# Autonomous Composition and Execution of REST APIs for Smart Sensors

**Daniela Ventura**[1], Ruben Verborgh[2], Vincenzo Catania[1], and Erik Mannens[2]

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

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

# Introduction

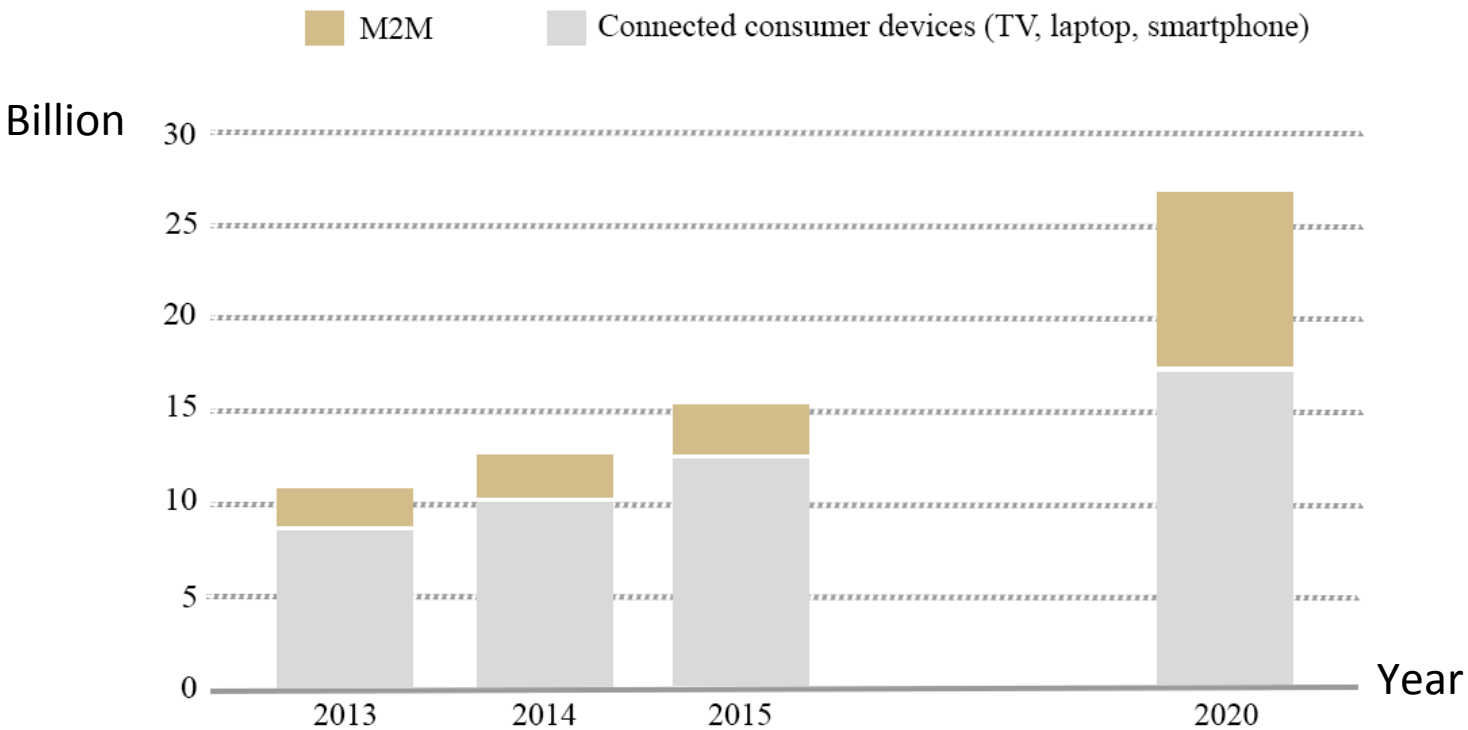
**Machine-to-Machine** (M2M) interactions: the ability of devices to exchange data in order to execute a task not explicitly required by a human being.

Focus on **Internet** connected devices. Benefits:

- stable performance;
- security mechanisms;
- global reach.

# Introduction

Estimated[1] number of smart internet connected "things" in use worldwide




[1]Ericsson's Media Vision 2020 research (2015)
http://www.ericsson.com/res/docs/2015/ericsson-mobility-report-june-2015.pdf

# What is the Challenge?

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 ***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.

