Executing Application Behaviour on the Web of Things – the Read-Write Linked Data Way

Tutorial Linked Data Techniques for the Web of Things – Part II
Andreas Harth and Tobias Käfer
8th International Internet of Things Conference, Santa Barbara (CA), USA, 2018
Agenda Part II

- Web of Things and Read-Write Linked Data
- Standards and practices around Read-Write Linked Data
- Building applications on Read-Write Linked Data
  - Rule-based
  - Workflow-based
- Related work
  - Building applications using web technologies and the cloud
  - Functional Descriptions around web technologies
- Conclusion
Web of Things Architecture [1]

[1] https://www.w3.org/TR/wot-architecture/
WoT, Semantic Web, Linked Data, and Agent Architectures

Our Dynamic Linked Data Layer Cake

- WiLD
- ASM4LD
- Read-Write Linked Data
- HTTP

SW Layer Cake

- RDFS/OWL LD
- RDF
- URI

Russell / Norvig's Agent Layer Cake

- Our Dynamic Linked Data Layer Cake
- WiLD
- ASM4LD
- Read-Write Linked Data
- HTTP

SW Layer Cake

- RDFS/OWL LD
- RDF
- URI
Read-Write Linked Data and WoT Thing Descriptions

- Functional (wot Thing) Descriptions:
  - Interaction Patterns and interaction verbs
    - Property
      - readproperty, writeproperty, observeproperty
    - Action
      - invokeaction
    - Event
      - subscribeevent, unsubscribeevent
  - Interaction Models
    - Request-response
    - Publish-subscribe
    - message passing
  - Protocols
    - HTTP
    - MQTT
    - ...

There is a 1:1 mapping between the red interaction verbs and HTTP methods GET, PUT, POST
Read-Write Linked Data and the WoT Scripting API

Web of Things (WoT) Scripting API

... 

2. Use Cases

- This section is non-normative.
- The following scripting use cases are supported in this specification:

2.1 Discovery

- ...

2.2 Consuming a Thing

- …Fetch a TD and consume the thing (read the descriptions about the low level access APIs)…
- On a consumed Thing,
  - Read the value of a Property or set of properties.
  - Set the value of a Property or a set of properties.
  - Observe value changes of a Property.
  - Invoke an Action.
  - Observe Events emitted by the Thing.
  - Observe changes to the Thing Description of the Thing.
  - Get the Thing Description.
  - Get the list of linked resources based on the Thing Description.

2.3 Exposing a Thing

+ Property value/action payload in RDF
\[ \approx \text{Read-Write Linked Data} = \text{HTTP access [RFC7230seqq]} \]
+ RDF data

Linked Data Techniques for the Web of Things (Tutorial) – Andreas Harth
and TOBIAS KÄFER @ 8th International Internet of Things Conference, 2018
Applications on the Layer Cakes?

- Declarative application specifications
  - Rule-based
  - Workflow-based

Our Dynamic Linked Data Layer Cake

- WiLD
- ASM4LD
- Read-Write Linked Data
- HTTP

SW Layer Cake

- RDFS/OWL LD
- RDF
- URI

Russell / Norvig's Agent Layer Cake

- Agents with goals
- Agents with internal state

Workflow Meta Model WiLD

Simple reflex agents

Model of Computation ASM4LD

Data Model + Access

Read - Write Linked Data

System Interaction HTTP

RDFS/OWL LD

RDF

URI
Why?

- Read-Write Linked Data
  - Uniform interface of HTTP
  - Uniform data model of RDF
  → Technologies for large-scale interoperability based on decentral information

- Rule-based specifications of applications
  - Declarative
  - Compatible with rule-based reasoning
  - Stateless

- Workflow-based specifications of applications
  - Where application state is required
So much on the big picture…

WEB STANDARDS AND PRACTICES AROUND READ-WRITE LINKED DATA
Read-Write Linked Data

There is an architecture in which a few existing or Web protocols are gathered together with some glue to make a world wide system in which applications (desktop or Web Application) can work on top of a layer of commodity read-write storage. The result is that storage becomes a commodity, independent of the application running on it.

