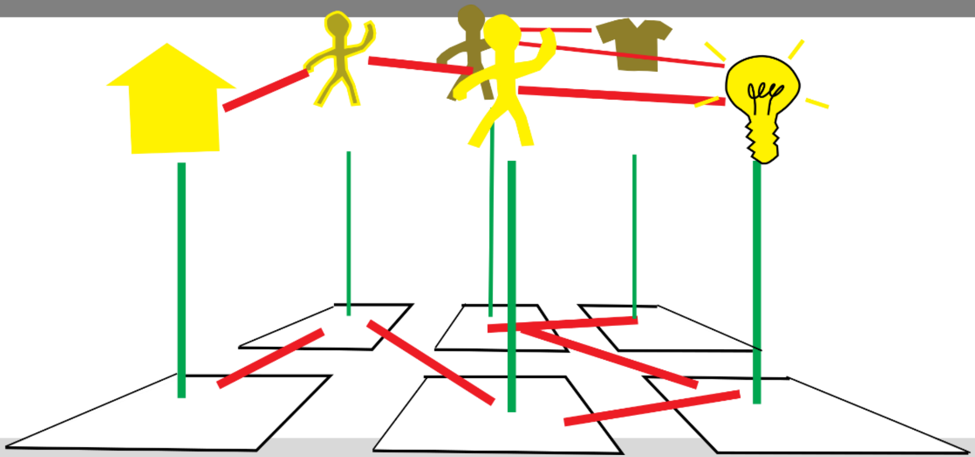
Rule-based Processing of Dynamic Linked Data

Tutorial at the European Semantic Web Conference (ESWC) 2017
Andreas Harth and Tobias Käfer

INSTITUTE AIFB, CHAIR FOR WEB SCIENCE



About the Presenters

- Andreas Harth

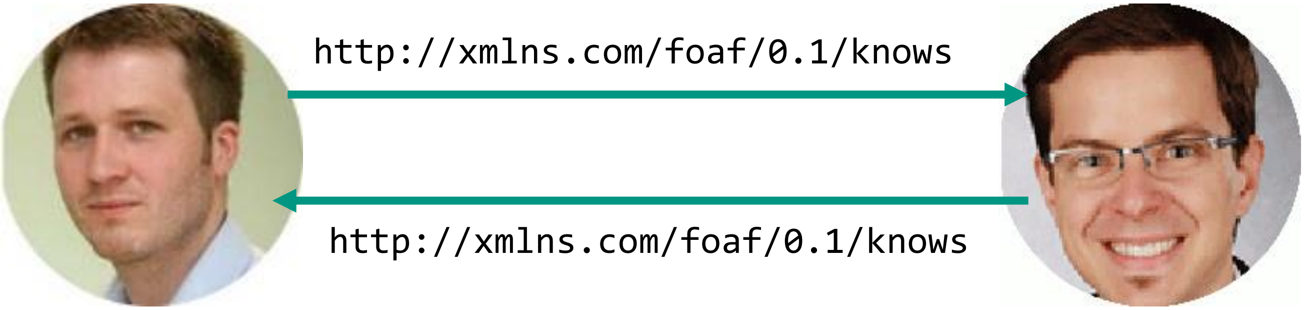
`http://harth.org/andreas/foaf#ah`

`http://www.aifb.kit.edu/id/Andreas_Harth`

- Tobias Käfer

`http://www.aifb.kit.edu/id/Tobias_K%C3%A4fer`

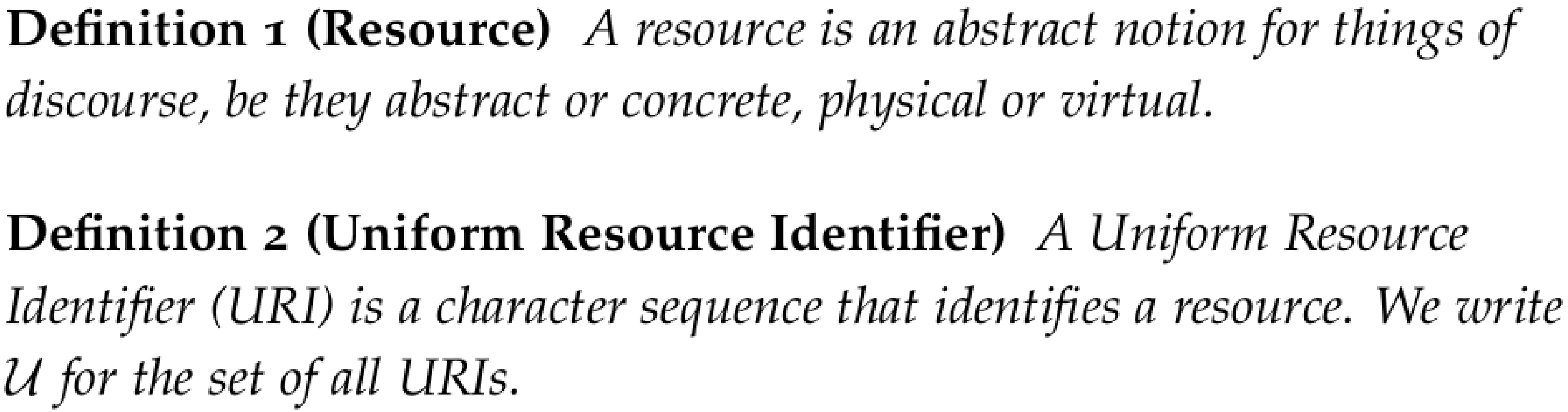
`http://www.aifb.kit.edu/id/Tobias_Käfer`



`http://xmlns.com/foaf/0.1/knows`

`http://xmlns.com/foaf/0.1/knows`

Resources and URIs



Definition 1 (Resource) *A resource is an abstract notion for things of discourse, be they abstract or concrete, physical or virtual.*

Definition 2 (Uniform Resource Identifier) *A Uniform Resource Identifier (URI) is a character sequence that identifies a resource. We write $\mathcal{U}$ for the set of all URIs.*

Outline Part I (Andreas)

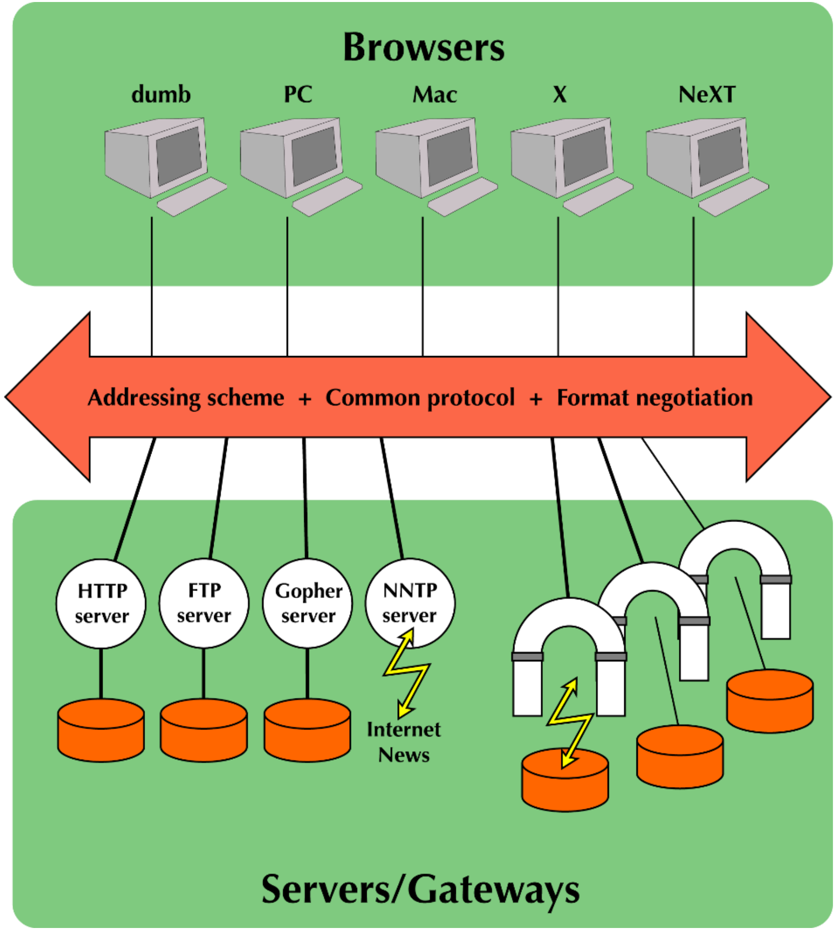
- Basic Concepts (Linked Data Principles)
- Query Evaluation on RDF Datasets (Principle #3)
- Query Evaluation with Entailment (Principle #3)
- Link Traversal Query Evaluation (Principle #4)
- Half-Time Summary

- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)

BASIC CONCEPTS

Web Architecture

- URI: RFC 1630 (1994), now RFC 3986
- HTTP: RFC 1945 (1996), now RFC 7230, 7231, 7232, 7234, 7235



https://www.w3.org/DesignIssues/diagrams/history/Architecture_crop.png
Redrawn from an image from 1990

Servers and User Agents

- Client

“A program that establishes connections for the purpose of sending requests.”

- User Agent

“The client which initiates a request. These are often browsers, editors, spiders (web-traversing robots), or other end user tools.”

- Server

“An application program that accepts connections in order to service requests by sending back responses.”

“Any given program may be capable of being both a client and a server; our use of these terms refers only to the role being performed by the program for a particular connection, rather than to the program's capabilities in general. […]”

All definitions from RFC2616 (obsolete)

Uniform Resource Identifiers (RFC 3986)

- Uniform
- Resource

“This specification does not limit the scope of what might be a resource; rather, the term ‘resource’ is used in a general sense for whatever might be identified by a URI.”

- an electronic document
- an image
- a source of information with a consistent purpose (e.g., ‘today's weather report for Los Angeles’)
- a service (e.g., an HTTP-to-SMS gateway)
- a collection of other resources
- the operators and operands of a mathematical equation
- the types of a relationship (e.g., ‘parent’ or ‘employee’)
- numeric values (e.g., zero, one, and infinity)

- Identifier

Linked Data

- Postulated by Tim Berners-Lee in 2006.

> *“The Semantic Web isn't just about putting data on the web. It is about making links, so that a person or machine can explore the web of data. With linked data, when you have some of it, you can find other, related, data.”* [1]

- Collection of rules governing the publication and consumption of data on the web
- Aim: unified method for describing and accessing resources
- Later we will also see how to manipulate resource state

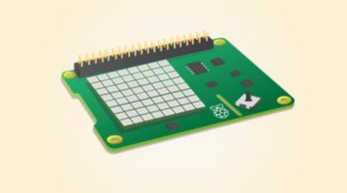
[1] http://www.w3.org/DesignIssues/LinkedData.html

Linked Data Principles

1. Use URIs as names for things
2. Use HTTP URIs so that people can look up those names.
3. When someone looks up a URI, provide useful information, using the standards (RDF*, SPARQL)
4. Include links to other URIs. so that they can discover more things.

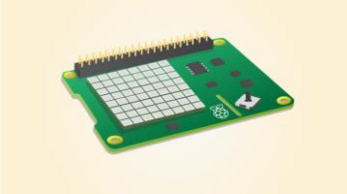
http://www.w3.org/DesignIssues/LinkedData.html

# 1 and # 2 Resources and URIs
Raspberry Pi Sense HAT



Resource	URI
The Raspbery Pi Sense HAT	`http://raspi.lan/hat#id`
The gyroscope	`/gyroscope#id`
The accelerometer	`/accelerometer#id`
The thermometer	`/thermometer#id`
The barometric pressure sensor	`/barometer#id`
The humidity sensor	`/hygrometer#id`
The magnetometer	`/magnetometer#id`
The LED panel	`/led-panel#id`

# 3: Provide Useful Information



- A User Agent performing a HTTP GET on `http://raspi.lan/hat` leads to:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix ldp: <http://www.w3.org/ns/ldp#> .

