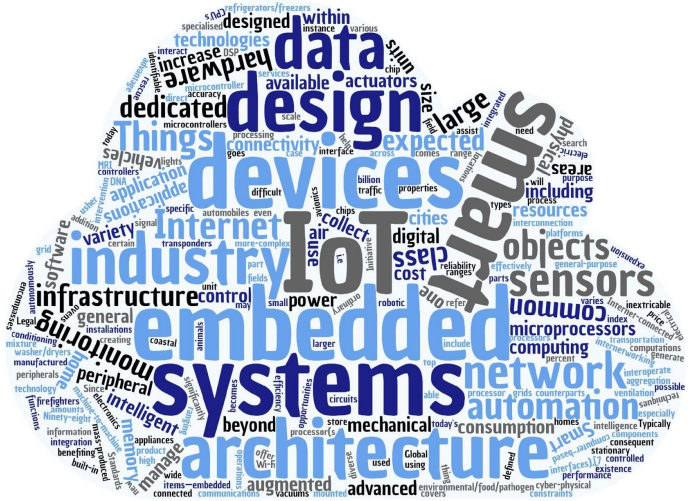
Associate Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

An Introduction to Thingweb node-wot

Ege Korkan
ege.korkan@tum.de

Technical University of Munich / Siemens AG

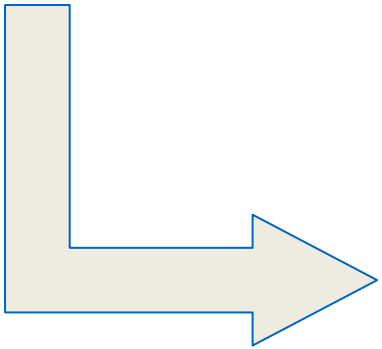


ToC

- Why node-wot, what is the motivation? Developing devices and Consumer applications. No fiddling with all HTTP, MQTT etc. frameworks
- What does it do? List features like protocols, media types
- How to write an Exposed Thing implementation
- How to write a Consumer implementation
- What more does it enable? Thing Simulation, Testing etc. (Dependents on npm)
- ?DANIEL? Open Source License, How to help evolve/contribute (GitHub, core, bindings, ...), et cetera

Why node-wot

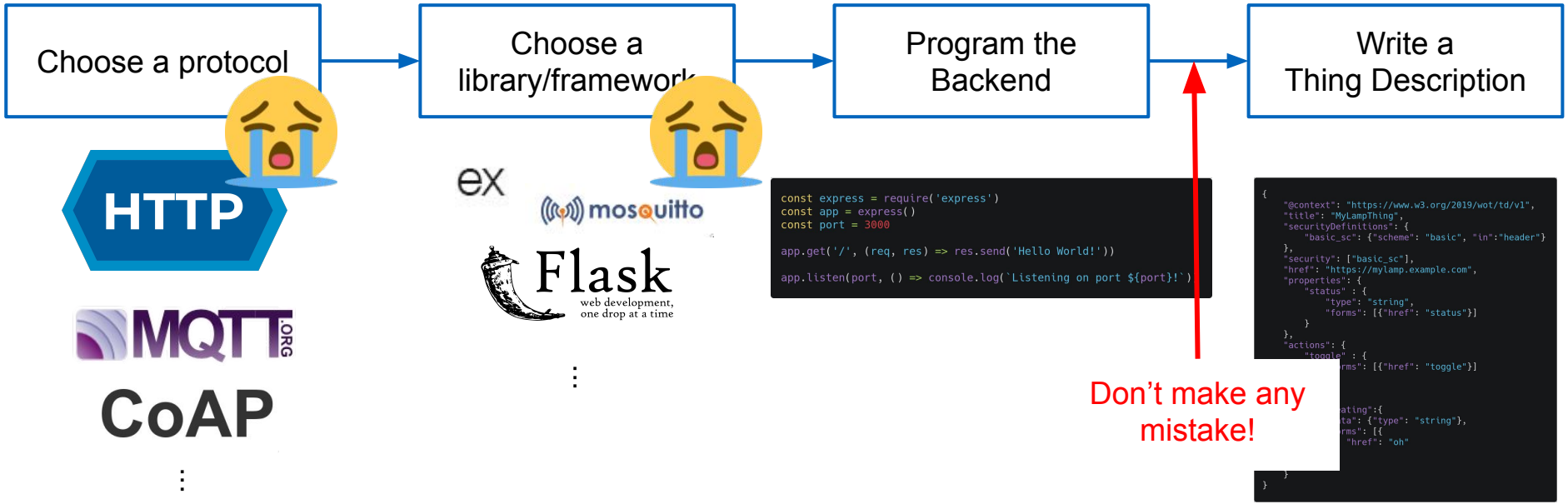
Thing Description hides the implementation details



- But what are the implementation details?
- How do I write a device implementation that is WoT compatible?
- How do I write programs that interact with WoT devices?

