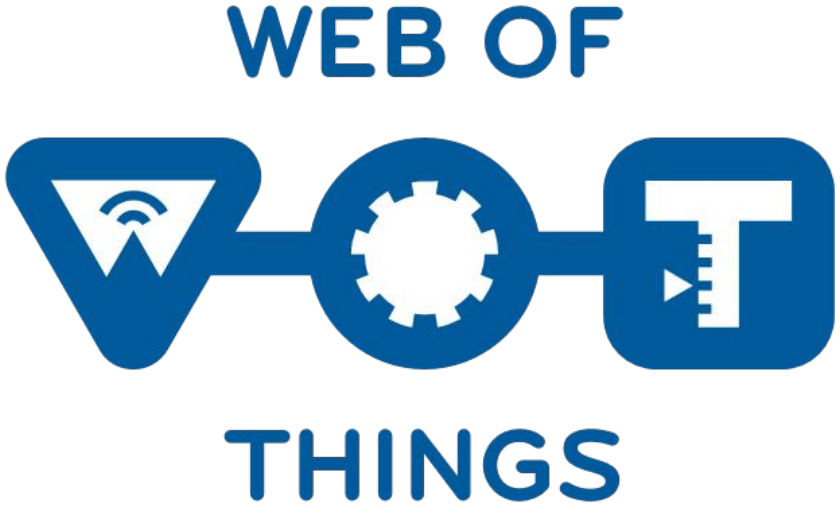


wot-rust introduction

Online Meeting

Luca Barbato

22.09.2022

Intro - Me

- Luca Barbato
  - Email: lu_zero@gentoo.org, lu_zero@videolan.org
  - Twitter: @lu_zero_, GitHub: @lu-zero

Part of some opensource communities (Gentoo, Videolan, …)

Lately I'm focused on rust.

And I'm participating to SIFIS-Home.

Intro - SIFIS-Home

- It is an Horizon 2020 project.
- We try to demonstrate you can have a trustworthy connected home.
- Focus on security and interoperability
  - No single point of failure
  - Open Standards
  - Software development guidelines
    - Among those we warmly suggest to use rust.

Intro - Rust

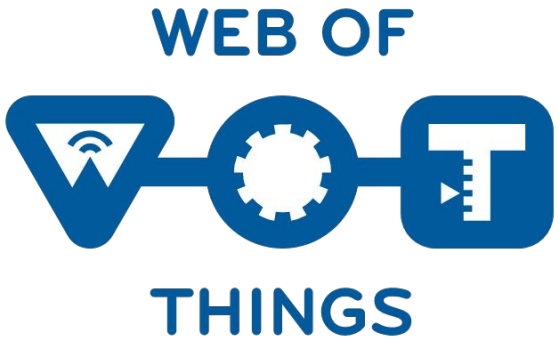
- We suggest to use it since more than half of the bugs in large software happens to be related to memory faults that rust makes impossible.
  - Ask Microsoft about it.

Intro - Rust

- On top of the safety features, the ecosystem around it makes fairly easy to write code that runs fast and pretty much everywhere
  - Including the embedded/bare-metal targets

