Yet another Action model

Trying to describe action queues and dynamic resources

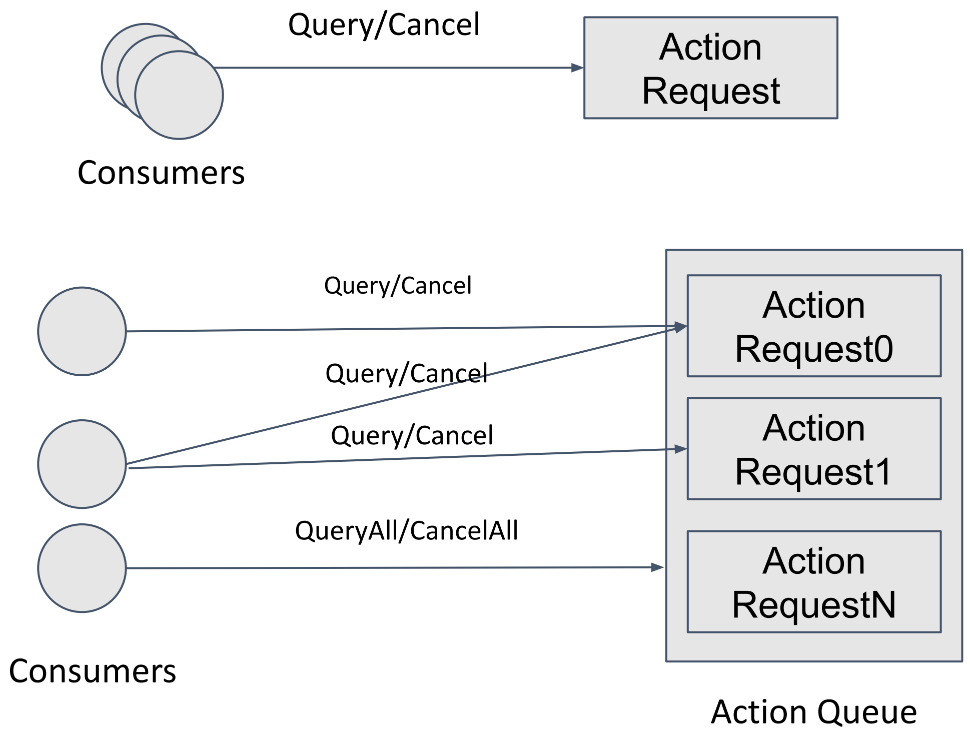
Problem

How to describe complex action interaction models of the IoT world.

Features of complex action models:

- Actions can last longer of a regular protocol request
- Users can stop or query or perform other operation on an action Request
- Users might queue an action requests

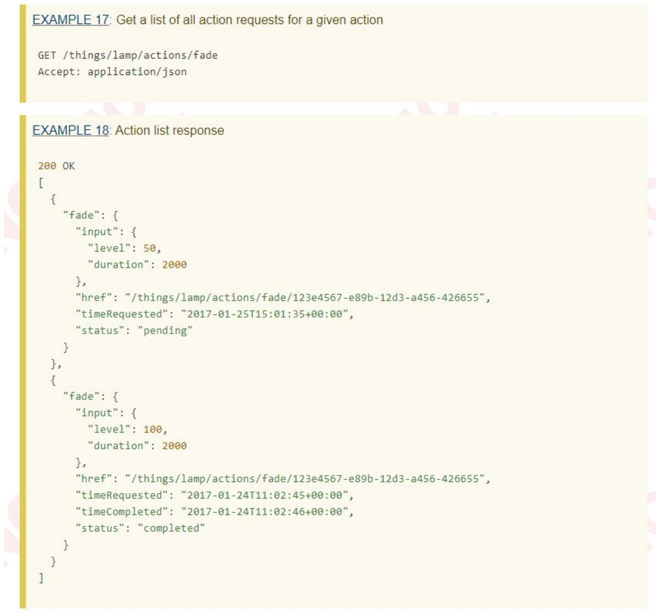
Problem


Query/Cancel
Action Request
Consumers
Query/Cancel
Query/Cancel
Query/Cancel
QueryAll/CancelAll
Action Request0
Action Request1
Action RequestN
Consumers
Action Queue

Use cases: webthings.io

Each action affordance has its own queue, there is a global queue per WebThing where all actions requests are scheduled.

