip von Representational State Transfer (REST) [1], [2], [3], SOAP [4], Stream Feeds [7] als

[1] Conseil Européen pour la Recherche Nucléaire

Beispiel für streambasierte Kommunikation mit Sensordaten und Smart object awareness and adaptation model (SOAM) [8] für agentenbasierte Kommunikation.
Die verschiedenen Konzepte werden jeweils detailliert vorgestellt und anhand praktischer Beispiele erläutert. Zum Schluss folgt eine allgemeine Diskussion.

2 SOAP

SOAP ist ein weit verbreiteter Standard zur Kommunikation zwischen Webservices. Es wird vereinfacht gezeigt, wo Schwächen von SOAP liegen und wo für eingebettete Systeme ein passenderes Konzept nötig wäre.
SOAP legt fest, wie Applikationen per Remote Procedure Call (RPC) Informationen austauschen können, sei dies über das Internet oder über ein lokales Netz (z.B. Ethernet). Abb. 1 visualisiert exemplarisch, wie zwei Applikationen per SOAP kommunizieren können. Sobald mit SOAP über das Internet kommuni-

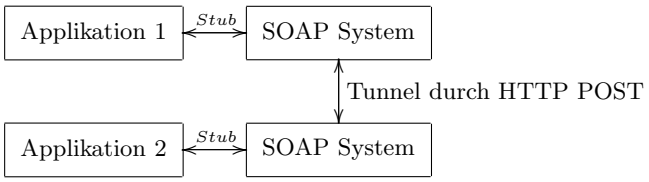

Abb. 1. Applikation 1 setzt einen RPC-Aufruf durch den SOAP-Stub ab. Der Stub übersetzt den RPC-Aufruf in ein XML-Format und sendet dieses XML über HTTP POST an den SOAP-Stub von Applikation 2. Der Stub entpackt den RPC-Aufruf aus dem XML und setzt lokal den Prozeduraufruf ab. Das Resultat des Prozeduraufrufs wird analog in XML verpackt und an Applikation 1 zurück gesendet.

ziert werden soll, wird in den meisten Fällen das HTTP-Protokoll als Transport-Protokoll verwendet. Dazu wird der Prozeduraufruf in XML serialisiert und in eine HTTP-POST-Anfrage verpackt. Im HTTP-Standard [6] sind neben POST zusätzlich OPTIONS, GET, HEAD, PUT, DELETE, TRACE und CONNECT spezifiziert. SOAP ignoriert also bereits auf Transportebene einen Grossteil der Flexibilität, welche das HTTP-Protokoll grundsätzlich bereitstellt, was wiederum impliziert, dass auf einer höheren Programm-Logik mehr Arbeit geleistet werden muss. Die Serialisierung resp. Deserialisierung einer Anfrage in XML benötigt Rechenleistung und Ressourcen, die unter Umständen auf eingebetteten Systemen nicht oder nur begrenzt zur Verfügung stehen.
Ein weiteres Problem für kleine eingebettete Systeme im Kontext des "Internets der Dinge" stellt sich in der Spezifikation der Schnittstellen und somit der Stubs. Ein SOAP Service wird durch die Webservices Description Language (WSDL) [5] beschrieben. Wenn also ein Service per SOAP bereitgestellt werden soll, so muss zuerst ein WSDL Interface erstellt werden, welches dann durch ein Interface

Compiler (z.B Apache Axis [2]) in ein SOAP Stub übersetzt wird. Ändert sich das Interface des Services, so muss das WSDL Interface angepasst, die Stubs neu generiert und die Anbindung erweitert werden. Vorallem im Kontext des Internets der Dinge ist aber gerade diese Anpassung oft schwierig oder nicht praktikabel. Auch wenn für SOAP bereits dedizierte Hardware existiert, welche die Serialisierung/Deserialisierung ressourcenschonend implementiert, stellt sich trotzdem die grundlegende Frage, ob nicht mit einem vereinfachten System unter voller Nutzung des HTTP-Protokolls ohne komplizierte Infrastruktur eine vergleichbare, für eingebettete Systeme und Sensoren besser geeignete, Architektur gebaut werden könnte.

