Fraunhofer IIS

Fraunhofer Institute for Integrated Circuits IIS

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

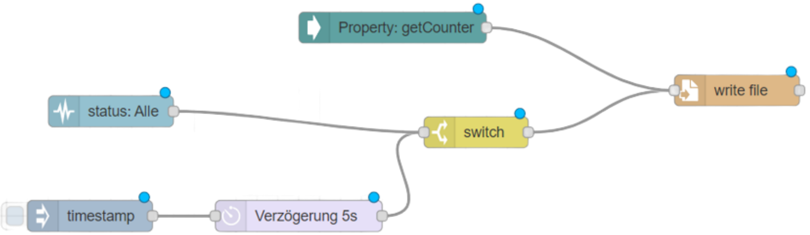
Initial Situation

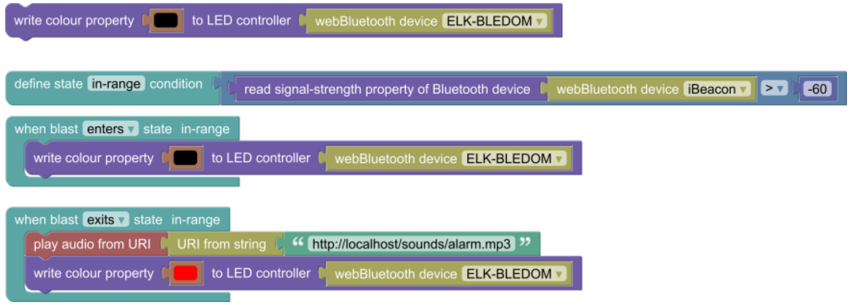
- Constrained devices are used in industry and consumer applications to sense and act on the environment
- W3C Web of Things:
  - Simplify device interaction
  - Utilize semantic API descriptions (TD)
- Experts can use WoT Scripting API for text based programming languages
- Everyday users can use graphical tools

```
const { Servient, Helpers } = require("@node-wot/core");
const { HttpClientFactory } = require('@node-wot/binding-http');

const servient = new Servient();
servient.addClientFactory(new HttpClientFactory(null));
const WoTHelpers = new Helpers(servient);

WoTHelpers.fetch("http://localhost:8080/example").then(async (td) => {
    try {
        servient.start().then(async (WoT) => {
            // Then from here on you can consume the thing
            // i.e let thing = await WoT.consume(td) ...
        });
    }
    catch (err) {
        console.error("Script error:", err);
    }
}).catch((err) => { console.error("Fetch error:", err); });
```





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

```
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];
  };
}
```

```
export function generateReadPropertyBlock(
  propertyName: string,
  deviceName: string,
  td: ThingDescription
) {
    Blocks[`${deviceName}_readPropertyBlock_${propertyName}`] = {
    init: function () {
      this.appendValueInput('thing')
        .setCheck('Thing')
        .appendField(blockName, 'label');
      this.setOutput(true, td.properties?.type ?? null);
      this.setColour(255);
      this.setTooltip(
        td.descriptions?.[langTag] ??
          td.description ??
          `Read the ${propertyName} property of ${deviceName}`
      );
    },
  };
}
```

  public

Fraunhofer IIS

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

Agenda

Fraunhofer IIS

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

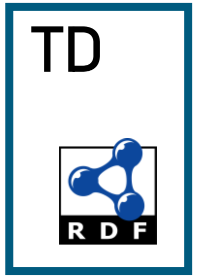
- To read a property:

```
thing.readProperty('status');
```

From TDs to Blocks and Code

Example TD

- Metadata



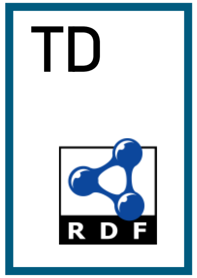

```
"@context": "https://www.w3.org/2022/wot/td/v1.1",
"@type": "Thing",
"id": "urn:dev:ops:32473-WoT-Thing-1234",
"title": "LampThing",
"titles": { "en": "LampThing", "de": "LampenDing"},
"description": "A lamp",
"descriptions": { "en": "A lamp", "de": "Eine Lampe"},
"version": "1.0", "created": "2020-10-10T17:00:00Z",
"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"],
```


From TDs to Blocks and Code

Example TD

- Metadata
- Properties:
  - status (read)

```
"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": [...]
   }
 },
```

