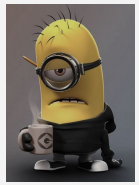
HiveOT - The Hive Of Things

1. Introduction and Why HiveOT
2. Security of Things
3. Using the WoT TD for Digital Twins
4. Messaging
5. Demo
6. TD Challenges
    a. Affordances
    b. Forms
    c. Other

Introduction - about me



- Author of HiveOT (https://github.com/hiveot)
- Educated as Electronics Engineer
- Switched to software development in ‘89
- Took a (wrong) turn somewhere into management
- Love Coffee, IoT, FOSS, Linux, Golang and now WoT
- Slowly gaining experience with WoT since 2023

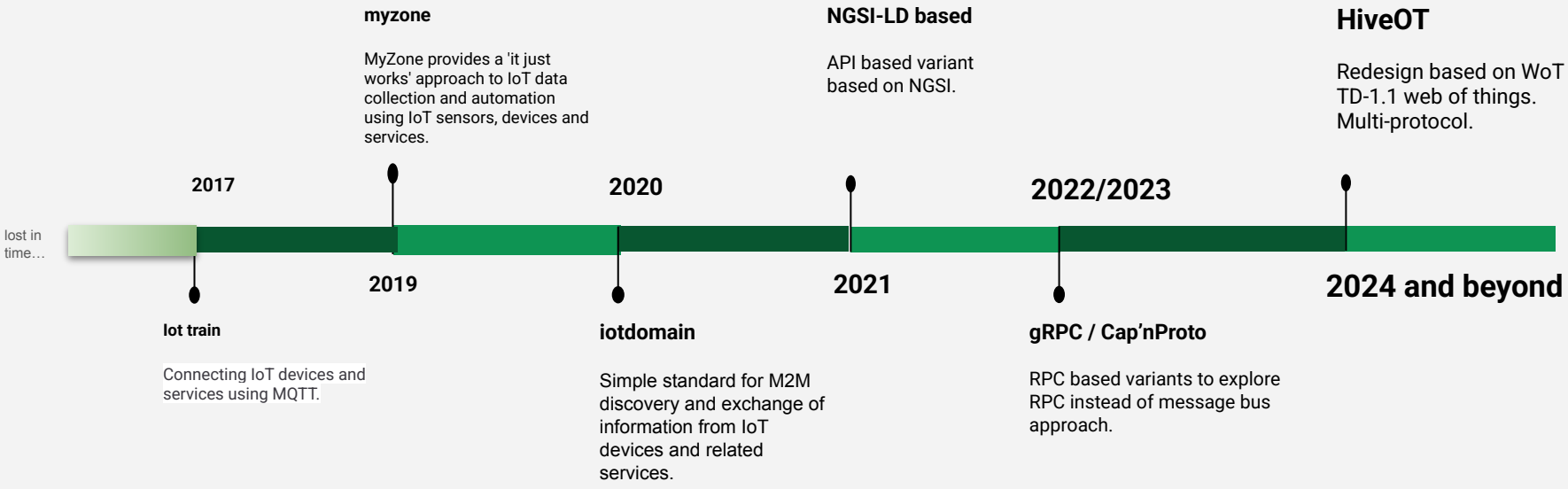
Introduction - Why HiveOT?

A long time ago, like in 2010, there was a pump house. With it, came the questions…

- Does the heater still work?
- Will it freeze in there?
- How much energy is it using?
- Any ‘unauthorized’ visitors?
- .. and more …

And so it started…

Introduction - Evolution



(Because its fun)

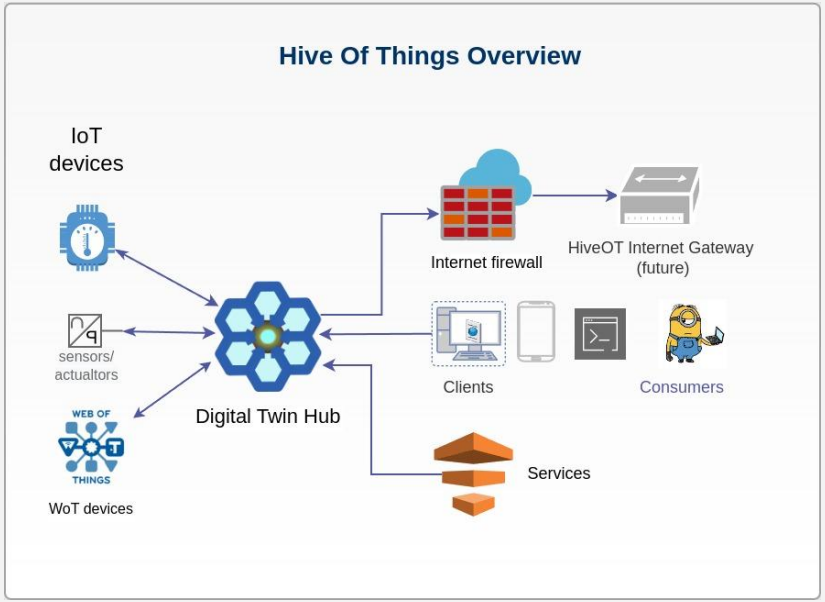
Introduction - What is HiveOT

What is HiveOT?