3 Representational State Transfer

Motiviert am Beispiel des Webs spezifiziert REST ein Regelwerk, welches helfen soll, verteilte Systeme zu bauen. Im Gegensatz zu SOAP sollen REST Applikationen ohne dedizierte Infrastruktur (WSDL, Stubs, etc.) auskommen und bestehende Systeme/Konzepte wiederverwenden.
Das Regelwerk umfasst im wesentlichen sechs Prinzipien:

- Jede Ressource wird über eine eindeutige URI identifiziert.
- Alle Ressourcen sind durch eine uniforme Schnittstelle ansprechbar:
  - GET (Zustand einer Ressource abfragen. Nur Lesen).
  - POST (Verändern einer Ressource).
  - PUT (Neue Ressource erzeugen).
  - DELETE (Eine Ressource löschen).
- Hyperlinks um den Applikations-Zustand zu beschreiben.
- Sämtliche Meldungen sollen „selbstbeschreibend“ sein (Metadaten).
- Die Kommunikation zwischen zwei Kommunikations-Partnern erfolgt zustandslos.
- Der Austausch von Ressourcen erfolgt über Repräsentationen von Ressourcen.

Eine Applikation, die dem Regelwerk entspricht, wird oft als RESTful Applikation bezeichnet.
In den folgenden Teilkapiteln wird exemplarisch auf zwei Architekturen eingegangen, wo REST erfolgreich eingesetzt wird. Zuerst wird jedoch kurz auf URI als Grundbaustein von REST eingegangen.

3.1 Uniform Resource Identifier

Damit Objekte und/oder Daten über das Internet für das Web verfügbar gemacht werden können, muss eine standardisierte Methode der Adressierung gefunden werden. Bereits 1994 stellte Berners-Lee einen ersten Entwurf einer solchen Adressierung vor [9], ein allgemeiner Standard wurde jedoch erst 2005 von

[2] http://ws.apache.org/axis/

der Internet Engineering Task Force (IETF) [3] verabschiedet.

Uniform Resource Identifier (URI) ist eine eindeutige Identifizierung einer physischen oder logischen Einheit. Dies kann z.B. ein Textdokument auf dem Web, ein Sensor-Knoten einer Messstation oder die Adresse eines Scripts auf einem Server darstellen. Eine URI besteht im Wesentlichen aus einem Schema, einer "Authority" (Zuständigkeit/Behörde) und einem Pfad. Abb. 2 zeigt den Aufbau einer URI. Das Schema beschreibt, wie die URI kodiert resp. dekodiert wer-

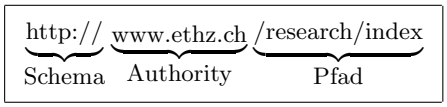

Abb. 2. Beispiel-Aufbau einer URI

den soll. Dies ist insbesondere wichtig, da verschiedene Schemen benutzt werden können (z.B. HTTPS für sichere End-zu-End-Kommunikation). Die "Authority" legt fest, wo die Ressource abgelegt ist (z.B ein Server im Internet). Über den Pfad wird die gewünschte Ressource ausgewählt.

3.2 Das Web

Die Begriffe Web und Internet werden fälschlicherweise häufig als Synonyme betrachtet, obwohl sie zwei grundlegend unterschiedliche Konzepte beschreiben. Das Internet ist ein Netz von Computern, die über das IP-Protokoll verbunden sind. Das Internet "beschäftigt" sich also um Fragen, wie ein Datenpaket von Computer A zu Computer B transportiert werden kann. Das Web hingegen ist eine Menge von Ressourcen, die über Hyperlinks verbunden sind, wobei eine Ressource in den meisten Fällen nicht durch ein Computer repräsentiert ist, sondern durch z.B. eine Datei.
Das Web kann als grösste RESTful Applikation und als das grösste verteilte System überhaupt verstanden werden. Anhand einzelner Entwurfskriterien des Webs wird gezeigt, warum REST als Architekturkonzept für grosse Systeme verstanden werden kann.
Wie von REST gefordert, wird im Web keine spezielle API zum Austausch von Informationen vorgegeben. Wenn ein Klient eine Ressource anfordert, so werden sämtliche für den Server nötigen Parameter in die URI kodiert. Auf diese Weise wird dem Server bei jedem Aufruf der vollständige Zustand der geforderten Ressource mitgeteilt und die Kommunikation erfolgt als Gesamtes zustandslos. Abb. 3 zeigt eine REST-Anfrage auf eine Ressource.
Eine für das Web charakteristische Eigenschaft ist das Austauschen von Ressourcenrepräsentationen. Wenn ein Klient per URI eine Ressource anfordert, so wird

