Digital Twin Workshop: Convention for Actuation Workflows

Giuseppe Tropea

ETSI ISG CIM

05/07/2022

Actuators and feedback to the application

There is currently no specified support for actuation in the NGSI-LD API

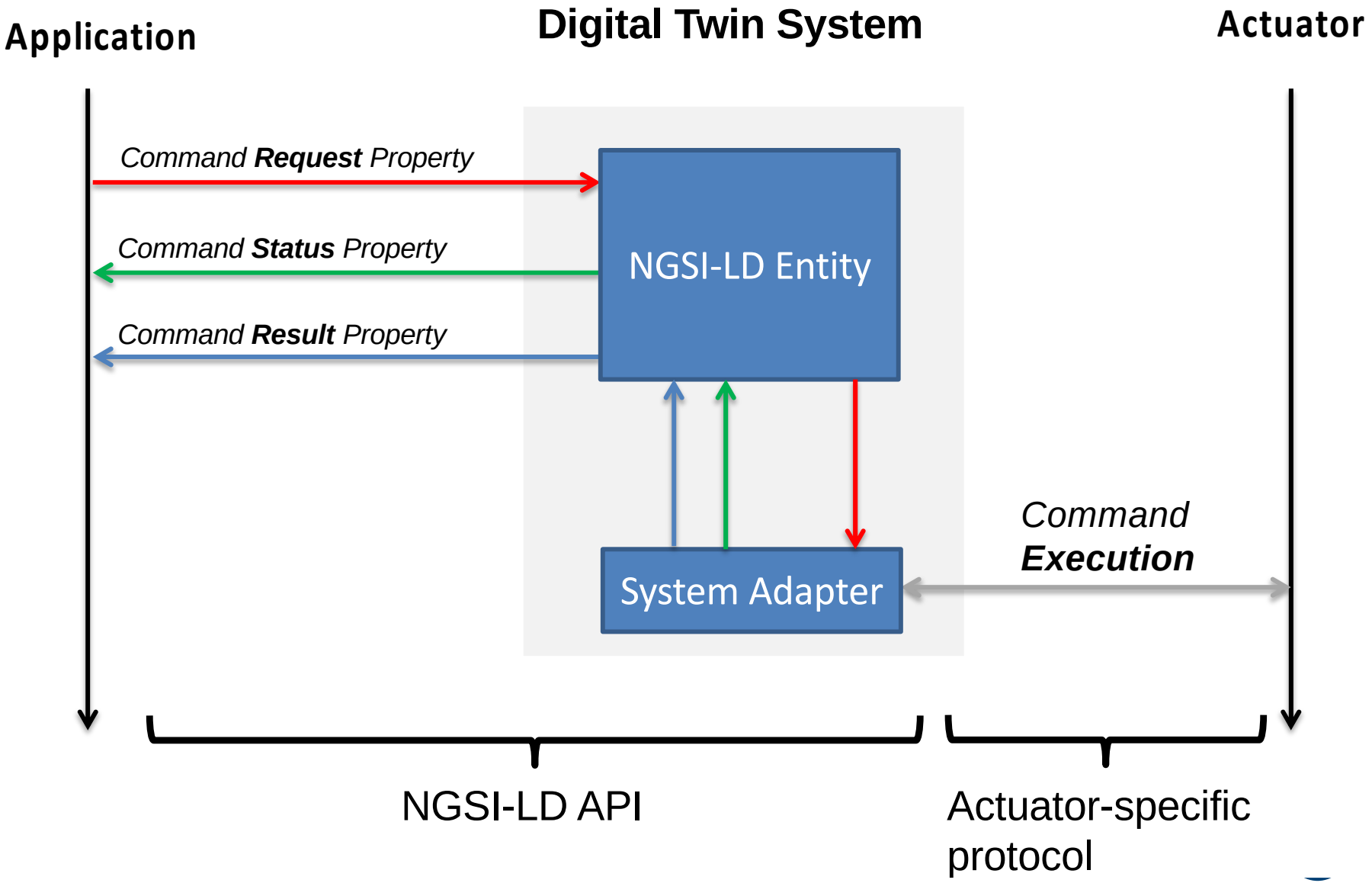
We propose a **best-practice** for the applications-actuators interaction