- Digital Twin IoT Hubs at the edge
- IOT devices and services
- Consumer clients

Objectives

- Share information between IoT Devices, Services and Consumers
- Based on WoT-1.1 Thing Specification
- Digital Twin representation
- Lightweight with minimal dependencies
- Extensible. Integrate with existing IoT Systems and services.
- Secure. Don’t touch my things.



Security of Things - Concerns

IoT devices have a large attack surface due to their internet-supported connectivity. (1)

Concerns:

- Missing firmware updates
- Weak authentication
- Shared network access
- Lack of encryption
- … on and on and on …

(Credits: https://www.youtube.com/watch?v=Cxrwpij2MSQ)

Some examples:

- The 2016 Mirai botnet was made possible by unsecured IoT devices.
- Target's credit card breach (2013) stole login credentials from an HVAC vendor IoT sensors.
- A Casino's Database Was Hacked Through A Smart Fish Tank Thermometer (*2)
- … on and on and on …

(1) https://www.techtarget.com/iotagenda/definition/IoT-security-Internet-of-Things-security

(2) https://interestingengineering.com/culture/a-casinos-database-was-hacked-through-a-smart-fish-tank-thermometer

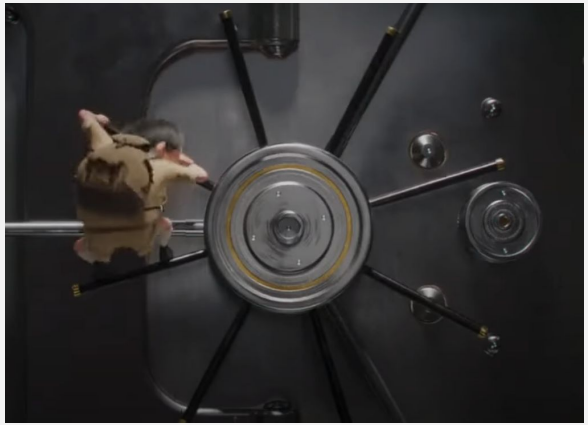
Security of Things - HiveOT

Rule #1: HiveOT Things do not run (network) servers.

- Offer an alternative to the model that all Things must run a server to be used, which raises a global security concern due to incorporation in botnets.
- IoT devices instead connect to a HiveOT Hub through a discovery and provisioning process.
- Greatly reduced attack surface. No direct access to the device.
- Encrypted network connections.

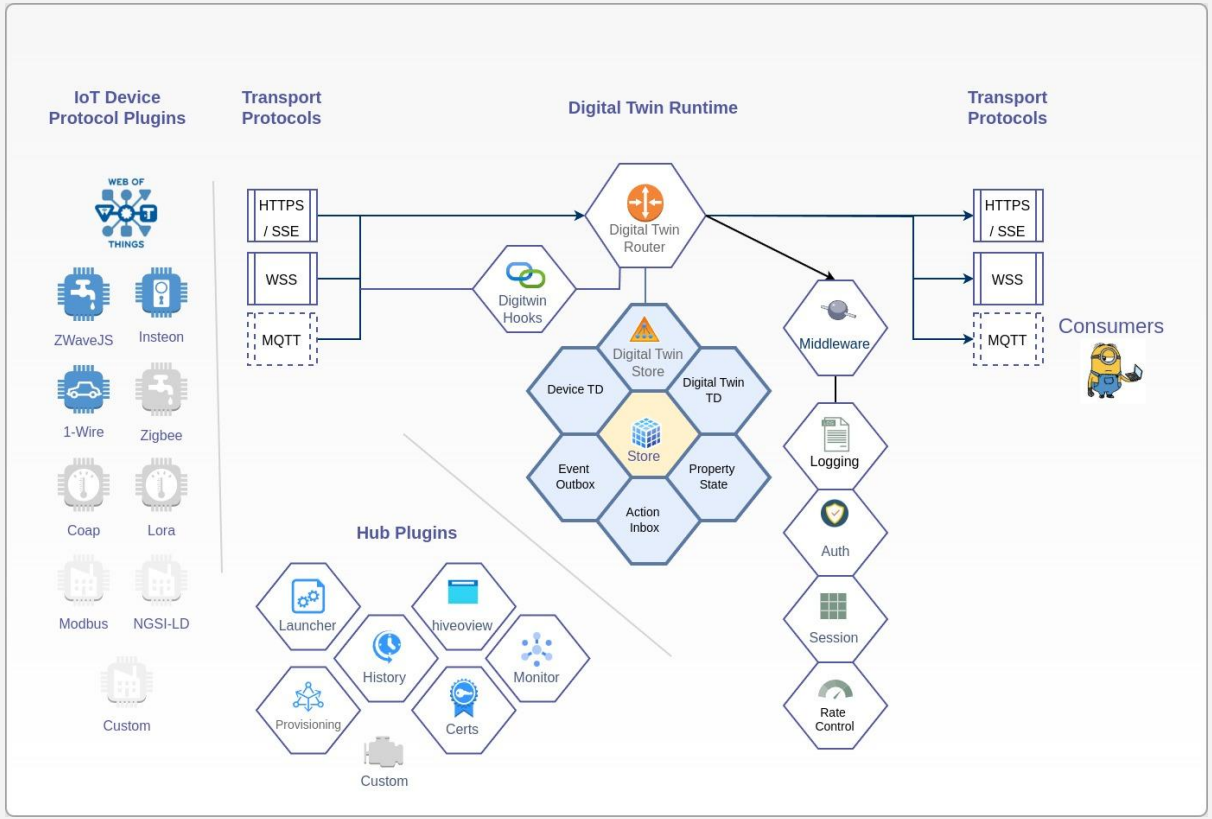
Benefits of using a Hub. Provide centralized:

