Autonomous Composition and Execution of REST APIs for Smart Sensors

Daniela Ventura¹, Ruben Verborgh², Vincenzo Catania¹, and Erik Mannens²

¹University of Catania (Italy) - Dpt. of Electrical, Electronics, and Computer Engineering
²Ghent University - iMinds - Multimedia Lab (Belgium)

SSN-TC, ISWC2015 - Bethlehem (USA) – 11 October 2015
**Introduction**

**Machine-to-Machine (M2M) interactions:** the ability of devices to exchange data in order to execute a task not explicitly required by a human being.

Focus on **Internet** connected devices. Benefits:
- stable performance;
- security mechanisms;
- global reach.
Introduction

Estimated\(^1\) number of smart internet connected “things” in use worldwide

<table>
<thead>
<tr>
<th>Year</th>
<th>M2M</th>
<th>Connected consumer devices (TV, laptop, smartphone)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>10</td>
<td>7</td>
</tr>
<tr>
<td>2014</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>2015</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>2020</td>
<td>30</td>
<td>23</td>
</tr>
</tbody>
</table>

\(^1\)Ericsson’s Media Vision 2020 research (2015)

Enabling machines to understand how to use a Web service and what the effect of the invocation is. Therefore, machines will be able to take decisions and cooperate with each other.
What are the issues?

• Describing what functionalities a device provides using a machine-interpretable format;

• Knowing at runtime the services’ interfaces => well-defined and universal semantics on the exchanged data;

• Composing REST APIs in order to execute tasks that satisfy goals => reasoning & plans’ generation and execution.
What are the issues?

• Describing what \textit{functionalities} a device provides using a machine-interpretable format;

• Knowing at runtime the services’ interfaces \implies well-defined and universal \textit{semantics on the exchanged data};

• \textit{Composing} REST APIs in order to execute tasks that satisfy goals \implies reasoning & plans’ generation and execution.
API Semantic Description

RESTdesc

\{ Precondition \}

\Rightarrow

\{ Postcondition \}
API Semantic Description: RESTdesc

@prefix vocab: <http://example.org/vocab#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix bonsai: <http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#>.

{ ?actuator a vocab:IrrigationPump. ?state a st:State;
   log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }.
?newValue a bonsai:SwitchAction;
   vocab:hasValue ?val.
} => {
   _:request http:methodName "PUT";
   http:requestURI (?actuator ?val);
[ a st:StateTransition; st:typeOperation "replacement";
 st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
 st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
 st:originalState ?state ].
}.

if a resource, whose type is "IrrigationPump", exists and a new SwitchingState would be set...
API Semantic Description: RESTdesc

```reasonml
@prefix vocab: <http://example.org/vocab#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix bonsai: <http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#>.

{ ?actuator a vocab:IrrigationPump.
  ?state a st:State;
    log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }.
  ?newValue a bonsai:SwitchAction;
    vocab:hasValue ?val.
} => {
  _:request http:methodName "PUT";
  http:requestURI (?actuator ?val);
  [ a st:StateTransition; st:typeOperation "replacement";
    st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
    st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
    st:originalState ?state ].
}.
```

...Invoking a HTTP PUT request to the URI which identifies the resource plus the value of new state...
API Semantic Description: RESTdesc

@prefix vocab: <http://example.org/vocab#>.
@prefix http: <http://www.w3.org/2011/http#>.
@prefix st: <http://www.mystates.org/states#>.
@prefix log: <http://www.w3.org/2000/10/swap/log#>.
@prefix bonsai: <http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#>.

{ ?actuator a vocab:IrrigationPump.
 ?state a st:State;
   log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }.
 ?newValue a bonsai:SwitchAction;
   vocab:hasValue ?val.
} => {
   _:request http:methodName "PUT";
   http:requestURI (?actuator ?val);
[ a st:StateTransition; st:typeOperation "replacement";
 st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
 st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
 st:originalState ?state ].
}.  

...The IrrigationPump will change Switching state.
What are the issues?

