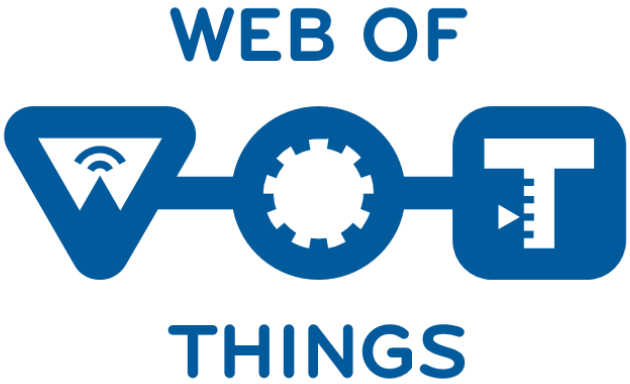


Yet another Action model

Trying to describe action queues and dynamic resources
TestFest resutls

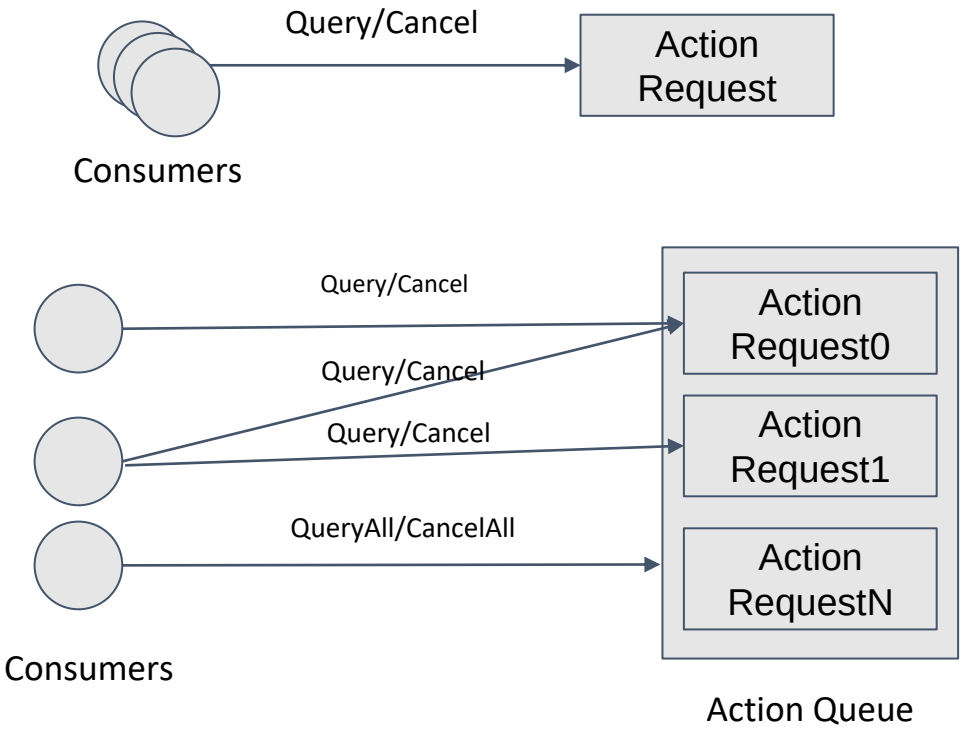
Problem

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

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"
                }
            ]
        }
    }
}
```

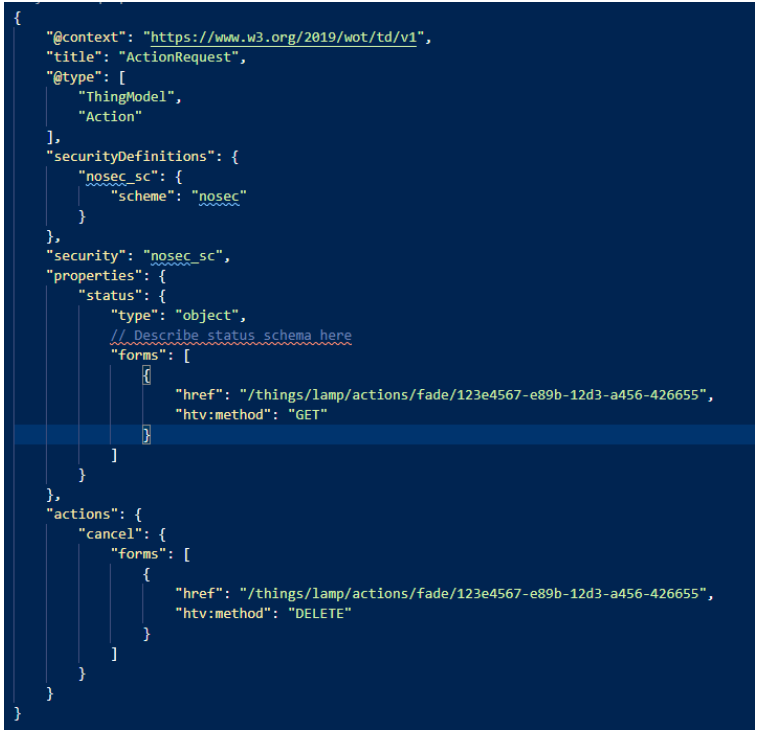
Proposal: GreenField (core profile?)

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.

Results

During the plug-fest I tried to create **two** different services using the proposed approach.

- Green field device that returns an ActionDescription when the action is asynchronous
- A simulated existing service which returns a json object after invoking the action

https://github.com/relu91/wot-hypermedia-test

Green Field device Exposed TMs

```
{      You, 5 days ago • first commit
    "@context": [
        "https://www.w3.org/2019/wot/td/v1"
    ],
    "@type": [
        "tm:ThingModel"
    ],
    "title": "TestingActions",
    "actions": {
        "longRunning":{
            "title": "long",
            "description": "",
            "input": {},
            "output": {
                "model": {
                    "href":"http://localhost:8888/model.tm.json"
                }
            }
        }
    }
}
```

Exposed Thing Model

```
{      You, 5 days ago • first commit
    "@context": [
        "http://www.w3.org/ns/td"
    ],
    "@type": [
        "tm:ThingModel"
    ],
    "title": "LongAction",
    "actions": {
        "cancel": {
            "title": "Cancel",
            "description": "Cancel action execution"
        }
    },
    "properties": {
        "status":{
            "type": "object",
            "properties": {
                "status": {
                    "type": "string",
                    "enum": ["pending","running","cancelled","completed"]
                },
                "id":{