# API Semantic Description

RESTdesc

{ Precondition }

=>

{ Postcondition }

n3

# 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);
    http:resp [http:body ?actuator].
[ 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

```
@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);
    http:resp [http:body ?actuator].
[ 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);
    http:resp [http:body ?actuator].
[ 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.**

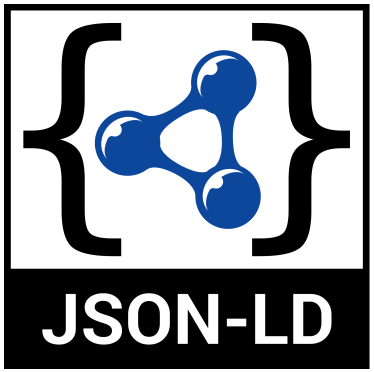
# What are the issues?

- 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


+


=



# Data Exchange Format: JSON-LD

```
{
 "@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 identify by @id**

# Why JSON-LD & RESTdesc?

**JSON-LD**

```
{ "@context" : “http://…/contexts/putPump.jsonld”,
  "@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>.
```

We know in advance…

**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);
      http:resp [http:body ?actuator].
[ a st:StateTransition;  st:typeOperation "replacement";
st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
st:originalState ?state ]. }.
```

1

**The type of the returned resource**

# Why JSON-LD & RESTdesc?

**JSON-LD**

```
{ "@context" : “http://…/contexts/putPump.jsonld”,
  "@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>.
```

We know in advance…

**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);
      http:resp [http:body ?actuator].
[ a st:StateTransition;  st:typeOperation "replacement";
st:oldComponent { ?actuator vocab:hasSwitchingState ?oldValue. };
st:newComponent { ?actuator vocab:hasSwitchingState ?newValue. };
st:originalState ?state ]. }.
```

2

**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.

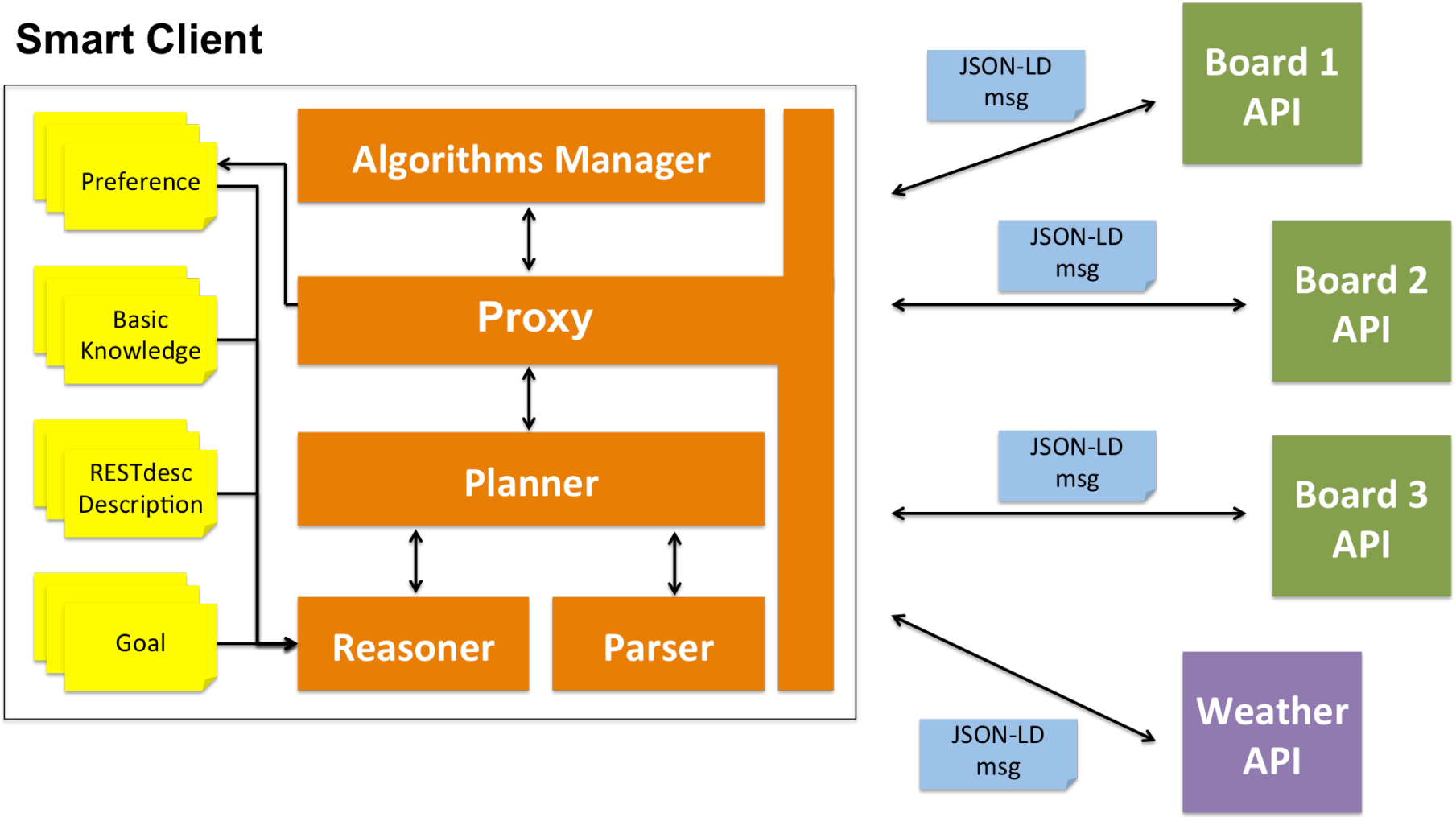
Entities involved:

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

# 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

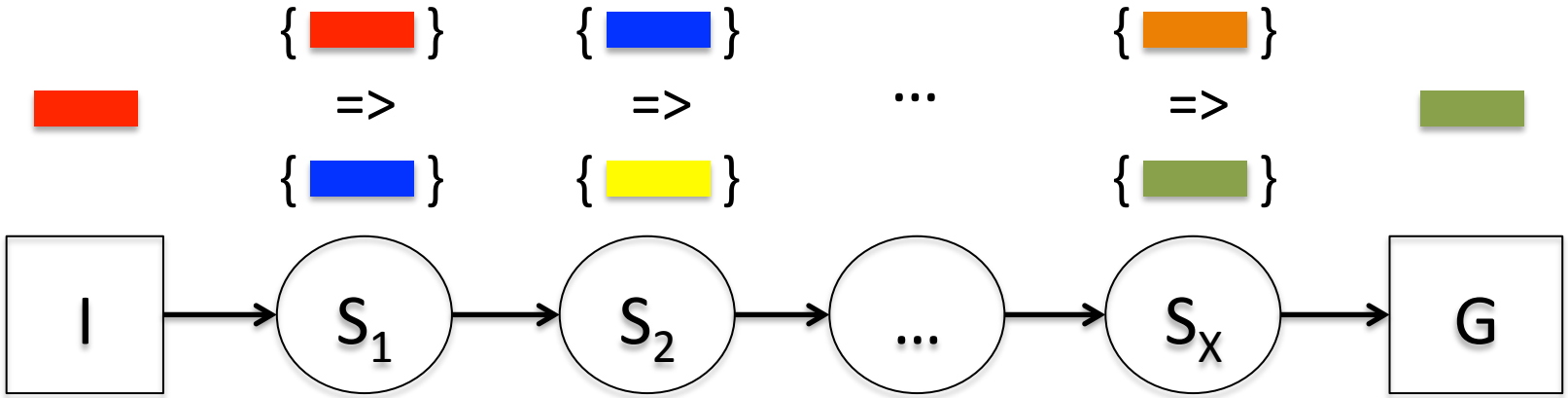


# Goal

*"Get sensors values associated with each plant monitored by each embedded board known by the client"*

# Reasoner

Task: to find **all** the possible **plans** that fulfill the goal.