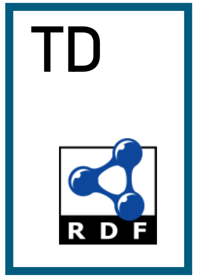


From TDs to Blocks and Code

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": [...],
   }
 },
```

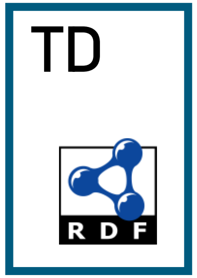


Fraunhofer IIS

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": [...],
   }
 },
```

  public

Fraunhofer IIS

From TDs to Blocks and Code

Mapping of Thing Vocabulary

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

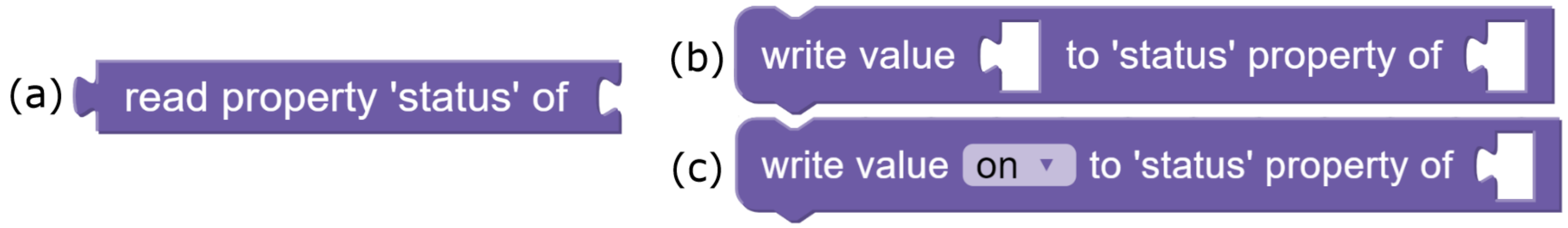
Block Property	TD Property Keyword	Additional Notes
Block Name	titles or title	If available
Block Color	-	Flexible to choose
Block Output	-	Output type 'thing'
Tooltip	description(s), version, modified, created	If available
Help URI	support	If available

From TDs to Blocks and Code

Mapping of Property Affordance Vocabulary

- Properties are available in two types: **readProperties** and **writeProperties**

Block Property	TD Property Keyword	Additional Notes
Block Name	titles, title, property affordance name, op	If available
Block Color	op	Dependent on op
Block Output	op, type	Only if op is read
Block Input	op, type, enum	Only if op is write
Tooltip	description(s), default	If available



  public

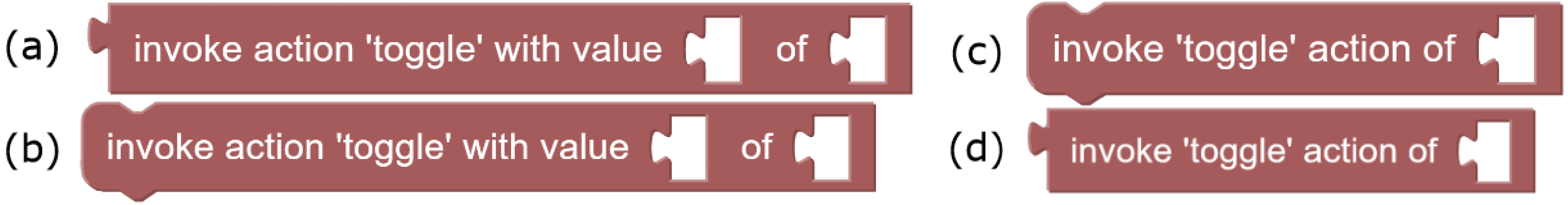
Fraunhofer IIS

From TDs to Blocks and Code

Mapping of Action Affordance Vocabulary

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

Block Property	TD Property Keyword	Additional Notes
Block Name	titles, title, action affordance name, op	If available
Block Color	op	Dependent on op
Block Output	output, type	If output provided
Block Input	input, type	If input provided
Tooltip	description(s), default	If available



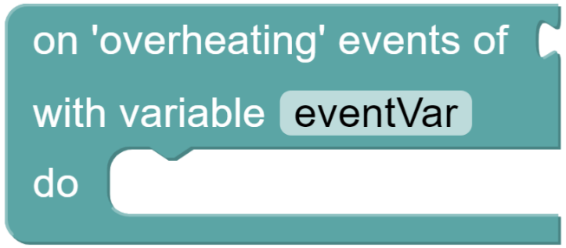
Fraunhofer IIS

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

Block Property	TD Property Keyword	Additional Notes
Block Name	titles, title, event affordance name, op	If available
Block Color	op	Dependent on op
Tooltip	descrioption(s), default	If available