Introduction

The Linked Data article gave simple rules for putting data on the web so that it is linked. This article follows on from that to discuss allowing applications to write as well as read data.

Architectures

File write-back

The model is that all data is stored in a document (virtual or actual file) named with a URI. One way of changing the data is to overwrite the whole file with an HTTP PUT operation. Whereas typical Apache servers are not configured out of the box to accept PUT, when they are configured for WebDAV (The Web Distributed Authoring and Versioning specs) then they do. Traffic on the server is bidirectional, the server can accept a PUT with an HTTP GET.


Tim Berners-Lee

Date: 2009-16-08, last change: $Date: 2013-08-16 18:45:00$

Status: personal view only. Editing status: Not finished at all @@

Up to Design Issues
The Hypertext Transfer Protocol (HTTP) [RFC7230]

- Selected Properties of HTTP
  - Stateless
  - Request/response messages
  - Interaction with resources
  - Message: the current state of a resource

- Focus: requests that implement CRUD
  - Create, Read, Update, Delete, the basic operations of persistent storage [1]

<table>
<thead>
<tr>
<th>CRUD Operation</th>
<th>HTTP Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>GET</td>
</tr>
<tr>
<td>Update</td>
<td>PUT</td>
</tr>
<tr>
<td>Create</td>
<td>POST / PUT</td>
</tr>
<tr>
<td>Delete</td>
<td>DELETE</td>
</tr>
</tbody>
</table>

CRUD – HTTP Correspondence

<table>
<thead>
<tr>
<th>HTTP Method</th>
<th>Safe?</th>
<th>Idempotent?</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>PUT</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>POST</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DELETE</td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

POST
- Append-to-collection vs. RPC

OPTIONS
- Describe communication options

NB: No events → polling

### When Resource State is (Not) Sent/Received? – HTTP Message Semantics [RFC7231]

<table>
<thead>
<tr>
<th>HTTP Request Method</th>
<th>HTTP Request, or Response Code</th>
<th>HTTP Message Semantics: The HTTP Message Body Contains…</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET</td>
<td>Request</td>
<td>Nothing</td>
</tr>
<tr>
<td>PUT</td>
<td>Request</td>
<td>State of the resource</td>
</tr>
<tr>
<td>POST</td>
<td>Request</td>
<td>Arbitrary data or state of resource</td>
</tr>
<tr>
<td>DELETE</td>
<td>Request</td>
<td>Nothing</td>
</tr>
<tr>
<td>any</td>
<td>Non-2xx</td>
<td>State of the request</td>
</tr>
<tr>
<td>GET</td>
<td>2xx</td>
<td>State of the resource</td>
</tr>
<tr>
<td>PUT</td>
<td>2xx</td>
<td>State of the resource or empty</td>
</tr>
<tr>
<td>POST</td>
<td>2xx</td>
<td>State of the request (refering to new resource)</td>
</tr>
<tr>
<td>DELETE</td>
<td>2xx</td>
<td>State of the request or empty</td>
</tr>
</tbody>
</table>
Abstract

Linked Data Platform (LDP) defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web.

1. Introduction

This section is non-normative.

This specification describes the use of HTTP for accessing, updating, creating and deleting resources from servers that expose their resources as Linked Data. It provides clarifications and extensions of the rules of Linked Data [LINKED-DATA]:
Linked Data Platform [1]

- Classification of resources →

- Clarifications for the use of the combination HTTP + RDF, eg.:
  - 4.2.8 HTTP OPTIONS and LDPR
    - 4.2.8.1 LDP servers MUST support the HTTP OPTIONS method.
    - 4.2.8.2 LDP servers MUST indicate their support for HTTP Methods by responding to a HTTP OPTIONS request on the LDPR’s URL with the HTTP Method tokens in the HTTP response header Allow.
  - 5.2.3 HTTP POST and LDPC
    - 5.2.3.1 LDP clients SHOULD create member resources by submitting a representation as the entity body of the HTTP POST to a known LDPC. If the resource was created successfully, LDP servers MUST respond with status code 201 (Created) and the Location header set to the new resource’s URL. Clients shall not expect any representation in the response entity body on a 201 (Created) response.