<#id> rdfs:comment "RasPi Sense HAT“ ;
      rdfs:seeAlso <gyroscope#id> , <accelerometer#id> ,
            <thermometer#id> , <barometer#id> ,
            <hygrometer#id> , <magnetometer#id> ,
            <led-panel#id> .
```

Linked Data Principles

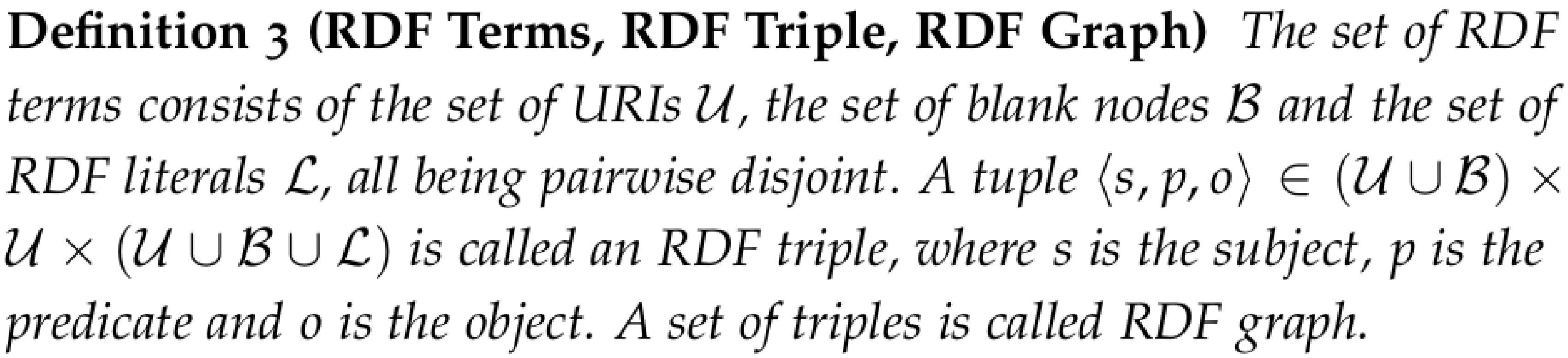
Server – Data Provider

1. Coin URIs to name things
2. Provide HTTP URIs
3. Provide useful information using the standards (RDF, RDFS, SPARQL) upon HTTP request
4. Include links to other URIs in the HTTP response

User Agent – Data Consumer

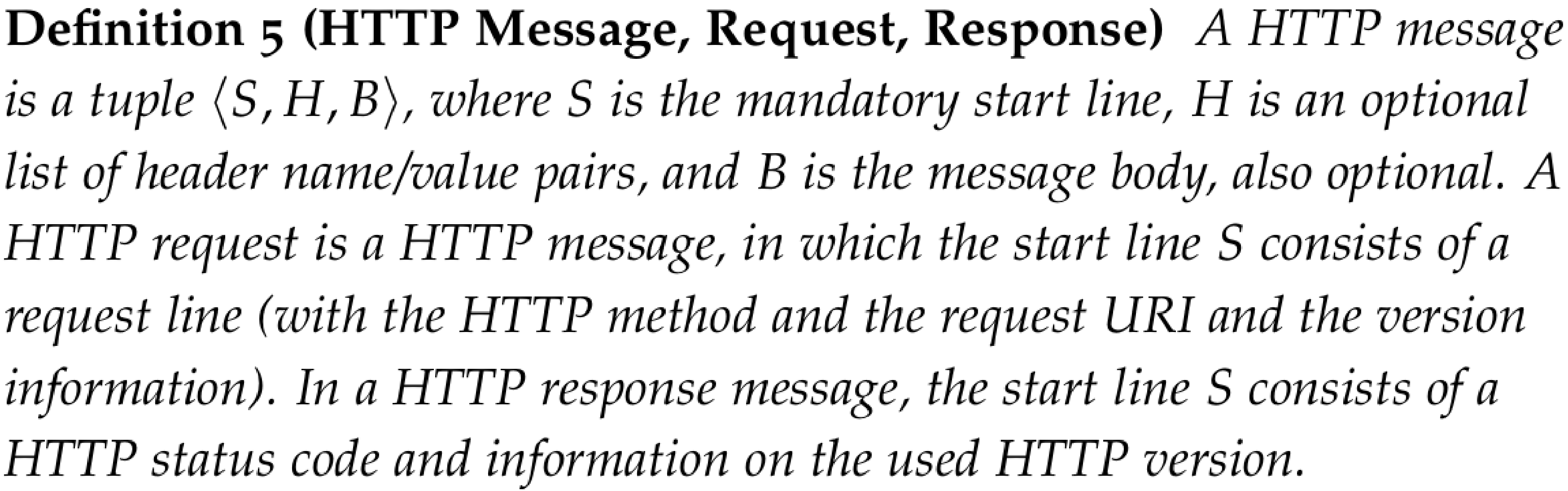
1. Use URIs as names
2. Operate on HTTP URIs
3. Look up URIs (emit HTTP request messages) and process the data returned in HTTP response messages using the standards (RDF, RDFS, SPARQL)
4. Use links to other URIs to discover more things

RDF Abstract Syntax



Definition 3 (RDF Terms, RDF Triple, RDF Graph) *The set of RDF terms consists of the set of URIs $\mathcal{U}$, the set of blank nodes $\mathcal{B}$ and the set of RDF literals $\mathcal{L}$, all being pairwise disjoint. A tuple $\langle s, p, o \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times (\mathcal{U} \cup \mathcal{B} \cup \mathcal{L})$ is called an RDF triple, where s is the subject, p is the predicate and o is the object. A set of triples is called RDF graph.*

HTTP Request/Response Abstract Syntax



Definition 5 (HTTP Message, Request, Response) *A HTTP message is a tuple $\langle S, H, B\rangle$, where S is the mandatory start line, H is an optional list of header name/value pairs, and B is the message body, also optional. A HTTP request is a HTTP message, in which the start line S consists of a request line (with the HTTP method and the request URI and the version information). In a HTTP response message, the start line S consists of a HTTP status code and information on the used HTTP version.*

Goal of the Tutorial

- We build on the basic notions of web architecture
  - HTTP URIs, Request/Response
  - Manipulation of resource state
  - Hyperlinks
- Our user agents operate on resources
  - Just current resource state („now“), no distinction between „static“ and „dynamic“ data
- We show how to specify user agent behaviour in rules
  - Declarative queries and rules, no imperative programming
  - Simple language can be basis for visual programming language
- Overall: we value simplicity to achieve web-scale
  - Focus on execution, no artificial intelligence planning

William of Ockham, 1287-1347

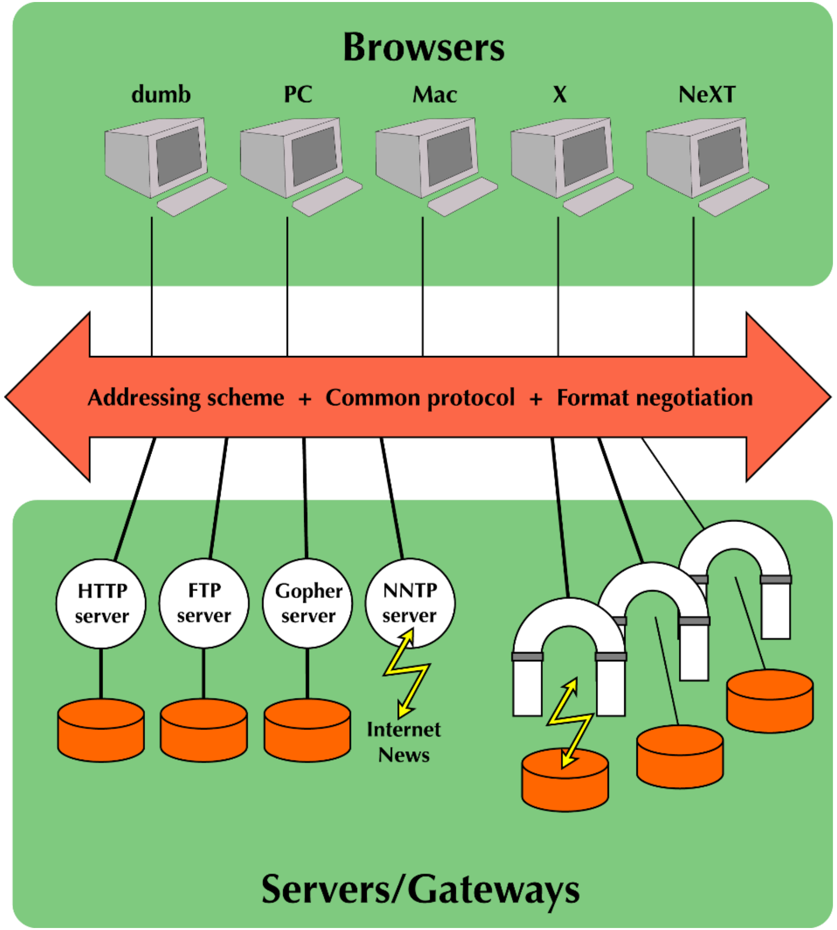
The simplest

system architecture

~~explanation~~ is usually the correct one.

Web Architecture

- URI: RFC 1630 (1994), now RFC 3986
- HTTP: RFC 1945 (1996), now RFC 7230, 7231, 7232, 7234, 7235

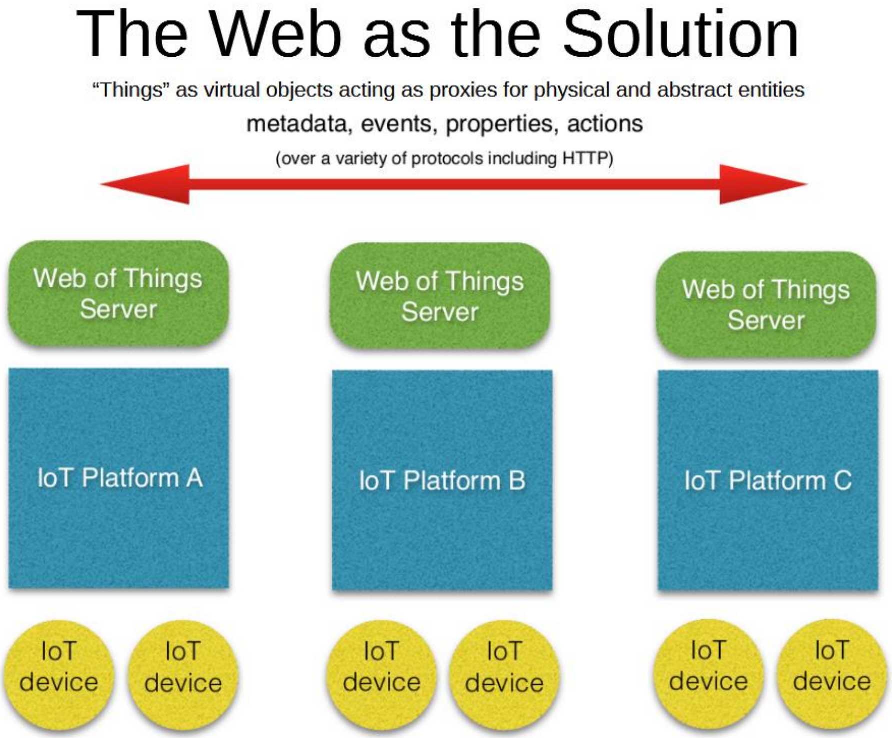


https://www.w3.org/DesignIssues/diagrams/history/Architecture_crop.png
Redrawn from an image from 1990

Linked Data Architecture (2009)

https://www.w3.org/2009/Talks/0204-ted-tbl/#(7)

Web of Things Architecture (2015)



The Web as the Solution

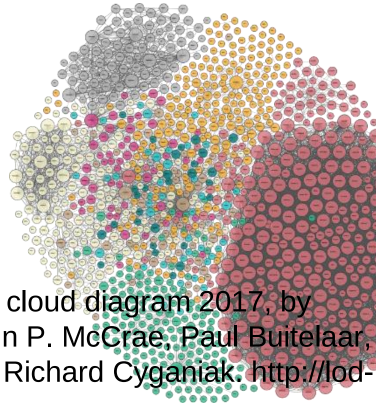
“Things” as virtual objects acting as proxies for physical and abstract entities

metadata, events, properties, actions

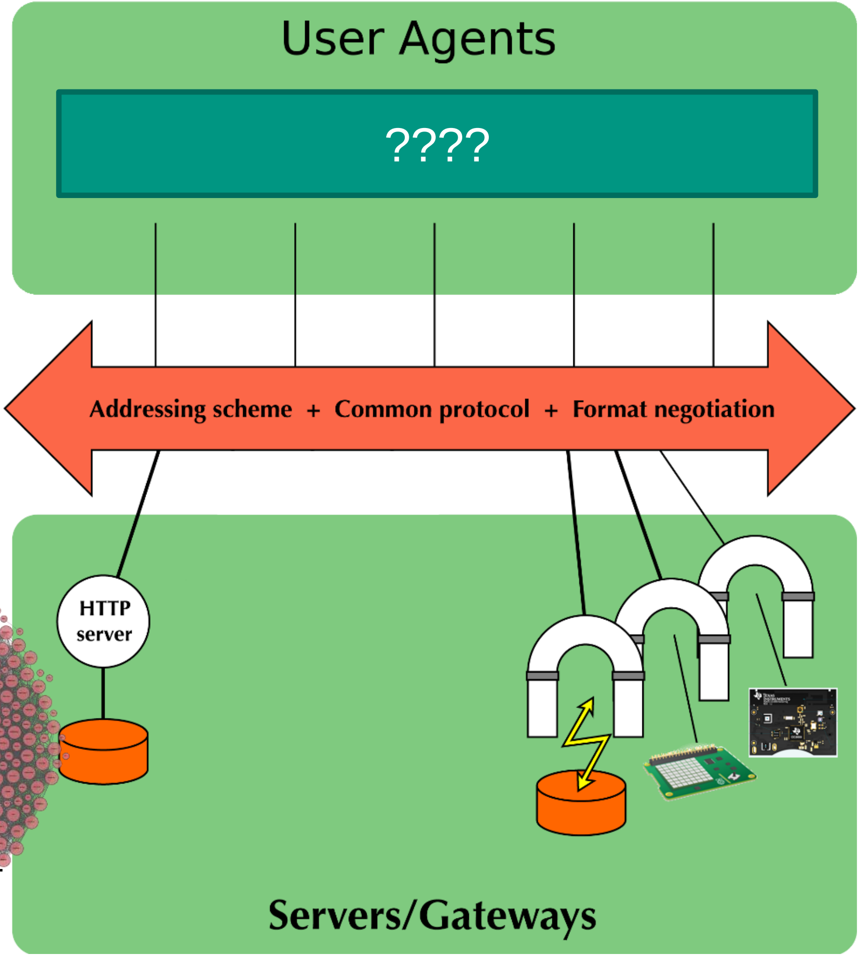
(over a variety of protocols including HTTP)

https://www.w3.org/2015/05/wot-framework.pdf

Web of Data/ Web of Things Architecture



Linking Open Data cloud diagram 2017, by Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch and Richard Cyganiak. http://lod-cloud.net/



Adapted from https://www.w3.org/DesignIssues/diagrams/history/Architecture_crop.png

Examples of Linked Data User Agents

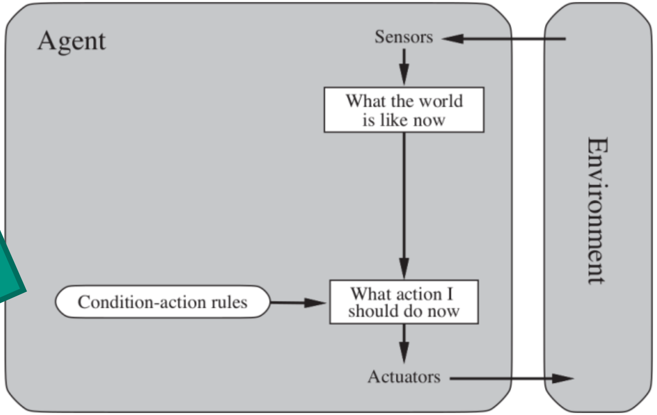
- Temperature recorder (for temperature of room, building, area…)
- Temperature display (for temperature of room, building, area…)
- Thermostat (for room temperature)
  - Depending on who is in the room
- Heat alarm (for temperature of room, building, area…)
- … (your ideas?)

- So-called „Recipes“ (WoT) or „Applets“ (on IFTTT) provide similar functionality

Cognitive Architectures

- SOAR (initially: State, Operator, Apply, Result),
- ACT-R (Adaptive Control of Though – Rational)
- Goal: to create „intelligent agents“
- In the tutorial, we only consider user agents that are
  - „simple reflex agents“ (Russel & Norvig, see figure),
  - aka „tropistic agents“ (Genesereth & Nilson)
- We explain how to use rules to control the agent‘s behaviour
- Part I: safe HTTP methods (GET)
- Part II: unsafe HTTP methods

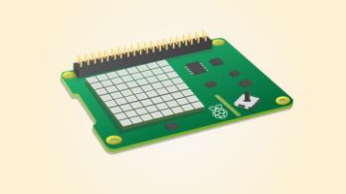
Russel and Norvig, Artificial Intelligence – A Modern Approach, Third Edition, 2010



QUERY EVALUATION ON RDF DATASETS

SPARQL Query

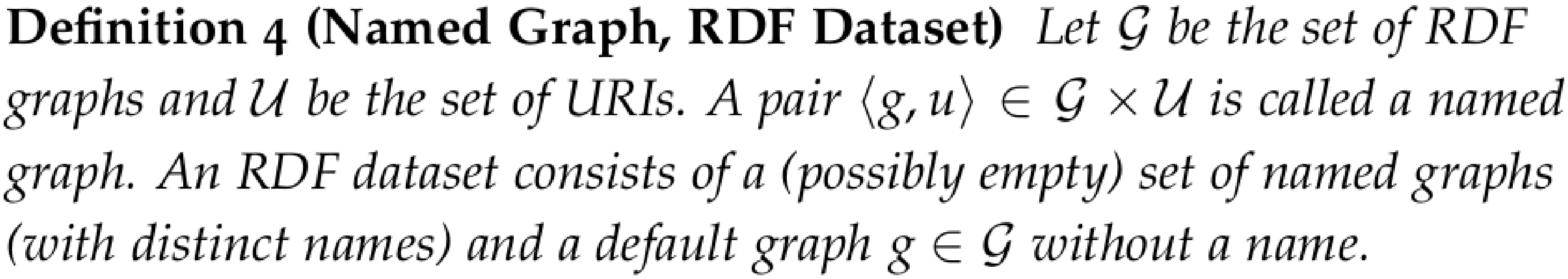
- Return the value and unit of the observation from the thermometer*:



```
PREFIX sosa: <http://www.w3.org/ns/sosa/>
PREFIX qudt: <http://qudt.org/1.1/schema/qudt#>

SELECT ?obs ?val ?unit
FROM <http://raspi.lan/thermometer>
WHERE {
   ?obs sosa:madeBySensor <http://raspi.lan/thermometer#id> ;
        sosa:hasResult [ qudt:numericValue ?val ;
                         qudt:unit ?unit ] .
}
```

* Subject to changes in the SOSA ontology.

SPARQL FROM and FROM NAMED



Definition 4 (Named Graph, RDF Dataset) *Let $\mathcal{G}$ be the set of RDF graphs and $\mathcal{U}$ be the set of URIs. A pair $\langle g, u \rangle \in \mathcal{G} \times \mathcal{U}$ is called a named graph. An RDF dataset consists of a (possibly empty) set of named graphs (with distinct names) and a default graph $g \in \mathcal{G}$ without a name.*

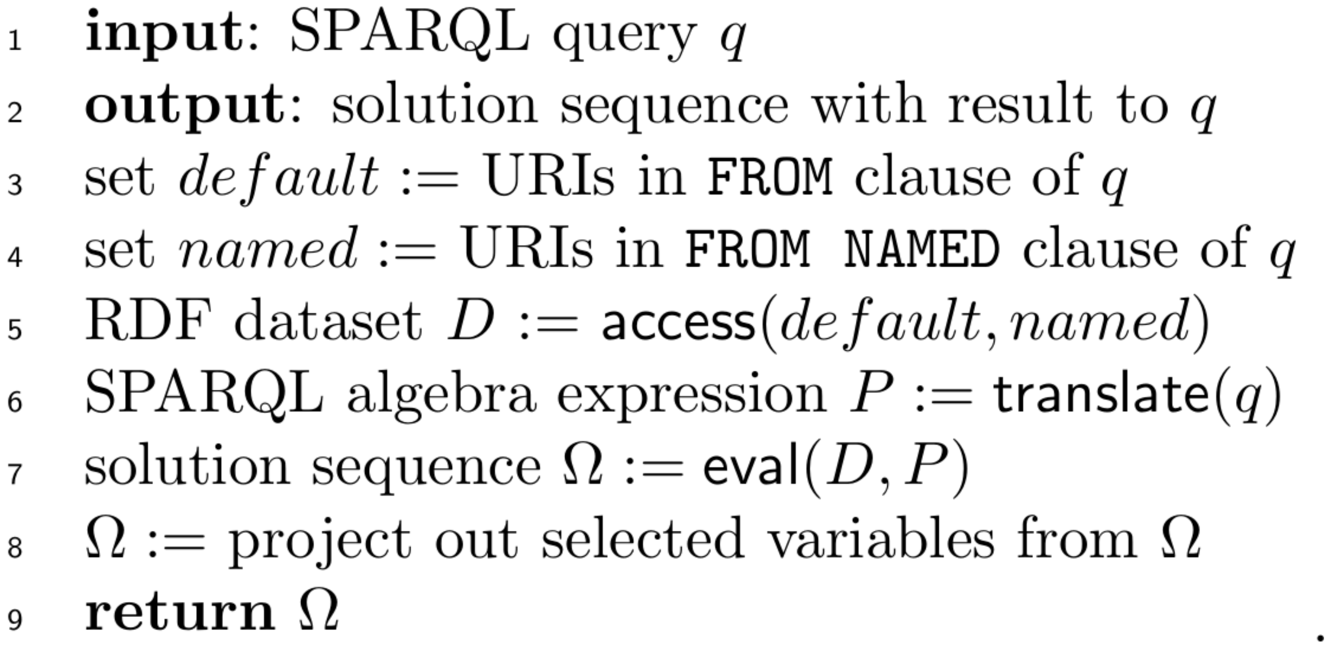
We assume the graph names are also addresses to locations with RDF documents.

Dave Beckett‘s roqet

- SPARQL processors operate on RDF datasets
- Many SPARQL processors operate on local RDF datasets
- A SPARQL „database“, to which you first import your data and then pose queries

- There are query processing user agent
- E.g., roqet (http://librdf.org/rasqal/roqet.html) (for years)
- Install on Debian: # apt-get install rasqal-utils
- Roqet dereferences the URIs of the graphs in the RDF dataset during query time
- Possible to access current data for query processing

Algorithm for SPARQL SELECT User Agent



```
**input**: SPARQL query $q$
**output**: solution sequence with result to $q$
set $default$ := URIs in FROM clause of $q$
set $named$ := URIs in FROM NAMED clause of $q$
RDF dataset $D := \mathsf{access}(default, named)$
SPARQL algebra expression $P := \mathsf{translate}(q)$
solution sequence $\Omega := \mathsf{eval}(D, P)$
$\Omega$ := project out selected variables from $\Omega$
**return** $\Omega$ .
```

Translating Queries to Algebra Expressions

- Need a translate() algorithm to get an algebra expression, given a query in SPARQL syntax
- Then, apply evalute() algorithm with dataset and algebra expression
- When dereferencing the URIs in the dataset, we always access the information resources (URIs identifying RDF documents, not people)

Hands-On!

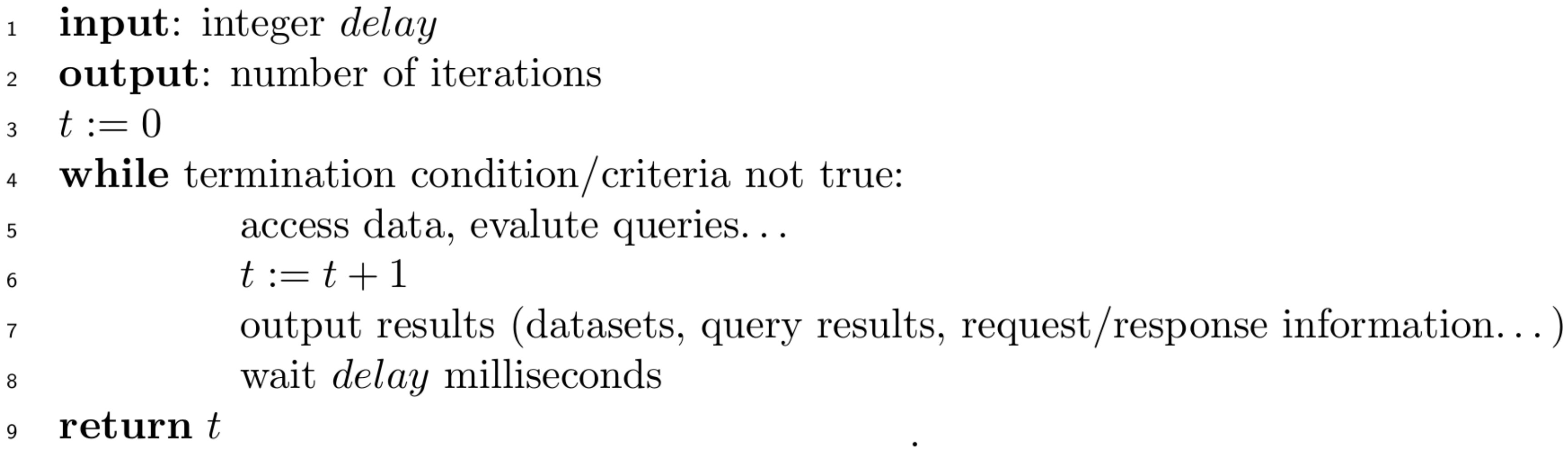
- Go to http://t2-ambient-relay.lan/
- Find the light sensor
- Write a SPARQL SELECT query that returns the light value

- Either install roqet (# apt-get install rasqal-utils on Debian/Ubuntu)
- Or:
- SSH into 192.168.254.214
- Username: pi
- Password: pi
- $ roqet light.rq

Repeated Query Evalution

```
SELECT ?value
FROM <http://t2-ambient-relay/ambient/light>
WHERE {
 ?x <http://example.org/hasLightValue> ?value .
}
```

Algorithm Agent Loop