Examples:

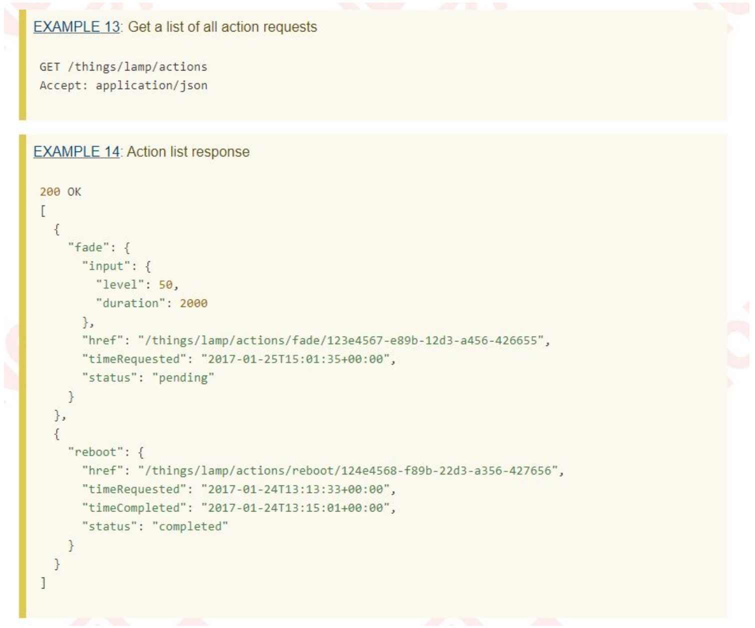


EXAMPLE 17: Get a list of all action requests for a given action

```
GET /things/lamp/actions/fade
Accept: application/json
```

EXAMPLE 18: Action list response

```
200 OK
[
  {
    "fade": {
      "input": {
        "level": 50,
        "duration": 2000
      },
      "href": "/things/lamp/actions/fade/123e4567-e89b-12d3-a456-426655",
      "timeRequested": "2017-01-25T15:01:35+00:00",
      "status": "pending"
    }
  },
  {
    "fade": {
      "input": {
        "level": 100,
        "duration": 2000
      },
      "href": "/things/lamp/actions/fade/123e4567-e89b-12d3-a456-426655",
      "timeRequested": "2017-01-24T11:02:45+00:00",
      "timeCompleted": "2017-01-24T11:02:46+00:00",
      "status": "completed"
    }
  }
]
```



EXAMPLE 13: Get a list of all action requests

```
GET /things/lamp/actions
Accept: application/json
```

