First International Workshop on Semantic Web on Constrained Things

Generating Visual Programming Blocks based on Semantics in W3C Thing Descriptions

Michael Freund, Justus Fries, Thomas Wehr, Andreas Harth
28/05/2023
Agenda

1. Introduction and Motivation
2. From TDs to Blocks and Code
3. Mapping Algorithm
4. Performance Evaluation
5. Conclusion and Future Work
Introduction and Motivation

Initial Situation

- Constrained devices are used in industry and consumer applications to sense and act on the environment.

- W3C Web of Things:
  - Simplify device interaction
  - Utilize semantic API descriptions (TD)

- Experts can use WoT Scripting API for text based programming languages

- Everyday users can use graphical tools
Introduction and Motivation

What is the problem?

- A Block requires:
  - a structural definition – describing the layout
  - a source code generator function – defining code that is generated

```javascript
export function generateReadPropertyCode(
    propertyName: string,
    deviceName: string
) {
    JavaScript[`\${deviceName}._readPropertyBlock_${propertyName}`] = function (block: Block) {
        const name = JavaScript.valueToCode(block, 'thing', JavaScript.ORDER_NONE) || null;
        const code = `\await (await things.get(${name}).readProperty('${propertyName}')).value()`;
        return [code, JavaScript.ORDER_NONE];
    }
}
```
Introduction and Motivation

What is the problem?

- A Block requires:
  - a structural definition – describing the layout
  - a source code generator function – defining code that is generated

→ In a Wot context: all interaction affordances of a device need a separate block and code definition.

**Problem I.** All definitions must be implemented by hand, even if a TD is available

→ Limits the number of supported devices in visual programming environments (VPE)

**Problem II.** Starting from a TD, it is hard to discover related devices

→ Limits the number of devices to interact with
Introduction and Motivation

Why is it interesting and important?

- TDs are implemented with machine readability in mind
- An algorithm could use the semantic information contained in a TD to generate blocks/code and follow links
  - Extends the flexibility of VPEs
  - Allows users to interact with arbitrary constrained devices (TD)
  - Improves device discoverability
1. Introduction and Motivation
2. From TDs to Blocks and Code
3. Mapping Algorithm
4. Performance Evaluation
5. Conclusion and Future Work
From TDs to Blocks and Code

Structure of a Thing Description

- RDF document in JSON-LD serialization
- Keywords are mapped to ontology terms via a context (e.g. title → td:title, op → hctl:hasOperationType)

Structure of generated Blocks

- TDs consist of mandatory and optional property keywords → information in generated blocks varies
- Follow abstraction of WoT Scripting API, to simplify the transition to text based programming
- Two Phases:
  
  **Creation phase:**
  - TD is consumed
  - Thing object is created
  
  **Interaction phase:**
  - Thing object used to call functions
  - readProperty, writeProperty, invokeAction, subscribeEvent

- To read a property:
  
  ```javascript
  thing.readProperty('status');
  ```

  ```javascript
  read property 'status' of LampThing thing
  ```
From TDs to Blocks and Code

Example TD

- Metadata

```
"@context": "https://www.w3.org/2022/wot/td/v1.1",
"@type": "Thing",
"id": "urn:dev:ops:32473-WoT-Thing-1234",
"title": "LampThing",
"titles": { "en": "LampThing", "de": "LampenDing"},
"description": "A lamp",
"descriptions": { "en": "A lamp", "de": "Eine Lampe"},
"version": "1.0", "created": "2020-10-10T17:00:00Z",
"modified": "2022-10-10T17:00:00Z",
"support": "https://example.org/lamp",
"links": [{
  "href": "http://example.com/related-td",
  "type": "application/td+json"
}]
"securityDefinitions": {
  "basic_sc": {"scheme": "basic", "in": "header"}
},
"security": ["basic_sc"],
```
Example TD

- Metadata
- Properties:
  - status (read)

```json
"properties": {
  "status": {
    "title": "status",
    "titles": {"en": "status", "de": "Zustand"},
    "description": "Read the status of the lamp",
    "descriptions": {
      "en": "Read the status of the thing",
      "de": "Auslesen des Lampenzustands"
    },
    "type": "string",
    "forms": [...]
  }
},
```
Example TD

- Metadata
- Properties:
  - status (read)
- Actions:
  - toggle

```
"actions": {
  "toggle": {
    "title": "toggle",
    "titles": {"en": "toggle", "de": "umschalten"},
    "description": "Toggle current lamp status",
    "descriptions": {
      "en": "Toggle current lamp status",
      "de": "Umschalten des aktuellen Lampenstatus"
    },
    "output": {"type": "string"},
    "forms": [...]}
}
```
From TDs to Blocks and Code

Example TD

- Metadata
- Properties:
  - status (read)
- Actions:
  - toggle
- Events:
  - overheating

```
"events": {
  "overheating": {
    "title": "overheating",
    "titles": {
      "en": "overheating",
      "de": "Ueberhitzung"
    },
    "description": "An overheating event of the lamp",
    "descriptions": {
      "en": "An overheating event of the lamp",
      "de": "Ein Ueberhitzungs Event der Lampe"
    },
    "data": {
      "type": "string"
    },
    "forms": [...]
  }
}
```
From TDs to Blocks and Code