Why node-wot

- How do I write a device implementation that is WoT compatible without node-wot ?

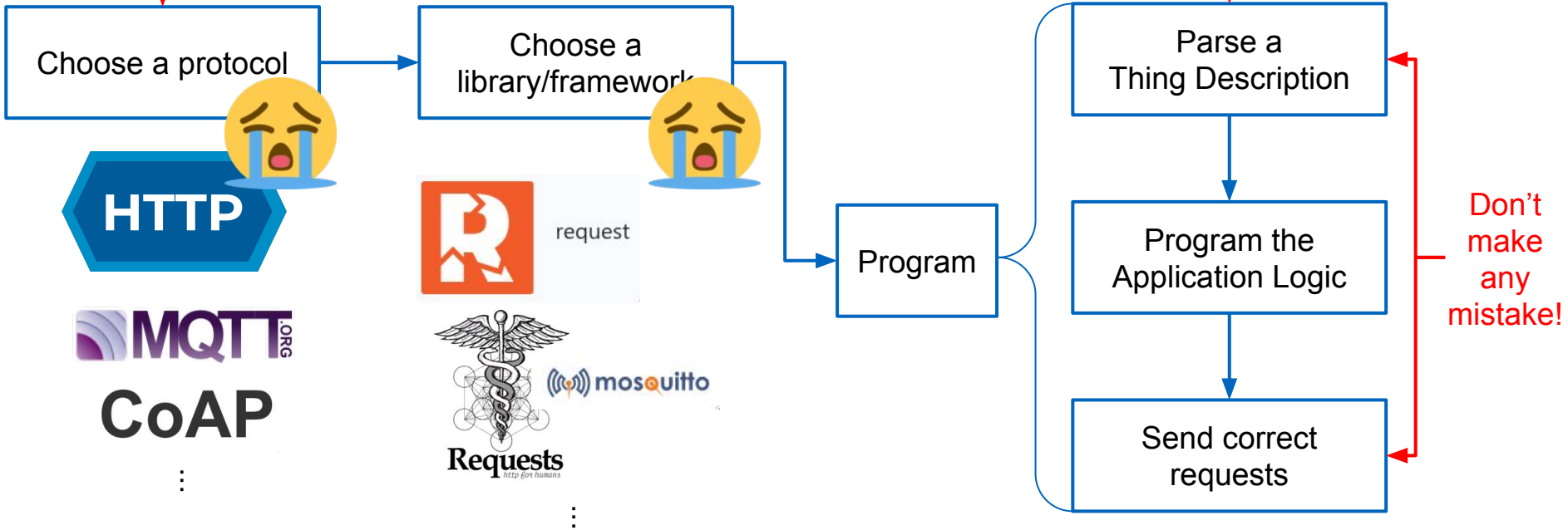


Associate Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

Why node-wot

Hopefully you support the protocol of *all* TDs

- How do I write programs that interact with WoT devices without node-wot ?



Why node-wot

- How do I write a device implementation that is WoT compatible with node-wot ?

Program the Backend

- How do I write programs that interact with WoT devices with node-wot ?

Program the Application Logic

What is node-wot ?

node-wot is an open source node.js library under the umbrella of the Eclipse Thingweb project, available at https://github.com/eclipse/thingweb.node-wot

- It is designed specifically for WoT applications, Thing or Consumer side alike and supports multiple protocols and data formats (media types). It is based on the W3C WoT Scripting API standard.

- It is highly modular and embraces software design patterns to provide the same programming interface for different protocols.

What is node-wot ?

Features:

- Protocols: HTTP, CoAP, MQTT, OPC-UA, Websockets with more that can be implemented
- Media Types: `application/json`, text/plain, image formats (experimental)
- WoT Operation Types: `readproperty`, `writeproperty`, `observeproperty`, `unobserveproperty`, `invokeaction`, `subscribeevent`, `unsubscribeevent`, `readallproperties`, `writeallproperties`, `readmultipleproperties`, `writemultipleproperties` (all of them)

What is node-wot ?

Features:

- Security: Basic Authentication, Bearer Tokens, API Key, PSK

Most importantly:

It is **not** an end to end framework!

You can write a Thing implementation and let another library,

Web browser, REST client interact with it

Now, let’s see some action!