- Describing what *functionalities* a device provides using a machine-interpretable format;

- Knowing at runtime the services’ interfaces => well-defined and universal *semantics on the exchanged data*;

- Composing REST APIs in order to execute tasks that satisfy goals => reasoning & plans’ generation and execution.
Data Exchange Format

Web APIs + Semantic Web = JSON-LD
Data Exchange Format: JSON-LD

```json
{
  "@context": {
    "vocab": "http://example.org/vocab#",
    "schema": "https://schema.org/",
    "bonsai": "http://lpis.csd.auth.gr/ontologies/bonsai/BOnSAI.owl#",
    "actuatorState": "vocab:hasSwitchingState",
    "hasValue": {
      "@id": "vocab:hasValue",
      "@type": "schema:Boolean"
    }
  },
  "@id": "http://localhost:3300/actuators/1",
  "@type": "vocab:IrrigationPump",
  "actuatorState": {
    "@type": "bonsai:SwitchAction",
    "hasValue": 1
  }
}
```

Result of a HTTP request to switch on the IrrigationPump identified by @id
Why JSON-LD & RESTdesc?

<table>
<thead>
<tr>
<th>JSON-LD</th>
<th>RDF</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;http://.../actuators/1&gt; vocab:hasSwitchingState _:b0. </code>_:b0 a bonsai:SwitchAction; vocab:hasValue &quot;1&quot;^^<a href="https://schema.org/Boolean">https://schema.org/Boolean</a>`.</td>
</tr>
</tbody>
</table>

We know in advance...

```
{ ?actuator a vocab:IrrigationPump. 
?state a st:State; 
  log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }. 
?newValue a bonsai:SwitchAction; 
  vocab:hasValue ?val. 
} => {
  _:request http:methodName "PUT";
  http:requestURI (?actuator ?val);
  [ a st:StateTransition; st:typeOperation "replacement"; 
    st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. }; 
    st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. }; 
    st:originalState ?state ]. }.
```
Why JSON-LD & RESTdesc?

JSON-LD

```
{ "@context": "http://.../contexts/putPump.j
  
  
  "@id": "http://.../actuators/1",
  "@type": "vocab:IrrigationPump",
  "actuatorState": {
    "@type": "bonsai:SwitchAction",
    "hasValue": 1 } }
```

RDF

```
<http://.../actuators/1> a vocab:IrrigationPump.
<http://.../actuators/1> vocab:hasSwitchingState _:_b0.
  _:_b0 a bonsai:SwitchAction;
  vocab:hasValue "1"^^<https://schema.org/Boolean>.
```

RESTdesc

```
{ ?actuator a vocab:IrrigationPump.
  ?state a st:State;
    log:includes { ?actuator vocab:hasSwitchingState ?oldValue. }.
  ?newValue a bonsai:SwitchAction;
    vocab:hasValue ?val.
} => {
  _:request http:methodName "PUT";
    http:requestURI (?actuator ?val);
  [ a st:StateTransition; st:typeOperation "replacement";
    st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
    st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
    st:originalState ?state ]. }
```

We know in advance...

The properties of the returned resource
What are the issues?

• Describing what functionalities a device provides using a machine-interpretable format;

• Knowing at runtime the services’ interfaces => well-defined and universal semantics on the exchanged data;

• Composing REST APIs in order to execute tasks that satisfy goals => reasoning & plans’ generation and execution.
How managing APIs’ composition and execution?

Use Case: **Smart Garden/Greenhouse**

Autonomously monitoring of environmental conditions and taking decisions about how and when to act in order to ensure plants thrive.

**Entities involved:**

- **Embedded board**: internet-connected, sensors (temperature, light, moisture) & actuators (irrigation pumps, lamps, heaters, automated windows), GPS module $\Rightarrow$ Garden API.
- **Weather web server**: current and forecast (daily or hourly) weather $\Rightarrow$ Weather API.
- **Smart client**: decision-algorithm to ensure the ideal environment conditions for each plant - No pre-knowledge about what the APIs do and how invoking their services.
What algorithms can the Smart Client execute?