Mapping of Thing Vocabulary

- Only `@context, title, security, and securityDefinitions` are mandatory

<table>
<thead>
<tr>
<th>Block Property</th>
<th>TD Property Keyword</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Name</td>
<td>titles or title</td>
<td>If available</td>
</tr>
<tr>
<td>Block Color</td>
<td></td>
<td>Flexible to choose</td>
</tr>
<tr>
<td>Block Output</td>
<td></td>
<td>Output type 'thing'</td>
</tr>
<tr>
<td>Tooltip</td>
<td>description(s), version, modified, created</td>
<td>If available</td>
</tr>
<tr>
<td>Help URI</td>
<td>support</td>
<td>If available</td>
</tr>
</tbody>
</table>

![LampThing and LampenDing images]
Mapping of Property Affordance Vocabulary

- Properties are available in two types: `readProperties` and `writeProperties`
From TDs to Blocks and Code

Mapping of Action Affordance Vocabulary

- 4 different layouts of action blocks (input, output, neither, both)

<table>
<thead>
<tr>
<th>Block Property</th>
<th>TD Property Keyword</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Name</td>
<td>titles, title, action affordance name, op</td>
<td>If available</td>
</tr>
<tr>
<td>Block Color</td>
<td>op</td>
<td>Dependent on op</td>
</tr>
<tr>
<td>Block Output</td>
<td>output, type</td>
<td>If output provided</td>
</tr>
<tr>
<td>Block Input</td>
<td>input, type</td>
<td>If input provided</td>
</tr>
<tr>
<td>Tooltip</td>
<td>description(s), default</td>
<td>If available</td>
</tr>
</tbody>
</table>

(a) invoke action 'toggle' with value of

(b) invoke action 'toggle' with value of

(c) invoke 'toggle' action of

(d) invoke 'toggle' action of
From TDs to Blocks and Code

Mapping of Event Affordance Vocabulary

- Event blocks are statement inputs instead of value inputs
- Data type of ‘eventVar’ defined via data property keyword

<table>
<thead>
<tr>
<th>Block Property</th>
<th>TD Property Keyword</th>
<th>Additional Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block Name</td>
<td>titles, title, event affordance name, op</td>
<td>If available</td>
</tr>
<tr>
<td>Block Color</td>
<td>op</td>
<td>Dependent on op</td>
</tr>
<tr>
<td>Tooltip</td>
<td>description(s), default</td>
<td>If available</td>
</tr>
</tbody>
</table>

on 'overheating' events of
with variable eventVar
do
Link Following Vocabulary

- Link following is a fundamental aspect of the Web to find and explore related Web resources
- Same concept can be used in the Web of Things via the links property keyword
- Only href mandatory

```
"links": [{
    "href": "http://example.com/related-td",
    "type": "application/td+json",
    "rel": "controlledBy"
}
]```
Agenda

1. Introduction and Motivation
2. From TDs to Blocks and Code
3. Mapping Algorithm
4. Performance Evaluation
5. Conclusion and Future Work
Mapping Algorithm

Implementation of an Algorithm

- PoC implementation using JavaScript, the defined mappings, and Google's Blockly library
- Analyse TD and call corresponding creation block and code functions
- Crawler based on focused crawling technique (only application/td+json)
- Crawler uses asynchronous features of JavaScript to follow links recursively

Limitations:
- Only HTTP(S) is supported
- Loading and saving of programs is not supported
- Crawler only follows links described with links property keyword
Agenda

1. Introduction and Motivation
2. From TDs to Blocks and Code
3. Mapping Algorithm
4. Performance Evaluation
5. Conclusion and Future Work
Performance Evaluation

Evaluation Setup

- Consumer hardware (i7-10610U, 16 GB RAM, Windows 10 21H2)
- Timing determined with `performance.now()` with millisecond time resolution
- Total acceptable run time should be below 200 ms
Performance Evaluation

Evaluation of link following algorithm

- Evaluation of run time with an increasing number of links to TDs
- Evaluation of 2 TD types:
  - With 1 link forming a link chain
  - With 2 links forming a link tree
- Discover about 30 Thing Descriptions in 0.1 s
Performance Evaluation

Evaluation of block and code generator

- Theoretical analysis of time complexity resulting in $O(n)$
- Empirical analysis resulting also in a linear timing behavior
- Generates about 4,000 interaction affordance blocks and code in 0.1s
Performance Evaluation

Combined Performance

![Graph showing combined performance with different affordances over time. The x-axis represents time in milliseconds, and the y-axis represents discovered and analyzed TDs. The graph includes data points for 32, 128, and 512 affordances.]
Agenda

1. Introduction and Motivation
2. From TDs to Blocks and Code
3. Mapping Algorithm
4. Performance Evaluation
5. Conclusion and Future Work
Conclusion and Future Work

Conclusion

Problem I.) All definitions must be implemented by hand, even if a TD is available
- Mapping of TD property keywords to block structure definitions and code generator functions
- Implementation of mapping algorithm

Problem II.) Starting from a TD, it is hard to discover related devices
- Link following algorithm to discover related and linked TDs
- In 0.2 seconds the algorithm can discover
  - 25 Thing Descriptions with
  - 128 interaction affordances

Future Work
- Expand generation algorithm to other protocol bindings
- Investigate the link following concept in Thing Descriptions
Thank you for your time
Contact

Michael Freund
Data Spaces and IoT
michael.freund@iis.fraunhofer.de

Fraunhofer IIS
Nordostpark 84
90411 Nürnberg
www.fraunhofer.de