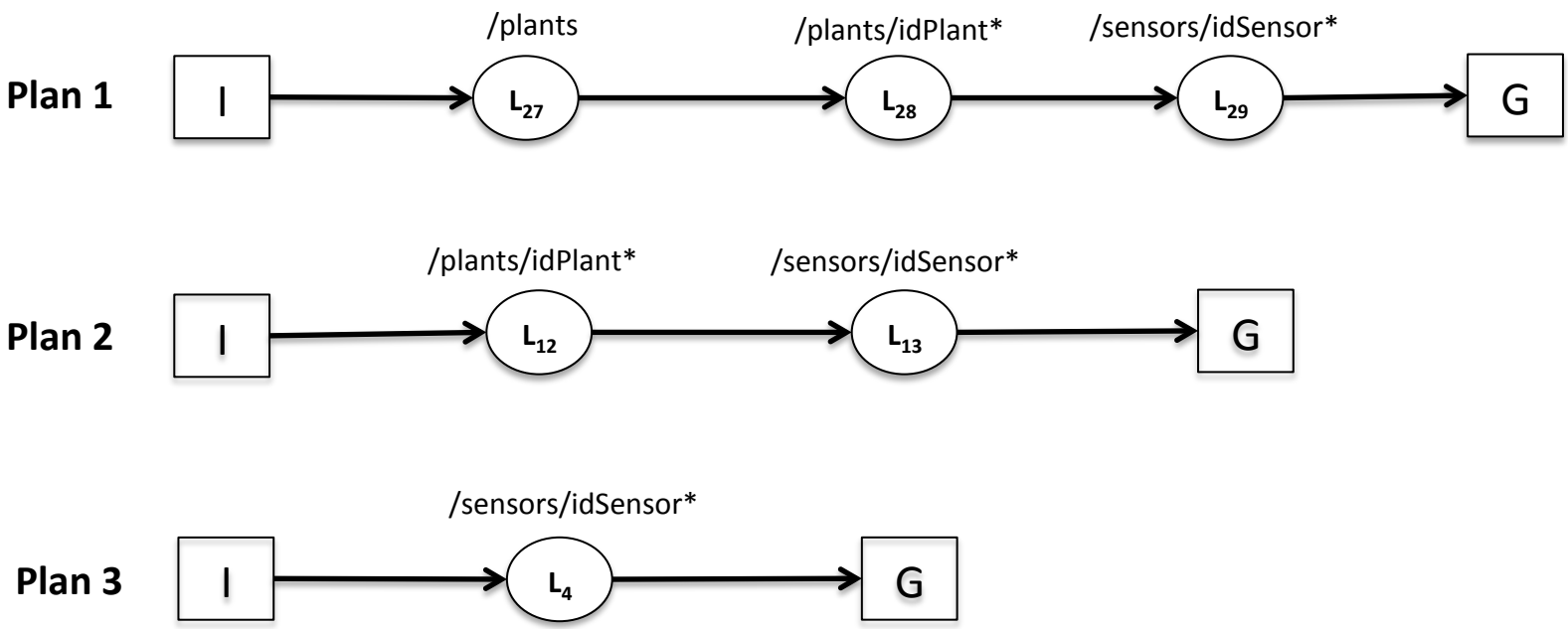



I = Initial State

$S_x$ = x State reached after having passed x-1 State

G = Goal State

# Reasoner – Use Case




* = To repeat for the number of plants/sensors

# Parser

Task: to **convert** the output generated by Reasoner in object properly formatted that can be understood by client.

```
p:lemma27 a c:ServiceCall.
p:lemma28 a c:ServiceCall.
p:lemma29 a c:ServiceCall.
p:lemma28 c:hasDependency p:lemma27.
p:lemma29 c:hasDependency p:lemma28.
p:lemma27 c:details {_:sk3_1 http:methodName "GET".
  _:sk3_1 http:requestURI <http://127.0.0.1:3300/plants>; http:resp _:sk4_1.
  _:sk4_1 http:body <http://127.0.0.1:3300/plants>.
  <http://127.0.0.1:3300/plants> hydra:member _:sk5_1. _:sk5_1 a vocab:Plant}.
p:lemma28 c:details {_:sk36_1 http:methodName "GET".
  _:sk36_1 http:requestURI _:sk5_2; http:resp _:sk37_1.
  _:sk37_1 http:body _:sk5_2.
  _:sk5_2 vocab:hasAssociatedSensors _:sk40_1; vocab:hasAssociatedActuators _:sk41_1.
  _:sk5_2 vocab:hasIdealTemperature _:sk42_1; vocab:hasIdealMoisture _:sk43_1.
  _:sk5_2 vocab:hasIdealLight _:sk44_1.
  _:sk40_1 a vocab:SensorsPlantCollection; hydra:member _:sk45_3. _:sk45_3 a vocab:Sensor.
  _:sk41_1 a vocab:ActuatorsPlantCollection; hydra:member _:sk46_1.
  _:sk46_1 a vocab:Actuator; vocab:hasSwitchingState _:sk47_1.}.
p:lemma29 c:details {_:sk104_1 http:methodName "GET".
  _:sk104_1 http:requestURI _:sk45_4; http:resp _:sk105_1.
  _:sk105_1 http:body _:sk45_4. _:sk45_4 vocab:madeObservation _:sk108_1.
  _:sk108_1 vocab:hasTimestamp _:sk109_1; vocab:outputObservation _:sk110_1}.
```