- The application-actuator interaction needs to be bidirectional
- Different levels of feedback can be requested
  - high operation rate but no feedback. QoS = 0
    - control the arms of a robot with a joystick.
  - get back a payload in response to the command. QoS = 1
    - switching on a light with confirmation
  - continuous status feedback. QoS = 2
    - a door opening (10% open, 50% open, …)

Architecture for actuation

To support actuation, there is a need to specify:

- Additional NGSI-LD Properties to manage command Request, Status, Result
- A communication model that allows commands to flow in forward direction and feedback to flow in reverse



Property for listing available commands

```
"commands": {
  "type": "Property",
  "value": ["<cmd_name1>","<cmd_name2>", …, "<cmd_nameN>"]
}
```

It is a Property whose value is an array of Strings, each string representing the unique name of a supported command

Properties for command endpoints

This convention dictates that:

- The Property that manages requests has the same name as the command, e.g. “<cmd_name1>”
- The Property that manages status the “-STATUS” suffix
- The Property that manages results the “-RESULT” suffix

Command Request, Status and Result endpoints

```
"<cmd_name>": {
  "datasetId": a URI uniquely identifying the specific command request
               (optional, if the use case does not need command queueing),
  "type":      "Property",
  "qos":       an Integer, representing the desired QoS (optional, default=0),
  "value":     custom parameters of the command (mandatory)
}

"<cmd_name>-STATUS": {
  "datasetId": a URI uniquely identifying the specific status feedback message
               (optional, if the use case does not need queueing them),
  "type":      "Property",
  "value":     custom status of the command (mandatory)
}

"<cmd_name>-RESULT": {
  "datasetId": a URI uniquely identifying the specific result feedback message
               (optional, if the use case does not need queueing them),
  "type":      "Property",
  "value":     custom result of the command (mandatory)
}
```

Example of an Entity representing a light that can change colour

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",

  REGULAR PROPERTIES
  "colorRGB": {"type": "Property", "value": "0xABABAB"},
  "is-on": {"type": "Property", "value": true},

  AVAILABLE COMMANDS
  "commands": {
    "type": "Property",
    "value": ["turn-on", "set-saturation", "set-hue", "set-brightness"]
  }

  COMMAND ENDPOINTS
  "turn-on": {"type": "Property", "value": <custom request>}
  "turn-on-STATUS": {"type": "Property", "value": <custom status>}
  "turn-on-RESULT": {"type": "Property", "value": <custom response>}
  "set-hue": ...
  "set-hue-STATUS": ...
  "set-hue-RESULT": ...
}
```

Example requests

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",
  "turn-on": {
    "type": "Property",
    "qos": {
      "type": "Property",
      "value": 1
    },
    "value": false
  }
}
```

Turn the light off

```
{
  "id": "urn:ngsi-ld:pHueActuator:light1",
  "type": "Lamp",
  "set-hue": {
    "type": "Property",
    "qos": {
      "type": "Property",
      "value": 1
    },
    "datasetId": {
      "type": "Property",
      "value": "myapp:mycommand:1342"
    },
    "value": {"red":"1 seconds", "green": "2 seconds"}
  }
}
```