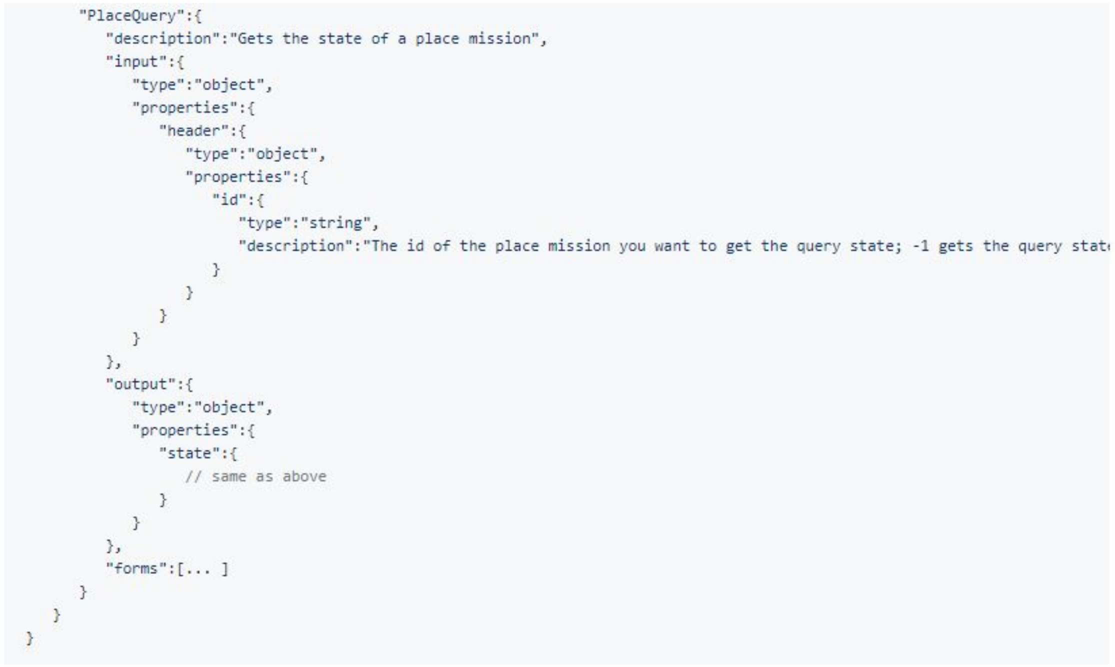
EXAMPLE 14: Action list response

```
200 OK
[
  {
    "fade": {
      "input": {
        "level": 50,
        "duration": 2000
      },
      "href": "/things/lamp/actions/fade/123e4567-e89b-12d3-a456-426655",
      "timeRequested": "2017-01-25T15:01:35+00:00",
      "status": "pending"
    }
  },
  {
    "reboot": {
      "href": "/things/lamp/actions/reboot/124e4568-f89b-22d3-a356-427656",
      "timeRequested": "2017-01-24T13:13:33+00:00",
      "timeCompleted": "2017-01-24T13:15:01+00:00",
      "status": "completed"
    }
  }
]
```

Use cases: BRAIN-IOT TD example

```
{
   "title":"robotnik",
   "description":"Robotnik REST Implementation for Brain-Iot",
   "actions":{
      "PlaceAdd":{
         "description":"Commands a robot to start place procedure",
         "input":{...},
         "output":{
            "type":"object",
            "properties":{
               "state":{
                  "type":"object",
                  "properties":{
                     "current_state":{
                        "type":"string",
                        "enum":["queued","running","paused","finished","unknown"]
                     }
                  }
               }
            },
         "forms":[...]
      },
      "PlaceCancel":{
         "description":"Cancels the current place mission",
         "input":{
            "type":"object",
            "properties":{
               "header":{
                  "type":"object",
                  "properties":{
                     "id":{
                        "type":"string",
                        "description":"The ID of the place mission you want to cancel; -1 cancels last mission"
                     }
                  }
               }
            }
         },
         "output":{
            "type":"object",
            "properties":{
               "state":{
                  // same as above
               }
            }
         },
         "forms":[...]
      },
```

Use cases: BRAIN-IOT TD example cont.



```
        "PlaceQuery":{
            "description":"Gets the state of a place mission",
            "input":{
                "type":"object",
                "properties":{
                    "header":{
                        "type":"object",
                        "properties":{
                            "id":{
                                "type":"string",
                                "description":"The id of the place mission you want to get the query state; -1 gets the query stat
                            }
                        }
                    }
                }
            },
            "output":{
                "type":"object",
                "properties":{
                    "state":{
                        // same as above
                    }
                }
            },
            "forms":[... ]
        }
    }
}
```

Use cases: OPCUA

Programs are complex *Functions* in a *Server* or underlying system that can be invoked and managed by a *Client*. *Programs* can represent any level of functionality within a system or process in which *Client* control or intervention is required and progress monitoring is desired.