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

# Plan Selection & Execution

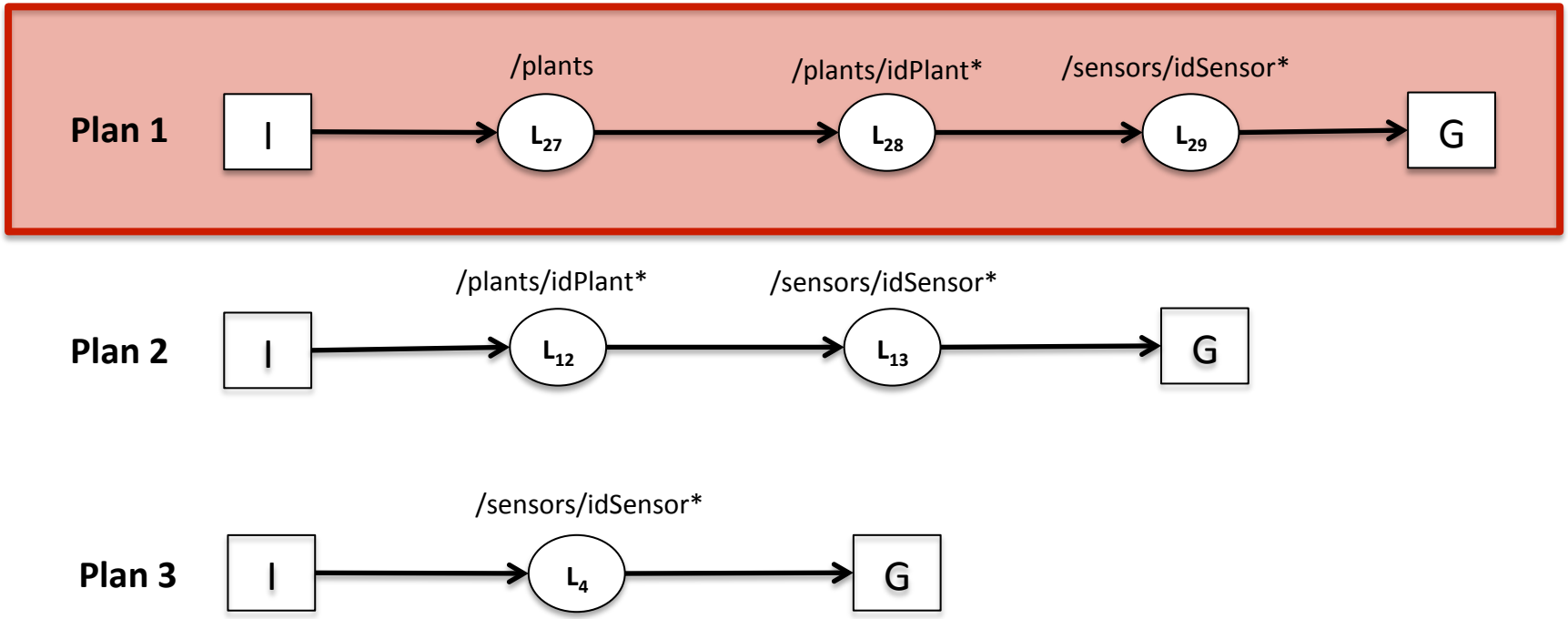
## Planner

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

## Proxy

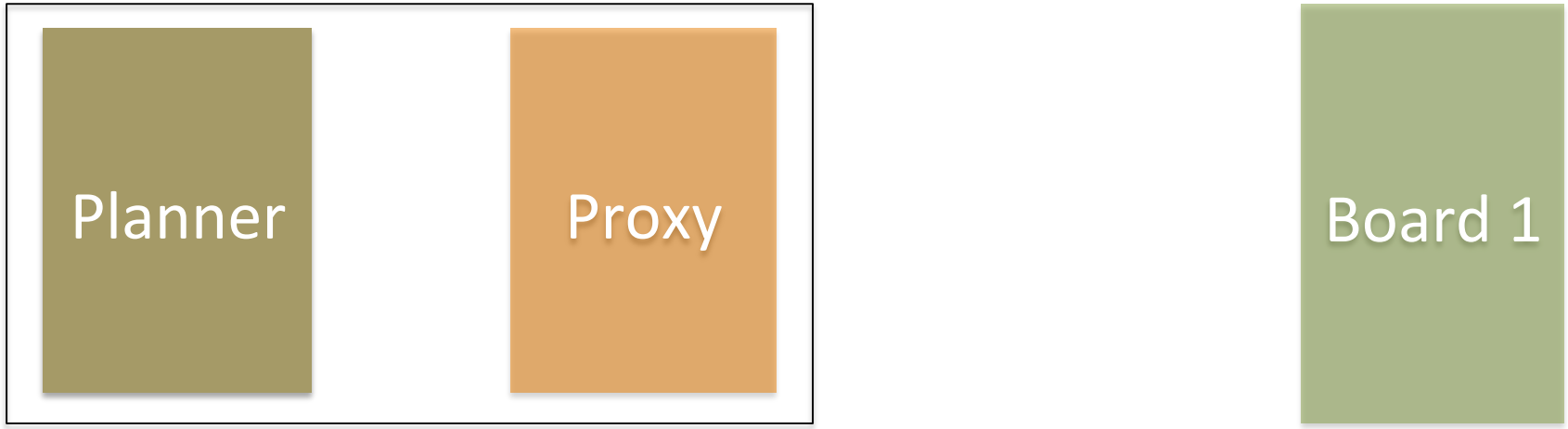
Task: to **call** HTTP requests following the directives of Planner in order to advance the plan **execution**