We will start with simplified concepts and then go into its more advanced uses

Installation

Prerequisites:

- Node.js (10.13.0 +) is needed for all operating systems

Linux

Meet the node-gyp requirements:

- Python v2.7, v3.5, v3.6, v3.7, or v3.8
- make
- A proper C/C++ compiler toolchain, like GCC

MacOS

Meet the node-gyp requirements:

- `xcode-select --install`

Windows

Windows build tools:

```
npm install -g --production windows-build-tools
```

Installation

Alternative installation as npm packages, explained in Slide 35

1. Clone the repository: `git clone https://github.com/eclipse/thingweb.node-wot`
2. Change into the directory: `cd thingweb.node-wot`
3. Install dependencies: `npm install`
4. Build the source code: `npm run build`
5. Link the packages to enable the `wot-servient` command:

   ```
   sudo npm run link
   ```

If the last command doesn't work, it is not mandatory! In the upcoming slides, replace the `wot-servient` command with `node packages\cli\dist\cli.js`

Let’s use it!

W3C WoT Scripting API standard is always a good reference to understand the use of different methods of node-wot

How to write a Thing implementation

In the following slides, we will learn how to program a temperature controlling Thing. It:

- provides temperature values as a property
- allows to increase or decrease the temperature via actions
- alerts when the temperature reaches 45°C

How to write a Thing implementation

We have to start by describing these interactions

```
WoT.produce({
    title: "TemperatureController",
    description: "A Thing to control the temperature of the room and also get alerts in too high
temperatures",
    properties: {
        temperature: {
            type: "integer",
            description: "Current temperature value",
            observable: true,
            readOnly: true,
            unit:"Celsius"
        }
    },
    actions: {
        increment: {
            description: "Incrementing the temperature of the room with 0 to 5 increments",
            input:{
                type: "integer",
                minimum: 0,
                maximum: 5
            }
        },
        decrement: {
            description: "Decrementing the temperature of the room with 0 to 5 increments",
            input:{
                type: "integer",
                minimum: 0,
                maximum: 5
            }
        }
    },
    events: {
        overheat: {
            description: "Alert sent when the room temperature is too high"
        }
    }
})
```

How to write a Thing implementation

Then, we have to program:

- How to read the temperature from the internal sensor
- What happens when increment or decrement actions are invoked
- Have a logic that emits the overheat alert when the temperature exceeds 45°C

Some dummy functions to mimic temperature related operations

```
    function getTemperature() {
        // normally, you would call the temperature sensor's function to read the actual
temperature value
        return Math.random() * Math.floor(50);
        // return 5; //uncomment to test incrementing etc.
    }

    function changeTemperature(newValue){
        // normally, you would do physical action to change the temperature
        //do nothing
        thing.writeProperty("temperature",newValue);
        return;
    }
```

```
thing.setPropertyReadHandler("temperature",function(){
    return new Promise((resolve, reject) => {
        resolve(getTemperature());
    });
});

// set action handlers
thing.setActionHandler("increment", function (value, options) {
    changeTemperature(getTemperature()+value)
});

thing.setActionHandler("decrement", function (value, options) {
    changeTemperature(getTemperature()-value)
});
```

```
setInterval(() => {
    var curTemp = getTemperature();
    thing.writeProperty("temperature",curTemp)
    if (curTemp > 45) {
        thing.emitEvent("overheat")
    }
}, 5000);

// expose the thing
thing.expose().then(function () { console.info(thing.getThingDescription().title +
                                                " ready"); });
```

Full code available at:
https://gist.github.com/egekorkan/ddf2e03f40fb976d9d4b925fbbb9d381

How to use the `wot-servient` command

The program cannot be run directly via node.js, i.e. via `node myTempController.js`, since the WoT object in the beginning is unknown for the node.js.

Huh?

How to use the `wot-servient` command

`wot-servient` command builds the necessary *infrastructure* and creates the WoT object for our scripts to use. It is the Command Line Interface (CLI) of node-wot !

It uses a default configuration file that can be changed where the desired protocols, security configuration, static address for the generated TD, etc. are specified.

Let’s give it a run!

Associate Professorship of Embedded Systems and Internet of Things
Department of Electrical and Computer Engineering
Technical University of Munich

How to use the `wot-servient` command

The command uses this configuration file that has default values.

You can change it and supply your own configuration file with the -f option.

Run `wot-servient -h` to learn all the options!

How to write code to interact with a Thing

Not needed!

THE END

How to write code to interact with a Thing

You can use any browser or REST Client software such as Postman, Insomnia, cURL for HTTP; Copper for Chrome for CoAP, MQTT.fx for MQTT, or Node-RED which has multiple protocols. As long as they use the protocol of the Thing, you can use it!