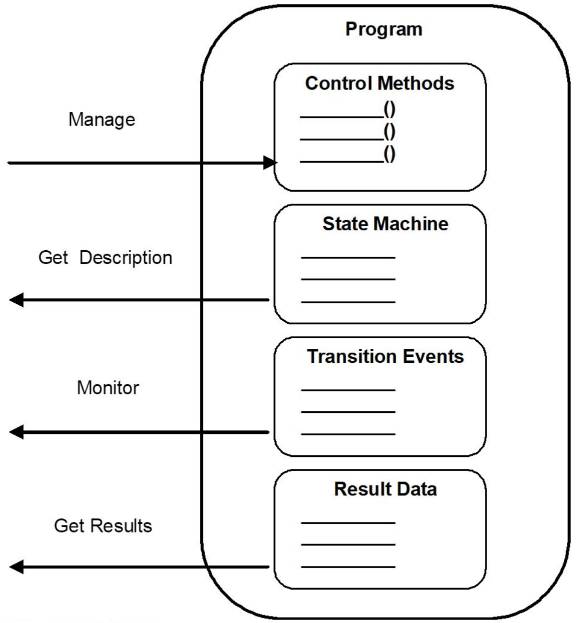


Figure 2 – Program illustration

State of the art

Dynamic TDs

Describe different forms to access **Action Request** and queues with a dynamic TD. The TD is update every once a new action request is issued.

Group found this solution to be troublesome:

- Caching issues
- Consumers needs to keep track of TD changes
- Security concerns (?)
- No static description of the possible operation. (Bad for documentation)

New Ops and URI templates

Describes **Action queues** and operations on **Action Requests** with new op member. Use URI template to describe dynamically changing Forms.

Group welcomed this solution. Nonetheless there are some concerns about how consumers would understand URI templates without out-of-band information.

Proposal: Leverage on Thing Models and Action Thing Descriptions

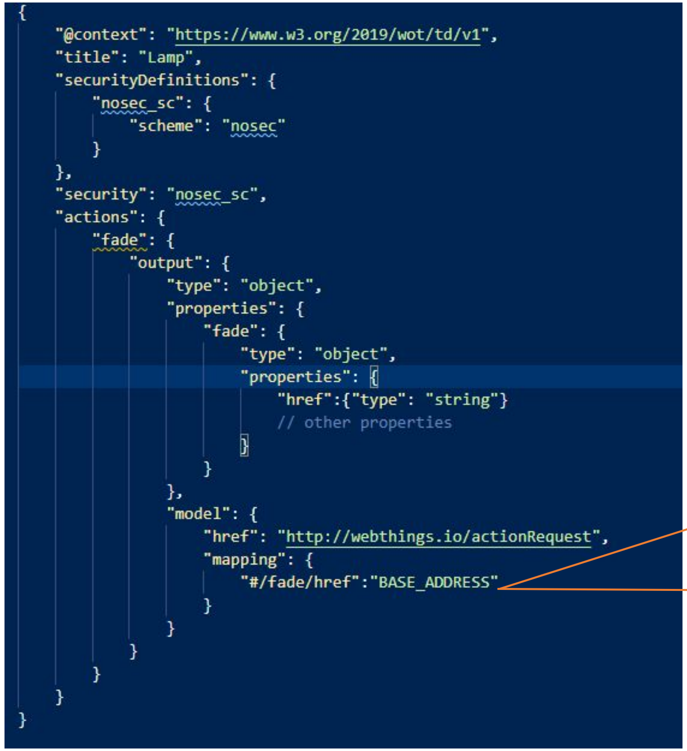
The idea is to use a Thing Model to describe further operations that the consumer can do after an invokeaction operation. At runtime consumers will build a Action Thing Description that can be use to access further operations like querying the status or cancel the action execution.

This approach should work for both green and brown field devices.