[3] http://www.ietf.org/

Abb. 3. Beispiel-Anfrage im Web. Besonders interessant ist der letzte Parameter. Decodiert heisst der Parameter meta=(lr=lang_de). Hier wird dem Server mitgeteilt, dass die Antwort auf Deutsch gewünscht ist.

der Server ein Abbild der geforderten Ressource erstellen und dieses Abbild dem Klienten zusenden. Für den Transport des Abbildes wird das HTTP-Protokoll verwendet, was zusätzlich ermöglicht, dass sich der Klient und der Server einigen können, in welchem Format das Abbild erstellt werden soll (Content negotiation [4]). Im HTTP-Header ist dafür ein spezielles Feld reserviert, wo der Klient das gewünschte Format dem Server direkt bei der Abfrage mitteilen kann. So kann erreicht werden, dass die Kommunikation bereits von Anfang an über das optimale Format erfolgt. Wenn Menschen einen Web-Browser bedient, so wird normalerweise eine HTML-Darstellung verwendet. Im Fall, wo Maschinen mit Maschinen interagieren, ist das Lesen und Interpretieren von HTML-Dokumenten mit viel Aufwand verbunden und so wird hier meist entweder XML oder JavaScript Object Notation (JSON)[5] verwendet. Sowohl HTML, JSON als auch XML sind von Maschinen lesbar, also „selbstbeschreibend".
Aus Sicht des Autors konnte das Web nur so gross werden, da sämtliche Ressourcen über URIs locker gekoppelt sind. So können Millionen von Ressourcen koexistieren und untereinander kommunizieren, ohne dass eine zentrale Steuereinheit nötig ist.

3.3 Stream Feeds

Im Web existieren viele Datenquellen, die sich über die Zeit verändern oder teilweise sogar über keinen statischen Zustand verfügen. Ein Beispiel für zeitveränderliche Datenquellen kann ein News-Ticker sein, der über ein RSS-Feed [11] jeweils die neusten Sportresultate auf einer Webseite publiziert. Ein Beispiel für eine konstant veränderliche Datenquelle könnte ein Online-Radio sein, welches per Real Time Streaming Protocol (RTSP) [12] rund um die Uhr sendet. Obwohl beide Beispiele durch eine zeitveränderliche Grösse charakterisierbar sind, kann trotzdem ein grundlegender Unterschied festgestellt werden: Ein RSS-Feed wird vom Nutzer bei einem speziellen Programm registriert (zb. im Web-Browser) und wird von Zeit zu Zeit nach Änderungen überprüft („pull"-Verfahren). Im Gegensatz dazu sendet ein Online-Radio permanent ein Datenstrom an den Empfänger („push"-Verfahren). Der Empfänger erhält die Änderungen also fortlaufend (auch wenn sich die Datenquelle in der Zwischenzeit nicht verändert hat). Versucht man nun Sensoren (z.B. Temperatur-Sensoren) in eine der beiden Kategorien einzuordnen, so stellt sich heraus, dass unglücklicherweise

[4] http://tools.ietf.org/html/rfc2616#section-12
[5] http://www.json.org

keine den Anforderungen gerecht wird. Will man beispielsweise laufend über die aktuellste Messung eines Temperatursensors informiert sein, so kann ein „push“-Verfahren angewendet werden. Möchte man hingegen jeweils einmal am Tag den Durchschnitt aller Messresultate erfassen, dann sollte man ein „pull“-Verfahren anwenden.
Stream Feeds versucht die Vorteile beider Kategorien in einer Abstraktion zu vereinen. Jede Ressource wird nach dem REST-Prinzip über eine eindeutige URI identifiziert und adressiert. Wünscht der Klient den momentanen Stand einer Ressource, dann fordert er über den URI ein Abbild der Ressource an. Ein Server bearbeitet die Anfrage und erstellt ein aktuelles Abbild der Ressource, welches er dem Klienten zusendet (siehe Abb. 4). Sollen die Daten zuerst vom

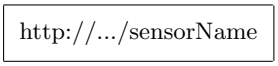

Abb. 4. Der Klient fordert eine aktuelle Kopie der Ressource ohne Aggregation beim Server.