- Authentication & authorization
- Consumer facing connectivity
- Security protocols
- Firmware updates



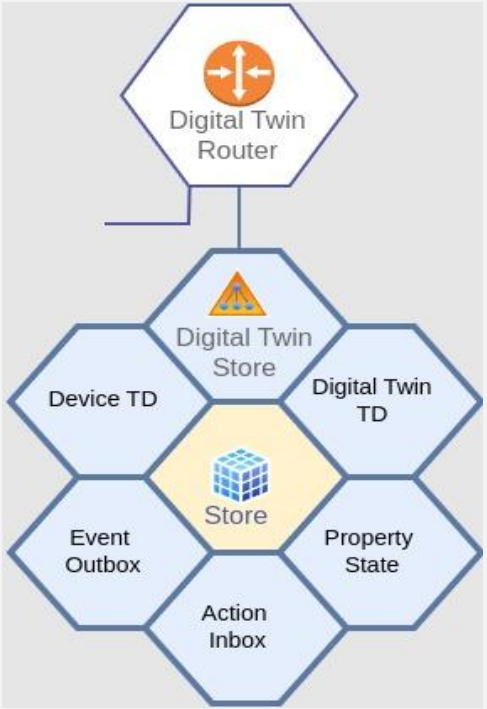
HiveOT Hub - An Edge Device

- Digital Twin Runtime
  - Transport Protocols
  - Router
  - Digital Twin Store
  - Digital Twin Hooks
- IoT Device Protocols
  - WoT
  - ZWave
  - 1-wire
  - …etc
- Services
  - Launcher
  - Provisioning
  - History
  - HiveOView (web view)
  - Monitoring