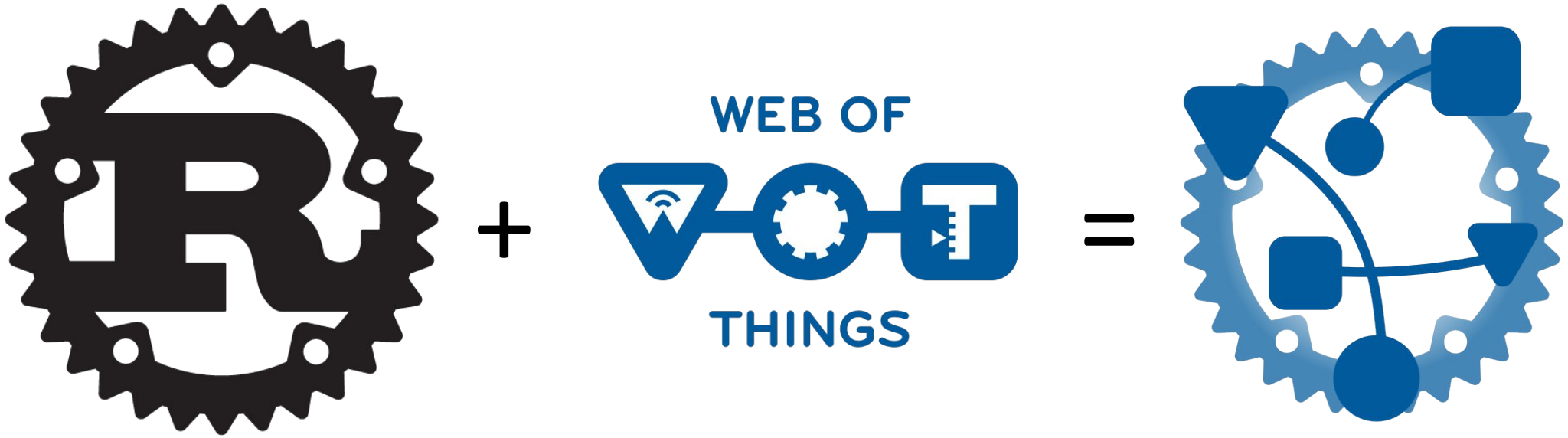
Intro - Web of Things

- SIFIS-Home also focuses on Open Standards
  - In the opinion of a number of the consortium partners WoT is one of the most promising.
  - And it fits perfectly our specific need of being able to describe with high granularity what are the risks in using a connected device.



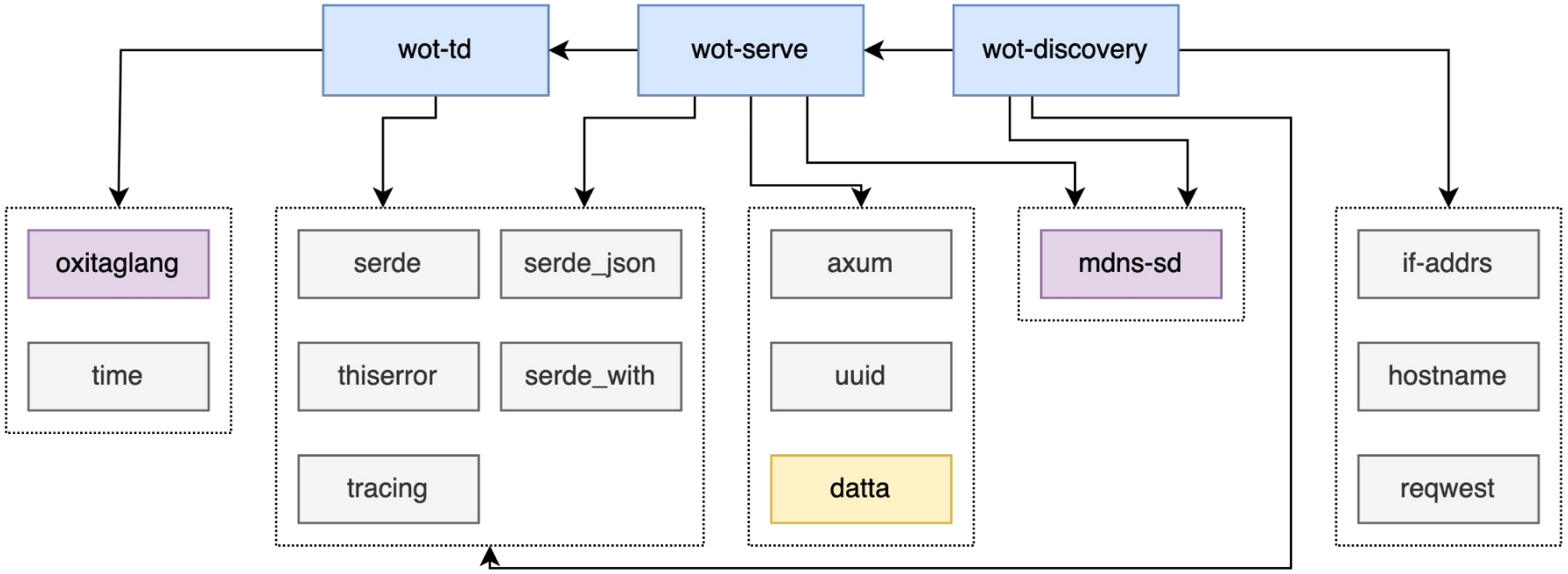
wot-rust

- After getting a proof-of-concept working using webthings-rust and some hacking, we decided to make an implementation from scratch
  - We needed a project large enough to test our developer guidelines
  - We wanted to satisfy our sifis-specific needs without imposing them to other projects