Server aggregiert werden, so kann dies vom Klienten analog einer Web-Anfrage durch die Angabe von Parametern in der URI dem Server mitgeteilt werden. In diesem Fall erstellt der Server ein Ressourcen-Abbild und berechnet dann entsprechend den mitgeteilten Parametern die gewünschte Aggregation (siehe Abb. 5). Die Parameter können also gewissermassen als Filter-Funktionen aufgefasst werden. Dem Klienten stehen vier verschiedene Filter-Funktionen zur

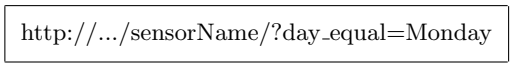

Abb. 5. Der Klient fordert eine aktuelle Kopie der Ressource, restriktiert jedoch die Anfrage auf Montag.

Verfügung: equal, not_equal, lower_bound, upper_bound. <TAG> steht jeweils für ein Parameter - z.B. „day“ in Abb. 5.
Die verschiedenen Filterfunktionen im Detail:

- <TAG>_equal: prüft auf Gleichheit.
- <TAG>_not_equal: prüft auf Ungleichheit.
- <TAG>_lower_bound: Gibt eine untere Schranke für einen Wert an.
- <TAG>_upper_bound: Gibt eine obere Schranke für einen Wert an.

Stream Feeds spezifiziert kein API mit vorgegebenen oder geforderten Parametern. Die Parameter sind abhängig vom jeweiligen Sensor und können daher den Bedürfnissen entsprechend gewählt gestellt werden.

Dem aufmerksamen Leser wird nicht entgangen sein, dass mit dieser Adressierungsform zwar sowohl aktuelle Daten als auch aggregierte Daten von Sensoren bezogen werden können, jedoch entspricht das Verfahren nur dem „pull"-Verfahren, wo der Klient die Ressource periodisch auf Änderungen überprüft. Stream Feeds bietet auch für das „push"-Verfahren eine Lösung mit Hilfe eines publish-subscribe-Verfahrens. Bei pub/sub registriert sich der Klient beim Server und hinterlegt dem Server eine URI, wohin Ereignisse der Ressource hingesendet werden sollen Abb. 6. Auch beim pub/sub-Verfahren können sämtliche Filter-Funktionen ausgeschöpft werden, da der Klient bei der Registrierung analog dem „pull"-Verfahren über eine URI auf den Sensor zugreift und der Server somit die geforderte Filterung durchführen kann.

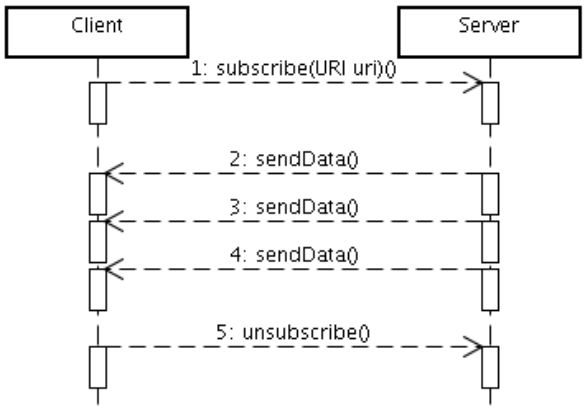

Abb. 6. Ein Klient registriert sich beim Server. Der Server sendet dem Klienten periodisch Daten. Sobald der Klient keine Daten mehr wünscht, meldet er sich beim Server ab.

Stream Feeds bietet eine aus Sicht des Autors besonders erwähnenswerte Fähigkeit mehrere Streams zu einem neuen Stream zu aggregieren. Dies soll am Beispiel des MetroNet-Projektes kurz erläutert werden.