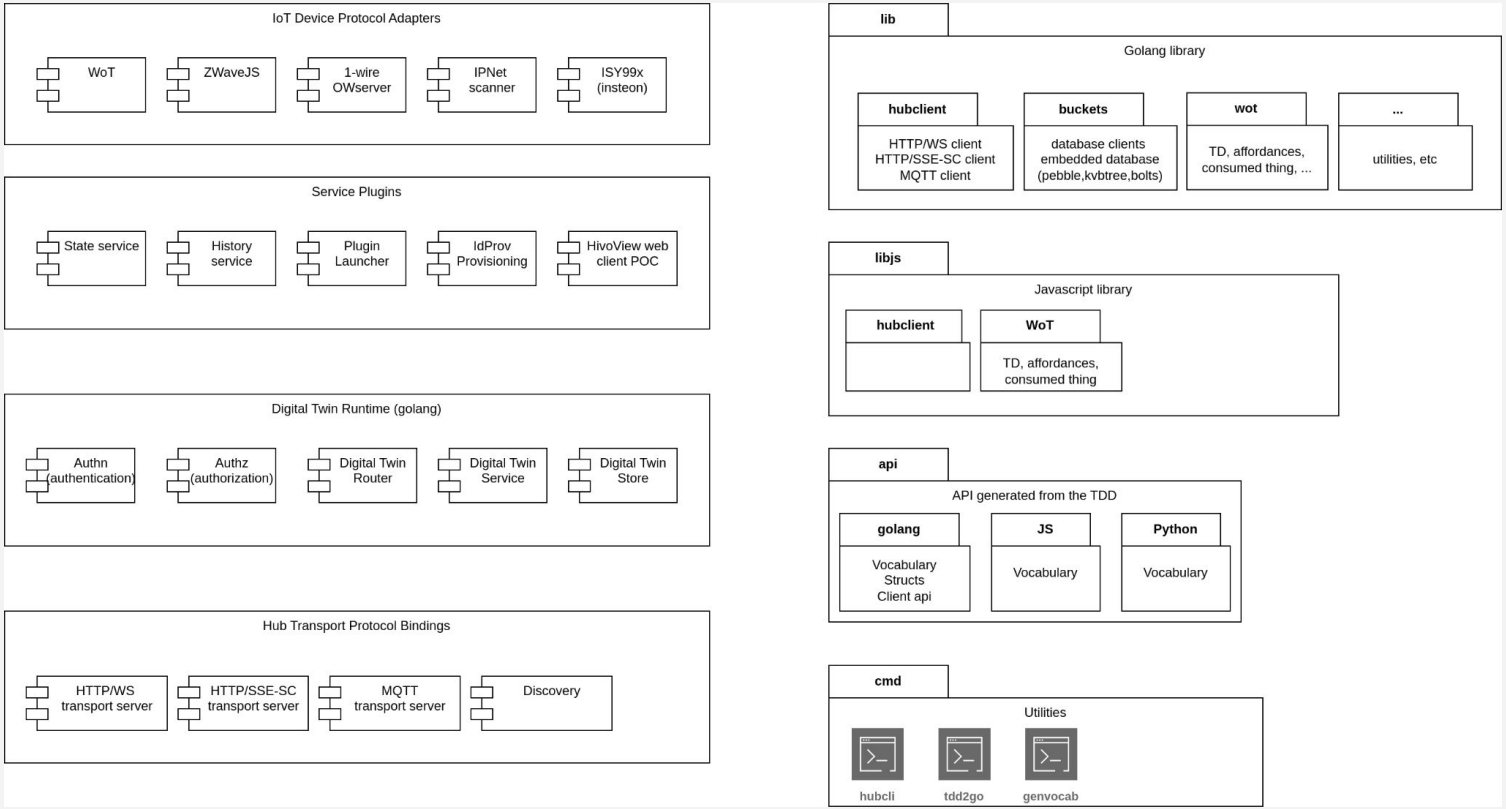


Digital Twin Runtime

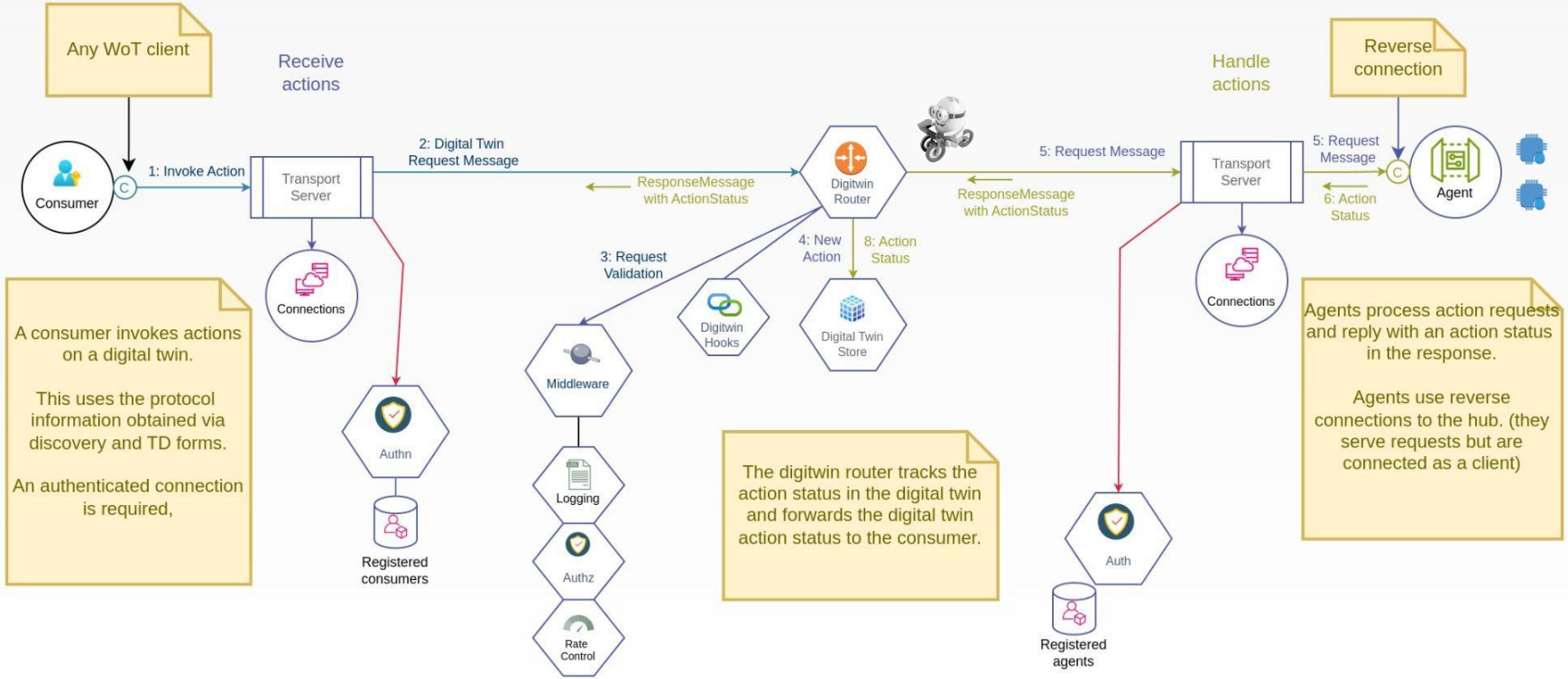
- HiveOT uses two TD's per Digital Twin:
  - Device TD
    - Sent by device agents
  - Digital Twin TD
    - Served to consumers
    - Forms updated with with Hub endpoints
    - Identify as digital-twin Thing-ID: dtw:{agentID}:{thingID}
- Consumers interact with the digital twin
  - Requests are routed by the digital twin router
  - The device is isolated from the outside
- Monitoring and simulation hooks
  - Monitor Thing interaction (*future)
  - Route Thing operations to a simulator (*future)