# Plan Selection




* = To repeat for the number of plants/sensors

# Plan Execution

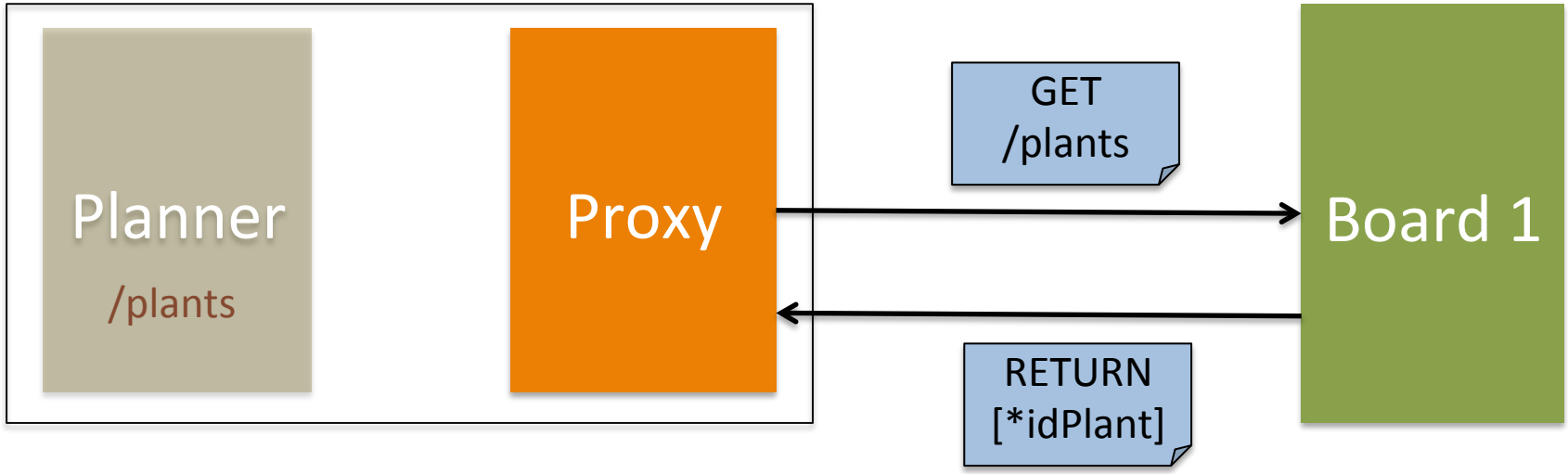



Plan => /plants

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

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

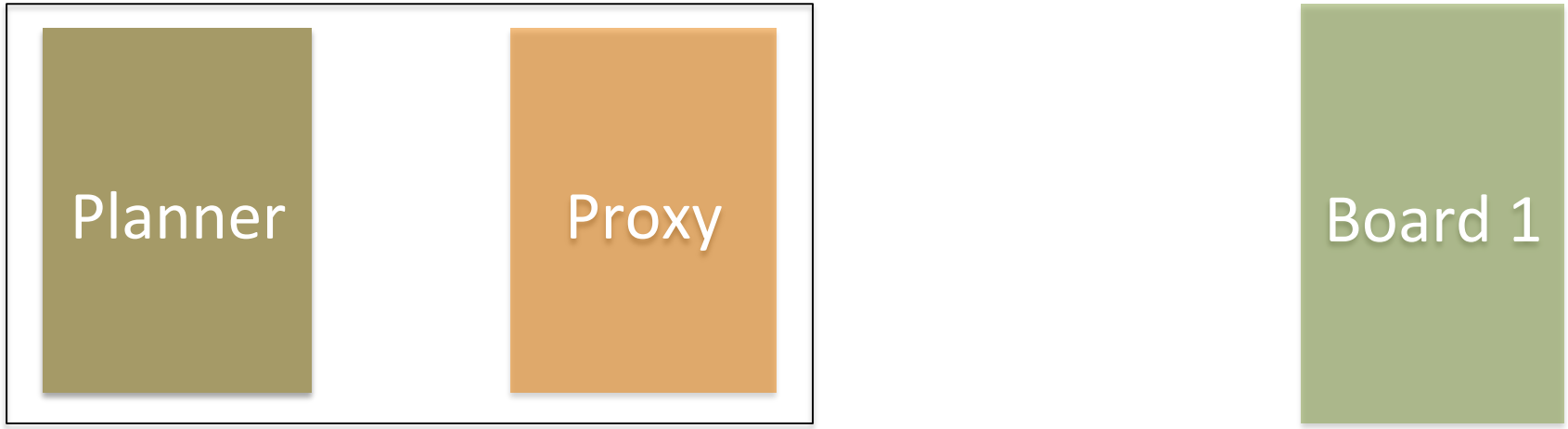
# Plan Execution




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

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

# Plan Execution




Plan + RETURN [*idPlant] => ??????