```
**input**: integer $delay$
**output**: number of iterations
$t := 0$
**while** termination condition/criteria not true:
        access data, evalute queries...
        $t := t + 1$
        output results (datasets, query results, request/response information...)
        wait $delay$ milliseconds
**return** $t$
```

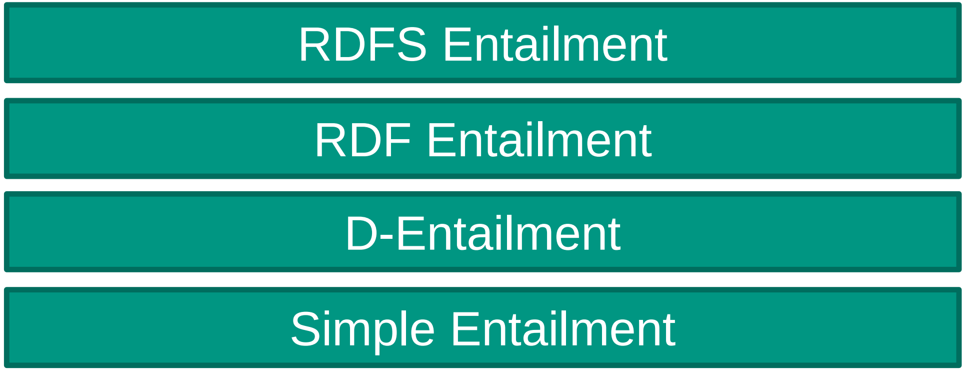
QUERY EVALUATION WITH ENTAILMENT

# 3: Provide Useful Information in RDF/RDFS

- RDF and RDFS have a model-theoretic semantics
- Requires appropriate implementation (datatypes, inference rules…)

- We could even do a bit of OWL (including owl:sameAs)

Layering of Entailment



RDFS Entailment

RDF Entailment

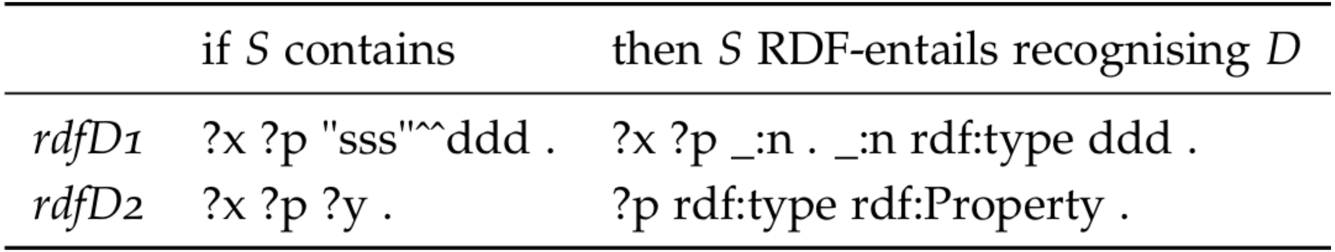
D-Entailment

Simple Entailment

- RDF 1.1 moved datatype entailment (D-entailment)
- Now, datatype entailment comes first, then RDF entailment, then RDFS entailment
- In the following, we present the entailment patterns for RDF and RDFS (and omit axiomatic triples)

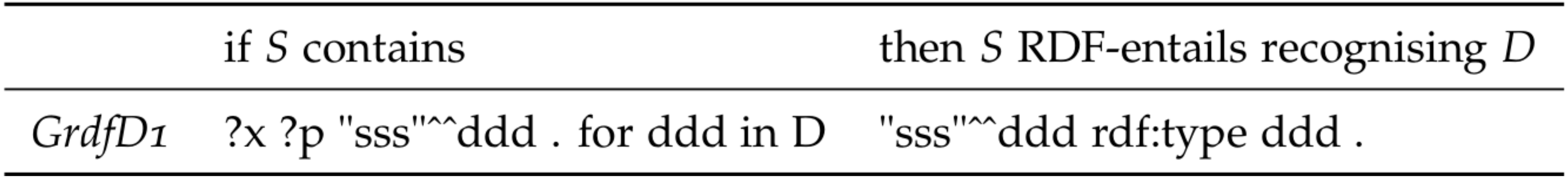
RDF Entailment Patterns

- S is a graph, D is the set of supported datatypes (at least rdf:langString and xsd:string)
- Patterns in conjunction with RDF data model:



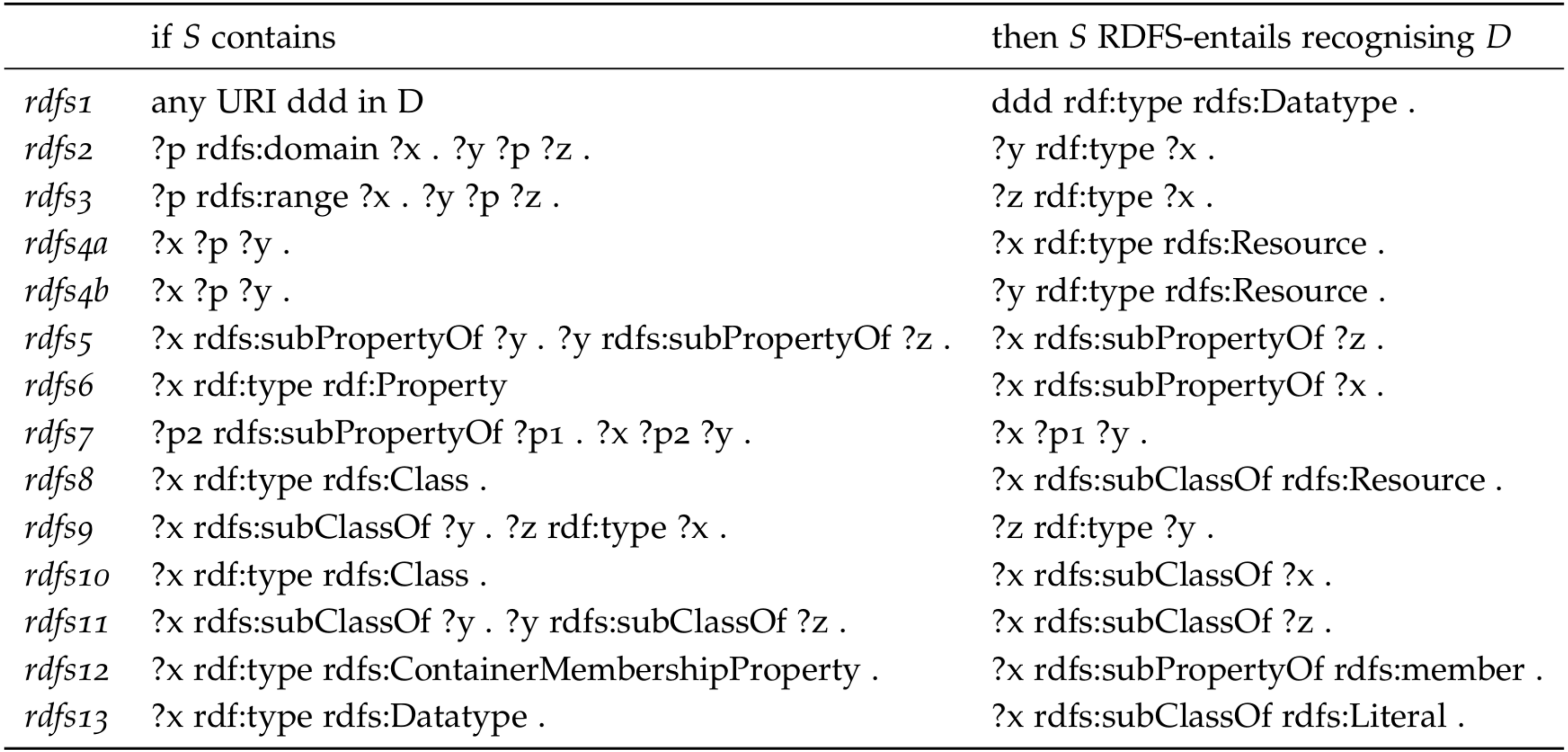
	if *S* contains	then *S* RDF-entails recognising *D*
rdfD1	?x ?p "sss"^^ddd .	?x ?p _:n . _:n rdf:type ddd .
rdfD2	?x ?p ?y .	?p rdf:type rdf:Property .

- Patterns in conjunction with generalised RDF data model:



	if *S* contains	then *S* RDF-entails recognising *D*
GrdfD1	?x ?p "sss"^^ddd . for ddd in D	"sss"^^ddd rdf:type ddd .

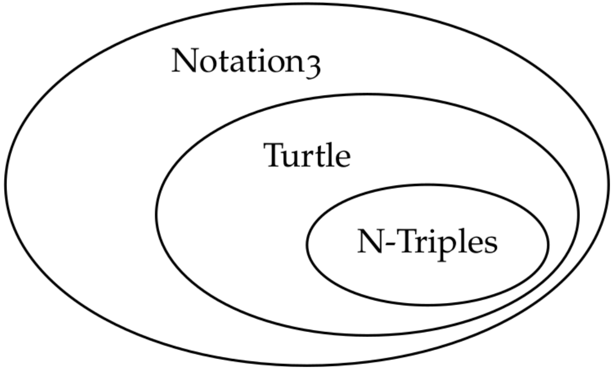
RDFS Entailment Patterns



	if *S* contains	then *S* RDFS-entails recognising *D*
rdfs1	any URI ddd in D	ddd rdf:type rdfs:Datatype .
rdfs2	?p rdfs:domain ?x . ?y ?p ?z .	?y rdf:type ?x .
rdfs3	?p rdfs:range ?x . ?y ?p ?z .	?z rdf:type ?x .
rdfs4a	?x ?p ?y .	?x rdf:type rdfs:Resource .
rdfs4b	?x ?p ?y .	?y rdf:type rdfs:Resource .
rdfs5	?x rdfs:subPropertyOf ?y . ?y rdfs:subPropertyOf ?z .	?x rdfs:subPropertyOf ?z .
rdfs6	?x rdf:type rdf:Property	?x rdfs:subPropertyOf ?x .
rdfs7	?p2 rdfs:subPropertyOf ?p1 . ?x ?p2 ?y .	?x ?p1 ?y .
rdfs8	?x rdf:type rdfs:Class .	?x rdfs:subClassOf rdfs:Resource .
rdfs9	?x rdfs:subClassOf ?y . ?z rdf:type ?x .	?z rdf:type ?y .
rdfs10	?x rdf:type rdfs:Class .	?x rdfs:subClassOf ?x .
rdfs11	?x rdfs:subClassOf ?y . ?y rdfs:subClassOf ?z .	?x rdfs:subClassOf ?z .
rdfs12	?x rdf:type rdfs:ContainerMembershipProperty .	?x rdfs:subPropertyOf rdfs:member .
rdfs13	?x rdf:type rdfs:Datatype .	?x rdfs:subClassOf rdfs:Literal .

Notation3 Syntax

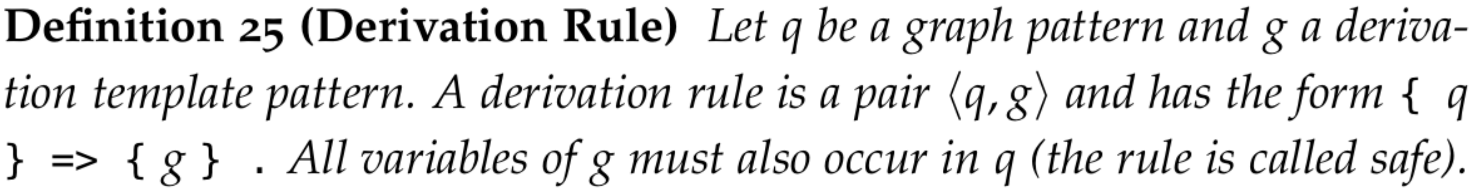
- Notation3 syntax is a superset of Turtle syntax
- N3 adds variables (prefixed with a question mark „?“)
- N3 adds graph quoting
  - Subject or object can consist of triple patterns



- Example

```
@prefix : <#> .
@prefix log: <http://www.w3.org/2000/10/swap/log#> .
{ ?x :p ?y } log:implies { ?y :q ?x . } .
```

Derivation Rule Abstact Syntax



Definition 25 (Derivation Rule) *Let q be a graph pattern and g a derivation template pattern. A derivation rule is a pair $\langle q, g \rangle$ and has the form* `{` q `} => {` g `} .` *All variables of g must also occur in q (the rule is called safe).*

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

# rdfD2
{ ?xxx ?aaa ?yyy . } => { ?aaa rdf:type rdf:Property . } .
```

Query Processing User Agent with Entailment (forward-chaining)

- Input: Dataset URIs, Query (as SPARQL algebra expression), Ruleset (as SPARQL algebra expression), L2V datatypes map
- Output: Query results (or error)

- Construct RDF dataset for local processing
- Materialise RDF dataset to default graph
- Evaluate SPARQL algebra expression over local (materialised) RDF dataset
- Return query results

Query Processing User Agent with Entailment (backward-chaining)

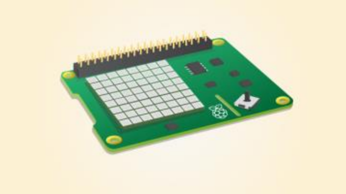
- Input: Dataset URIs, Query (as SPARQL algebra expression), Ruleset (as SPARQL algebra expression), L2V datatypes map
- Output: Query results (or error)

- Construct RDF dataset for local processing
- Rewrite Query to take into account Ruleset
- Evaluate SPARQL algebra expression over local RDF dataset
- Return query results

LINK TRAVERSAL QUERY PROCESSORS

# 4: Discovering More Things (Recap)

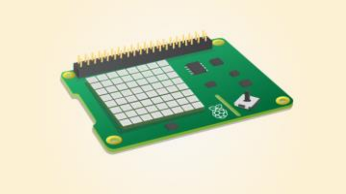
- A User Agent performing a HTTP GET on `http://raspi.lan/hat` leads to:



```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix dcterms: <http://purl.org/dc/terms> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .

<> dcterms:date "2017-05-28" .
<#id> rdfs:seeAlso <thermometer#id> , <barometer#id> .
<thermometer#id> a sosa:Sensor .
<barometer#id> a sosa:Sensor .
```

# 3: Provide Useful Information (Recap)

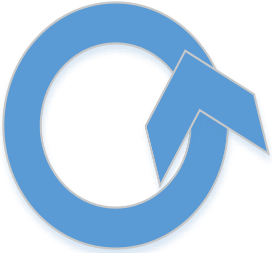
- A User Agent performing a HTTP GET on `http://192.168.0.1/thermometer` leads to:



```
@prefix dcterms: <http://purl.org/dc/terms> .
@prefix qudt: <http://qudt.org/2.0/schema/qudt> .
@prefix qudt-u: <http://qudt.org/1.1/vocab/unit#> .
@prefix sosa: <http://www.w3.org/ns/sosa/> .

<> dcterms:date "2017-05-28" .
<#id> a sosa:Sensor .
[] sosa:madeBySensor <#id> ;
   sosa:hasResult [ qudt:numericValue 20 ;
                    qudt:hasUnit qudt-u:DegreeCelcius ] .
```

Link Traversal User Agent Model

1. The user agent starts its interaction based on a specified seed URI.
2. The user agent performs HTTP requests[1] on URIs and parses the response message.
3. Based on the response, the user agent has the choice as to which URIs to dereference next.
4. The user agent decides which link(s) to follow and initiates a new request/new requests.



[1] We restrict HTTP requests to GET requests for now.

- Follow one link to arrive at a path through the web of information resources (e.g., depth-first search)
- Follow all links to arrive at a tree of information resources (e.g., breadth-first search)

Link Traversal Query Processing User Agent

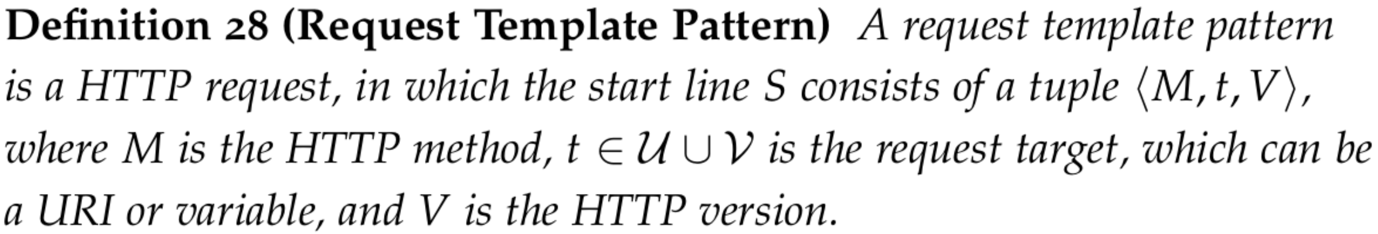
- Input: Query (as SPARQL algebra expression)
- Output: Query results

- Construct RDF dataset for local processing from URIs in Query
- Evaluate SPARQL algebra expression over local RDF dataset
- Return query results

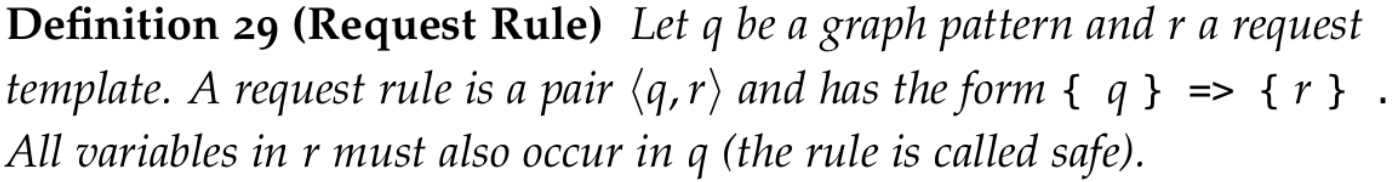
How to Specify which Links to Traverse?

- Function that maps current state to additional requests
- Implicitely encoded in the algorithm/system
- Or: encode the function using rules

Request Rule Abstract Syntax



Definition 28 (Request Template Pattern) *A request template pattern is a HTTP request, in which the start line S consists of a tuple $\langle M, t, V \rangle$, where M is the HTTP method, $t \in \mathcal{U} \cup \mathcal{V}$ is the request target, which can be a URI or variable, and V is the HTTP version.*



Definition 29 (Request Rule) *Let q be a graph pattern and r a request template. A request rule is a pair $\langle q, r \rangle$ and has the form* `{ q } => { r } .` *All variables in r must also occur in q (the rule is called safe).*

Example Traversal Rule

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix http: <http://www.w3.org/2011/http#> .
@prefix httpm: <http://www.w3.org/2011/http-methods#> .