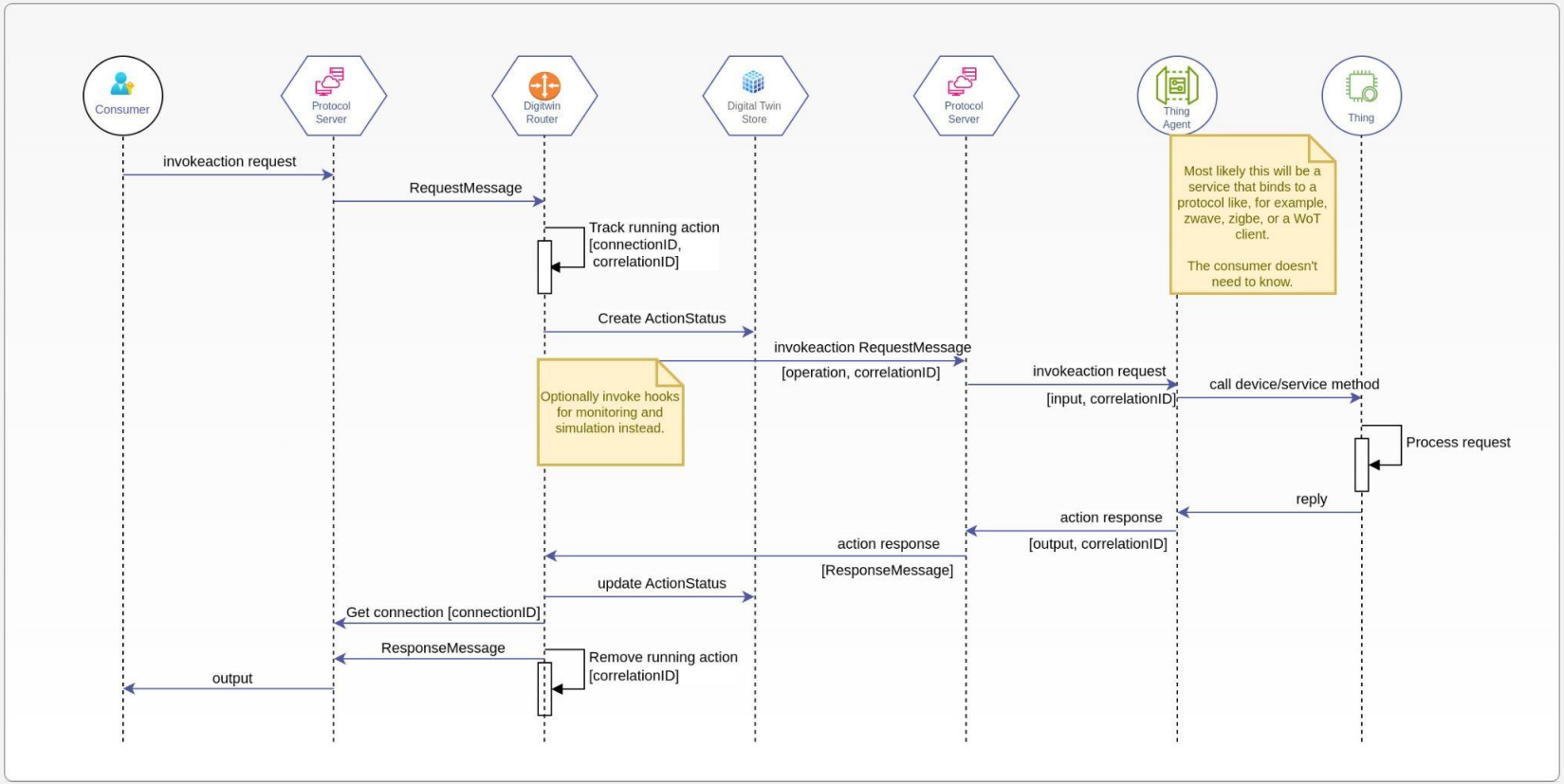
What's in the box - Runtime, Messaging Protocols and Plugins



