Orchestration of Network-accessible Components with Condition-Action Rules
Andreas Harth
Joint work with Tobias Käfer
Motivation: System Interoperation

- Many scenarios require the combination of data and functionality from different components
- Integration architectures include wrappers (aka administration shells, lifting/lowering) to provide access to components via common interface
- Applications are built in programming language such as Python or JavaScript
- Standardising interfaces to components reduces cost
- Specifying applications in a high-level abstraction increases flexiblity

http://ivision-project.eu/  http://arvida.de/
Many Choices…

- Network protocol: resource-based vs. event-based vs. publish-subscribe
- Access capabilities: get/set state vs. query
- Interface descriptions: none vs. input/output/precondition/effect
- Knowledge representation: XML/JSON vs. OWL 2 profiles
- Agent architecture: simple reflex agents vs. goal-based learning agents
Uniform Interfaces (Web Architecture)

1990

- Browsers: dumb, PC, Mac, X, NeXT
- Addressing scheme + Common protocol + Format negotiation
- Servers/Gateways: HTTP, FTP, Gopher, NNTP, Internet News

[Berners-Lee 1996]

2015

“Things” as virtual objects acting as proxies for physical and abstract entities
metadata, events, properties, actions
(over a variety of protocols including HTTP)

- Web of Things Server
- IoT Platform A
- IoT Platform B
- IoT Platform C

[Raggett 2015]
Web of Data/
Web of Things
Architecture

User Agents

Addressing scheme + Common protocol + Format negotiation

Servers/Gateways


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

- SOAR (initially: State, Operator, Apply, Result),
- ACT-R (Adaptive Control of Though – Rational)
- Goal: to create „intelligent agents“
- In the following, 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
- Sense: safe HTTP methods (GET)
- Act: unsafe HTTP methods
Resource-based Interface to Devices

- Tessel 2, WiFi and Ethernet interfaces
- Read-Write Linked Data interface to four LEDs at https://github.com/kaefer3000/t2-rest-leds

<table>
<thead>
<tr>
<th>Resource</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index page</td>
<td><a href="http://t2-rest-leds.lan/">http://t2-rest-leds.lan/</a></td>
</tr>
<tr>
<td>The Tessel 2</td>
<td><a href="http://t2-rest-leds.lan/#tessel2">http://t2-rest-leds.lan/#tessel2</a></td>
</tr>
<tr>
<td>Set of LEDs</td>
<td><a href="http://t2-rest-leds.lan/leds/#bar">http://t2-rest-leds.lan/leds/#bar</a></td>
</tr>
<tr>
<td>LED 0</td>
<td><a href="http://t2-rest-leds.lan/leds/0#led">http://t2-rest-leds.lan/leds/0#led</a></td>
</tr>
<tr>
<td>LED 1</td>
<td><a href="http://t2-rest-leds.lan/leds/1#led">http://t2-rest-leds.lan/leds/1#led</a></td>
</tr>
<tr>
<td>LED 2</td>
<td><a href="http://t2-rest-leds.lan/leds/2#led">http://t2-rest-leds.lan/leds/2#led</a></td>
</tr>
<tr>
<td>LED 3</td>
<td><a href="http://t2-rest-leds.lan/leds/3#led">http://t2-rest-leds.lan/leds/3#led</a></td>
</tr>
</tbody>
</table>
Device Described in RDF

- HTTP GET on http://t2-rest-leds.lan/ returns RDF with links to LEDs:

  ```rdfs
@prefix sosa:http://www.w3.org/ns/sosa/ .
<#tessel2> a sosa:Platform ;
  sosa:hosts <leds/#bar> .
  ```

- Querying with SPARQL (e.g. using roqet, # apt-get install rasqal-utils):

  ```sparql
SELECT ?y
FROM <http://t2-rest-leds.lan/>
WHERE { ?x sosa:hosts ?y . }
```
Link to LEDs

- GET on http://t2-rest-leds.lan/leds/ returns links to individual LEDs:

  @prefix sosa: <http://www.w3.org/ns/sosa/> .
  <#bar> a sosa:Platform ;

  sosa:hosts <0#led> , <1#led> , <2#led> , <3#led> .

- SPARQL query:

  SELECT ?x ?y
  FROM <http://t2-rest-leds.lan/leds/0>
  FROM <http://t2-rest-leds.lan/leds/1>
  WHERE { ?x <https://w3id.org/saref#hasState> ?y . }
Getting and Setting the State of an LED

- HTTP GET on http://t2-rest-leds.lan/leds/0 returns the current state:

```html
@prefix saref: <https://w3id.org/saref#> .
<#led> a saref:LightingDevice ;
    saref:hasState saref:Off .
```

- PUT on http://t2-rest-leds.lan/leds/0 with the following body sets the current state:

```html
@prefix saref: <https://w3id.org/saref#> .
<#led> saref:hasState saref:On .
```
Hello World

IoT: Blinking Light

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


{ <http://t2-rest-leds.lan/leds/0#led> saref:hasState saref:Off .
} => {
    http:body { <http://t2-rest-leds.lan/leds/0#led> saref:hasState saref:On . } .
} .

{ <http://t2-rest-leds.lan/leds/0#led> saref:hasState saref:On .
} => {
    http:body { <http://t2-rest-leds.lan/leds/0#led> saref:hasState saref:Off . } .
} .

Orchestration of Network-accessible Components with Condition-Action Rules
Link-Following to Access Resource State

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

# GET index page
[] http:mthd httpm:GET ;

# for each triple with predicate sosa:hosts
{
} => {
# do a GET on the object of the triple
   [] http:mthd httpm:GET ;
   http:requestURI ?y .
} .

Orchestration of Network-accessible Components with Condition-Action Rules
Linked Data-Fu Architecture

- Approach for accessing, integrating, querying and manipulating resource state on the Web of Things
- The language allows developers to specify interactions using rules
- The engine executes desired interactions in parallel

- **Request rules** specify how and when to interact with resources, i.e., retrieve the state of resources (sense) or manipulate the state of resources (act)
- **Derivation rules** support reasoning constructs, e.g., transitivity, reflexivity of properties
Conclusion

- We have designed and implemented a rule-base language to specify user agents on Read-Write Linked Data
- Rule-based programs encode a state machine that emits HTTP requests (user agents)
- The program operates in a sense-act cycle:
  - First acquire the state of resources via GET
  - Then decide which action to carry out
- Graph-structured data model (RDF), vocabulary description languages (RDFS) and ontology languages (OWL) provide basis for data modelling, integration and exchange
- Rules and state machines provide rigorous formal background for specifying and reasoning with application behaviour
- A (subset of) Semantic Web and Linked Data technologies provides a solid basis for specifying application behaviour
- We conduct research on advanced topics, for example pushing application behaviour to components
References

- Tessel2 REST+Linked Data interface: https://github.com/kaefer3000/t2-rest-leds
- Linked Data-Fu: https://linked-data-fu.github.io/