Early ideas can be found here

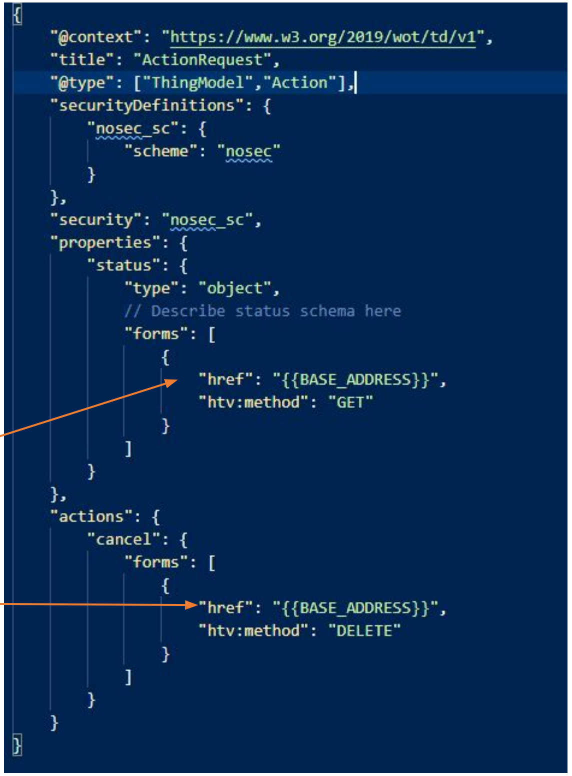
Proposal: Leverage on Thing Models and Action Thing Descriptions

Thing Description



```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "Lamp",
    "securityDefinitions": {
        "nosec_sc": {
            "scheme": "nosec"
        }
    },
    "security": "nosec_sc",
    "actions": {
        "fade": {
            "output": {
                "type": "object",
                "properties": {
                    "fade": {
                        "type": "object",
                        "properties": {
                            "href":{"type": "string"}
                            // other properties
                        }
                    }
                },
                "model": {
                    "href": "http://webthings.io/actionRequest",
                    "mapping": {
                        "#/fade/href":"BASE_ADDRESS"
                    }
                }
            }
        }
    }
}
```

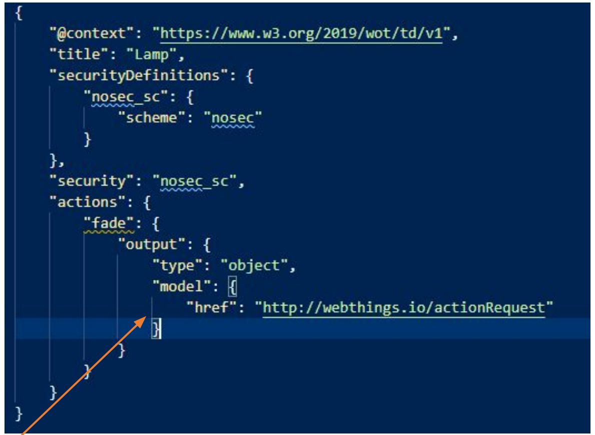
Thing Model



```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "ActionRequest",
    "@type": ["ThingModel","Action"],
    "securityDefinitions": {
        "nosec_sc": {
            "scheme": "nosec"
        }
    },
    "security": "nosec_sc",
    "properties": {
        "status": {
            "type": "object",
            // Describe status schema here
            "forms": [
                {
                    "href": "{{BASE_ADDRESS}}",
                    "htv:method": "GET"
                }
            ]
        }
    },
    "actions": {
        "cancel": {
            "forms": [
                {
                    "href": "{{BASE_ADDRESS}}",
                    "htv:method": "DELETE"
                }
            ]
        }
    }
}
```

Proposal: GreenField (core profile?)