{ [] http:mthd httpm:GET ;              # access index (entry) page
   http:requestURI <http://raspi.lan/hat> . }

{
  <http://raspi.lan/hat#id> rdfs:seeAlso ?x .       # match rdfs:seeAlso triples
} => {
  [] http:mthd httpm:GET ;              # access linked pages
     http:requestURI ?x .
} .
```

Link Traversal Query Processors

- Hartig et al. ISWC 2009
  - Index nested loops joins
  - Iterator model (bolted on top of ARQ/Jena)
- Harth et al. WWW 2010
  - Repeated evaluation
  - Improve source index over time
- Ladwig and Tran ISWC 2010
  - Architecture for link-traversal query processor (push, SHJ)
  - But no recursion
- Fionda, Gutierrez and Pirro WWW 2012 (also: NautiLOD)
  - No notion Information Resource Graph
  - Function calls on results (no REST state manipulation)
  - But: recursion on graph traversal paths
- Hartig and Perez 2016: LDQL
  - Language to specify link traversal
  - Keep link traversal specification and query specification separate

Link Traversal Query Processing User Agent

- Input: Query (as SPARQL algebra expression), link expansion specification
- Output: Query results

- Construct RDF dataset for local processing from Query
- Evaluate SPARQL algebra expression over local RDF dataset
- Return query results

- Loop: do repeated query evaluation
- 2nd time: know sources already

Hands-On!

- Download Linked Data-Fu
- You need JDK 1.7+
- Unzip archive

- Run
- $ ./linked-data-fu-0.9.12/bin/ldfu.sh -p get-all.n3 -q props.rq /tmp/out.tsv -p ./linked-data-fu-0.9.12/rulesets/rdf.n3

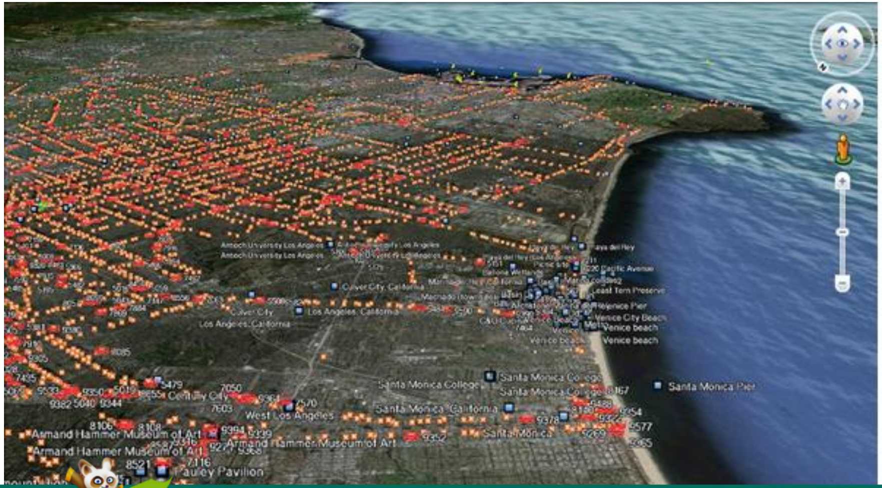
HALF-TIME SUMMARY

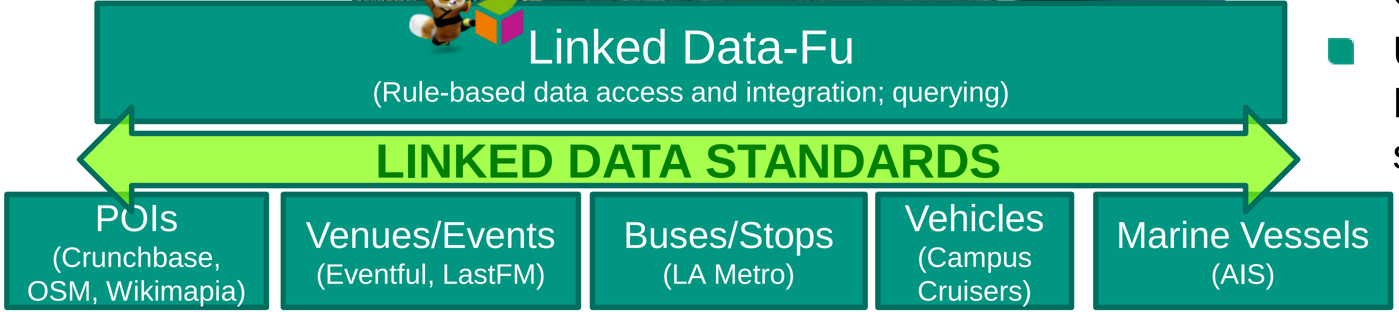
Data Integration on APIs

- Now you can easily do data integration on Linked Data
- If you work on (REST) APIs, you need wrappers, but they are reasonably easy to build
- If you have software libraries that abstract from the API, you can just build an interface to the software library
- You need to keep in mind to use a resource-oriented interface on the web

Near-Realtime Data Integration for Los Angeles

Andreas Harth, Craig Knoblock, Steffen Stadtmüller, Rudi Studer and Pedro Szekely. "On-the-fly Integration of Static and Dynamic Linked Data". Fourth International Workshop on Consuming Linked Data (COLD 2013).





Linked Data-Fu

(Rule-based data access and integration; querying)

LINKED DATA STANDARDS

POIs (Crunchbase, OSM, Wikimapia)

Venues/Events (Eventful, LastFM)

Buses/Stops (LA Metro)

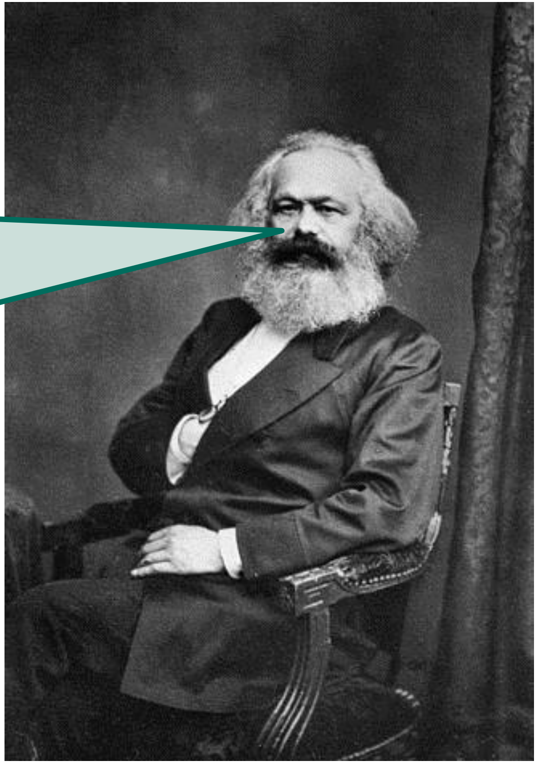
Vehicles (Campus Cruisers)

Marine Vessels (AIS)

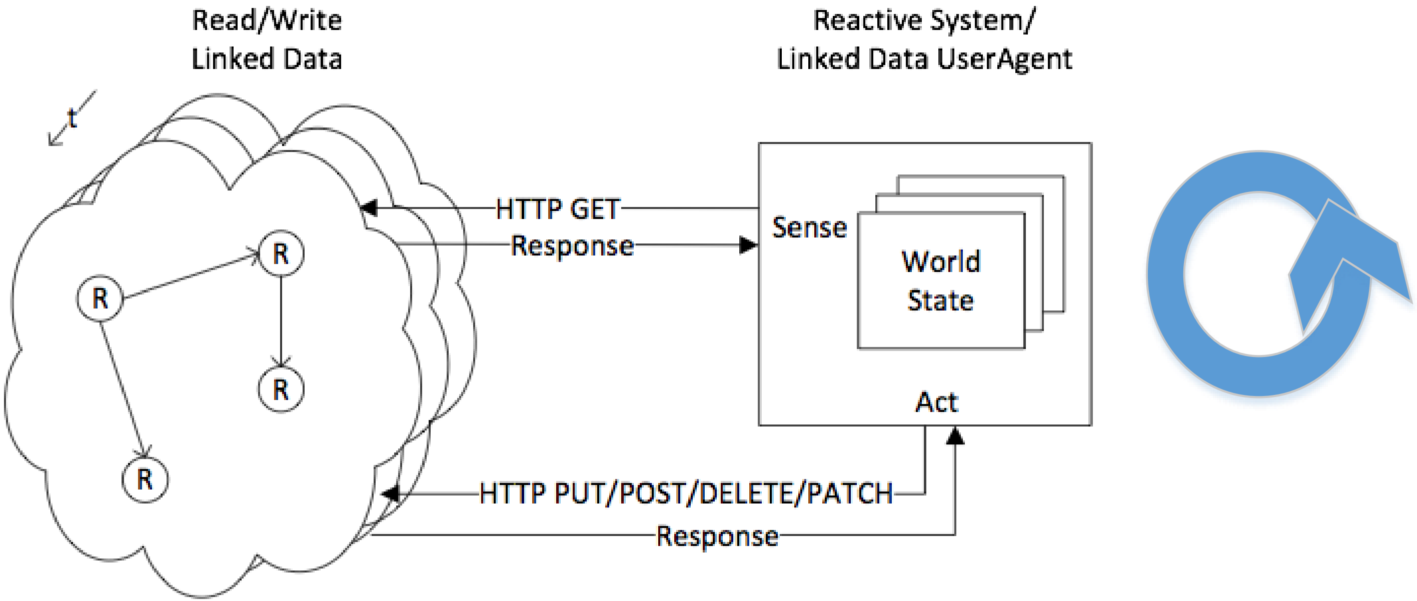
- Visualisation in Google Earth
- Repeated query evaluation (update frequency in minutes to weeks)
- Mediated schema to represent points of interest
- Uniform Linked Data interface to sources (web APIs)

Karl Marx (1818 – 1883)

The ~~philosophers~~ ontologists have only interpreted the world, in various ways. The point, however, is to change it.



Summary of First Half, Outlook to Second Half



Read/Write
Linked Data

Reactive System/
Linked Data UserAgent

t

R

R

R

R

HTTP GET

Response

Sense

World
State

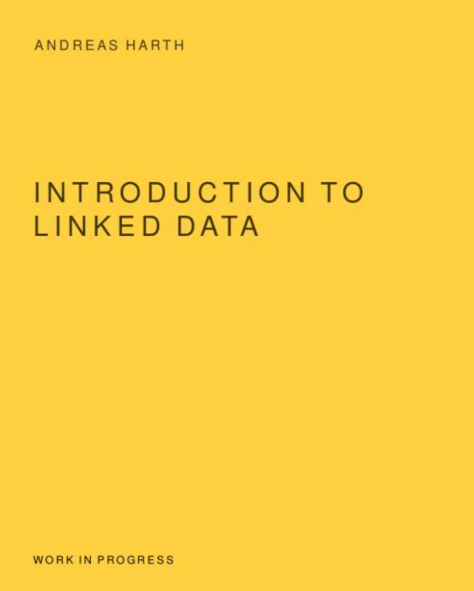
Act

HTTP PUT/POST/DELETE/PATCH

Response

Commercial Break

- Some figures taken from the upcoming book „Introduction to Linked Data“
- Estimated publication in autumn 2017



ANDREAS HARTH

INTRODUCTION TO LINKED DATA

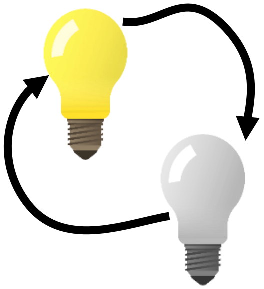
WORK IN PROGRESS

Karlsruhe Institute of Technology

Traversing and Reasoning on Read-Write Linked Data

Rule-based Processing of Dynamic Linked Data – Part II
Andreas Harth and Tobias Käfer
Tutorial at the 14th European Semantic Web Conference (ESWC), Portorož, 2017

INSTITUT FOR APPLIED INFORMATICS AND FORMAL DESCRIPTION METHODS, CHAIR FOR WEB SCIENCE

{ < 0.5}=> 

Why?

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

- User agents for Read-Write Linked Data
  - → Applications that leverage data and functionality available as Linked Data

- Rule-Based specifications of user agents for Read-Write Linked Data
  - Declarative
  - Compatible with rule-based reasoning
  - Allow for model-based analysis
  - Can serve as the formal basis for execution in planning/description-based approaches

- Isn‘t your user agent for Read-Write Linked Data a bit like service orchestration? – Yes

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)

  → Repeated evaluation of queries and rules over Linked Data from the Web

- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - Running Example
  - Theory: Transition Systems and Read-Write Linked Data
  - Approaches
  - Hands-on

  → Rule-based user agents for Read-write Linked Data

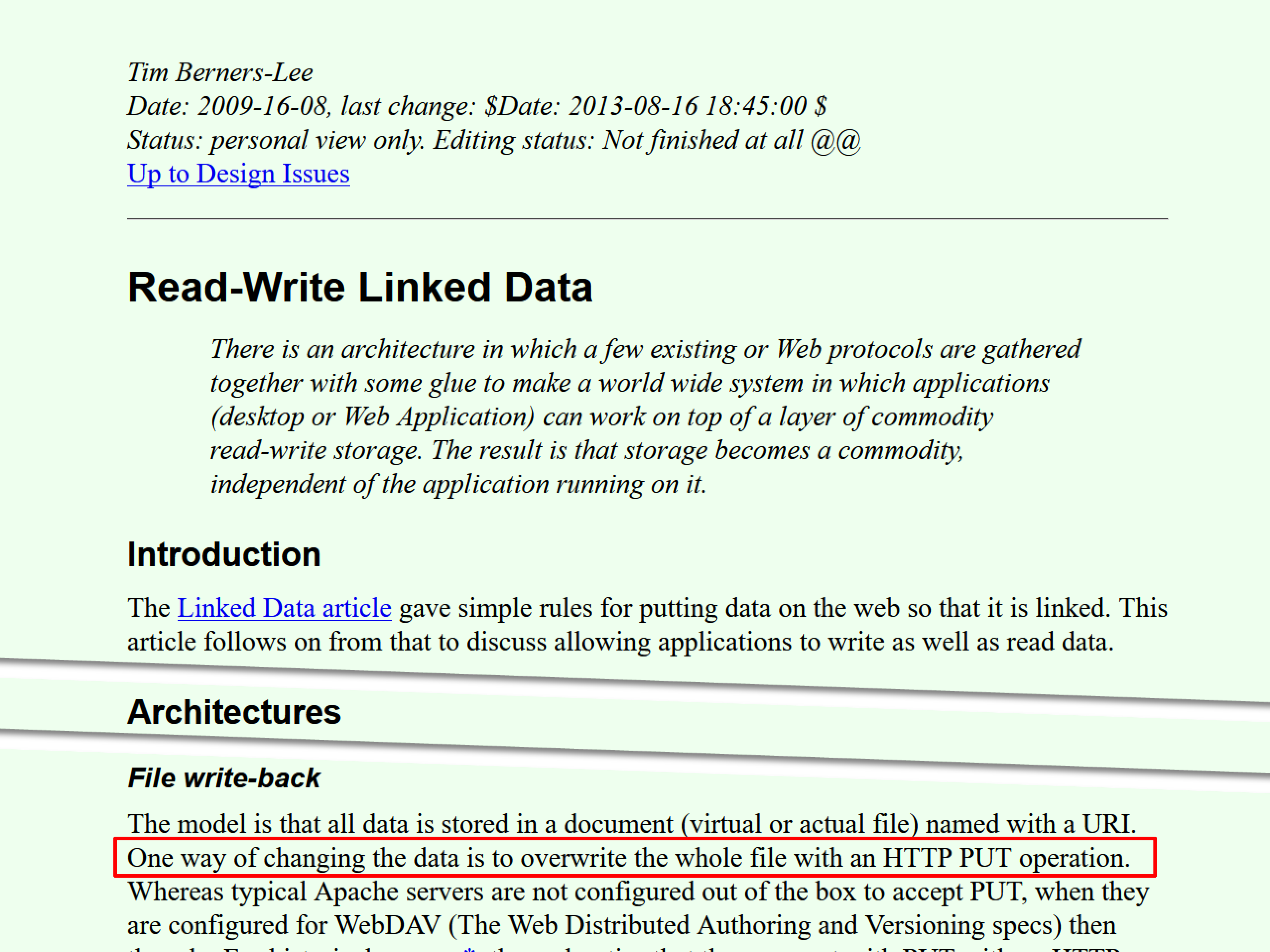
- Conclusion

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)

  → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - **Standards, Recommendations, Practices**
