The Semantic Definition Format (SDF):
A brief tutorial and status

Carsten Bormann
The need for One Data Model

- IoT standardization is dominated by ecosystem-specific SDOs
- Each ecosystem has their own data models, and their own way to document them
- IoT applications may need to work with things from multiple ecosystems: No single ecosystem can supply the whole variety needed
- Can build protocol translators; harder to translate hundreds of data models
The One Data Model liaison group

- People from different SDOs meet in an informal liaison group
- Bring together hundreds of ecosystem-specific data models
  - Express in common format
  - Work on merging and harmonizing data models
  - Make harmonized data models available for all SDOs (BSD license!)
  - Working in the open: https://github.com/one-data-model
- Inevitably: standardize on a common format: SDF
SDF: The Semantic Definition Format

- [https://github.com/ietf-wg-asdf/SDF](https://github.com/ietf-wg-asdf/SDF)
- Defines classes of *things* (sdfObject, combine into sdfThing)
- Things don’t have data, they have **interactions** with their *clients*(*), provided by **affordances**
- Interaction affordances grouped into **interaction patterns**: For now, **Property, Action, Event**
- Interactions input and output **data** (groupable into sdfData)

(*) Not a oneDM term
Overall Specification Structure

- One or more JSON documents; linked together with JSON pointers [RFC6901]
- An SDF specification can **reuse** elements (such as sdfData definitions) of other SDF specifications
  - Goal: define a basic core set that every specification can reference (“common reusable definitions”)
Interaction Patterns

- SDF is about modeling data

- Interaction Patterns mostly defined along input and output data

<table>
<thead>
<tr>
<th>Name</th>
<th>cf. REST</th>
<th>Initiative</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>GET</td>
<td>Client</td>
<td>—</td>
<td>Data</td>
</tr>
<tr>
<td>Property (writable)</td>
<td>PUT</td>
<td>Client</td>
<td>Data</td>
<td>(Data)</td>
</tr>
<tr>
<td>Action</td>
<td>POST</td>
<td>Client</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>Event</td>
<td>?</td>
<td>Thing</td>
<td>—</td>
<td>Output</td>
</tr>
</tbody>
</table>
- Actions can have different input and output data.

- Some actions take time (not modeled): Initiative to return output moved to Thing (~ Event).

<table>
<thead>
<tr>
<th>Action</th>
<th>cf. REST</th>
<th>Initiative</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>GET</td>
<td>Client</td>
<td>—</td>
<td>Data</td>
</tr>
<tr>
<td>Property (writable)</td>
<td>PUT</td>
<td>Client</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Action</td>
<td>POST</td>
<td>Client</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>Event</td>
<td>?</td>
<td>Thing</td>
<td>—</td>
<td>Output</td>
</tr>
</tbody>
</table>
### Property

- Property is used for data items that can be read by the client.

- Writable properties can also be “set” (no special output).

- Observable properties look like an Event.

<table>
<thead>
<tr>
<th>Name</th>
<th>cf. REST</th>
<th>Initiative</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>GET</td>
<td>Client</td>
<td>—</td>
<td>Data</td>
</tr>
<tr>
<td>Property (writable)</td>
<td>PUT</td>
<td>Client</td>
<td>Data</td>
<td>(Data)</td>
</tr>
<tr>
<td>Property (observable)</td>
<td>GET (observe)</td>
<td>Client, Thing</td>
<td>—</td>
<td>Data</td>
</tr>
<tr>
<td>Event</td>
<td>?</td>
<td>Thing</td>
<td>—</td>
<td>Output</td>
</tr>
</tbody>
</table>
Event

- Least well-defined interaction pattern

- Is an Event just a notification (similar to observable property)?

- Are Events just status updates (temperature) or is any single one of them precious (coin insertion)?

<table>
<thead>
<tr>
<th>Name</th>
<th>cf. REST</th>
<th>Initiative</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>GET</td>
<td>Client</td>
<td>—</td>
<td>Data</td>
</tr>
<tr>
<td>Property (writable)</td>
<td>PUT</td>
<td>Client</td>
<td>Data</td>
<td>Data</td>
</tr>
<tr>
<td>Action</td>
<td>POST</td>
<td>Client</td>
<td>Input</td>
<td>Output</td>
</tr>
<tr>
<td>Event</td>
<td>?</td>
<td>Thing</td>
<td>—</td>
<td>Output</td>
</tr>
</tbody>
</table>
Data

• Data is defined by their shape (as in data definition/“schema” languages)

• Data definitions can be made inline in an affordance definition or separately, for later reference

• Definitions can use curated subset of json-schema.org terms, and/or SDF-specific terms such as contentFormat, nullable, scale…

• Mapping information (protocol bindings) helps bind these data to ecosystem specific formats and encodings
Data Model vs. Information Model (1)

• SDF 1 uses json-schema.org-style data modeling, enhanced by SDF qualities

• Really: This should be information models (RFC 3444):
  • Abstract from arbitrary representation decisions
  • Don’t commit to specific numbers, strings, etc. (bindings can do that)
  • Bind to semantics via RDF-style links

SDF next
Data Model vs. Information Model (2)

- "Enums": choices of values (strings, integers), each usually denoting some specific concept

- Information model: Don’t commit to specific representation (bindings can do that)

- Do bind to semantics via RDF-style links


sdfThing, sdfProduct

• sdfObject definitions can be combined into top-level structures

• sdfThing can contain sdfObject and sdfThing

• sdfProduct similar, as a (not to be harmonized) top-level product definition