                    "type": "string"
                }
            },
            "required": ["status","id"],
            "title": "status",
            "description": "Action status"
        }
    },
    "events": {}
}
```

Action Description model

Green field device-consumer

```
You, 5 days ago • first commit
WoTHelpers.fetch("http://localhost:8080/testingactions").then( async td => {
    const thing = await WoT.consume(td);

    if (!td.actions.longRunning || td.actions.longRunning.output.model.href !== "http://localhost:8888/model.tm.json") {
        console.log("Long Running action is synchronous");
        await thing.invokeAction("longRunning");
        return;
    }

    console.log("Long Running action is asynchronous");

    const actionInstance = await (await thing.invokeAction("longRunning")).value();
    console.log("Action invoked",actionInstance.id);

    const actionThingHandler = await WoT.consume(actionInstance);
    console.log("Monitoring", actionInstance.id);

    const interval = setInterval(async () => {
        try {
            const status = await (await actionThingHandler.readProperty("status")).value();
            console.log("Status", status);

            if (status.status === "completed" || status.status === "cancelled") {
                console.log("Action has stopped:", status.status);
                clearInterval(interval);
            }

            if(Math.random() < 0.1){
                // cancel with action with 10% probability
                console.log("Cancelling action");
                await actionThingHandler.invokeAction("cancel");
            }

        } catch (error) {
            console.log("Can't read", error);
        }

    }, 1000);

}).catch(e => {
    console.log(e);
});
```

Green field device-exposer

```
const model = require('./expose.tm.json');
const actionModel = require('./model.tm.json');
model['@type'] = 'Thing';
actionModel['@type'] = 'Thing';
const actions = [];