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

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

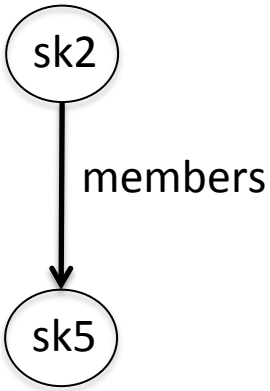
# Plan Execution – Bottom-Up Algorithm

**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” }
]
```



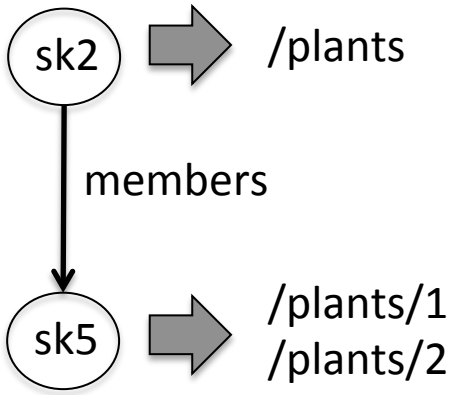
# Plan Execution – Bottom-Up Algorithm

**Plan converted in JSON-LD**

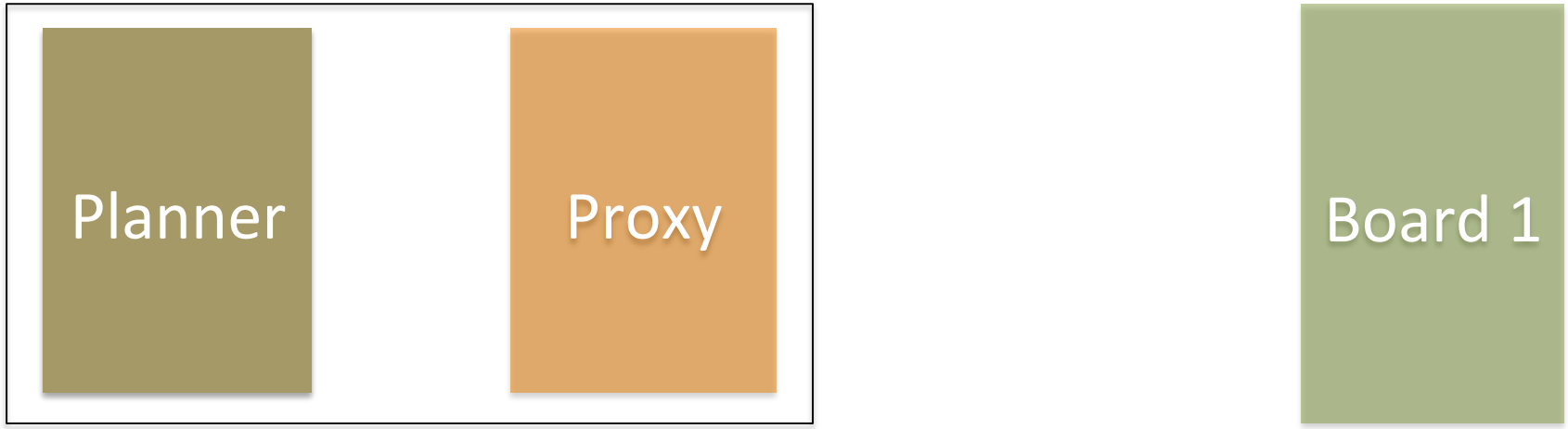
```
context : {…}
lemma27 : {
  methodName : “GET”,
  requestURI : “/plants”,
  resp: “sk2_1”,
  sk2_1 : { members : “sk5_1” }
  sk5_1 : { @type : “Plant” }
},
lemma28 : {
  methodName : “GET”,
  requestURI : “sk5_6”,
  resp: “sk5_6”,
  sk5_6 : { hasAssociatedSensors : “sk10_2”,
            hasAssociatedActuators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



# Plan Execution




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

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

**Continue…**

Plant-1: { associatedSensors : [1,2] }
Plant-2: { associatedSensors : [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**!

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

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