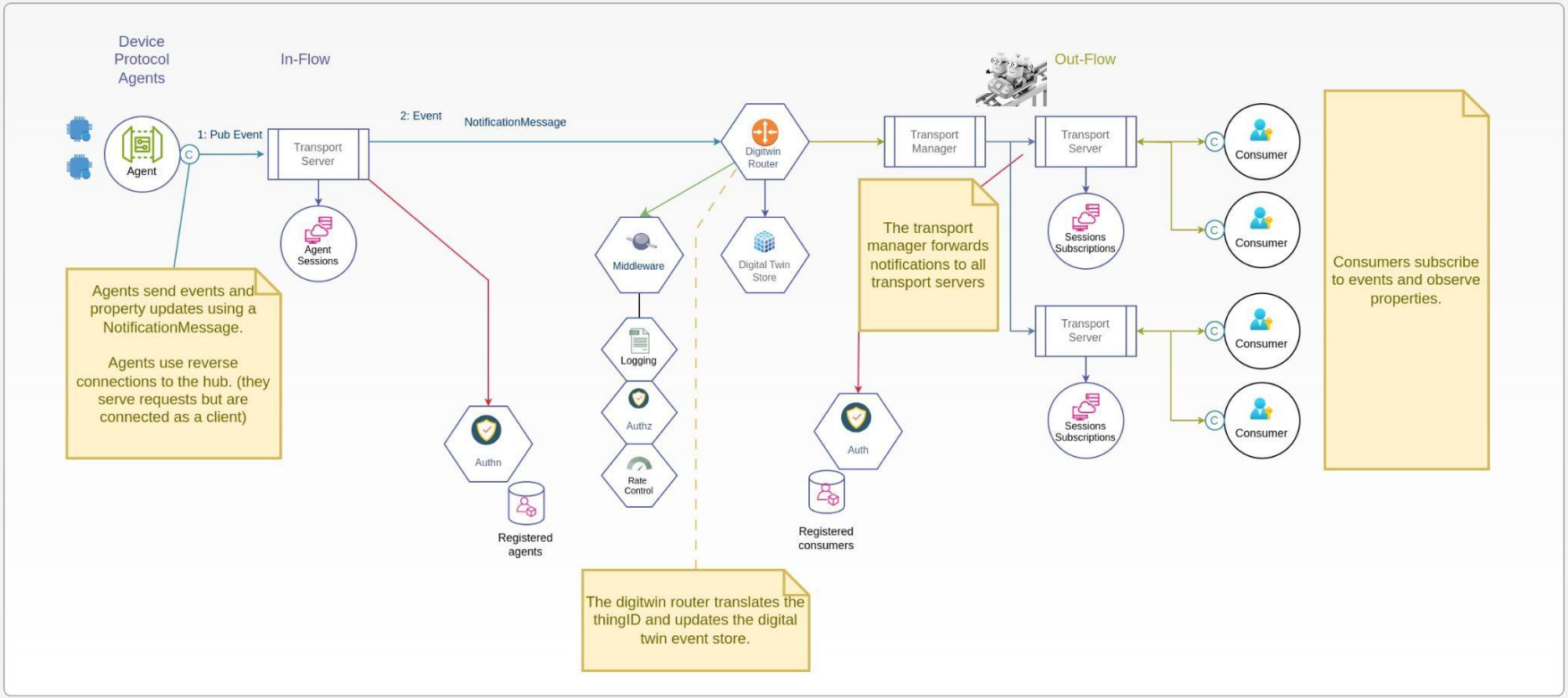
Hub Messaging - Action Flow



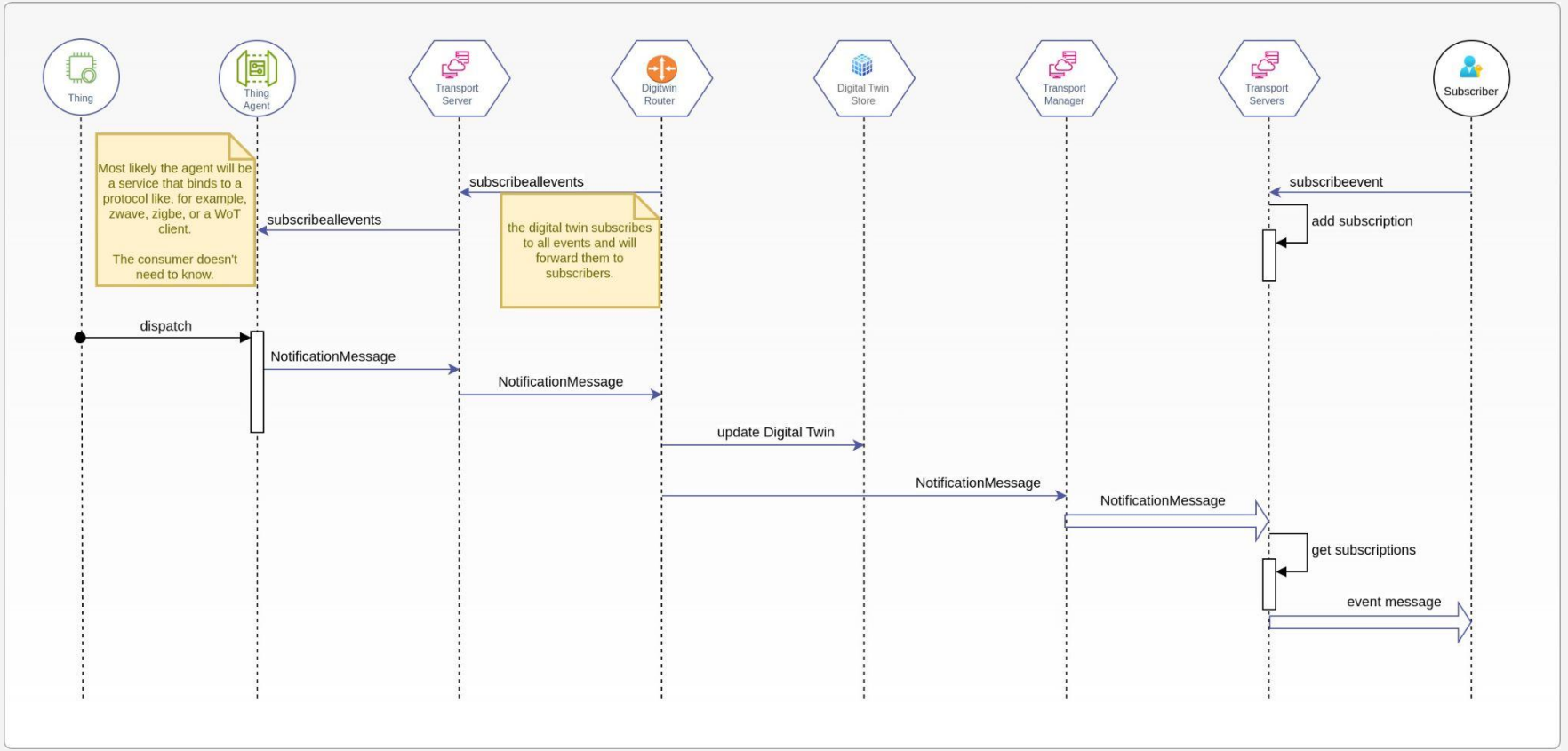
Hub Messaging - Action sequence



Hub Messaging - Event Flow



Hub Messaging - Event Sequence



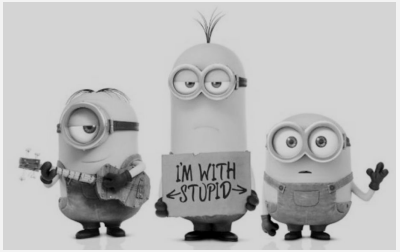
Hub Services - Batteries Included