From TDs to Blocks and Code

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": "contolledBy"
          }]
```

  public

Fraunhofer IIS

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

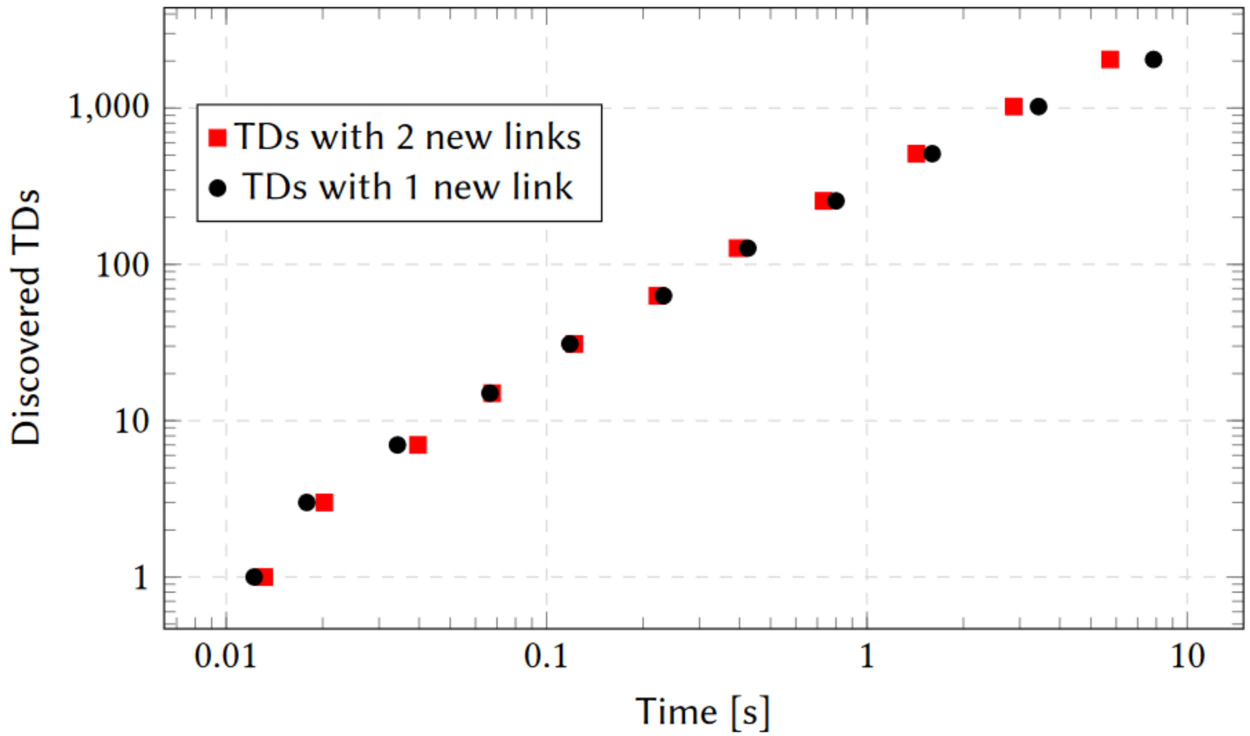
 11/07/2022 public

Fraunhofer IIS

Performance Evaluation

Evaluation of link following algorithm

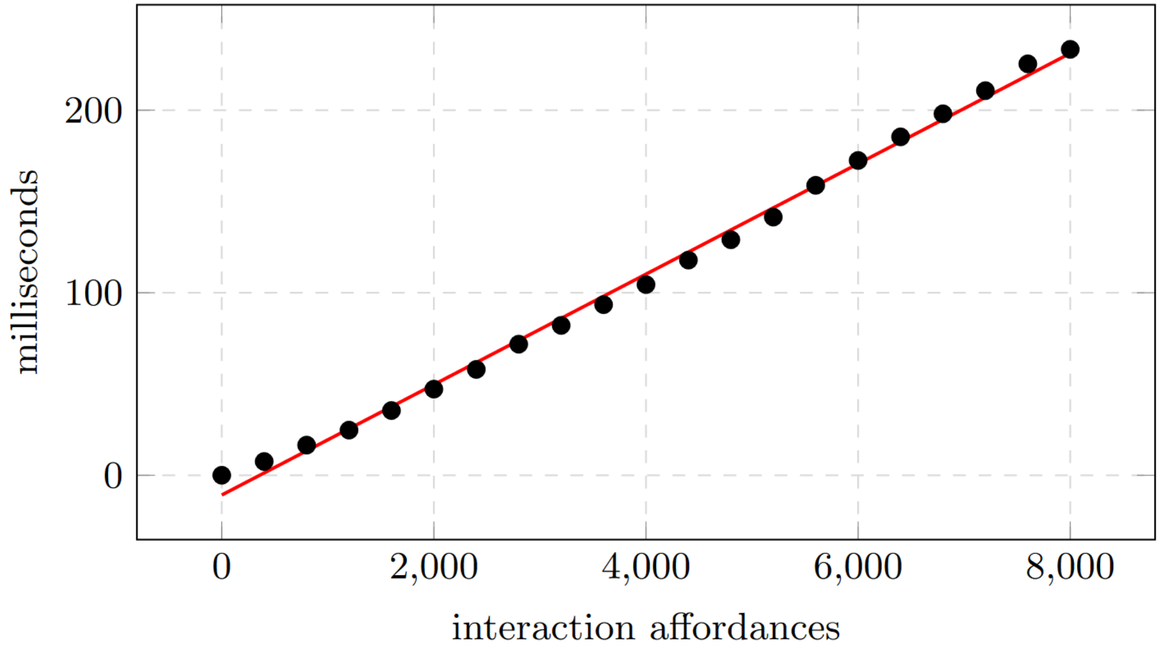
- Evaluation of run time with an increasing number of links to TDs
- Evaluation of 2 TD types:
  - With 1 link forming a link chain
  - With 2 links forming a link tree