3.4 MetroNet

MetroNet [6] ist ein Forschungsprojekt der Universität Virginia gesponsert durch Microsoft Research. Mit einem System von Sensoren wird der Fussgänger-Strom vor Geschäften gemessen (mit Bewegungssensoren, Thermalkameras etc.). Anhand der Sensordaten soll die Effizienz von Werbeflächen, Lichtreklamen und anderem ermittelt werden. Aus dem Verhältnis der Personen vor dem Geschäft und den Personen, die ins Geschäft eintreten, wird ein Indikatorwert berechnet,

[6] http://www.cs.virginia.edu/ whitehouse/research/metronet/

der die Effizienz wiederspiegeln soll.
Die Sensoren können nun per Stream Feeds angeboten werden. Indem in einem Server mehrere dieser Streams aggregiert werden, kann aus den einzelnen (unabhängigen) Streams ein neuer Stream erstellt werden. Ein Beispiel wäre hier das Folgende: Mehrere Geschäftsbesitzer stellen ihre Sensoren als Rohdaten zur Verfügung. Drittpersonen können die Rohdaten zusammenführen und damit die Anzahl der Fussgänger in der Strasse berechnen. Diese abstraktere Information könnte nun erneut als Stream bereitgestellt werden und als Grundlage für die Stadtplanung genutzt werden.
MetroNet wird zu Test- und Forschungszwecken in Charlottesville bereits eingesetzt.

4 Smarte Objekte

Wie in der Einleitung bereits beschrieben, gibt es immer mehr mobile Geräte, die sowohl mit Kommunikationsschnittstellen als auch mit einer beschränkten Rechenkapazität ausgestattet sind. Solche Geräte, auch smarte Objekte genannt, können untereinander Informationen austauschen und so ohne direkte Intervention eines Menschen einen Mehrwert für die Umgebung bewirken. So könnte z.B. das Licht beim Betreten eines Raumes automatisch auf die vom Besucher präferierte Intensität gedimmt werden. Bei Höhrgeschädigten könnte das Hörgerät die Stereoanlage automatisch veranlassen, die Lautstärke herunterzudrehen, falls die Musik für die Person zu laut oder gesundheitsschädigend ist.
Doch nach welchem Standard sollen smarte Objekte Informationen austauschen? Wie sollen Informationen überhaupt ausgedrückt werden, so dass ein smartes Objekt diese interpretieren kann?
Die folgenden zwei Sektionen beschreiben zwei etablierte Methoden für das Beschreiben von Ressourcen im „semantic web“, namentlich Resource Description Framework (RDF) [13], [14] und Web Ontology Language (OWL) [15], [16]. Nachher wird mit SOAM ein Projekt beschrieben, welches erläutert, wie RDF und OWL bei smarten Objekten eingesetzt werden könnten.

4.1 RDF

RDF ist eine Sprache, die es Maschinen ermöglicht, Informationen in Form von Metadaten über Ressourcen zu erhalten und zu interpretieren. Ein RDF-Ausdruck besteht immer aus drei Literalen: Dem Subjekt, dem Prädikat und dem Objekt. Jedes Literal wird durch eine URI eindeutig bestimmt. Möchten wir also die Aussage „John Smith hat die Webseite http://www.xyz.ch/index.html programmiert.“ in RDF ausdrücken, so muss die Aussage zuerst in ein (Subjekt, Prädikat, Objekt)-Tripel zerlegt werden. Dies ergibt in unserem Beispiel („John Smith“, „creator“, „http://www.xyz.ch/index.html“). Abb. 7 zeigt eine mögliche RDF-Schreibweise in XML-Form dieses Tripels.
Kombiniert man nun mehrere RDF-Tripel in ein RDF Dokument, so kann eine Webseite mit zusätzlichem Inhalt (also mit Semantik) versehen werden, welche von einem Computer ausgelesen und interpretiert werden kann.

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:exterms="http://www.example.org/terms/">