    - Read-Write Linked Data
    - Hypertext Transfer Protocol (HTTP)
    - Linked Data Platform (LDP)
    - REST / Richardson Maturity Model
  - Running Example
  - Theory: Transition Systems and Read-Write Linked Data
  - Approaches
  - Hands-on

  → Rule-based user agents for Read-write Linked Data
- Conclusion



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

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

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]

CRUD Operation	HTTP Method
Read	GET
Update	PUT
Create	POST / PUT
Delete	DELETE

CRUD – HTTP Correspponcence

HTTP Method	Safe?	Idempotent?
GET	✓	✓
PUT		✓
POST		
DELETE		✓

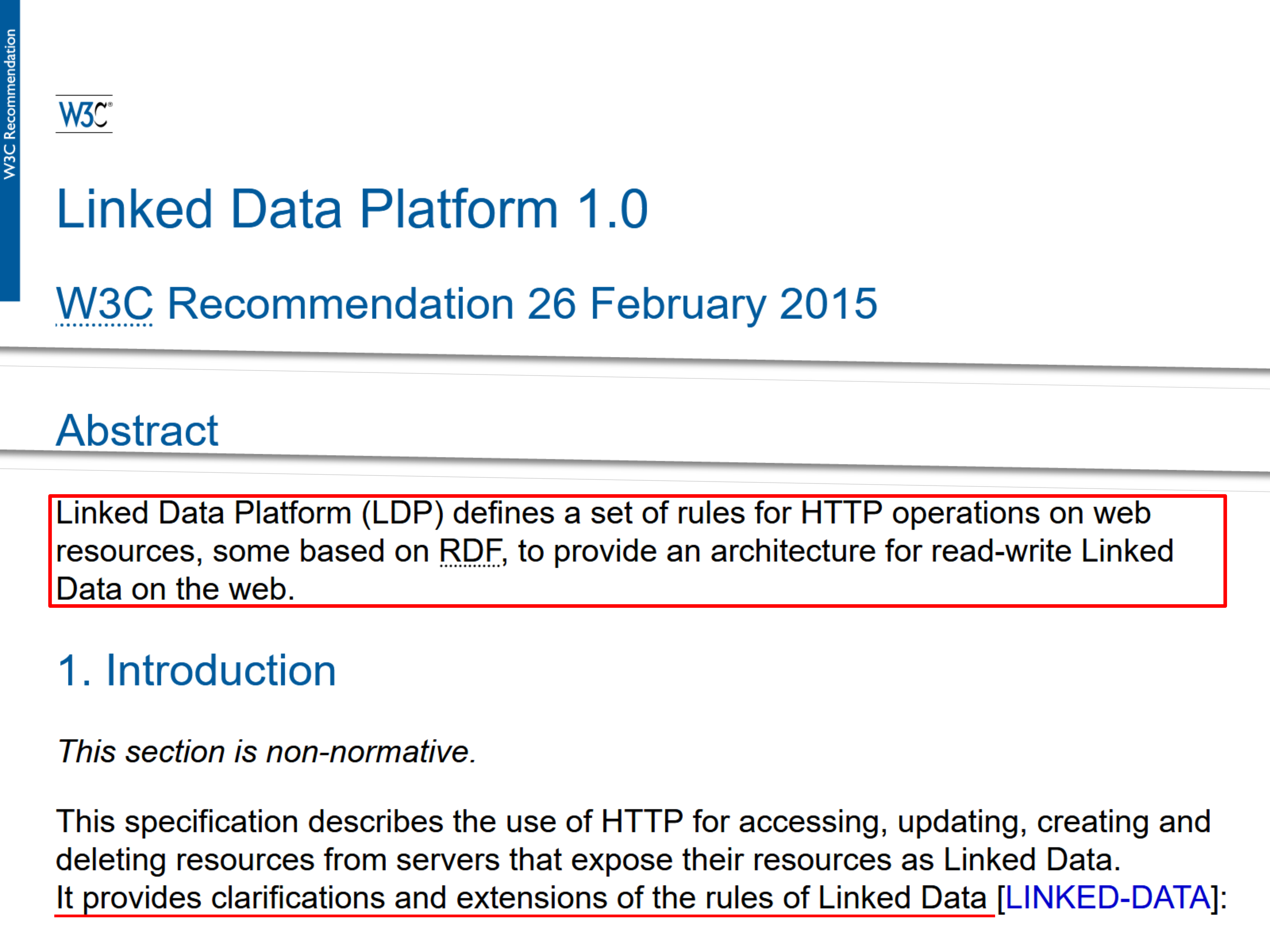
Properties of HTTP Requests

  - POST
    - Append-to-collection vs. RPC
  - OPTIONS
    - Describe communication options
- NB: No events → polling

[1] James Martin: Managing the Data-Base Environment, Pearson (1983)

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

HTTP Request Method	HTTP Request, or Response Code	HTTP Message Semantics: The HTTP Message Body Contains…
GET	Request	Nothing
PUT	Request	State of the resource
POST	Request	Arbitrary data or state of resource
DELETE	Request	Nothing
any	Non-2xx	State of the request
GET	2xx	State of the resource
PUT	2xx	State of the resource or empty
POST	2xx	State of the request (refering to new resource)
DELETE	2xx	State of the request or empty



W3C Recommendation

W3C®

Linked Data Platform 1.0

W3C Recommendation 26 February 2015

Abstract

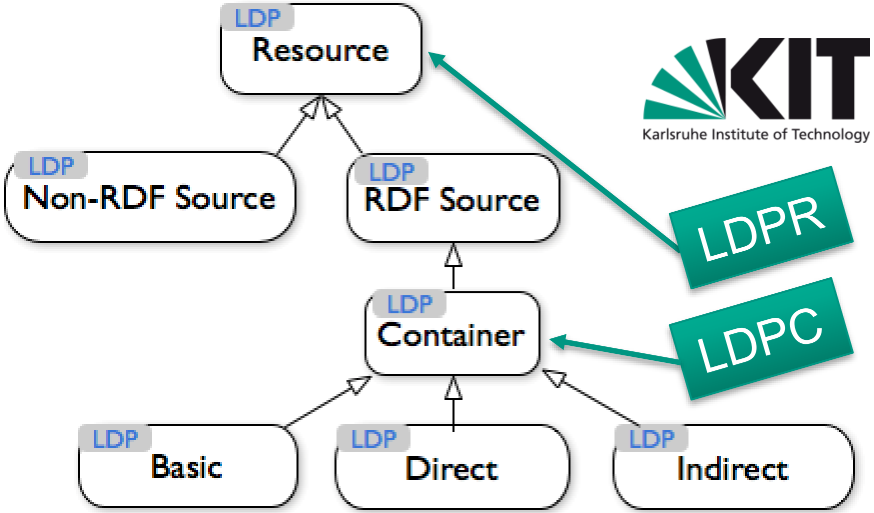
Linked Data Platform (LDP) defines a set of rules for HTTP operations on web resources, some based on RDF, to provide an architecture for read-write Linked Data on the web.

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]:

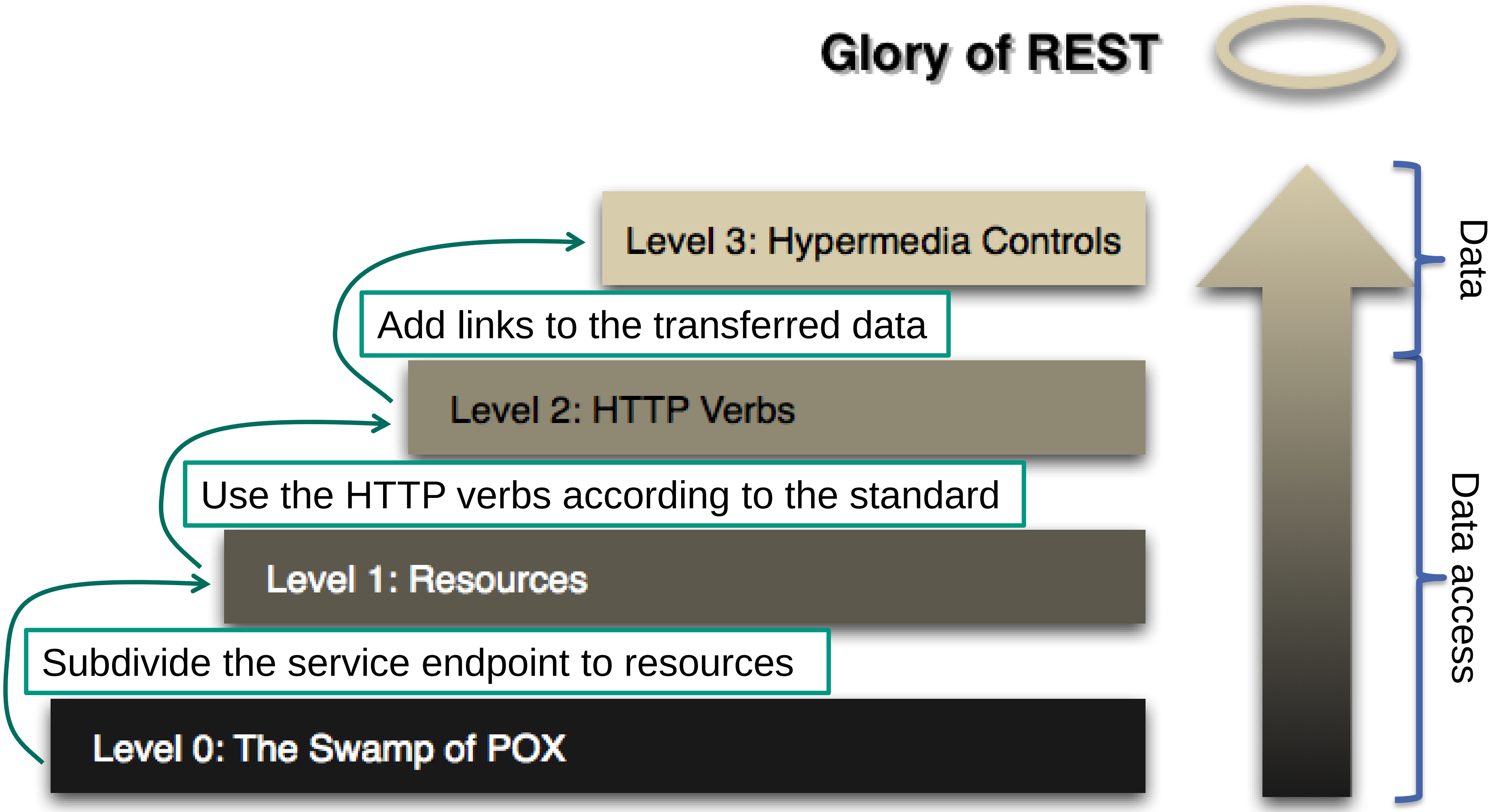
Linked Data Platform [1]



- 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

[1] Speicher, Arwe, Malhotra: "Linked Data Platform 1.0" W3C Recommendation (2015)

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


Glory of REST
Level 3: Hypermedia Controls
Add links to the transferred data
Level 2: HTTP Verbs
Use the HTTP verbs according to the standard
Level 1: Resources
Subdivide the service endpoint to resources
Level 0: The Swamp of POX
Data
Data access

[1] Fielding: “Architectural Styles and the Design of Network-based Software Architectures”. PhD Thesis, UC Irvine, USA (2000)
[2] Fowler: “Richardson Maturity Model” (2010) available from http://martinfowler.com/articles/richardsonMaturityModel.html

Rule-Based User Agents for Read-Write Linked Data Should Respect the Standards + Be Formal

- …should communicate on RMM2 for *data access*
  - …to make use of HTTP semantics
- …should have RDF as *data model*
  - …to make use of RDF data integration techniques
- …should have a formal grounding in its *dynamics model*
  - …to layer higher-level works on top
  - …to make statements on the expressivity
  - …for formal analysis

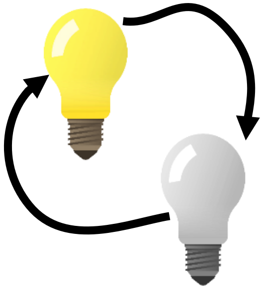
Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)
  → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - **Running Example**
  - Theory: Transition Systems and Read-Write Linked Data
  - Approaches
  - Hands-on

  → Rule-based user agents for Read-write Linked Data
- Conclusion

Running Example

- Local, for being independent from conference connectivity
- A light sensor available as Linked Data
- A relay available as Read-Write Linked Data with a light connected

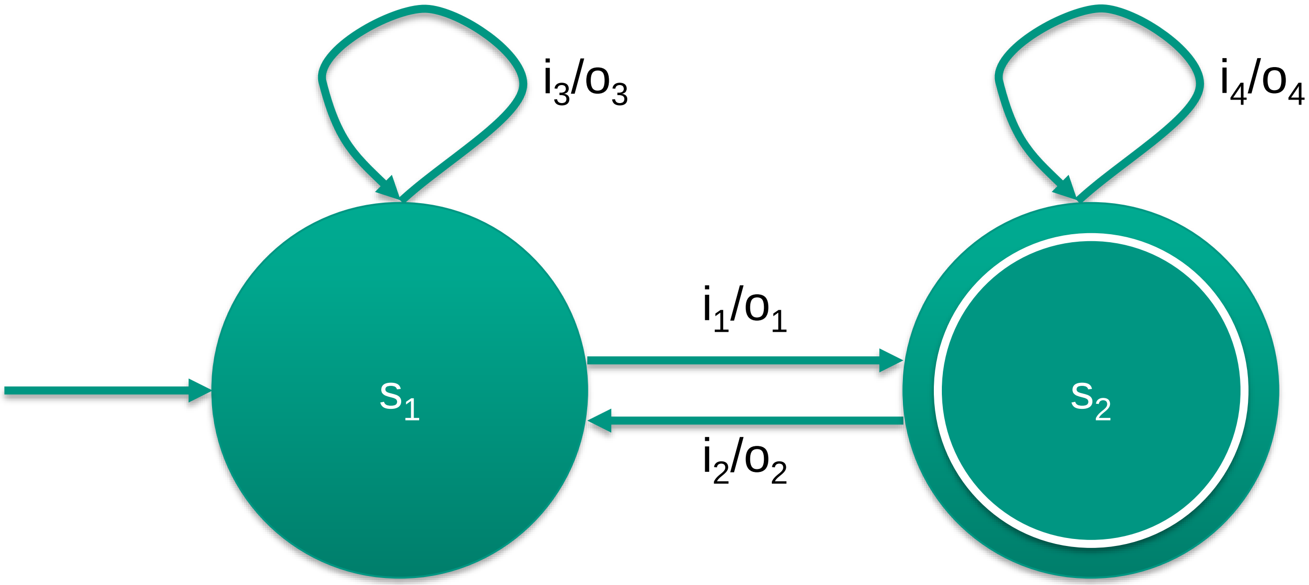
{ < 0.5}=> 

- Turn on the light if the light sensor‘s value drops below a certain threshold
- Excursus: Model the relay with some theory

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)
  - → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - Running Example
  - **Theory: Transition Systems and Read-Write Linked Data**
  - Approaches
  - Hands-on
  - → Rule-based user agents for Read-write Linked Data
- Conclusion

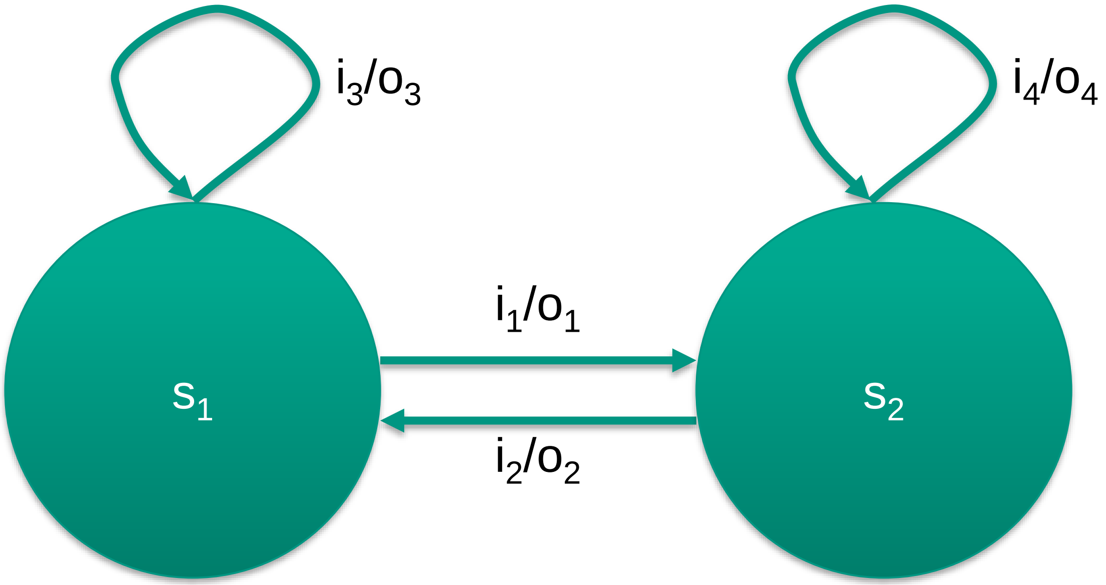
Finite State Machines (Mealy Automata) and Transition Systems



A State Machine

i_n: input
o_n: output

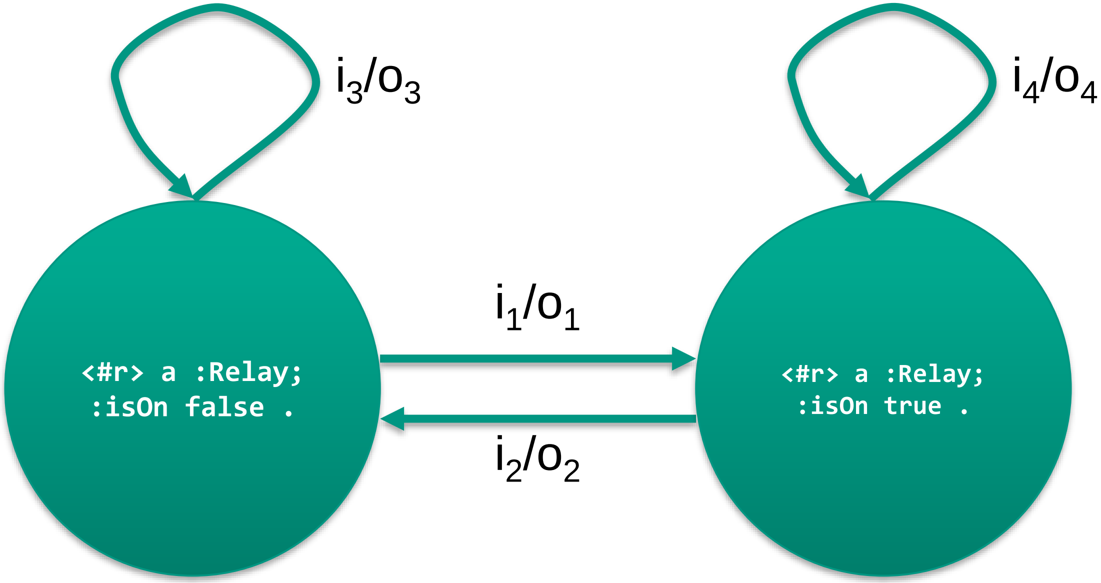
Finite State Machines (Mealy Automata) and Transition Systems



A Transition System

i_n: input
o_n: output

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

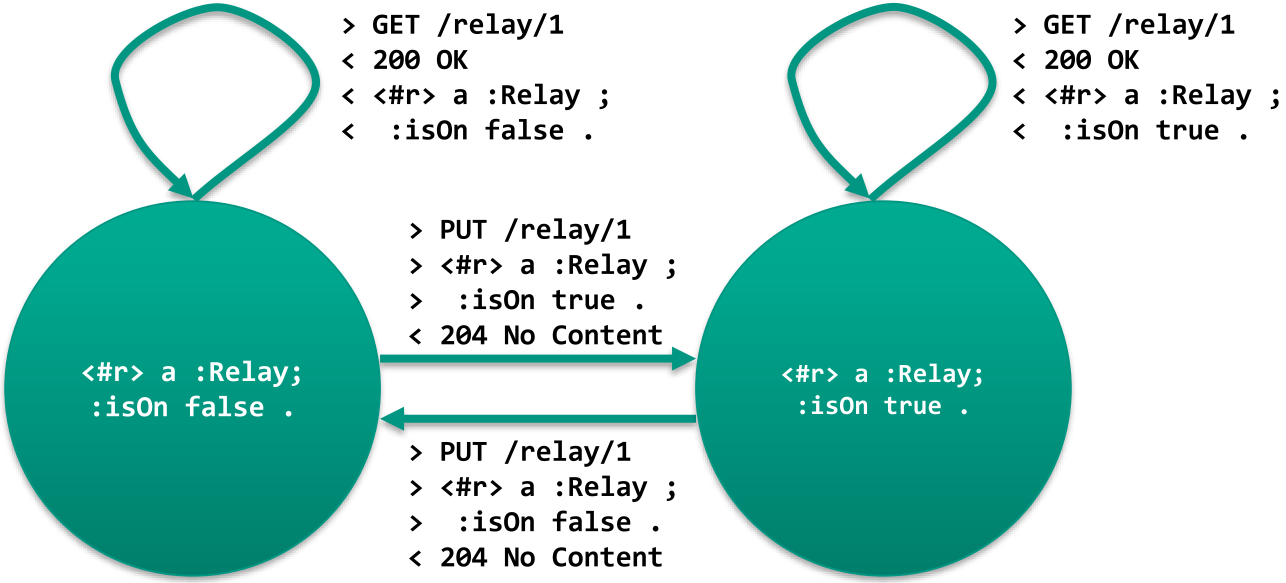


Linked Data Transition System of Resource /relay/1 on server http://t2-ambient-relay.lan/

i_n: input
o_n: output

[1] Harth and Käfer: "Towards Specification and Execution of Linked Systems". Proceedings of the 28th GI-Workshop on Foundations of Database Systems (Grundlagen von Datenbanken, GvD), May 24 - 27, 2016, Nörten-Hardenberg, Germany.

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