# 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).

| Tasks | | Algorithm 1 | | Algorithm 2 | | Algorithm 3 | | Algorithm 4 | |
|---|---|---|---|---|---|---|---|---|---|
| **Goal** | **Operation** | **A (ms)** | **B (ms)** | **A (ms)** | **B (ms)** | **A (ms)** | **B (ms)** | **A (ms)** | **B (ms)** |
| **Get Plants** | Reas. + Pars. | 361 | 366 | 426 | 417 | 403 | 408 | 394 | 399 |
| | Selec. + Exec. | 33 | 79 | 46 | 96 | 50 | 85 | 47 | 83 |
| **Get Sensors** | Reas. + Pars. | 380 | 387 | 465 | 459 | 401 | 420 | 413 | 420 |
| | Selec. + Exec. | 35 | 80 | 49 | 104 | 42 | 93 | 50 | 94 |
| **Get Actuators** | Reas. + Pars. | – | – | 572 | 560 | 489 | 504 | 482 | 498 |
| | Selec. + Exec. | – | – | 35 | 98 | 38 | 87 | 39 | 98 |
| **Get Current Weather** | Reas. + Pars. | – | – | 464 | 470 | 552 | 514 | 467 | 501 |
| | Selec. + Exec. | – | – | 657 | 702 | 776 | 785 | 837 | 799 |
| **Get Current Weather** | Reas. + Pars. | – | – | – | – | 451 | 456 | 500 | 459 |
| | Selec. + Exec. | – | – | – | – | 835 | 850 | 805 | 865 |

# Results

The number of plants, sensors or actuators:

- Does not influence the Reasoner and Parser's tasks:
  Average Reasoner + Parser time = 0.4 seconds → Depends from the number of dependencies

- 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