   <rdf:Description rdf:about="http://www.xyz/index.html">
       <exterms:creator>John Smith</exterms:creator>
   </rdf:Description>
</rdf:RDF>
```

Abb. 7. RDF-Beispiel. Jedes Literal ist durch eine eindeutige URI gebunden.

4.2 OWL

OWL ist, wie RDF, eine von Maschinen lesbare Sprache, mit deren Hilfe Ontologien [7] auf dem Web erstellt und verteilt werden können. OWL ist eine Weiterentwicklung/Erweiterung von RDF um semantische Aussagen genauer ausdrücken zu können. Dies umfasst z.B. Teile der Prädikatenlogik um Entscheidungsprobleme lösen zu können, was mit RDF nicht möglich ist.
Eine OWL Ontologie kann als eine Menge von Klassen und Axiomen verstanden werden. Axiome erstellen die Relationen zwischen den einzelnen Klassen und beschreiben deren gültige Beziehungen. Abb. 8 zeigt ein Beispiel einer OWL Ontologie.
Man stelle sich vor, ein Programm soll automatisch überprüfen, dass eine Instanz der Klasse Person immer eine Vater- und eine Mutterinstanz besitzen muss. Je nach Programmiersprache kann dies mehr oder weniger umständlich implementiert werden. Viel einfacher kann dies mit einer Ontologie erreicht werden. Analog zu Abb. 8 könnte dem OWL Dokument eine zusätzliche Ontologie hinzugefügt werden, welche das Eltern-Prädikat beschreibt. Ein Programm in einer beliebigen Programmiersprache kann nun die Korrektheit der Personen-Instanz verifizieren.

4.3 SOAM

SOAM ist ein Forschungsprojekt der Universität Deusto und beschreibt ein Framework, welches versucht das semantische Web für smarte Objekte nutzbar zu machen. Grundsätzlich entspricht SOAM einem Regelwerk vergleichbar mit REST, welches die Eckpfeiler der Kommunikation zwischen smarten Objekten regelt. Die Kommunikation zwischen Teilnehmern in SOAM findet über HTTP nach den Regeln von REST statt. So können smarte Objekte lose gekoppelt im Verbund ohne zentrale Steuereinheit interagieren.
Ein Teilnehmer in SOAM muss drei Datenstrukturen bereitstellen:

- Kontext-Informationen: Informationen, die durch Sensoren von der Umgebung gewonnen werden.

[7] Eine kurze Einführung: http://en.wikipedia.org/wiki/Ontology_(information_science)

```
<rdf:RDF>
  <owl:Class rdf:ID="Gender"/>
  <owl:Class rdf:ID="Person"/>
  <owl:Class rdf:ID="Woman">
    <rdfs:subClassOf rdf:resource="#Person"/>
    <owl:equivalentClass>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#Gender"/>
        <owl:hasValue rdf:resource="#female" rdf:type="#Gender"/>
      </owl:Restriction>
    </owl:equivalentClass>
  </owl:Class>

  <owl:ObjectProperty rdf:ID="gender"
     rdf:type="http://www.w3.org/2002/07/owl#FunctionalProperty">
    <rdfs:range rdf:resource="#Gender"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Abb. 8. OWL-Beispiel. Eine Frau ist eine Subklasse einer Person und hat die Eigenschaft „weiblich“. (Aus Platzgründen wurden Teile des XMLs weggelassen)

- Capabilities: Eine Beschreibung, über welche Kontext-Informationen das smarte Objekt verfügt.
- Constraints: Eine Datenstruktur, die den gewünschten Umgebungszustand des smarten Objekts beschreibt.

Kontext-Informationen können zwischen den smarten Objekten ausgetauscht werden. Damit diese Informationen für Maschinen lesbar sind, werden sie in RDF und OWL Ontologien beschrieben. Abb. 9 zeigt ein Beispiel für eine Kontext-Information. Mit Capabilities können smarte Objekte herausfinden, ob und wie sie ihre Umgebung beeinflussen können. Constraints geben die Möglichkeit den gewünschten Zustand anderen smarten Objekten mitzuteilen oder um deren Zustand zu beeinflussen resp. zu ändern.

Um die smarten Objekte zu „steuern“, werden zusätzlich zwei Steuereinheiten benötigt, der „Orchestrator“ und der „Adaptation User-agent“. Der Adaptation User-agent verwaltet ein vom Benutzer erstelltes Nutzungsprofil und versucht dieses mit den umliegenden Benutzern zu koordinieren. Ein Nutzungsprofil legt Constraints für smarte Objekte fest (z.B. die maximale Lautstärke für ein Hörgerät oder die minimale Beleuchtung, die ein Nutzer im Raum wünscht). Für die Steuerung der smarten Objekte selbst ist der Orchestrator zuständig. Anhand des Nutzungsprofils versucht der Orchestrator die smarten Objekte so einzustellen, so dass den Wünschen des Benutzers möglichst Rechnung getragen wird.