wot-rust

- wot-rust is a collection of crates (packages in rust-jargon)
- We try to split the concerns
  - wot-td is focused only on deserializing/serializing and manipulating Thing Descriptions
  - wot-serve builds on top of it and tries to streamline building and running Servients.
  - wot-discovery uses both to implement discovery and directory



wot-td

- It implements a builder pattern that tries to prevent at compile time most of the misuses

```
let thing = Thing::build("My Lamp") <- (title) => Thing, ThingBuilder<Nil, ToExtend>
    .finish_extend() => ThingBuilder<Nil, Extended>
    .id("urn:dev:ops:my-lamp-1234") => ThingBuilder<Nil, Extended>
    .attype("OnOffSwitch") => ThingBuilder<Nil, Extended>
    .attype("Light") => ThingBuilder<Nil, Extended>
    .description("A web connected lamp") => ThingBuilder<Nil, Extended>
    .security(|b| b.no_sec().with_key("nosec_sc").required()) <- (name) => ThingBuilder<Nil, Extended>, SecuritySchemeBuilder<()>
    .property("on", |b| { <- (name, f) => PropertyAffordanceBuilder<…>
        b.finish_extend_data_schema() => PropertyAffordanceBuilder<…>
            .attype("OnOffProperty") => PropertyAffordanceBuilder<…>
            .title("On/Off") => PropertyAffordanceBuilder<…>
            .description("Whether the lamp is turned on") => PropertyAffordanceBuilder<…>
            .form(|b| b.href("/properties/on")) => PropertyAffordanceBuilder<…>, FormBuilder<Nil, (), Nil>
            .bool()
    }) => ThingBuilder<Nil, Extended>
    .link_with(f)
    link(…)~      Method fn(self, impl Into<String>) -> ThingBuilder<Other, Status>  fn(self, F) -> ThingBuilder<Other, Status>
    link_with(…)~ Method fn(self, F) -> ThingBuilder<Other, Status>                  Add an additional link to the Thing Description, with specified
            .attype("BrightnessProperty")                                            optional fields.
            .title("Brightness")
            .description("The level of light from 0-100")
            .form(|b| b.href("/properties/brightness"))
```

wot-td

- It implements a builder pattern that tries to prevent at compile time most of the misuses
- It leverages serde to serialise and deserialise from and to json, or any other format that has a serde implementation e.g. ciborium.

```
.property("on", |b| {
    b.finish_extend_data_schema()
        .attype("OnOffProperty")
        .title("On/Off")
        .description("Whether the lamp is turned on")
        .form(|b| {
            b.href("/properties/on")
                .http_get(get_on_property)
                .http_put(put_on_property)
                .op(wot_td::thing::FormOperation::ReadProperty)
                .op(wot_td::thing::FormOperation::WriteProperty)
        })
        .bool()
})
.property("brightness", |b| {
    b.finish_extend_data_schema()
        .attype("BrightnessProperty")
        .title("Brightness")
        .description("The level of light from 0-100")
        .form(|b| {
            b.href("/properties/brightness")
                .http_get(get_brightness_property)
                .http_put(put_brightness_property)
                .op(wot_td::thing::FormOperation::ReadProperty)
                .op(wot_td::thing::FormOperation::WriteProperty)
        })
        .integer()
        .minimum(0)
```

```
  "properties": {
    "brightness": {
      "@type": "BrightnessProperty",
      "description": "The level of light from 0-100",
      "forms": [
        {
          "href": "/properties/brightness",
          "op": [
            "readproperty",
            "writeproperty"
          ]
        }
      ],
      "maximum": 100,
      "minimum": 0,
      "readOnly": false,
      "title": "Brightness",
      "type": "integer",
      "unit": "percent",
      "writeOnly": false
    },
    "on": {
      "@type": "OnOffProperty",
      "description": "Whether the lamp is turned on",
      "forms": [
        {
          "href": "/properties/on",
```