- Discover about 30 Thing Descriptions in 0.1 s



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

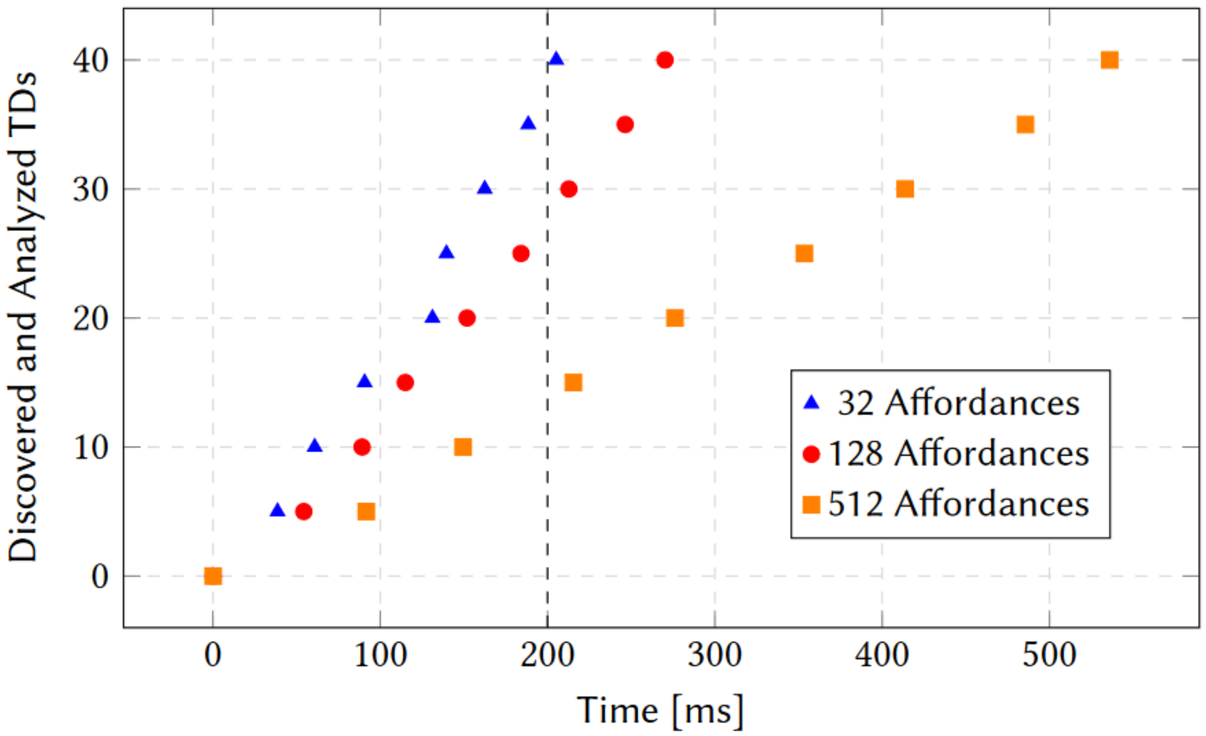


  public

Fraunhofer IIS

Performance Evaluation

Combined Performance



  public

Agenda

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

Fraunhofer IIS

Fraunhofer Institute for Integrated Circuits IIS

Contact

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

Fraunhofer IIS
Nordostpark 84
90411 Nürnberg
www.fraunhofer.de

Fraunhofer IIS

Fraunhofer Institute for Integrated Circuits IIS