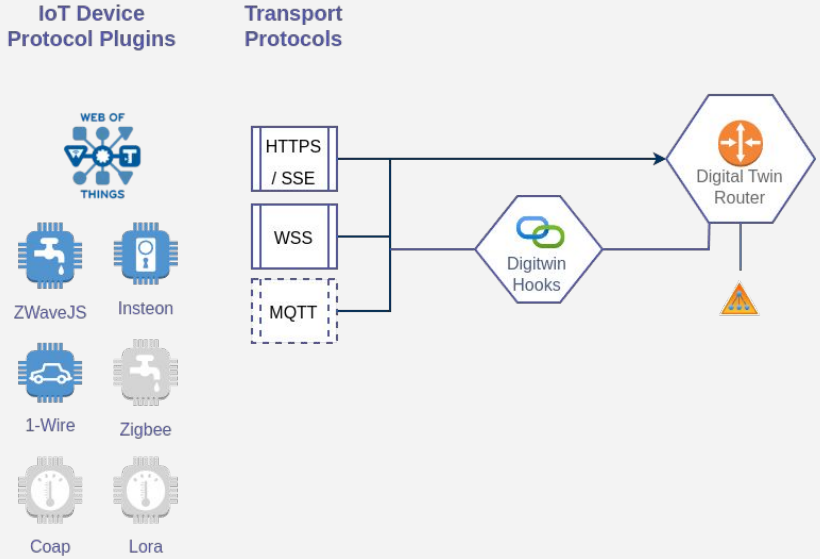
- Launcher - starting and stopping services
- History - event history
- Hiveoview - example web ui
- Certificates - server certificate management
- Provisioning - provisioning of devices
- Monitor - monitor Thing status and interactions *(*)*
- Custom - easily extensible