SOAM wurde experimentell bereits erfolgreich eingesetzt. Der Adaptation User-

```
<rdf:Description rdf:about="urn:uuid:light1">
  <lit:luminance rdf:datatype="http://www.w3.org/2001/XMLSchema#int">
     30</lit:luminance>
  <lit:color rdf:resource="http://www.awareit.com/onto/light#White"/>
  <rdf:type rdf:resource="http://www.awareit.com/onto/light#Light"/>
</rdf:Description>
```

Abb. 9. Kontext-Information: Ein weisses Licht mit Leuchtkraft 30.

agent wurde auf einem handelsüblichen Mobiltelefon installiert, der Orchestrator auf einem standard Desktop-Computer.

5 Diskussion

Obwohl REST kein neues Konzept ist und am Beispiel des Webs bereits seine Stärken gezeigt hat, wird REST trotzdem oft als „Webservices für Arme“ bezeichnet. Aus Sicht des Authors zu Unrecht. Mit der Verbreitung des Internets der Dinge werden eine Vielzahl von kommunikationsfähigen Geräten Einzug in unser Alltag halten. Alle diese Geräte sollten möglichst keinen Wartungs- und Unterhaltsaufwand generieren. Die beschränkt vorhandenen Ressourcen (z.B. Batteriekapazität oder Rechenleistung des Prozessors) müssen effizient genutzt werden und so eingesetzt werden, dass eine möglichst lange Laufzeit gewährleistet ist. Am Beispiel von SOAP wurde gezeigt, dass der Einsatz von Webservices genau bei diesem Einsatzgebiet erhebliches Verbesserungspotential birgt. SOAP wurde entwickelt, um das Konzept eines RPC-Aufrufes zu generalisieren und so zu verallgemeinern, so dass die Kommunikation zwischen beliebigen Applikationen über beliebige Netze ermöglicht wird. Applikationen verschiedener Firmen sollten über das Internet Geschäftsprozesse ermöglichen und vereinfachen. Da bei solchen Einsatzgebieten Server mit grosser Rechenkapazität zur Verfügung stehen, musste dem Aspekt der Ressourcenknappheit nur wenig bis gar keine Aufmerksamkeit geschenkt werden. Dies ist jedoch bei kleinen eingebetteten Systemen und Sensoren grundlegend anders. REST könnte hier als leichtgewichtige Alternative eingesetzt werden.
Um die Diskussion von REST abzuschliessen, soll an dieser Stelle noch auf diverse Schwachstellen von REST eingegangen werden. SOAP wurde speziell entwickelt, um beliebige Zusatzprotokolle in den SOAP-Stack zwischen zwei SOAP-Endpunkten einzubinden. Wünschen zwei Applikationen eine verschlüsselte Verbindung, so kann z.B. das WS-Security Modul verwendet werden und schon ist eine sichere Verbindung über eine beliebige Anzahl von Knoten gewährleistet. REST bietet hier nur das HTTPS-Protokoll, womit nur die direkte Verbindung zweier adjazenter Knoten gewährleistet werden kann. Soll die Verbindung über mehrere Zwischenknoten geleitet werden, so ist dies „per se“ nicht möglich. Weitere Schwachpunkte von REST sind inhärent im Protokoll enthalten. Dadurch, dass die Kommunikation zustandslos erfolgt, muss bei einer Konversation immer

der gesamte Zustand hin und her gesendet werden, was wiederum viel Sendeleistung verbraucht. Gerade bei smarten Dingen möchte man diese Sendeleistung möglichst gering halten, um die Batterie zu schonen. Die per URI übermittelten Parameter müssen sowohl vom Sender als auch vom Empfänger verstanden werden. Es ist also schwierig, herauszufinden, welche Parameter in welcher Form übermittelt werden müssen, zudem kann die Typsicherheit nicht gewährleistet werden.