- Cf. ATOM publishing protocol [RFC5023]: interactions with collections

REST (Representational State Transfer) [1] and the Richardson Maturity Model (RMM) for Services [2]

 Glory of REST

- **Level 0: The Swamp of POX**
  - Subdivide the service endpoint to resources

- **Level 1: Resources**
  - Use the HTTP verbs according to the standard

- **Level 2: HTTP Verbs**
  - Add links to the transferred data

- **Level 3: Hypermedia Controls**


SPECIFYING APPLICATIONS FOR READ-WRITE LINKED DATA USING RULES
Example

- A light sensor available as Linked Data
- A LED available as Read-Write Linked Data

{ < 0.5 } =>

- Turn on the LED if the light sensor’s value drops below a certain threshold
Developing ASM4LD

- HTTP GET / Readproperty
  - To retrieve the CURRENT state of one device
  - To retrieve the CURRENT state of multiple devices / systems
    → Retrieve the world state in RDF

- ASK Queries on the world state → Conditions for actions

- HTTP PUT / Writeproperty
  → Actions
  → Set the state of components
ASM4LD-based User Agents for Read-Write Linked Data

- **Aim**: Execution of agent specifications on Read-Write Linked Data
- **Approach**:
  - Directly operate on the world state
  - Inspired by Simple Reflex Agents [1]
- **In a nutshell**:
  ```
  while(true):
    sense()
    HTTP-GET
  think()
    Queries in rule bodies
  act()
    HTTP-PUT/POST/DELETE
  ```

ASM4LD – a Model of Computation for Read-Write Linked Data [LDOW2018]

- Abstract state machines [2]
- Model-theoretic semantics of RDF Graphs
- Semantics of RDF Datasets
- Semantics of HTTP

ASM4LD [1]
- Supports Read-Write Linked Data:
  - URIs to identify things
  - HTTP for interaction
  - RDF for describing data
  - Writing to Linked Data
- Embraces reasoning
- Embraces link following
- Turing complete

SEMSE: Retrieve the world state
THINK: Conditionally...
ACT: ...manipulate the world state

Käfer and Harth: Rule-based programming of user agents for Linked Data. Proc. 11th LDOW@TheWebConf (2018)
ASM4LD DEMO
A Binary Counter in ASM4LD for 2 LEDs of a Tessel 2 [1]

```
@prefix http: <http://www.w3.org/2011/http#>.
@prefix http_m: <http://www.w3.org/2011/http-methods#>.
@prefix saref: <https://w3id.org/saref#>.

# For the URIs to the LEDs
@prefix leds: <http://t2-rest-leds.lan/leds/>.

# Data retrieval:
{ _:h http:mthd http_m:GET ; http:requestURI leds: . }

# The logic:
{ leds:2#led saref:hasState saref:Off . }
=>
{ _:h http:mthd http_m:PUT ; http:requestURI leds:2 ; http:body { leds:2#led saref:hasState saref:On . } . }.

{ leds:2#led saref:hasState saref:On . }
=>
{ _:h http:mthd http_m:PUT ; http:requestURI leds:2 ; http:body { leds:2#led saref:hasState saref:Off . } . }.

{ leds:2#led saref:hasState saref:On .
  leds:3#led saref:hasState saref:Off . }
=>
{ _:h http:mthd http_m:PUT ; http:requestURI leds:3 ; http:body { leds:2#led saref:hasState saref:On . } . }.

{ leds:2#led saref:hasState saref:On .
  leds:3#led saref:hasState saref:On . }
=>
{ _:h http:mthd http_m:PUT ; http:requestURI leds:3 ; http:body { leds:2#led saref:hasState saref:Off . } . }.
```