```
> GET /relay/1
< 200 OK
< <#r> a :Relay ;
<   :isOn false .
```

```
<#r> a :Relay;
:isOn false .
```

```
> PUT /relay/1
> <#r> a :Relay ;
>   :isOn true .
< 204 No Content
```

```
> GET /relay/1
< 200 OK
< <#r> a :Relay ;
<   :isOn true .
```

```
<#r> a :Relay;
:isOn true .
```

```
> PUT /relay/1
> <#r> a :Relay ;
>   :isOn false .
< 204 No Content
```

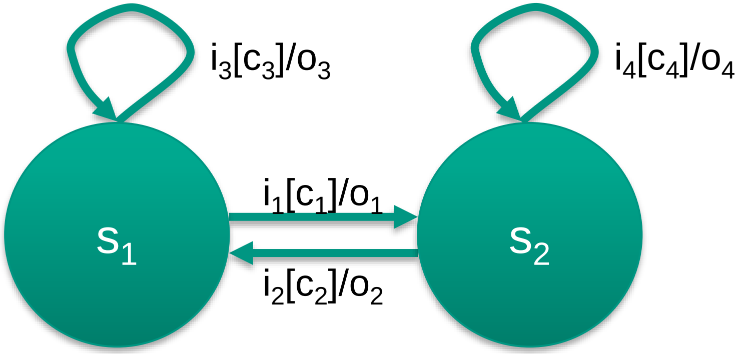
Transition System of Resource /relay/1 on server http://t2-ambient-relay.lan/

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

[1] Harth and Käfer: “Towards Specification and Execution of Linked Systems”. Proceedings of the 28th GI-Workshop on Foundations of Database Systems (Grundlagen von Datenbanken, GvD), May 24 - 27, 2016, Nörten-Hardenberg, Germany.

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
  - $\rightarrow \subset (S \times L \times S)$: Transition Relation



A Transition System

→ 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

Name	RDF Graph
u_1	G_{u1}

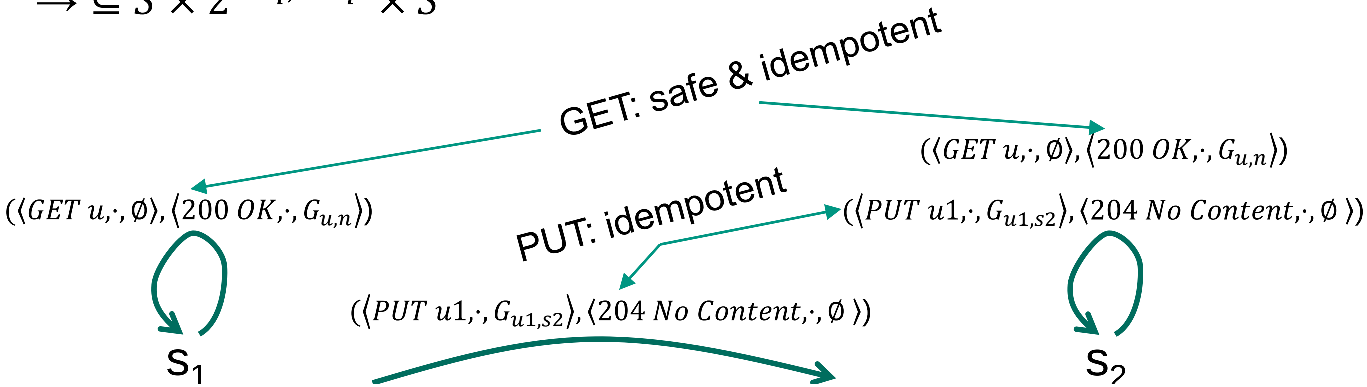
`/relay/1`	`<#r> a :Relay;`
	`  :isOn false .`

An RDF Dataset

[1] Cyganiak, Wood, Lanthaler (eds.): "RDF 1.1 Concepts and Abstract Syntax". W3C Recommendation (2014)
[2] Zimmermann (ed.): "RDF 1.1 On Semantics of RDF Datasets". W3C WG Note (2014)

Linked Data Transition System [1]

- Linked Data Transition System $LDTS \coloneqq (S, \rightarrow)$
  - 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
    - $\rightarrow \subseteq S \times 2^{Req,Resp} \times S$


GET: safe & idempotent
$(\langle GET\ u,\cdot,\emptyset\rangle, \langle 200\ OK,\cdot,G_{u,n}\rangle)$
$(\langle GET\ u,\cdot,\emptyset\rangle, \langle 200\ OK,\cdot,G_{u,n}\rangle)$
PUT: idempotent
$(\langle PUT\ u1,\cdot,G_{u1,s2}\rangle, \langle 204\ No\ Content,\cdot,\emptyset\ \rangle)$
$(\langle PUT\ u1,\cdot,G_{u1,s2}\rangle, \langle 204\ No\ Content,\cdot,\emptyset\ \rangle)$
s_1
s_2

Name	RDF Graph
u_1	$G_{u1,s1}$

`/relay/1`	`<#r> a :Relay;`
	`  :isOn false .`

Name	RDF Graph
u_1	$G_{u1,s2}$

`/relay/1`	`<#r> a :Relay;`
	`  :isOn true .`

[1] Harth and Käfer: “Towards Specification and Execution of Linked Systems”. Proceedings of the 28th GI-Workshop on Foundations of Database Systems (Grundlagen von Datenbanken, GvD), May 24 - 27, 2016, Nörten-Hardenberg, Germany.

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)

  → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - Running Example
  - Theory: Transition Systems and Read-Write Linked Data
  - **Approaches**
    - The Swamp of POX: BPEL
    - Automation on the Web: IFTTT, ARKTIK
    - Service descriptions: OWL-S & Co.
    - Affordances: Hydra, schema.org Potential Actions, Lids
    - RESTdesc
    - Linked Data-Fu
  - Hands-on

  → Rule-based user agents for Read-write Linked Data
- Conclusion

Mapping the Field of Service Composition and Web Agent Specification

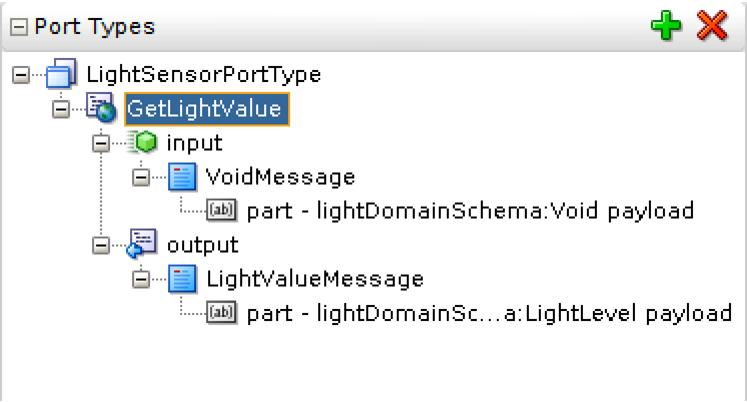
<table>
<tr><th>Level</th><th colspan="13">Foundational approaches / categories</th></tr>
<tr><td>Capability description</td><td colspan="8">Input, Output, Precondition, Effect (for automated composition)</td><td colspan="4">Affordance (for manual composition)</td><td>…</td></tr>
<tr><td>Composition description</td><td>Rules</td><td colspan="2">BPEL *</td><td>Pi calculus</td><td colspan="3">Petri Nets</td><td colspan="3">(Temporal) logic</td><td colspan="2">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Dynamics model</td><td colspan="2">ASM</td><td colspan="3">LTS</td><td colspan="4">Situation Calculus</td><td colspan="3">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Data model</td><td colspan="6">Graph (RDF)</td><td colspan="6">Nested (JSON, XML)</td><td>…</td></tr>
<tr><td>Data access</td><td colspan="3">RMM2</td><td colspan="4">RMM1</td><td colspan="4">RMM0</td><td>push</td><td>…</td></tr>
</table>

Cf. requirements

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

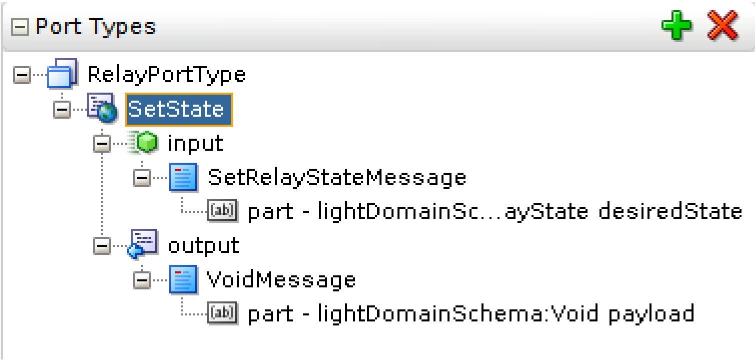
The Swamp of POX: Turn the Lights On in BPEL

- BPEL service orchestration, eg. the output of a planner
- Prerequisites: Descriptions
  - Define XML schemas for exchanged data
  - Define WSDL messages to be exchanged with the relay and sensor services
  - Define the WSDL operations for the relay and the sensor (getter and setter)



Port Types

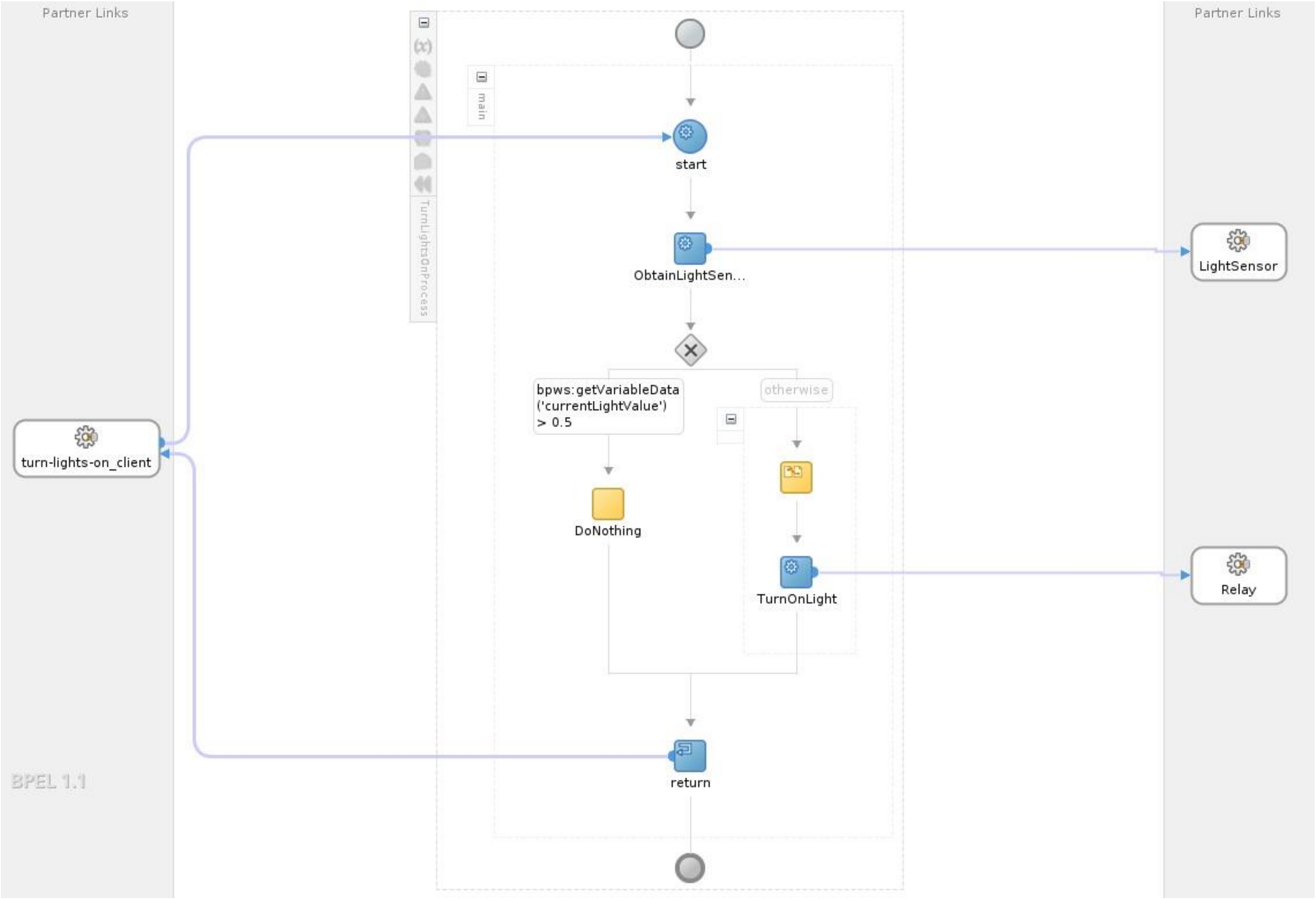
- LightSensorPortType
  - GetLightValue
    - input
      - VoidMessage
        - part - lightDomainSchema:Void payload
    - output
      - LightValueMessage
        - part - lightDomainSc…a:LightLevel payload



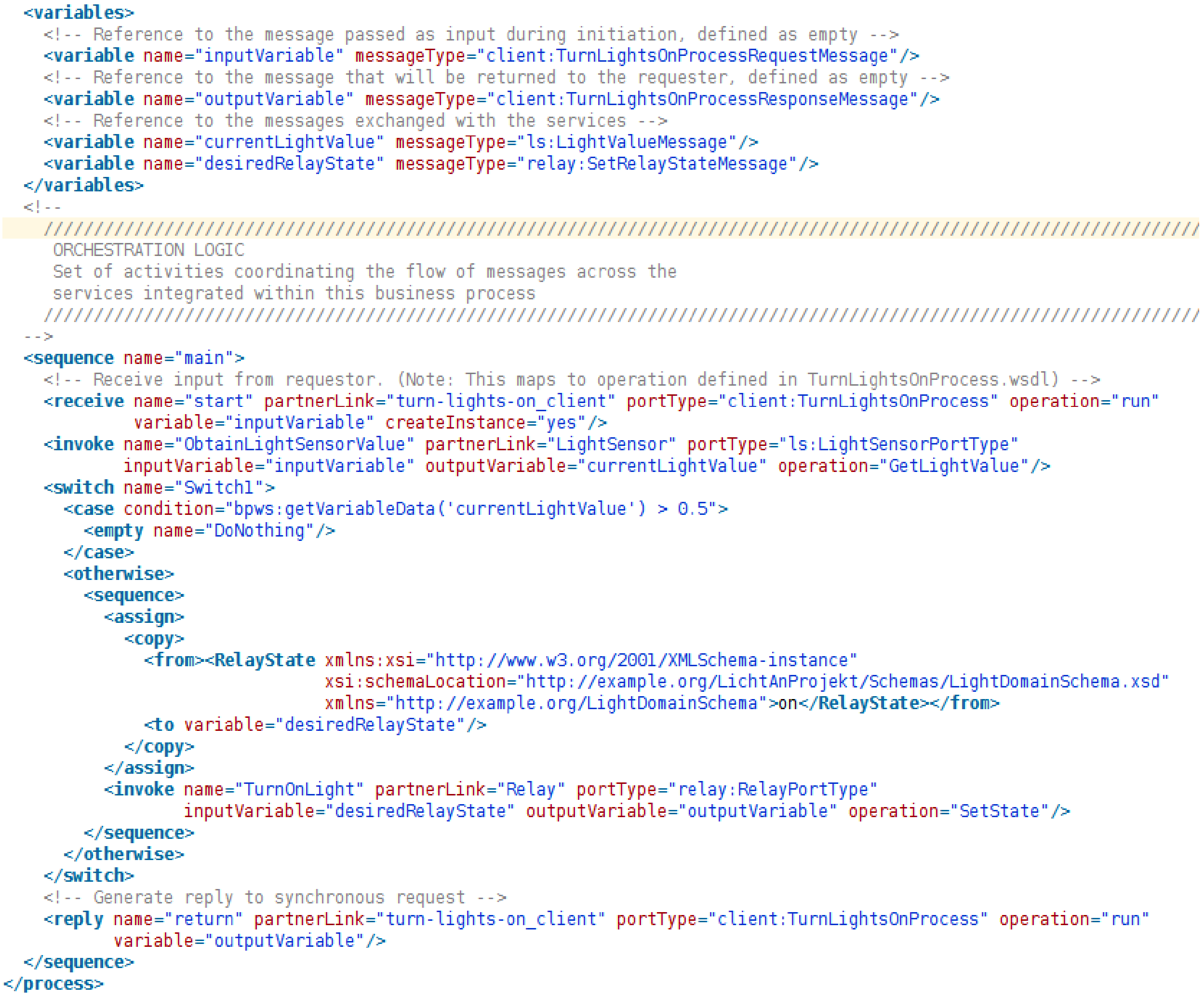
Port Types

- RelayPortType
  - SetState
    - input
      - SetRelayStateMessage
        - part - lightDomainSc…ayState desiredState
    - output
      - VoidMessage
        - part - lightDomainSchema:Void payload

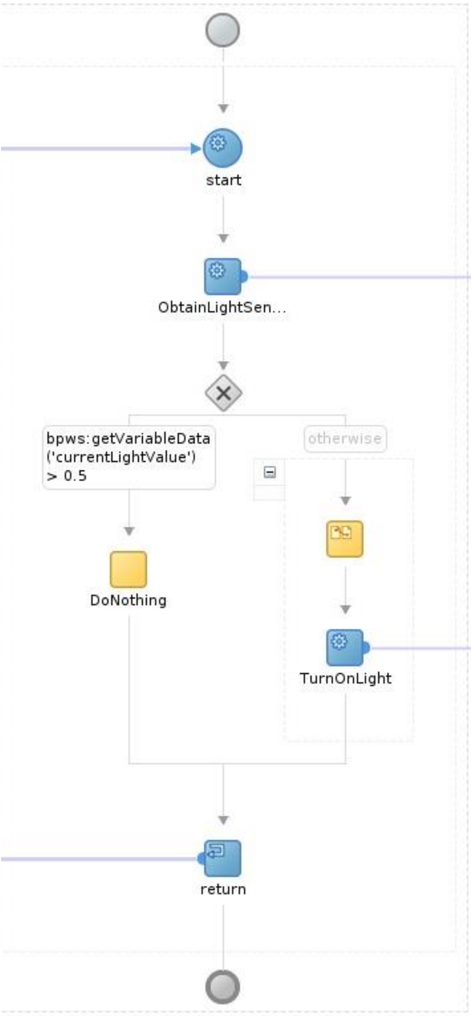
BPEL Orchestration to Turn the Lights On



BPEL Orchestration to Turn the Lights On – the Code (w/o imports etc.)

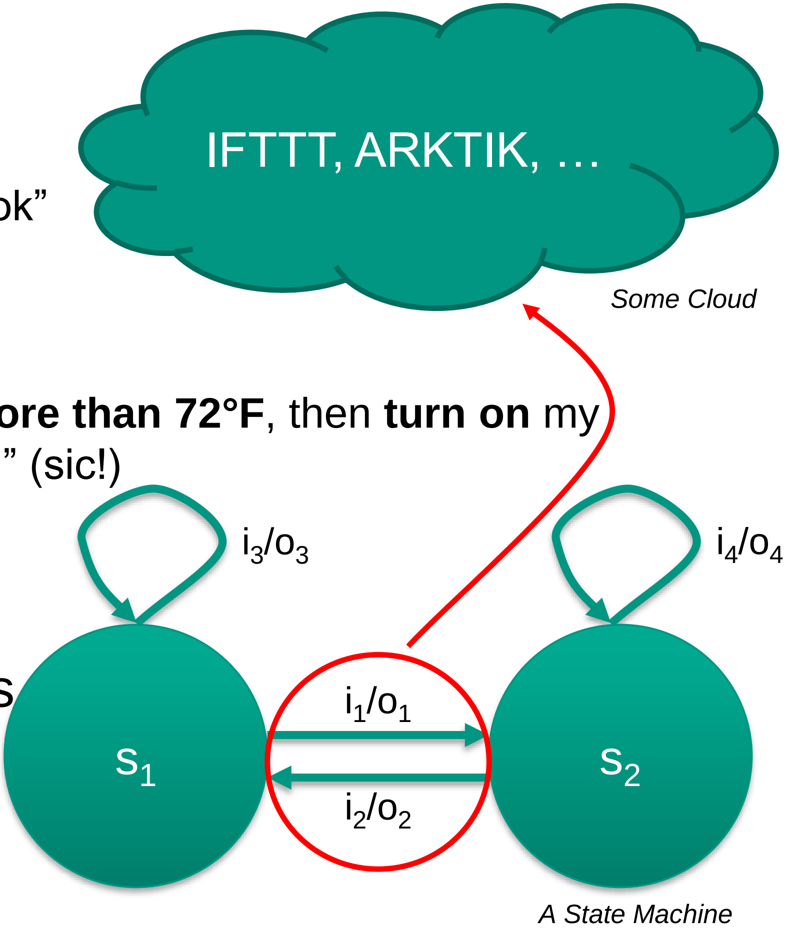