wot-td

- It implements a builder pattern that tries to prevent at compile time most of the misuses
- It leverages serde to serialise and deserialise from and to json, or any other format that has a serde implementation e.g. ciborium.
- It has an extension system
  - It lets you attach structured data as you build the thing
  - It lets you deserialize the same structured data

```rust
#[serde_as]
#[skip_serializing_none]
#[derive(Debug, Clone, Deserialize, Serialize, PartialEq, Eq, Hash, D
struct MessageHeader {
    #[serde(rename = "htv:fieldName")]
    pub field_name: Option<String>,
    #[serde(rename = "htv:fieldValue")]
    pub field_value: Option<String>,
}

#[serde_as]
#[skip_serializing_none]
#[derive(Debug, Clone, Deserialize, Serialize, Defaul
struct Response {
    #[serde(rename = "htv:headers")]
    #[serde(skip_serializing_if = "Vec::is_empty")]
    pub headers: Vec<MessageHeader>,
    #[serde(rename = "htv:statusCodeValue")]
    pub status_code_value: Option<usize>,
}

#[serde_as]
#[skip_serializing_none]
#[derive(Debug, Clone, Deserialize, Serialize, Default, PartialEq, Eq
struct Form {
    #[serde(rename = "htv:methodName")]
    pub method_name: Option<Method>,
}
```

```rust
#[derive(Debug, Clone, Deserialize, Serialize, PartialEq, Eq, Has
struct HttpProtocol {}

impl ExtendableThing for HttpProtocol {
    type InteractionAffordance = ();
    type PropertyAffordance = ();
    type ActionAffordance = ();
    type EventAffordance = ();
    type Form = Form;
    type ExpectedResponse = Response;
    type DataSchema = ();
    type ObjectSchema = ();
    type ArraySchema = ();
}
```

```rust
let property = r#" => &str
{
    "href": "/things{?offset,limit,format,sort_by,sort_order}",
    "htv:methodName": "GET",
    "response": {
        "description": "Success response",
        "htv:statusCodeValue": 200,
        "contentType": "application/ld+json",
        "htv:headers": [
            {
                "htv:fieldName": "Link"
            }
        ]
    }
}
"#;

let expected = Form { => Form<HttpProtocol>
    href: "/things{?offset,limit,format,sort_by,sort_order}".into(),
    response: Some(ExpectedResponse {
        content_type: "application/ld+json".into(),
        other: super::Response {
            headers: vec![super::MessageHeader {
                field_name: Some("Link".into()),
                field_value: None,
            }],
            status_code_value: Some(200),
        }
```

wot-serve

- It leverages the **wot-td** extension system to build the application server router as you build the **Thing Description** itself.

```
.property("on", |b| {
    b.finish_extend_data_schema()
        .attype("OnOffProperty")
        .title("On/Off")
        .description("Whether the lamp is turned
        .form(|b| {
            b.href("/properties/on")
                .http_get(get_on_property)
                .http_put(put_on_property)
                .op(wot_td::thing::FormOperation
                .op(wot_td::thing::FormOperation
        })
        .bool()
})
.property("brightness", |b| {
    b.finish_extend_data_schema()
        .attype("BrightnessProperty")
        .title("Brightness")
        .description("The level of light from 0-100 ")
        .form(|b| {
            b.href("/properties/brightness")
                .http_get(get_brightness_property)
                .http_put(put_brightness_property)
                .op(wot_td::thing::FormOperation::ReadProperty)
                .op(wot_td::thing::FormOperation::WriteProperty)
        })
        .integer()
        .minimum(0)
```

```
#[derive(Debug, Default, Serialize, Deserialize, Clone)]
pub struct ServientExtension {
    /// Listening address
    #[serde(skip)]
    addr: Option<SocketAddr>,
    /// Thing type
    #[serde(skip)]
    thing_type: ThingType,
}

#[doc(hidden)]
/// Form Extension
#[derive(Debug, Default, Serial
pub struct Form {
    #[serde(skip)]
    method_router: MethodRouter
}
```

```
impl<Other, Href, OtherForm> HttpRouter for FormBuilder<Other, Href,
where
    Other: ExtendableThing + Holder<ServientExtension>,
    OtherForm: Holder<Form>,
{
    type Target = FormBuilder<Other, Href, OtherForm>;

    /// Route GET requests to the given handler.
    fn http_get<H, T>(mut self, handler: H) -> Self::Target
    where
        H: Handler<T, axum::body::Body>,
        T: 'static,
    {
        let method_router = std::mem::take(&mut self.other.field_mut(
        self.other.field_mut().method_router = method_router.get(hand
        self
    }
```