[1] https://github.com/kaefer3000/t2-rest- leds
Linked Data-Fu Example: Distributed VR System Composition [1]

We encoded in Linked Data-Fu rules:
- Movement of the avatar according to Kinect data
- Detection of user gestures
- Movement of the map according to gestures
- Loading of concert data from the web
- Data integration between VR RWLD API, concert LD API, Kinect LD API
- Execution at Kinect sensor refresh rate (30Hz)

Finite State Machines (Mealy Automata) and Transition Systems

A State Machine

\[ i_1/o_1 \quad i_2/o_2 \]

\[ i_3/o_3 \quad i_4/o_4 \]

\[ s_1 \quad s_2 \]

\( i_n \): input
\( o_n \): output
Finite State Machines (Mealy Automata) and Transition Systems

A Transition System

\[
\begin{align*}
&i_3/o_3 \\
&S_1 \\
&i_1/o_1 \\
&S_2 \\
&i_2/o_2 \\
&i_4/o_4
\end{align*}
\]

\( i_n \): input \\
\( o_n \): output
Describing an Origin Server in an Linked Data Transition System [1]

Linked Data Transition System of Resource /relay/1 on server http://t2-ambient-relay.lan/

i₁/o₁

i₂/o₂

i₃/o₃

i₄/o₄

iᵢᵢ: input
oᵢᵣ: output

Describing an Origin Server in an Linked Data Transition System [1]

> GET /leds/1
< 200 OK
< <#led> a :LED ;
< :isOn false .

> PUT /leds/1
> <#led> a :LED ;
> :isOn true .
< 204 No Content

<#led> a :LED ; :isOn false .

> GET /leds/1
< 200 OK
< <#led> a :LED ;
< :isOn true .

> PUT /leds/1
> <#led> a :LED ;
> :isOn false .
< 204 No Content

<#led> a :LED ; :isOn true .

Transition System of Resource /leds/1 on server http://t2-led.lan/

From the perspective of the resource on the server:
> denote incoming requests, cf. inputs in automata terminology
< denote outgoing responses, cf. outputs in automata terminology

Labelled Transition Systems

Labelled Transition System $LTS = (S, L, \rightarrow)$

- $S$: Set of States
- $L$: Set of Labels
  - Typically some of: input/event, condition, output/action
- $\rightarrow \subseteq (S \times L \times S)$: Transition Relation

→ How can we describe Dynamic Linked Data as Transition System?
RDF Dataset

Definition [1]
- A collection of RDF graphs $G$
- Each graph has a URI $u$ as name
- The default graph has an empty name
- No restriction on the relation graph – name

A Linked Data view [2, section 3.5]:
- Name = the information resource's URI where the graph can get obtained

<table>
<thead>
<tr>
<th>Name</th>
<th>RDF Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>$u_1$</td>
<td>$G_{u1}$</td>
</tr>
<tr>
<td>/leds/1</td>
<td>&lt;#led&gt; a :LED ; :isOn false .</td>
</tr>
</tbody>
</table>

Linked Data Transition System \( LDTS \) := \( (S, \rightarrow) \)

- \( S \) : Possible states of all resources (set of RDF datasets)
  - \( s_n = \{(u, G_{u,n})\} \in S \): RDF dataset at point in time \( n \)
- \( \rightarrow \) : Transition Relation
  - \( \rightarrow \subseteq S \times 2^{Req,Resp} \times S \)

\[ \text{Name} \quad \text{RDF Graph} \]

\[ \begin{array}{ll}
    u_1 & G_{u1,s1} \\
    /\text{leds}/1 & <\#\text{led}> a :\text{LED}; :\text{isOn} \text{ false} . \\
\end{array} \]

\[ \begin{array}{ll}
    u_1 & G_{u1,s2} \\
    /\text{leds}/1 & <\#\text{led}> a :\text{LED}; :\text{isOn} \text{ true} . \\
\end{array} \]