1. Real-time sensors’ data.
2. Real-time sensors’ data + Current weather conditions => to overcome the eventually lack of sensors.
3. Real-time sensors’ data + Current weather conditions + Hourly forecast weather conditions => more efficient use of actuators.
4. Real-time sensors’ data + Current weather conditions + Daily forecast weather conditions => more efficient use of actuators.
Architecture

Smart Client

Preferences

Basic Knowledge

RESTdesc Description

Goal

Algorithms Manager

Proxy

Planner

Reasoner

Parser

JSON-LD msg

Board 1 API

Board 2 API

Board 3 API

Weather API

SSN-TC, ISWC2015 - Bethlehem (USA) – 11 October 2015
“Get sensors values associated with each plant monitored by each embedded board known by the client”
Task: to find all the possible plans that fulfill the goal.

I = Initial State
$S_x = x$ State reached after having passed $x-1$ State
G = Goal State
Reasoner – Use Case

Plan 1

\[ /\text{plants} \rightarrow L_{27} \rightarrow L_{28} \rightarrow L_{29} \rightarrow G \]

Plan 2

\[ /\text{plants/idPlant}* \rightarrow L_{12} \rightarrow L_{13} \rightarrow G \]

Plan 3

\[ /\text{sensors/idSensor}* \rightarrow L_{4} \rightarrow G \]

* = To repeat for the number of plants/sensors
Task: to **convert** the output generated by Reasoner in object properly formatted that can be understood by client.

```
lemmas : [
  lemma27 : {...},
  lemma28 : {...},
  ...
],
plans : [
  plan-1:[
    lemma27, lemma28, lemma29
  ]
  ...
]
```
Plan Selection & Execution

Planner

Task: to **select** and **manage** the plan **execution**

Proxy

Task: to **call** HTTP requests following the directives of Planner in order to advance the plan **execution**
Plan Selection

Plan 1:

\[ \text{Plan 1} \]

\[ \text{I} \rightarrow L_{27} \rightarrow L_{28} \rightarrow L_{29} \rightarrow G \]

Plan 2:

\[ \text{Plan 2} \]

\[ \text{I} \rightarrow L_{12} \rightarrow L_{13} \rightarrow G \]

Plan 3:

\[ \text{Plan 3} \]

\[ \text{I} \rightarrow L_{4} \rightarrow G \]

* = To repeat for the number of plants/sensors
Plan Execution

Plan $\Rightarrow$ /plants

/plants
/plants/1
/plants/2
/sensors/1
/sensors/2

Plant-1: { associatedSensors: [1,2] }
Plan Execution

/plants
/plants/1
/plants/2
/sensors/1
/sensors/2

Plant-1: { associatedSensors : [1,2] }
Plan Execution

```
/plants
/plants/1
/plants/2
/sensors/1
/sensors/2
```

Plant-1: \{ associatedSensors : [1,2] \}

Plan \+ \(\text{RETURN} \ [*\text{idPlant}]\) => ???????
Plan Execution – Bottom-Up Algorithm

Plan converted in JSON-LD

context : {...}
lemma27 : {
  methodName : “GET”,
  requestURI : “/plants”,
  resp: “sk2_1”,
  sk2_1 : { members : “sk5_1” }
  sk5_1 : { @type : “Plant” }
},
lemma28 : {
  methodName : “GET”,
  requestURI : “sk5_6”,
  resp: “sk5_6”,
  sk5_6 : { hasAssociatedSensors : “sk10_2”,
          hasAssociatedActuators : “sk11_2,
                      ...
  }
},
lemma29 : {...}
plan : [ lemma27, lemma28, lemma29 ]

JSON-LD returned by Board 1

context : {...},
"@type": "Collection",
"@id": "/plants",
"members" : [
  { “@type” : “Plant”,
    “@id” : “/plants/1” },
  { “@type” : “Plant”,
    “@id” : “/plants/2” }
]

SSN-TC, ISWC2015 - Bethlehem (USA) – 11 October 2015
Plan Execution – Bottom-Up Algorithm