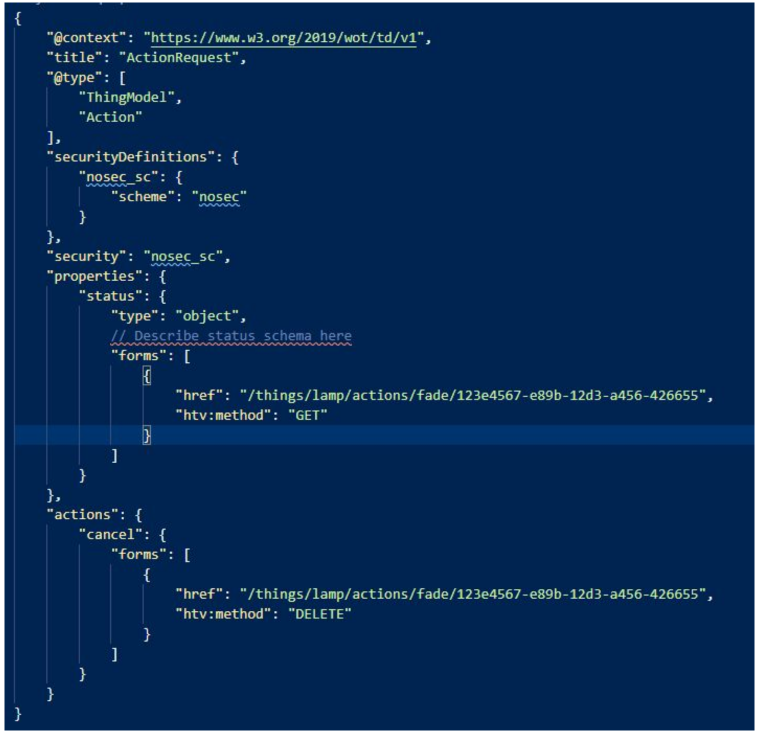
Thing Description



```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "Lamp",
    "securityDefinitions": {
        "nosec_sc": {
            "scheme": "nosec"
        }
    },
    "security": "nosec_sc",
    "actions": {
        "fade": {
            "output": {
                "type": "object",
                "model": {
                    "href": "http://webthings.io/actionRequest"
                }
            }
        }
    }
}
```

Use **model** as a validation hint for the returned json object. Thanks to the advancement in json schema community it is possible to define new validation terms.

Action Thing Description



```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "ActionRequest",
    "@type": [
        "ThingModel",
        "Action"
    ],
    "securityDefinitions": {
        "nosec_sc": {
            "scheme": "nosec"
        }
    },
    "security": "nosec_sc",
    "properties": {
        "status": {
            "type": "object",
            // Describe status schema here
            "forms": [
                {
                    "href": "/things/lamp/actions/fade/123e4567-e89b-12d3-a456-426655",
                    "htv:method": "GET"
                }
            ]
        }
    },
    "actions": {
        "cancel": {
            "forms": [
                {
                    "href": "/things/lamp/actions/fade/123e4567-e89b-12d3-a456-426655",
                    "htv:method": "DELETE"
                }
            ]
        }
    }
}
```

Pros

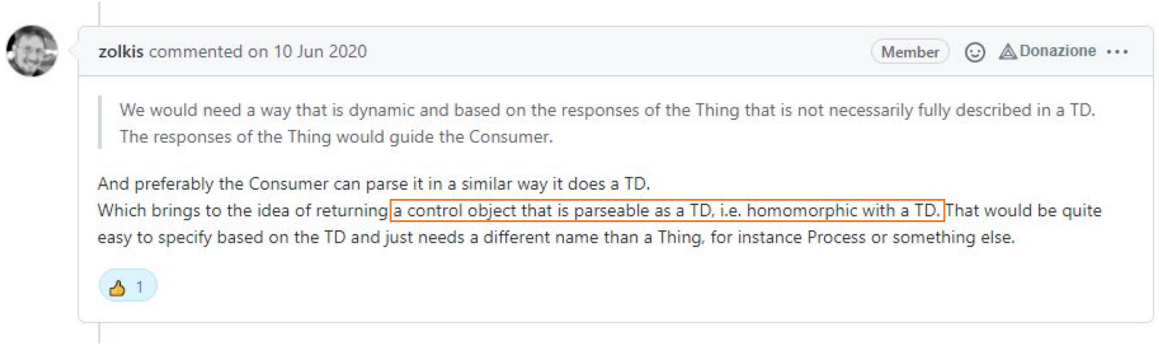
- No id tracking
- Statically describe **Action Request** operations thanks to the TM
- Backward compatible
- Flexible (it covers **green field** and **brown filed** devices). it might as well be used for other use cases: used in properties it might describes operations that can be performed on collections of Web Things (to be verified)
- As a side effect we get the ability to specify a TM as a validation parameter of affordances (w3c/wot-discovery#182)
- Compact: it does not add too many new parameters
- Lightweight: Consumers of a particular profile can just use the model reference as a tag and process the returned object according to their specification.