```
  <variables>
    <!-- Reference to the message passed as input during initiation, defined as empty -->
    <variable name="inputVariable" messageType="client:TurnLightsOnProcessRequestMessage"/>
    <!-- Reference to the message that will be returned to the requester, defined as empty -->
    <variable name="outputVariable" messageType="client:TurnLightsOnProcessResponseMessage"/>
    <!-- Reference to the messages exchanged with the services -->
    <variable name="currentLightValue" messageType="ls:LightValueMessage"/>
    <variable name="desiredRelayState" messageType="relay:SetRelayStateMessage"/>
  </variables>
  <!--
    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
     ORCHESTRATION LOGIC
     Set of activities coordinating the flow of messages across the
     services integrated within this business process
    ///////////////////////////////////////////////////////////////////////////////////////////////////////////
  -->
  <sequence name="main">
    <!-- Receive input from requestor. (Note: This maps to operation defined in TurnLightsOnProcess.wsdl) -->
    <receive name="start" partnerLink="turn-lights-on_client" portType="client:TurnLightsOnProcess" operation="run"
             variable="inputVariable" createInstance="yes"/>
    <invoke name="ObtainLightSensorValue" partnerLink="LightSensor" portType="ls:LightSensorPortType"
            inputVariable="inputVariable" outputVariable="currentLightValue" operation="GetLightValue"/>
    <switch name="Switch1">
      <case condition="bpws:getVariableData('currentLightValue') > 0.5">
        <empty name="DoNothing"/>
      </case>
      <otherwise>
        <sequence>
          <assign>
            <copy>
              <from><RelayState xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                                xsi:schemaLocation="http://example.org/LichtAnProjekt/Schemas/LightDomainSchema.xsd"
                                xmlns="http://example.org/LightDomainSchema">on</RelayState></from>
              <to variable="desiredRelayState"/>
            </copy>
          </assign>
          <invoke name="TurnOnLight" partnerLink="Relay" portType="relay:RelayPortType"
                  inputVariable="desiredRelayState" outputVariable="outputVariable" operation="SetState"/>
        </sequence>
      </otherwise>
    </switch>
    <!-- Generate reply to synchronous request -->
    <reply name="return" partnerLink="turn-lights-on_client" portType="client:TurnLightsOnProcess" operation="run"
           variable="outputVariable"/>
  </sequence>
</process>
```



IFTTT [1] and ARKTIK [2] (&Co.)

- 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



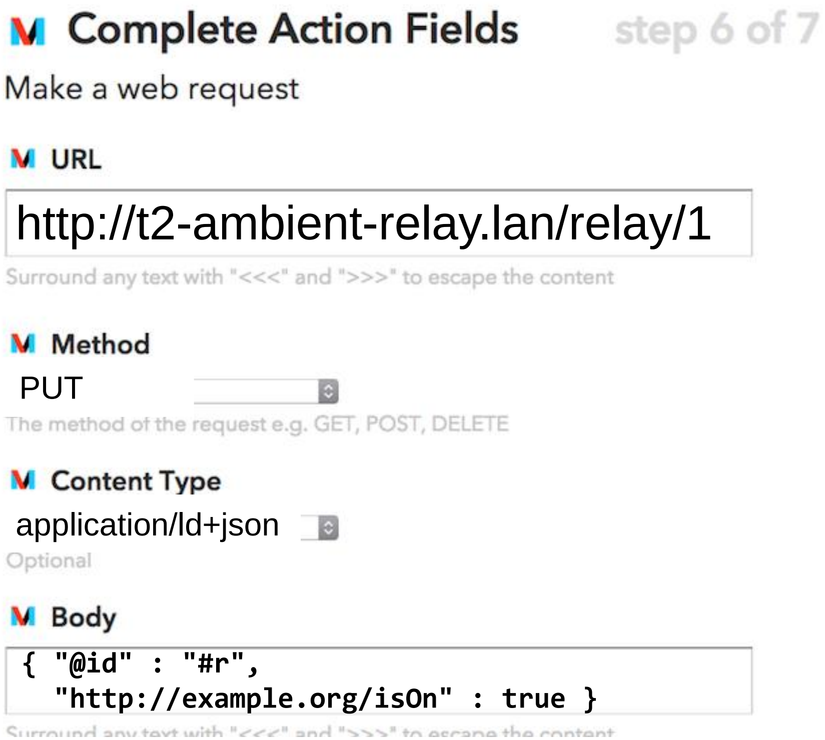
[1] https://ifttt.com/maker_webhooks
[2] https://developer.artik.cloud/documentation/data-management/develop-rules-for-devices.html

Turn on the Light using IFTTT Maker Channel

Maker event "light_state_change"

- 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):

    ```
    { "value1“ : "test",
      "value2“ : 0.5 ,
      "value3“ : True }
    ```



Complete Action Fields

step 6 of 7

Make a web request

URL

http://t2-ambient-relay.lan/relay/1

Surround any text with "<<<" and ">>>" to escape the content

Method

PUT

The method of the request e.g. GET, POST, DELETE

Content Type

application/ld+json

Optional

Body

```
{ "@id" : "#r",
  "http://example.org/isOn" : true }
```

Surround any text with "<<<" and ">>>" to escape the content

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
    }
   }
  } ]
}
```

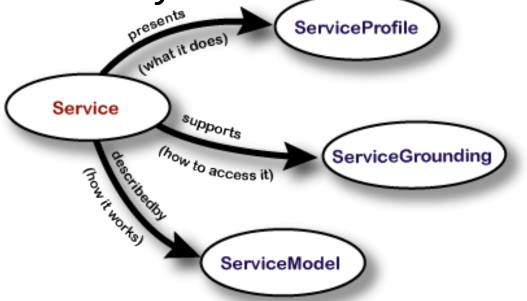
Classifying IFTTT and ARKTIK (&Co.)

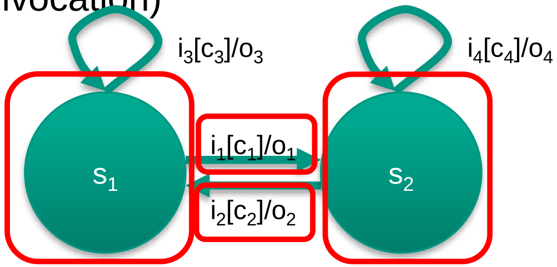
Level	Foundational approaches / categories						
Capability description	Input, Output, Precondition, Effect (for automated composition)	Affordance (for manual composition)					…
Composition description	Rules	BPEL *	Pi calculus	Petri Nets	(Temporal) logic	Unformalised Implementation	…
Dynamics model	ASM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

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

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]





A Transition System

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

[1] Baryannis and Plexousakis: “Automated Web Service Composition: State of the Art and Research Challenges”. ICS-FORTH/TR-409 (2010)

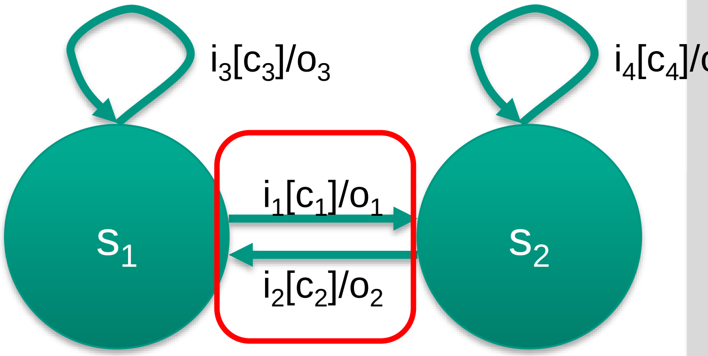
Classifying OWL-S and WSMO

<table>
<tr><th>Level</th><th colspan="13">Foundational approaches / categories</th></tr>
<tr><td>Capability description</td><td colspan="8">Input, Output, Precondition, Effect (for automated composition)</td><td colspan="4">Affordance (for manual composition)</td><td>…</td></tr>
<tr><td>Composition description</td><td>Rules</td><td colspan="2">BPEL *</td><td>Pi calculus</td><td colspan="3">Petri Nets</td><td colspan="3">(Temporal) logic</td><td colspan="2">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Dynamics model</td><td colspan="2">ASM</td><td colspan="3">LTS</td><td colspan="4">Situation Calculus</td><td colspan="3">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Data model</td><td colspan="6">Graph (RDF)</td><td colspan="6">Nested (JSON, XML)</td><td>…</td></tr>
<tr><td>Data access</td><td colspan="3">RMM2</td><td colspan="4">RMM1</td><td colspan="4">RMM0</td><td>push</td><td>…</td></tr>
</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

Level	Foundational approaches / categories					
Capability description	Input, Output, Precondition, Effect (for automated composition)	Affordance (for manual composition)				…
Composition description	Rules	BPEL *	Pi cal-culus	Petri Nets	(Temporal) logic	Unformalised Implementation	…
Dynamics model	ASM	LTS	Situation Calculus	Unformalised Implementation		…
Data model	Graph (RDF)	Nested (JSON, XML)				…
Data access	RMM2	RMM1	RMM0	push		…

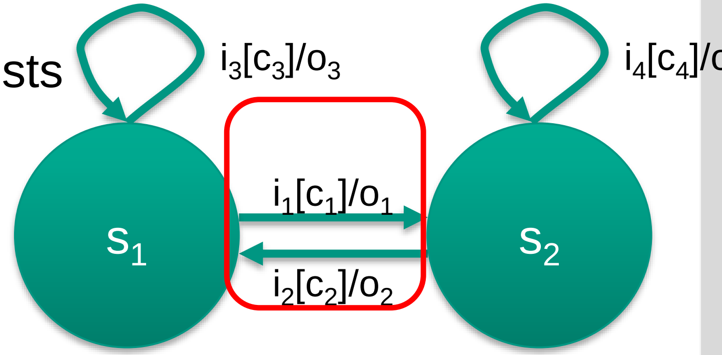


A Transition System

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

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)

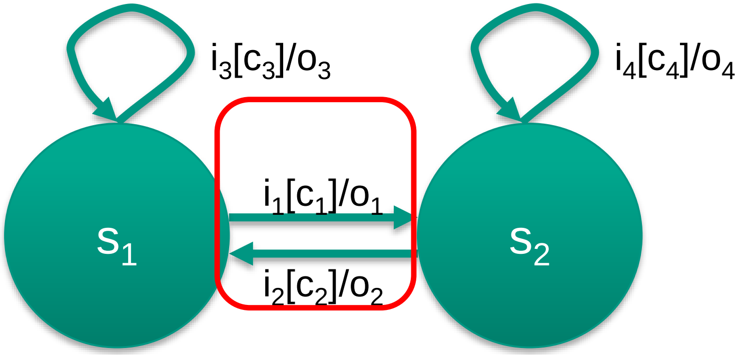


A Transition System

http://www.hydra-cg.com/

schema.org Potential Actions and WoT Thing Descriptions

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



A Transition System

- WoT Thing Descriptions
  - Defines (for a thing):
    - Actions
    - Properties
    - …
  - 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/

LIDS [1]

- 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

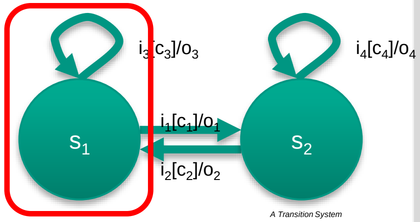
```
CONSTRUCT { ?point foaf:based_near ?feature }
FROM      <http://geowrap.openlids.org/findNearbyWikipedia>
WHERE     { ?point geo:lat ?lat . ?point geo:long ?lng }
```

  - We query for the WHERE clause in the data we already have

```
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

```
> 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 .
```



A Transition System

[1] Speiser, Harth: “Integrating Linked Data and Services with Linked Data Services”. Proc.8th ESWC (2011)

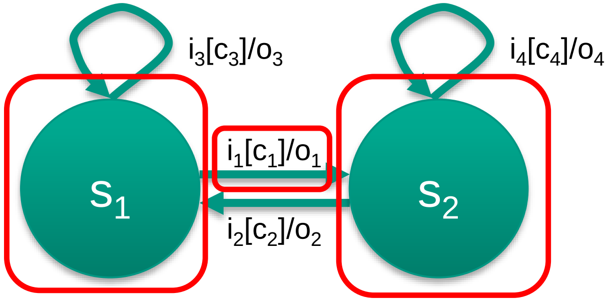
RESTdesc [1]

- Aim: Automated service composition and composition execution in the presence of hyperlinks in HTTP responses
- Composition problem:
  - Initial knowledge

    `<#r> :isOn false .`
  - API descriptions:

    `{` *preconditions* `} => {` *HTTP-request* `.` *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 } => {<#r> :isOn true } .`
  - Background knowledge, eg. ontologies



A Transition System

```
@prefix : <http://example.org/>.
@prefix http: <http://www.w3.org/2011/http#>.

{
  <#r> :isOn false .
}
=>
{
  _:request http:methodName "PUT";
            http:requestURI /relay/1 ;
            http:body "<#r> :isOn true ."
            http:resp [ http:body ?b1 ].
  <#r> :isOn true .
}.
```

[1] Verborgh, Steiner, Van Deursen, Coppens, Vallés, Van de Walle: “Functional descriptions as the bridge between hypermedia APIs and the Semantic Web”. In Proc. 3rd International Workshop on RESTful Design (WS-REST) (2012)

RESTdesc Algorithm [1]

Proof before an HTTP request

Set of API descriptions in rules

Goal

World/Application state

Background knowledge

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_{pre}
2) Check n_{pre}:
   a) If $n_{pre} = 0$, 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_{post}:
   a) If the reasoner was not able to generate a proof, set $n_{post} \coloneqq n_{pre}$.
   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_{post}.
7) Compare n_{post} with n_{pre}:
   a) If $n_{post} \geq n_{pre}$, go back to 1) with the new API composition problem $(R \setminus \{r\}, g, H, B)$.
   b) If $n_{post} < n_{pre}$, the post-proof can be used as the next pre-proof. Set $n_{pre} \coloneqq n_{post}$ and continue with 2)

[1] Verborgh, Arndt, Van Hoecke, De Roo, Mels, Steiner, Gabarró: “The pragmatic proof: Hypermedia API composition and execution“. *Theory and Practice of Logic Programming*, *17*(1), 1-48. (2017)

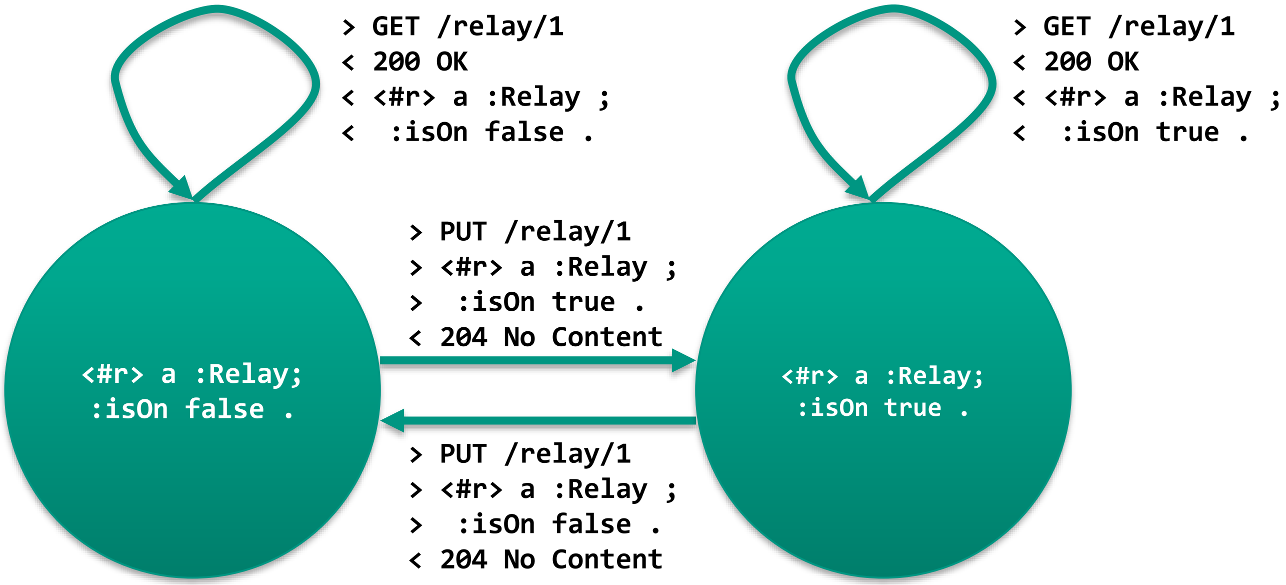
Classifying RESTdesc

More than mere affordances, but not full IOPE

Level	Foundational approaches / categories						
Capability description	Input, Output, Precondition, Effect (for automated composition)	Affordance (for manual composition)	…				
Composition description	Rules	BPEL *	Pi cal-culus	Petri Nets	(Temporal) logic	Unformalised Implementation	…
Dynamics model	ASM	LTS	Situation Calculus	Unformalised Implementation	…		
Data model	Graph (RDF)	Nested (JSON, XML)	…				
Data access	RMM2	RMM1	RMM0	push	…		

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

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