Plan converted in JSON-LD

```
context : {...}
lemma27 : {
    methodName : “GET”,
    requestURI : “/plants”,
    resp : “sk2_1”,
    sk2_1 : { members : “sk5_1” }
    sk5_1 : { @type : “Plant” }
},
lemma28 : {
    methodName : “GET”,
    requestURI : “sk5_6”,
    resp : “sk5_6”,
    sk5_6 : { hasAssociatedSensors : “sk10_2”,
                 hasAssociatedActuators : “sk11_2”,
                 … }
},
lemma29 : {...}
plan : [ lemma27, lemma28, lemma29 ]
```

JSON-LD returned by Board 1

```
context : {...},
"@type": "Collection",
"@id": "/plants",
"members" : [
    { "@type" : "Plant",
      "@id" : "/plants/1"
    },
    { "@type" : "Plant",
      "@id" : "/plants/2"
    }
]
```

Graph:

```
sk5
  └── members
    └── /plants/1
        └── /plants/2
```

SSN-TC, ISWC2015 - Bethlehem (USA) – 11 October 2015
Plan Execution

/plants
/plants/1
/plants/2
/sensors/1
/sensors/2

Plan + RETURN [*idPlant] => /plants/1
/plants/2

Continue...

Plant-1: { associatedSensors : [1,2] }
Plan Execution

- Proxy calls /plants/1 & receives Plant-1 by Board;
- Proxy calls /plants/2 & receives Plant-2 by Board;
- Parser advances in plan execution => find the new URIs to call
- Proxy calls /sensors/1 & receives Sensor-1 by Board;
- Proxy calls /sensors/2 & receives Sensor-2 by Board;
- Parser understands the **GOAL is satisfied!**
Evaluation

A) One plant, one sensor and one actuator (low load).
B) Three plants, each of them has associated three sensors and actuators (high load).

<table>
<thead>
<tr>
<th>Tasks Goal</th>
<th>Operation</th>
<th>Algorithm 1</th>
<th>Algorithm 2</th>
<th>Algorithm 3</th>
<th>Algorithm 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>A (ms)</td>
<td>B (ms)</td>
<td>A (ms)</td>
<td>B (ms)</td>
</tr>
<tr>
<td>Get Plants</td>
<td>Reas. + Pars.</td>
<td>361</td>
<td>366</td>
<td>426</td>
<td>417</td>
</tr>
<tr>
<td></td>
<td>Selec. + Exec.</td>
<td>33</td>
<td>79</td>
<td>46</td>
<td>96</td>
</tr>
<tr>
<td>Get Sensors</td>
<td>Reas. + Pars.</td>
<td>380</td>
<td>387</td>
<td>465</td>
<td>459</td>
</tr>
<tr>
<td></td>
<td>Selec. + Exec.</td>
<td>35</td>
<td>80</td>
<td>49</td>
<td>104</td>
</tr>
<tr>
<td>Get Actuators</td>
<td>Reas. + Pars.</td>
<td>–</td>
<td>–</td>
<td>572</td>
<td>560</td>
</tr>
<tr>
<td></td>
<td>Selec. + Exec.</td>
<td>–</td>
<td>–</td>
<td>35</td>
<td>98</td>
</tr>
<tr>
<td>Get Current Weather</td>
<td>Reas. + Pars.</td>
<td>–</td>
<td>–</td>
<td>464</td>
<td>470</td>
</tr>
<tr>
<td></td>
<td>Selec. + Exec.</td>
<td>–</td>
<td>–</td>
<td>657</td>
<td>702</td>
</tr>
<tr>
<td>Get Current Weather</td>
<td>Reas. + Pars.</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>Selec. + Exec.</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Results

The number of plants, sensors or actuators:

- Does not influence the Reasoner and Parser’s tasks: Average Reasoner + Parser time = 0.4 seconds

- Influences greatly the time spent to complete the plan execution: Average Execution Time for execution B is two or three times more that execution A

Tot. Time Algorithm 1 => less than 1 sec.

Tot. Time Algorithm 4 => 5 sec.
Thanks For Your Attention!

Daniela Ventura
PhD Student
University of Catania (Italy)
daniela.ventura@dieei.unict.it

SSN-TC, ISWC2015 - Bethlehem (USA) – 11 October 2015