State Machines, Transition Systems, and Linked Data [1]


SPECIFYING APPLICATIONS FOR READ-WRITE LINKED DATA USING WORKFLOWS
Fine-granular distributed components with REST / RWLD interfaces everywhere

- How to create integrated applications that maintain application state? Use workflows [1]?
- → We need a solution that combines:
  - Workflows
  - Semantic reasoning
  - RESTful interaction

Uniform Interface: Read-Write Linked Data
- REST as uniform interaction mechanism between systems
- RDF as uniform data model, ready for semantic data integration

WEB OF THINGS
- Interfaces to IoT sensors/actuators
- Built on Linked Data
- Interfaces to personal data storages
- Built on Linked Data
- Interfaces to core company functions
- Built on REST (lift to Linked Data)

## WiLD in our Layer Cakes

<table>
<thead>
<tr>
<th>Russell / Norvig‘s Agent Layer Cake</th>
<th>Our Dynamic Linked Data Layer Cake</th>
<th>SW Layer Cake</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agents with goals</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Agents with internal state</strong></td>
<td><strong>Workflow Meta Model</strong></td>
<td><strong>WiLD</strong></td>
</tr>
<tr>
<td>Simple reflex agents</td>
<td><strong>Model of Computation</strong></td>
<td><strong>ASM4LD</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Data Model + Access</strong></td>
<td><strong>Read-Write Linked Data</strong></td>
</tr>
<tr>
<td></td>
<td><strong>System Interaction</strong></td>
<td><strong>HTTP</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>RDFS/OWL LD</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>RDF</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>URI</strong></td>
</tr>
</tbody>
</table>
On our Workflow Management System Architecture and Overall System State

WiLD Operational Semantics

ASM4LD Implementation

OWLLD Semantics

Workflow Instances using WiLD Vocabulary

Linked Data Platform Container

Linked Data Component 1

Linked Data Component n

Workflow Models using WiLD Vocabulary

System under Observation and Control

Linked Data Component
The WiLD Ontology – our Workflow Language for Workflow Models and Instances

[Diagram showing relationships between workflow components such as Activity, CompositeActivity, SequentialActivity, ParallelActivity, and WorkflowModel.]
On our Workflow Management System Architecture and Overall System State

WiLD Operational Semantics

ASM4LD Implementation

OWLLD Semantics

Workflow Instances using WiLD Vocabulary

Linked Data Platform Container

Linked Data Component 1

Linked Data Component n

Workflow Models using WiLD Vocabulary

System under Observation and Control

**WiLD DEMO**
Set-up

WiLD Operational Semantics

Linked Data-Fu

OWLLD Semantics

Workflow Model and Instances

LDP Server

Tessel 2 with Loudspeaker

Tessel 2 with RFID reader

Tessel 2 with RFID reader

Tessel 2 with Light Sensor and RFID reader

System under Observation and Control

Linked Data Techniques for the Web of Things (Tutorial) – Andreas Harth and TOBIAS KÄFER @ 8th International Internet of Things Conference, 2018
Question whether dark or bright

Please cover the sensor

Well done!

Please do not cover the sensor

Well done!

Oh dear!

Oh dear!

Correct

Incorrect

Correct

Incorrect
WiLD Example: i-VISION

“SELECT the push-buttons in the Virtual Reality that are involved in the upcoming steps of the currently running take-off workflow and highlight them”

http://www.ivision-project.eu/
RELATED WORK
## Mapping the Field of Service Composition and Web Agent Specification

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability description</td>
<td>Input, Output, Precondition, Effect (for automated composition)</td>
</tr>
<tr>
<td></td>
<td>Affordance (for manual composition)</td>
</tr>
<tr>
<td>Composition description</td>
<td>Rules, BPEL, Pi calculus, Petri Nets (Temporal) logic</td>
</tr>
<tr>
<td></td>
<td>Unformalised Implementation</td>
</tr>
<tr>
<td>Dynamics model</td>
<td>ASM, LTS, Situation Calculus, Unformalised Implementation</td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF), Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2, RMM1, RMM0, push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions.*
Related Work

