Thing Description pain points

Luca Barbato - Luminem SRLs

Thing Description pain points

I found few areas in which the Thing Description is painful to implement

- **Uniformity**
- **Cross-specification interactions**
- **Interaction with other subspecification (profiles, protocol bindings)**
- **Expectations regarding resource usage**

Thing Description pain points

This presentation focuses on:

- Uniformity
  - Ideally you'd like that each and every part of your specification that is conceptually similar behaves in the same way.
  - We have few places in which the behavior is different even if the underlying concept is similar.
- Cross-specification interactions
  - We refer to other specification
    - but sometimes it is not clear what is the expectation in their regard.

Uniformity - DataSchema

Different meaning of one or array

- In the Thing Description some fields have the pattern of being `Item` or `Array<Item>`
  - the single item and an array of a single item providing the same meaning.
  - `ArraySchema::items` is the exception and should be clearly marked so.

Uniformity - Affordances

We have 3 kind of affordances, but their structure and usage are non-uniform.

- They all hold some `DataSchema` in some different way
  - For describing the input (`property` , `action`)
  - For describing the output (`event` , `action`)
- They all have to relate with their `Form` and `uriVariables` to map the information they would exchange with consumers

Affordances - Property and Event

- **Event** has 3 DataSchema fields:
  - `data`, equivalent to what the Property inherits directly
  - `dataResponse` that has unclear usage
  - `cancellation` that is useful to unsubscribe
- **Property** inherits a DataSchema and that's all
  - But you can observe a property...
    - And we'd need at least a `cancellation` field to make it as usable as `Event`.

Affordances - Action and Event

- **Action** has an `input` and an `output` fields
  - They are a bit more clear compared to `data` and `dataResponse` in **Event**.
- **Action** can be asyncronous, but we do not have a `subscribeaction`, while we have `cancelaction`
  - Polling using the `queryaction` isn't great.
    - And it is even more annoying if your protocol is *pub/sub*-oriented.

Affordances - Action and Property

Actions and R/W Properties are very similar in concept, but very different structurally.

- **Action** should let you indirectly manipulate the internal states of the thing over time.
    - Yet you cannot subscribe to get updates on how your action invocation is going.
        - The simplest workaround is to have a *read-only* **Property** and a matching **Action**
            - But there is no way to express the relationship between the two beside additional vocabularies.
- **Property** should expose an internal state of a Thing and let you directly read, observe or manipulate it.
    - Implies that the change is immediate and syncronous
        - even if it is never the case.

Affordances - Forms and Affordances

- All the `Affordances` have a `uriVariables` that clash/interact badly with `input` and the implicit **Property** `DataSchema`
- The `Form` contains both `response` and `additionalResponse` fields that clash with `dataResponse` / `data` and `output`.
- The `Form` does not have a good way to express which parts of the Affordance `DataSchema`s should map to its protocol if it has ancillary channels
    - (e.g. in HTTP we have Headers vs URI vs actual body)
- `uriVariables` seem to be additive over the Affordance `DataSchema`s, but they are specific to a certain use of URIs.

Relationship other specifications

- We reference to plenty of specifications
  - Many from IETF
  - As many from W3C
- The normal behavior is to use the specification referred as it is and in full.
  - In two cases we do not
    - JSON-LD and json-schema

Relationship other specifications

Relationship with JSON-LD

- The consensus is that you should be able to consume a TD without the requirement of a full JSON-LD processor
  - We'd like to allow the serialization of a TD in not-json sooner or later (e.g. CBOR)
- We should make clear how we can ensure interoperability
  - We plan to keep a registry of prefixes for `Protocol/Platform bindings`
  - `Profiles` may/should require that you must use those prefixes
    - disallow JSON-LD transformations
    - require to provide them explicitly in the context

Relationship with json-schema

- The DataSchema is related to the json-schema
  - It is unclear if it is a subset of it or a superset of it
    - What to do if what is in the TD and in the json-schema specification conflicts?
- The json-schema specification is also evolving
  - We cannot sensibly just point to it.
    - Can we?

That's all for today!