Das Beispiel Stream Feeds gibt Aufschluss darauf, dass mit REST-Prinzipien durchaus auch dynamische Elemente, wie z.B. Sensoren, über eine einfach Schnittstelle angesteuert werden können, ohne dass zuerst eine applikationsspezifische Schnittstelle implementiert werden muss. Indem das HTTP-Protokoll wiederverwendet wird können Sensordaten zwischen unterschiedlichen Teilnehmern des Webs optimal ausgetauscht werden. Hierbei soll vor allem content negotiation nochmals besonders erwähnt werden. Mit content negotiation können smarte Objekte das beste Format für den Datenaustausch wählen oder sogar deren Präferenzen für spezifische Versionen der Daten mitteilen (z.B. die Sprache eines HTML-Dokuments).

Angereichert mit RDF- und OWL-Dokumenten können Daten im Web mit Semantik versehen werden. Smarte Objekte erhalten dadurch die Fähigkeit Informationen von anderen smarten Objekten zu erhalten, diese zu interpretieren und danach zu handeln. SOAM formuliert hier ein Framework, wie solche Informationen in RDF und OWL verfasst und ausgetauscht werden können.

5.1 Fazit

Auch wenn REST diverse Schwächen und Nachteile hat, bietet das Regelwerk interessante Ansätze für die Kommunikation zwischen smarten Objekten des Internets der Dinge. Basierend auf bereits gut etablierten Mechanismen des Webs, erlaubt REST eine einfache und ressourcenschonende Kommunikationsbasis für das zukünftige Web der Dinge. Aufbauend auf dem Web können smarte Objekte mit Technologien des Webs - z.B. mit SOAM - semantische Informationen austauschen und so dem Web der Zukunft neben Inhalt auch Bedeutung verleihen.

Literatur

1. Wilde, E.: *Putting Things to REST*. Technical Report UCB iSchool Report 2007-015, School of Information, UC Berkeley, 2007.
2. Fielding, R. T.: *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.
3. *Representational_State_Transfer*: Wikipedia, die freie Enzyklopadie, zugegriffen 11.04.2008 auf http://en.wikipedia.org/wiki/Representational_State_Transfer.
4. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*: W3C Recommendation, http://www.w3.org/TR/2007/REC-soap12-part1-20070427/, April, 2007.
5. *Web Services Description Language (WSDL) 1.1*: W3C Note, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, March, 2001.
6. Fielding, R., Gettys, J., et.al: *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616, http://tools.ietf.org/html/rfc2616, 1999
7. Dickerson, R., Lu, J., Whitehouse, K.: *Stream Feeds - An Abstraction for the World Wide Sensor Web*. In Proceedings of the First Internet of Things Conference, Zurich, 2008.
8. Vazquez, J. I., Lopez de Ipiña, D. , Sedano, I.: *SOAM: An Environment Adaption Model for the Pervasive Semantic Web*. In UWSI 2006: The Second Ubiquitous Web Systems and Intelligence Workshop, 2006
9. Berners-Lee, T.: *Universal Resource Identifiers in WWW*. RFC 1630, http://tools.ietf.org/html/rfc1630, 1994.
10. Berners-Lee, T., Fielding, R., et al.: *Uniform Resource Identifier (URI): Generic Syntax*. RFC 3986, http://tools.ietf.org/html/rfc3986, 2005.
11. *RSS 2.0 Specification*: RSS Advisory Board, Version 2.0.10, 2007 zugegriffen 27.04.2008 auf http://www.rssboard.org/rss-specification
12. Schulzrinne H., Rao, A., Lanphier, R.: *Real Time Streaming Protocol (RTSP)*. RFC 2326, http://tools.ietf.org/html/rfc2326, 1998.
13. World Wide Web Consortium: *RDF Primer. W3C Recommendation*. World Wide Web Consortium, http://www.w3.org/TR/2004/REC-rdf-primer-20040210/, 2004.
14. *Resource_Description_Framework*: Wikipedia, die freie Enzyklopadie, zugegriffen 03.05.2008 auf http://de.wikipedia.org/wiki/Resource_Description_Framework
15. World Wide Web Consortium: *OWL Web Ontology Language Overview. W3C Recommendation*. World Wide Web Consortium, http://www.w3.org/TR/2004/REC-owl-features-20040210/, 2004.
16. *Web_Ontology_Language*: Wikipedia, die freie Enzyklopadie, zugegriffen 03.05.2008 auf http://de.wikipedia.org/wiki/Web_Ontology_Language