BUILDING APPLICATIONS

- IFTTT “if-this-then-that”
  - Automate tasks on the web
    - Eg. “If I tweet, post that also on Facebook”
  - Samsung ARKTIK rules
    - Automate on the Internet of Things
      - Eg. “If the temperature of the room is more than 72°F, then turn on my bedroom’s light and set the color to red” (sic!)

- …Centralised Platforms
- Event-Action rules
- Events = Notifications from devices/APIs

---

[1] https://ifttt.com/maker_webhooks
Turn on the Light using IFTTT Maker Channel

![Maker event “light_state_change”](image)

- Create an account, register key
- Register event type, eg. `light_state_change`
- Whenever there is a change:
  - POST to `https://maker.ifttt.com/trigger/light_state_change/with/key/{secret_key}`
  - HTTP body (must be JSON, keys have to be named exactly like that):
    ```json
    { "value1" : "test",
      "value2" : 0.5,
      "value3" : True }
    ```

Adapted from http://www.makeuseof.com/tag/ifttt-connect-anything-maker-channel/
Turn on the Light using ARKTIK

{  
  "if": {  
  "and": [ {  
  "sdid": "sensor123" ,  
  "field": "value",  
  "operator": "<",  
  "operand": 0.5
  } ]
  },
  "then": [ {  
  "action": "httpRequest",
  "parameters": {  
  "method": { "value": "PUT" },
  "url": {  
  "value": "http://t2-ambient-relay.lan/relay/1"
  },
  "body": {  
  "@id": "#r",  
  "http://example.org/isOn": true
  }
  }
  ]
  }
}
## Classifying IFTTT and ARKTIK (&Co.)

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability description</td>
<td>Input, Output, Precondition, Effect (for automated composition) Affordance (for manual composition)</td>
</tr>
<tr>
<td>Composition description</td>
<td>Rules BPEL * Pi calculus Petri Nets (Temporal) logic Unformalised Implementation</td>
</tr>
<tr>
<td>Dynamics model</td>
<td>ASM LTS Situation Calculus Unformalised Implementation</td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF) Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2 RMM1 RMM0 push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions*
## ASM4LD / Looped Linked Data-Fu

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability description</td>
<td>Input, Output, Precondition, Effect (for automated composition)</td>
</tr>
<tr>
<td>Composition description</td>
<td>Rules, BPEL *, Pi calculus, Petri Nets (Temporal) logic, Unformalised Implementation</td>
</tr>
<tr>
<td>Dynamics model</td>
<td>ASM, LTS, Situation Calculus, Unformalised Implementation</td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF), Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2, RMM1, RMM0, push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions*
Related Work

SERVICE AND AFFORDANCE DESCRIPTIONS
OWL-S

- OWL-S: for descriptions of (SOAP) web services
  - Aim: Automated web service discovery, invocation, composition, monitoring
  - WSDL descriptions of web services (SOAP) do not suffice

- OWL-S Service Profile / Model:
  - Functionality description of a service
  - Profile: “Advertising” eg. to be put in a registry for service discovery
  - Model: For service composition and invocation
  - Contents (~ for both Profile and Model):
    - Input (what to give to a service when invoking)
    - Output (what the service will return when invoked)
    - Precondition (what has to hold before the service invocation)
    - Effect / Postcondition / Result (what holds after the service invocation)