About Action Thing Description semantic

Thing Descriptions should describe physical devices.
Action Requests are not physical devices.

Semantically speaking we should expand our ontology model to cover also this use case. Options:

- Be less restrictive with what can be described by a Thing Description (we already forcing the definition a little bit with TDDs and ThingLinks)
- Introduce a new entity type



zolkis commented on 10 Jun 2020

> We would need a way that is dynamic and based on the responses of the Thing that is not necessarily fully described in a TD. The responses of the Thing would guide the Consumer.

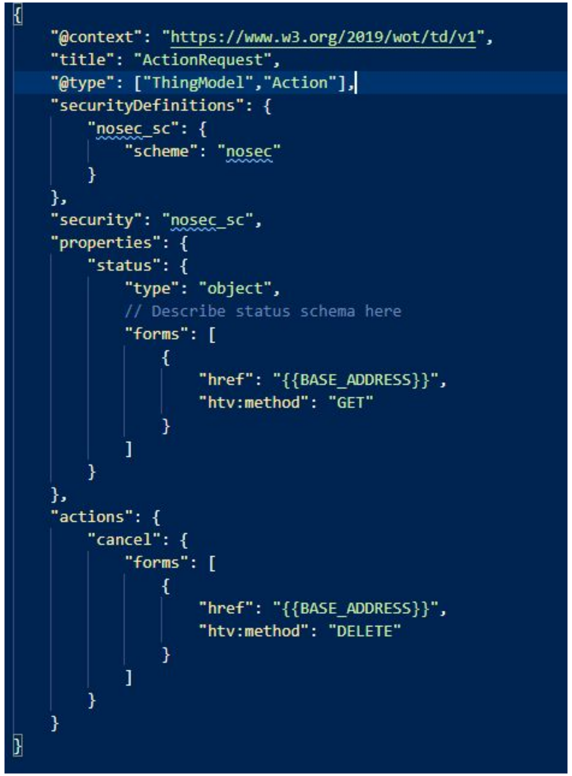
And preferably the Consumer can parse it in a similar way it does a TD.
Which brings to the idea of returning a control object that is parseable as a TD, i.e. homomorphic with a TD. That would be quite easy to specify based on the TD and just needs a different name than a Thing, for instance Process or something else.

Open question do we still need new operation types?

Operation types might be used in the TM as well. But are they useful?

Consumers will use the TM as a semantic annotation for forms.

We might need to standardize a common Thing Model for actions or define **Affordances Types** to tag Action TM .



```
{
    "@context": "https://www.w3.org/2019/wot/td/v1",
    "title": "ActionRequest",
    "@type": ["ThingModel","Action"],
    "securityDefinitions": {
        "nosec_sc": {
            "scheme": "nosec"
        }
    },
    "security": "nosec_sc",
    "properties": {
        "status": {
            "type": "object",
            // Describe status schema here
            "forms": [
                {
                    "href": "{{BASE_ADDRESS}}",
                    "htv:method": "GET"
                }
            ]
        }
    },
    "actions": {
        "cancel": {
            "forms": [
                {
                    "href": "{{BASE_ADDRESS}}",
                    "htv:method": "DELETE"
                }
            ]
        }
    }
}
```

Open question do we still need new operation types?

Action Thing Descriptions work well for indicating operations that can be performed on a **Action Request**. However, we are still missing to describe endpoints for action queues.

In this case, a new operation type works well because the action queue is a static resource. For example: **queryactions** and **cancelactions**

Conclusions

The proposal is taking the best of the current approaching. It is able to describe dynamic resources without losing the ability to statically describe the operations. Moreover, it solves the id tracking problem leveraging on **Thing Models** mapping rules. Finally it uses the new operation types to describe **action queues**.