How to write code to interact with a Thing

But you can do this easier, faster with node-wot and the code you will write will be protocol independent!

```
const TemperatureThingAddress = "http://localhost:8080/TemperatureController";

WoTHelpers.fetch(TemperatureThingAddress).then(async (TD) => {

    try{
        let temperatureThing = await WoT.consume(TD);
```

```
setInterval(async() => {
    let curTemp = await temperatureThing.readProperty("temperature");

    console.log("Room's Current Temperature is ", curTemp);

    if (curTemp < 20) {
        await temperatureThing.invokeAction("increment",4)
    }
}, 1000);
```

```
temperatureThing.subscribeEvent("overheat",
x => console.log("!!!CONTACT THE FIRE DEPARTMENT!!!!"),
e => console.error("Error: %s", e),
() => console.info("Completed")
);
```

How to write code to interact with a Thing

You can use the same configuration file.

The full code is available at:

https://gist.github.com/egekorkan/dfd0f999c22396a997eb10994e11aed6

Let’s get a bit more advanced ;)

How to use as an npm dependency

If you are building anything more complex than our examples, we recommend you to use it as an npm dependency and import into your project code via require statements such as:

```
Servient = require("@node-wot/core").Servient
```

This also means that you would **not** use the `wot-servient` command but build the WoT object yourself!

How to use as an npm dependency

Same with any npm package, you should download the required packages from npm. You should run the following in your project's folder:

`npm install @node-wot/core` (mandatory core component)

`npm install @node-wot/binding-coap` (optional bindings)

Let's see how the previous code look like:

```
Servient = require("@node-wot/core").Servient
HttpServer = require("@node-wot/binding-http").HttpServer

Helpers = require("@node-wot/core").Helpers

// create Servient add HTTP binding with port configuration
let servient = new Servient();
servient.addServer(new HttpServer({
    port: 8081 // (default 8080)
}));

servient.start().then((WoT) => {
    WoT.produce({
        title: "TemperatureController",
    description: "A Thing to control the temperature of the room and also get alerts in too high
temperatures",
    properties: {
        temperature: {
            type: "integer",
            description: "Current temperature value",
            observable: true,
            readOnly: true,
```

Exactly same as before!

```
Servient = require("@node-wot/core").Servient
HttpClientFactory = require("@node-wot/binding-http").HttpClientFactory

Helpers = require("@node-wot/core").Helpers

// create Servient and add HTTP binding
let servient = new Servient();
servient.addClientFactory(new HttpClientFactory(null));

let wotHelper = new Helpers(servient);

const TemperatureThingAddress = "http://localhost:8080/TemperatureController";
wotHelper.fetch(TemperatureThingAddress).then(async (td) => {
    // using await for serial execution (note 'async' in then() of fetch())
    try {
        servient.start().then((WoT) => {
            WoT.consume(td).then((thing) => {
                // read a property "string" and print the value
                setInterval(async() => {
                  let curTemp = await temperatureThing.readProperty("temperature");

                  console.log("Room's Current Temperature is ", curTemp);

                  if (curTemp < 20) {
                      await temperatureThing.invokeAction("increment",4)
```

Exactly same as before!

How to use as an npm dependency

Advantages:

- Better version control of node-wot
- More suitable for installing on more constrained devices, like Raspberry Pi
- Install only what your project needs
- Better compatibility when combined with other npm packages
- Self-contained
- Possibility to use Typescript and Intellisense

What more does node-wot enable?

Node-wot is meant for building bigger applications for WoT, it is only a stepping stone. Some node-wot based implementations:

WADE: A GUI or API Development Environment, based on node-wot.

Shadow-Thing: Automatic Reverse Proxy and Thing simulation

WoT-Testbench: Blackbox testing of Things based on their TDs

WoT Application Manager (WAM): Kickstarting WoT applications

How to Contribute

- New protocol bindings
  - However, we discourage new protocol dialects. So a new protocol such as AMQP is very welcome but a CoAP dialect for IKEA TRÅDFRI is not
- Bug fixes :) Some good first issues to tackle are #117, #151 or #161
- Using it and submitting new issues!

Further Information

GitHub repository: eclipse/thingweb.node-wot: thingweb.node-wot

npm packages: core, http, coap, mqtt, td-tools and more!

Thingweb Homepage: http://www.thingweb.io/

Eclipse Project Page: https://projects.eclipse.org/projects/iot.thingweb

WoT Scripting API: https://www.w3.org/TR/wot-scripting-api/

WoT Thing Description: https://www.w3.org/TR/wot-thing-description/

W3C WoT Homepage: https://www.w3.org/WoT/