- Results of a composition:
  - BPEL, Proofs, … [1]

http://www.w3.org/Submission/OWL-S/

### Classifying OWL-S and WSMO

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability</td>
<td>Input, Output, Precondition, Effect (for automated</td>
</tr>
<tr>
<td>description</td>
<td>composition)</td>
</tr>
<tr>
<td></td>
<td>Affordance (for manual composition)</td>
</tr>
<tr>
<td>Composition</td>
<td>Rules, BPEL, Pi calculus, Petri Nets, (Temporal)</td>
</tr>
<tr>
<td>description</td>
<td>logic, Unformalised Implementation</td>
</tr>
<tr>
<td>Dynamics</td>
<td>ASM, LTS, Situation Calculus, Unformalised Implementation</td>
</tr>
<tr>
<td>model</td>
<td></td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF), Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2, RMM1, RMM0, push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions*
## Next: Affordance Descriptions

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability description</td>
<td>Input, Output, Precondition, Effect</td>
</tr>
<tr>
<td></td>
<td>(for automated composition)</td>
</tr>
<tr>
<td></td>
<td>Affordance</td>
</tr>
<tr>
<td></td>
<td>(for manual composition)</td>
</tr>
<tr>
<td>Composition description</td>
<td>Rules</td>
</tr>
<tr>
<td></td>
<td>BPEL</td>
</tr>
<tr>
<td></td>
<td>Pi calculus</td>
</tr>
<tr>
<td></td>
<td>Petri Nets</td>
</tr>
<tr>
<td></td>
<td>(Temporal) logic</td>
</tr>
<tr>
<td></td>
<td>Unformalised Implementation</td>
</tr>
<tr>
<td>Dynamics model</td>
<td>ASM</td>
</tr>
<tr>
<td></td>
<td>LTS</td>
</tr>
<tr>
<td></td>
<td>Situation Calculus</td>
</tr>
<tr>
<td></td>
<td>Unformalised Implementation</td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF)</td>
</tr>
<tr>
<td></td>
<td>Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2</td>
</tr>
<tr>
<td></td>
<td>RMM1</td>
</tr>
<tr>
<td></td>
<td>RMM0</td>
</tr>
<tr>
<td></td>
<td>push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions.

A Transition System

\[
\begin{align*}
S_1 & \xrightarrow{i_1[c_1]/o_1} S_2 \\
S_2 & \xrightarrow{i_2[c_2]/o_2} S_1 \\
S_1 & \xrightarrow{i_3[c_3]/o_3} S_2 \\
S_2 & \xrightarrow{i_4[c_4]/o_4} S_1
\end{align*}
\]
Hydra

- Motivation:
  - Many web APIs do essentially similar things using differing terminology
  - With some standardisation, we could build generic agents
- Hydra: an API documentation standardisation effort building on established technologies:
  - Linked Data vocabularies, JSON-LD, and HTTP headers
- Contents
  - Links between resources that allow for requests
  - Possible requests
  - Required data in requests
  - Detailing out HTTP status information
- Similar and related concepts
  - LDP, ATOM (Collections)
  - HTTP headers (Allow)
schema.org Potential Actions and WoT Actions

- schema.org Potential actions
  - Typed actions (e.g. BuyAction)
  - Optional fields include:
    - Input and output schema
    - Result
    - Target
    - HTTP method

- WoT Thing Descriptions
  - Defines (for a thing):
    - Actions (horizontal arrows)
    - Properties
    - Events
  - Well-known relative URIs for actions and properties of a thing
  - Requirements on the use of HTTP and resource representations

http://schema.org
http://w3c.github.io/wot-thing-description/
Motivation:
- Web APIs provide non-RDF output data for some input values
- Even if we lift the output to RDF, the relation between input and output is missing
- With some descriptions of the Web API, we can relate the inputs to the lifted output