WoT.produce(model).then(async exposer => {

    exposer.setActionHandler('longRunning', async () => {
        console.log('Spawn new action');
        const actionInstance = new Action(uniqueID());

        actions.push(actionInstance);

        actionModel['@id'] = `urn:dev:${actionInstance.id}`;
        actionModel['id'] = `urn:dev:${actionInstance.id}`;

        const actionThing = await WoT.produce(actionModel)

        actionThing.setPropertyReadHandler('status', async () => {
            return {
                id: `urn:dev:${actionInstance.id}`,
                status: actionInstance.status
            }
        });

        actionThing.setActionHandler('cancel', async () => {
            actionInstance.cancel();
        });

        await actionThing.expose();
        actionInstance.start();
        return actionThing.getThingDescription();
    });

    await exposer.expose();
}).catch(err => {
    console.error(err);
});
```

Existing service/device TMs

```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1"
    ],
    "@type": [
        "tm:ThingModel"
    ],
    "title": "TestingActions",
    "actions": {
        "longRunning": {
            "title": "long",
            "description": "",
            "input": {},
            "output": {
                "type": "object",
                "properties": {
                    "status": {
                        "type": "string",
                        "enum": [
                            "pending",
                            "running",
                            "cancelled",
                            "completed"
                        ]
                    },
                    "id": {
                        "type": "string"
                    }
                },
                "model": {
                    "href":"http://localhost:3000/model-brown.tm.json",
                    "mappings": {
                        "/id" : "ID"
                    }
                }
            }
        }
    }
}
```

Exposed Thing Model

```
{
    "@context": [
        "http://www.w3.org/ns/td"
    ],
    "@type": [
        "tm:ThingModel"
    ],
    "title": "LongAction",
    "actions": {
        "cancel": {
            "title": "Cancel",
            "description": "Cancel action execution",
            "forms": [
                {
                    "href": "http://localhost:3000/actions/{{ID}}/cancel"
                }
            ]
        }
    },
    "properties": {
        "status": {
            "type": "object",
            "properties": {
                "status": {
                    "type": "string",
                    "enum": [
                        "pending",
                        "running",
                        "cancelled",
                        "completed"
                    ]
                },
                "id": {
                    "type": "number"
                }
            },
            "required": [
                "status",
                "id"
            ],
            "title": "status",
            "description": "Action status",
            "forms": [
                {
                    "href": "http://localhost:3000/actions/{{ID}}/status"
                }
            ]
        }
    },
    "events": {}
}
```

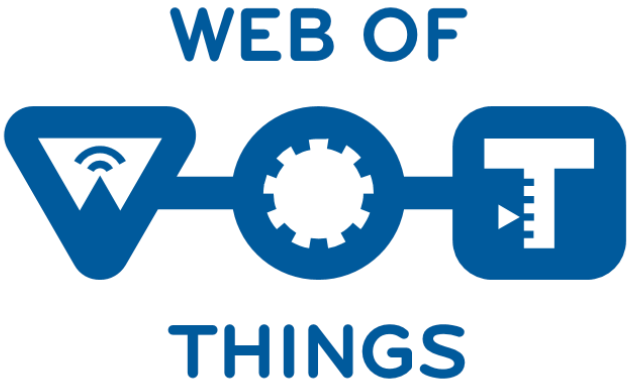
Action Description model

Existing service/device consumer

```
22      const actionInstance = await (await thing.invokeAction("longRunning")).value();
23      console.log("Action invoked", actionInstance.id);
24
25      //Create a "Action Description" from actionInstance and tm
26      const tm = await WoTHelpers.fetch(td.actions.longRunning.output.model.href);
27      let tmString = JSON.stringify(tm);
28
29      const mappings = td.actions.longRunning.output.model.mappings;
30      for (const p in td.actions.longRunning.output.model.mappings) {
31          const variable = mappings[p];
32          const value = pointer.get(actionInstance, p);
33          const exp = new RegExp(`{{${variable}}}`, 'g');
34
35          tmString = tmString.replace(exp, value);
36
37      }
38
39      const actionDescriptor = JSON.parse(tmString);
40      tm['@type'] = 'Thing'; // workaround for node-wot
41
42      const actionThingHandler = await WoT.consume(actionDescriptor);
43
```

What we need to standardize

- A common action model
- The concept of Action Description
- The TD keywords and semantics used in this proposal
- Operation types for queues are still needed



Yet another Action model

Trying to describe action queues and dynamic resources
TestFest resutls