```
> GET /relay/1
< 200 OK
< <#r> a :Relay ;
<   :isOn false .
```

```
> GET /relay/1
< 200 OK
< <#r> a :Relay ;
<   :isOn true .
```

```
<#r> a :Relay;
:isOn false .
```

```
> PUT /relay/1
> <#r> a :Relay ;
>   :isOn true .
< 204 No Content
```

```
<#r> a :Relay;
:isOn true .
```

```
> PUT /relay/1
> <#r> a :Relay ;
>   :isOn false .
< 204 No Content
```

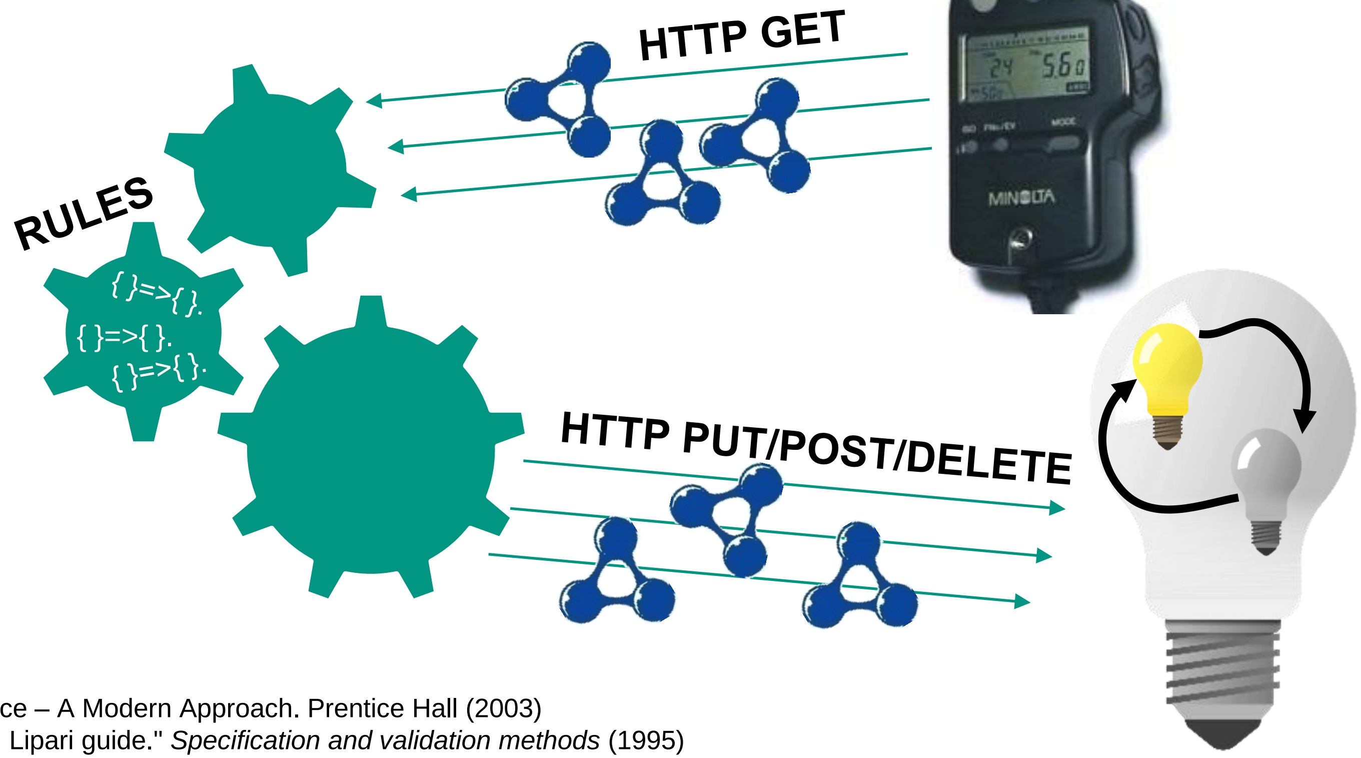
Transition System of Resource /relay/1

[1] Harth and Käfer: “Towards Specification and Execution of Linked Systems”. Proc. 28th GI-Workshop on Foundations of Database Systems (GvD) (2016)

Linked Data-Fu -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] and Abstract State Machines [2]
- In a nutshell:

```
while(true):
    sense()
    think()
    act()
```



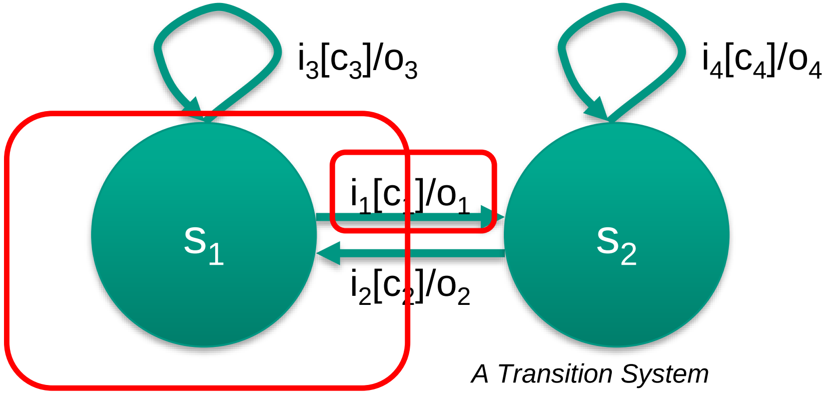
[1] Russell & Norvig: Artificial Intelligence – A Modern Approach. Prentice Hall (2003)
[2] Gurevich:. "Evolving algebras 1993: Lipari guide." *Specification and validation methods* (1995)

Programming Clients in Linked Data-Fu

- Background knowledge: RDF triples + N3 production rules (eg. for entailment and link traversal, see part I of this tutorial)

- SENSE
  - State how the world state represented in RDF is to be downloaded using HTTP requests
- THINK
  - Specify conditions on the world state in BGPs…
- ACT
  - …for actions (again, HTTP requests)

- …and REPEAT



A Transition System

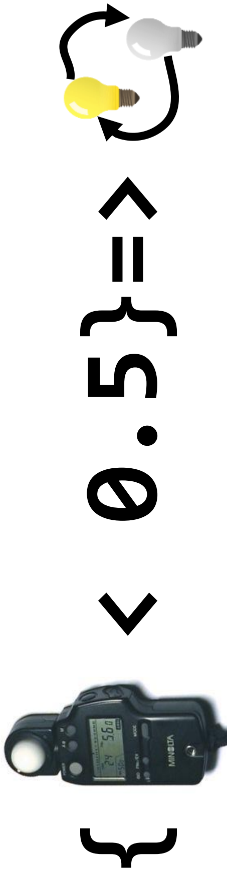
Looped Linked Data-Fu

<table>
<tr><th>Level</th><th colspan="13">Foundational approaches / categories</th></tr>
<tr><td>Capability description</td><td colspan="8">Input, Output, Precondition, Effect (for automated composition)</td><td colspan="4">Affordance (for manual composition)</td><td>…</td></tr>
<tr><td>Composition description</td><td>Rules</td><td colspan="2">BPEL *</td><td>Pi cal-culus</td><td colspan="3">Petri Nets</td><td colspan="3">(Temporal) logic</td><td colspan="2">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Dynamics model</td><td colspan="2">ASM</td><td colspan="3">LTS</td><td colspan="4">Situation Calculus</td><td colspan="3">Unformalised Implementation</td><td>…</td></tr>
<tr><td>Data model</td><td colspan="6">Graph (RDF)</td><td colspan="6">Nested (JSON, XML)</td><td>…</td></tr>
<tr><td>Data access</td><td colspan="3">RMM2</td><td colspan="4">RMM1</td><td colspan="4">RMM0</td><td>push</td><td>…</td></tr>
</table>

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

Turn the Light On in Linked Data-Fu

- Loop



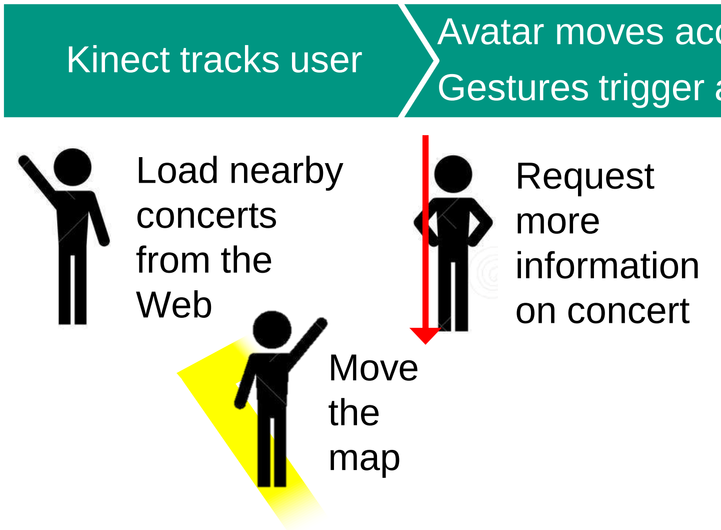
```
{ [] a http:Request ;
     http:hasMethod httpM:GET ;
     http:requestURI </ambient/light> . }
{ [] a http:Request ;
     http:hasMethod httpM:GET ;
     http:requestURI </relay/1> . }

{ </ambient/light> rdf:value ?val .
  ?val math:lessThan 0.5 .
  </relay/1#r> :isOn false . }
=>
{ [] a http:Request ;
     http:hasMethod httpM:PUT ;
     http:requestURI </relay/1> ;
     http:body
       { </relay/1#r> :isOn true . } . } .
```

SENSE:
Retrieve the
world state

THINK:
Conditionally…

ACT:
…manipulate
the world state

Linked Data-Fu Example: Distributed VR System Composition [1]



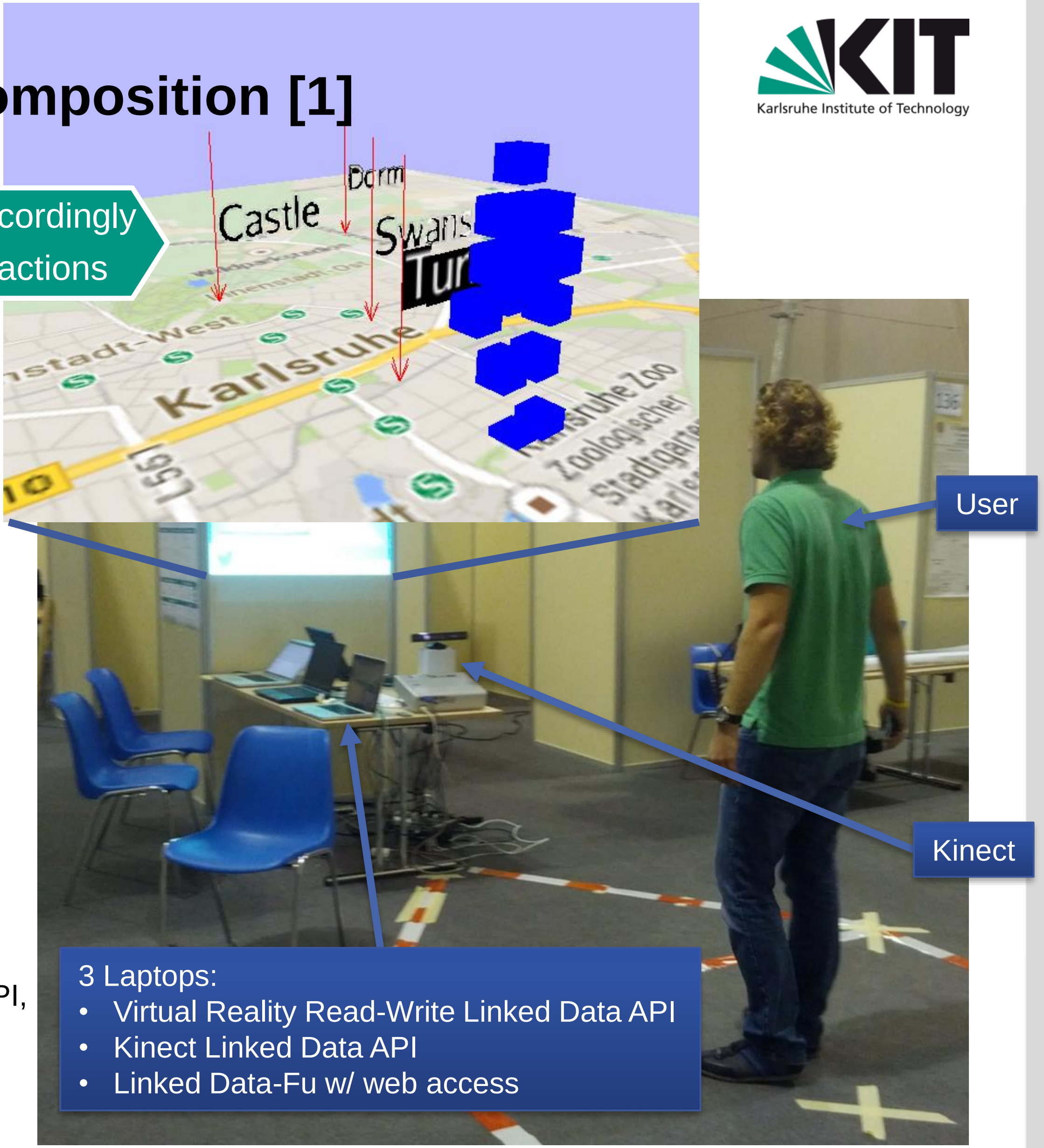
Kinect tracks user

Avatar moves accordingly
Gestures trigger actions

Load nearby concerts from the Web

Request more information on concert

Move the map



3 Laptops:

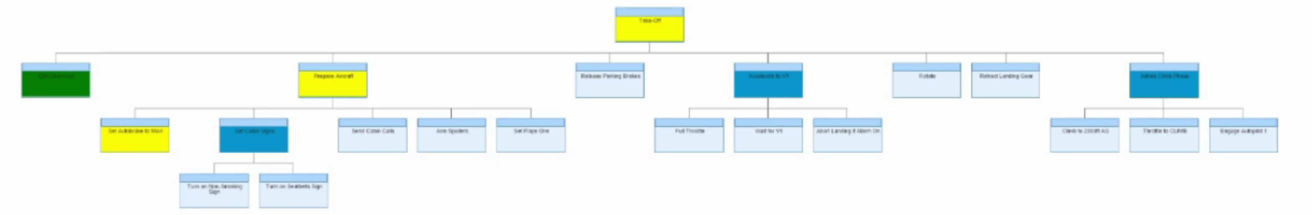
- Virtual Reality Read-Write Linked Data API
- Kinect Linked Data API
- Linked Data-Fu w/ web access

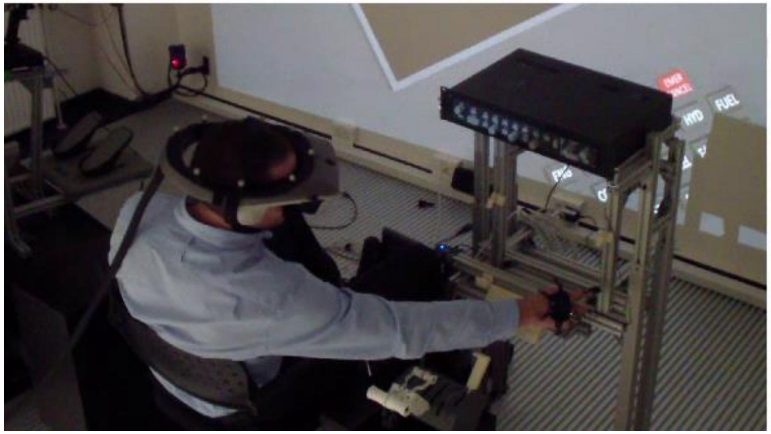
- We encoded in Linked Data-Fu rules:
  - Movement of the avatar according to Kinect data
  - Detection of user gestures
  - Movement of the map according to gestures
  - Loading of concert data from the web
  - Data integration between VR RWLD API, concert LD API, Kinect LD API
- Execution at Kinect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

[1] Keppmann, Käfer, Stadtmüller, Schubotz, Harth: "High Performance Linked Data Processing for Virtual Reality Environments". P&D ISWC 2014.

Linked Data-Fu 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/

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)
  → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - Running Example
  - Theory: Transition Systems and Read-Write Linked Data
  - Approaches
    - The Swamp of POX: BPEL
    - Automation on the Web: IFTTT, ARKTIK
    - Service descriptions: OWL-S & Co.
    - Affordances: Hydra, schema.org Potential Actions, Lids
    - RESTdesc
    - Linked Data-Fu
  - **Hands-on**

  → Rule-based user agents for Read-write Linked Data
- Conclusion

Read-Write Linked Data Hands-On

- We split you up in groups
- Each group gets an Tessel2 IoT board that
  - Has LEDs
  - Has a web server that serves and accepts Read-Write Linked Data exposing the data and functionality from the board
  - Is connected to the tutorial‘s WiFi
- There is another IoT board on the presenter‘s desk with the same configuration but
  - Has a RFID sensor
- Exercises:
  - Write a Read-Write Linked Data user agent in Linked Data-Fu that…
    1. Makes a LED on your team‘s board blink from your laptop also connected to the tutorial‘s WiFi
    2. Make the same LED blink only if there is a card detected by the RFID sensor

Solution to Exercise 2: Have a LED on a Tessel2 Blink if There is a RFID Card on Another Tessel2

```
@prefix ex: <http://example.org/> .
@prefix http: <http://www.w3.org/2011/http#>.
@prefix http_m: <http://www.w3.org/2011/http-methods#>.
```

SENSE:

```
{ [] http:mthd http_m:GET ; http:requestURI <http://t2-team-1/led/2> . }
{ [] http:mthd http_m:GET ; http:requestURI <http://t2-rfid/> . }
```

THINK:

```
{  <http://t2-rfid/#rfidSensor> ex:senses ?card .
   <http://t2-team-1/led/2#led> ex:isOn "false"^^xsd:boolean . }
=>
```

ACT:

```
{ [] http:mthd http_m:PUT ; http:requestURI <http://t2-team-1/led/2> ;
     http:body
       { <http://t2-team-1/led/2#led> ex:isOn "true"^^xsd:boolean . } . } .
```

THINK:

```
{  <http://t2-rfid/#rfidSensor> ex:senses ?card .
   <http://t2-team-1/led/2#led> ex:isSwitchedOn "true"^^xsd:boolean . }
=>
```

ACT:

```
{ [] http:mthd http_m:PUT ; http:requestURI <http://t2-team-1/led/2> ;
     http:body
      { <http://t2-team-1/led/2#led> ex:isOn "false"^^xsd:boolean . } . } .
```

Agenda

- Part I: Traversing and Reasoning on Linked Data (Andreas)
  - → Repeated evaluation of queries and rules over Linked Data from the Web
- Part II: Traversing and Reasoning on Read-Write Linked Data (Tobias)
  - Standards, Recommendations, Practices
  - Running Example
  - Theory: Transition Systems and Read-Write Linked Data
  - Approaches
  - Hands-on
  - → Rule-based user agents for Read-write Linked Data
- **Conclusion**

Summary, Conclusion and Future Work

- Summary of Part II
  - Presented Read-Write Linked Data and relevant standards
  - Formalised Linked Data as Transition System
  - Classified and discussed approaches from (Rule-Based) (Semantic) Web data and API interaction
  - Hands-on with simple reflex agents
- Conclusion
  - Linked Data is increasingly writeable
  - The Web architecture brings unique challenges to building systems, which contradict assumptions of traditional information systems. Here, we worked without events, the open-world assumption is another issue
- Current / future work
  - Provide formal basis for user agents using ASMs
  - Workflow languages for higher-level programming

Acknowledgements

- This tutorial is partially supported by the German federal ministry of education and research (BMBF) in AFAP, a Software Campus project (FKZ 01IS12051).