Example:
- We want `foaf:based_near` triples for places characterised using `geo:lat` and `geo:long`
- We have the description
  ```sparql
  CONSTRUCT { ?point foaf:based_near ?feature }
  WHERE { ?point geo:lat ?lat . ?point geo:long ?lng }
  ```
- We query for the WHERE clause in the data we already have
  ```sql
  SELECT * WHERE { ?point geo:lat ?lat . ?point geo:long ?lng }
  ```
- We call the API with the variables from the WHERE clause (that do not appear in the CONSTRUCT) as parameters and get back data like described in the CONSTRUCT
  ```http
  > GET /findNearbyWikipedia?lat=37.416&lng=-122.152#point HTTP/1.1
  > Host: geowrap.openlids.org
  < 200 OK
  <http://geowrap...Wikipedia?lat=37.416&lng=-122.152#point>
  foaf:based_near dbp:Palo_Alto%2C_California ;
  foaf:based_near dbp:Packard%27s_garage .
  ```

RESTdesc [1]

- **Aim**: Automated service composition and composition execution in the presence of hyperlinks in HTTP responses

- **Composition problem**:
  - **Initial knowledge**
    
    $$<\#r> : isOn \; false .$$

  - **API descriptions**:
    
    $$\{ \text{preconditions} \} \Rightarrow \{ \text{HTTP-request} . \text{postconditions} \} .$$

    - Precondition, Postcondition: ~ BGP; Postcondition ~ HTTP response’s body
    - HTTP-Request: (Method, URI + optional parameters)
      
      - Optional: eg. body: URIs or literals

  - **Goal specification**

    $$\{ <\#r> : isOn \; true \} \Rightarrow \{ <\#r> : isOn \; true \} .$$

  - **Background knowledge, eg. ontologies**

RESTdesc Algorithm [1]

1) Start an N3 reasoner to generate a pre-proof for \((R, g, H, B)\):
   a) If the reasoner is not able to generate a proof, halt with failure.
   b) Else scan the pre-proof for applications of rules of \(R\), set the number of these applications to \(n_{\text{pre}}\).

2) Check \(n_{\text{pre}}\):
   a) If \(n_{\text{pre}} = 0\), halt with success.
   b) Else continue with 3).

3) Out of the pre-proof, select a sufficiently specified HTTP request description which is part of the application of a rule \(r \in R\).

4) Execute the described HTTP request and parse the (possibly empty) server response to a set of ground formulas \(G\).

5) Invoke the reasoner with the new API composition problem \((R, g, H \cup G, B)\) to produce a post-proof.

6) Determine \(n_{\text{post}}\):
   a) If the reasoner was not able to generate a proof, set \(n_{\text{post}} := n_{\text{pre}}\).
   b) Else scan the proof for the number of inference steps which are using rules from \(R\) and set this number of steps to \(n_{\text{post}}\).

7) Compare \(n_{\text{post}}\) with \(n_{\text{pre}}\):
   a) If \(n_{\text{post}} \geq n_{\text{pre}}\), go back to 1) with the new API composition problem \((R \setminus \{r\}, g, H, B)\).
   b) If \(n_{\text{post}} < n_{\text{pre}}\), the post-proof can be used as the next pre-proof. Set \(n_{\text{pre}} := n_{\text{post}}\) and continue with 2)

Classifying RESTdesc

<table>
<thead>
<tr>
<th>Level</th>
<th>Foundational approaches / categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capability description</td>
<td>Input, Output, Precondition, Effect (for automated composition)</td>
</tr>
<tr>
<td>Composition description</td>
<td>Rules, BPEL * Pi calculus, Petri Nets (Temporal) logic</td>
</tr>
<tr>
<td>Dynamics model</td>
<td>ASM, LTS, Situation Calculus, Unformalised Implementation</td>
</tr>
<tr>
<td>Data model</td>
<td>Graph (RDF), Nested (JSON, XML)</td>
</tr>
<tr>
<td>Data access</td>
<td>RMM2, RMM1, RMM0, push</td>
</tr>
</tbody>
</table>

*Semantics of BPEL have been given eg. in Petri Nets and ASMs, but Petri Nets are also used to describe compositions*
Conclusion

- Correspondence of Read-Write Linked Data to the WoT architecture
- Standards and practices around Read-Write Linked Data
- ASM4LD and WiLD for specifying behaviour on Read-Write Linked Data
- Demo
- Related work