wot-serve

- It leverages the **wot-td** extension system to build the application server router as you build the **Thing Description** itself.
- It manages the mDNS advertisement

```
    Running `target/debug/lamp`
2022-09-19T16:39:22.278330Z DEBUG mdns_sd::service_daemon: new socket bind to 0.0.0.0:5353
2022-09-19T16:39:22.278714Z DEBUG lamp: listening on 0.0.0.0:3000
2022-09-19T16:39:22.299215Z DEBUG mdns_sd::service_daemon: register service ServiceInfo { ty_domain: "_wot._tcp.local.", sub_domain:
None, fullname: "mybf2fafe3139540509dbecfbb1207666c._wot._tcp.local.", server: "enyo.lan.local", addresses: {192.168.1.212}, port: 30
00, host_ttl: 120, other_ttl: 4500, priority: 0, weight: 0, properties: {"td": "/.well-known/wot", "type": "Thing"}, last_update: 166
3605562278 }
2022-09-19T16:39:22.299344Z DEBUG mdns_sd::service_daemon: broadcast service mybf2fafe3139540509dbecfbb1207666c._wot._tcp.local.
```

```
› dns-sd -B _wot._tcp
Browsing for _wot._tcp
DATE: ---Mon 19 Sep 2022---
18:40:34.085  ...STARTING...
Timestamp     A/R    Flags  if Domain               Service Type         Instance Name
18:40:34.086  Add        3   6 local.              _wot._tcp.           mybd729b6943214caeb527e15626162967
18:40:34.086  Add        2   6 local.              _wot._tcp.           mybf2fafe3139540509dbecfbb1207666c
```

wot-discovery

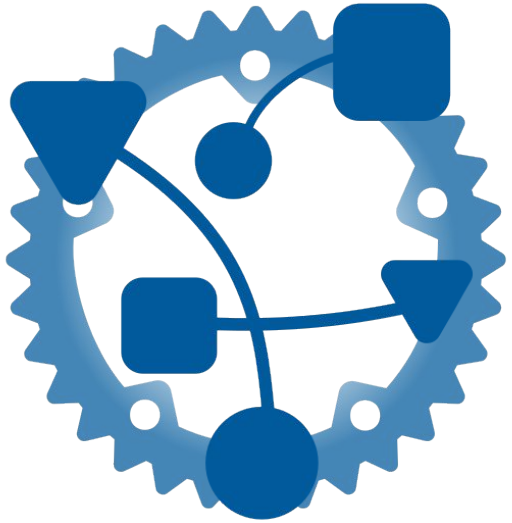
- Very Work in Progress
- Reacts to mDNS broadcasts
- Fetches the Thing Descriptions and stores them
- Uses **wot-serve** to provide the directory service as a Thing

```
    Running `target/debug/lamp`
2022-09-19T16:39:22.278330Z DEBUG mdns_sd::service_daemon: new socket bind to 0.0.0.0:5353
2022-09-19T16:39:22.278714Z DEBUG lamp: listening on 0.0.0.0:3000
2022-09-19T16:39:22.299215Z DEBUG mdns_sd::service_daemon: register service ServiceInfo { ty_domain: "_wot._tcp.local.", sub_domain:
None, fullname: "mybf2fafe3139540509dbecfbb1207666c._wot._tcp.local.", server: "enyo.lan.local", addresses: {192.168.1.212}, port: 30
00, host_ttl: 120, other_ttl: 4500, priority: 0, weight: 0, properties: {"td": "/.well-known/wot", "type": "Thing"}, last_update: 166
3605562278 }
2022-09-19T16:39:22.299344Z DEBUG mdns_sd::service_daemon: broadcast service mybf2fafe3139540509dbecfbb1207666c._wot._tcp.local.
```

```
    Running `target/debug/examples/list`
2022-09-19T16:37:53.398687Z  INFO list: found "My Lamp" Some("urn:dev:ops:my-lamp-1234")
```

What’s next

- The wot-td and wot-serve 0.2 supporting WoT Thing Description 1.1 will be released soon
- Incoming work
  - Add more pre-made building blocks in **wot-serve** (TLS, other security schemes)
  - Provide more built-in extensions such as protocols in **wot-td** and **wot-serve**
  - Complete the **wot-discovery** directory pending the integration with oxigraph



Questions?