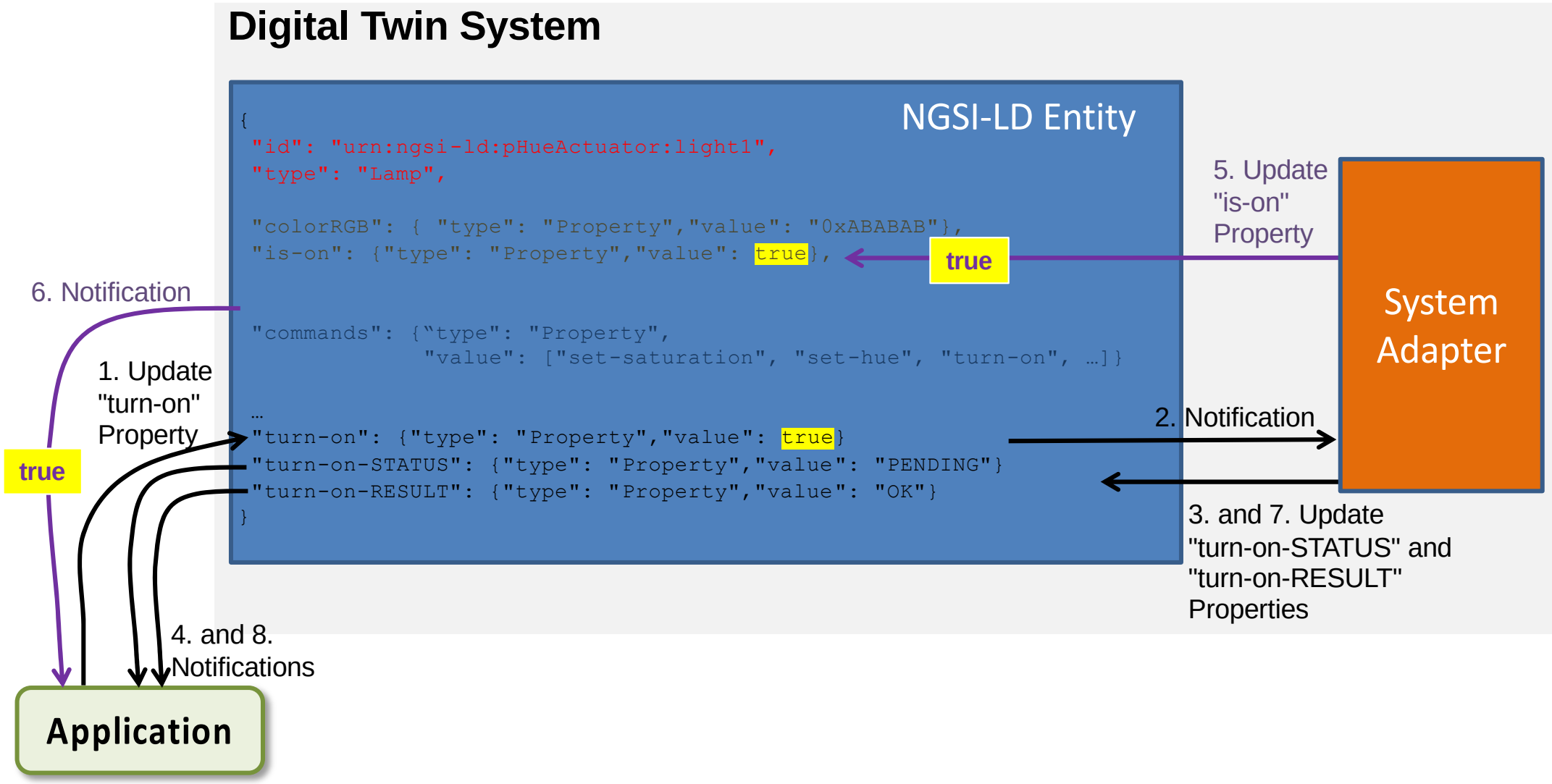
Send complex command to light, from a specific application

Communication model

This convention can be leveraged by two different communication models:

- Subscription/notification
  - where both the **application and the System Adapter use NGSI-LD Subscriptions** to have the command requests delivered to the appropriate handler within the System Adapter and vice-versa;
- Forwarding
  - which uses the **NGSI-LD Registry and a System Adapter able to federate** itself with the Context Broker holding the Digital Twin's Entity, as a means to deliver the commands.

Subscription/notification model



Forwarding model