(*) future

IoT Protocol Bindings - Integrations

- ZWave
- Insteon
- 1-wire
- … many more to come …
  - WoT binding using TDs (smart Things) (*)
  - Coap devices
  - LoRa devices
  - Internet services
  - Node-wot
  - …





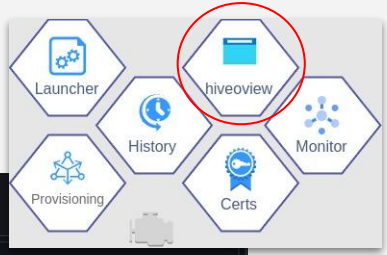
(*) future as there are currently few native WoT IoT devices

Hub Commandline Utilities

- hubcli - console admin tool to manage users and device, view events and inspect TDs.
- td2go - generate golang consumer and agent from a TD
- genvocab - generate vocabulary in golang, javascript and python

```
NAME:
   hubcli - Hub Commandline Interface

USAGE:
   hubcli [global options] command [command options]

VERSION:
   0.1-alpha

COMMANDS:
   auth:
     addu     Add a user with role and generate a temporary password
     addsvc   Add a service with its key and auth token in the certs folder.
     lu       List users
     rm       Remove a user or service. (careful, no confirmation)
     setrole  Set a new role
     setpass  Set password. (careful, no confirmation)
   certs:
     cca  Create a new Hub CA certificate
     vca  View CA and server certificate info
   directory:
     ld     List directory of Things
     disco  List discovered WoT Thing and Directories
   history:
     hev  History of Thing events
   launcher:
     ls     List services and their runtime status
     start  Start a service or all services
     stop   Stop a service or all services
   provisioning:
     idplist        List provisioning requests
     idpsubmit      Submit a provisioning request
     idpapprove     Approve a pending provisioning request
     idppreapprove  Preapprove a device for automated provisioning
   pubsub:
     pub    Publish action for Thing
     subev  SubscribeEvent to Thing events
     subtd  SubscribeEvent to TD publications

GLOBAL OPTIONS:
   --home value      Path to application home directory (default: "/home/her
   --nowrap          Disable konsole wrapping (default: false)
```

A Little Demo



HiveOT | DASHBOARD 1 | DIRECTORY

Porch/Deck

Porch Temperature °C | Porch Atmospheric Pressure mbar | Deck Air temperature °C

Shelter

Tackroom Temperature °C | Tackroom Humidity %

Cabin Temperature

Porch	10.5 °C
Basement Rack	18.6 °C
Bedroom Wall	17.6 °C
Deck	9 °C
Workshop	18.1 °C
Stairwell	21.2 °C
Living Room	22.1 °C

Cabin Power

Loft Power [W]	158.416 W
RO Pump [W]	0 W

Lights

- Stairwell Lights
- Freezer?

Outdoor Lights

- Cabin Floodlights
- Patio Lights
- Front door lights
- Yard lights

Shelter Temperatures

Pumphouse Air temperature	9.4 °C
Tackroom Air temperature	18.5 °C

Shelter

- Shelter floodlights
- Shelter post lights
- EBV105 Water Shutoff Valve

TD Challenges

These challenges were encountered in trying to implement the Digital Twin Hub with transport protocol bindings.

The challenges described are those that are understood sufficiently to formulate it as a challenge. (you know only what you know, not what you don’t know.

Additional challenges are expected to be uncovered once interoperability testing takes place.

- Challenges with TD Affordances
- Challenges with TD Forms
- Other challenges

TD Challenges - Affordances (1)

Is there a correlation between properties, actions and events?

- WoT says: There is no intended relationship between affordances with the same name
- So, what does it mean if they can have the same name?

HiveOT uses ‘twin affordances’ to improve interactions

- HiveOT assumes an implicit relationship. An action named ‘switch’ can result in an event named ‘switch’ and updates a state property named ‘switch’.
- It would be confusing to repurpose the same name with a different meanings between actions, events and properties, so this is discouraged in HiveOT..
- Defined as ‘**twin affordances**’.

TD Challenges - Affordances - actions vs properties (2)

When to use actions vs properties?

- Should a light switch be a property or an action?
- Can it be both? (hint: twin affordances)

HiveOT's rules

- If it triggers an **act**uator then it is an **act**ion
- If it has no inputs or requires multiple inputs then it is an action
- Mimics behavior in the real world. If humans have to 'act' then it is an action.
  - volume control, mute buttons, mixer sliders, media start/stop …
- Actions that affect state have a corresponding read-only property
  - Simply because anything that has state has a corresponding property

TD Challenges - Affordances - events vs properties (3)

When to use events vs properties?

- Should a light switch status be a property or an event?
- Should a temperature be a property or an event?
- Should a temperature alarm be a property or an event?

HiveOT rules

- If it reflects an **external** state then changes cause an **event** (environmental, manual controls)
- If an external state affects internal state then it is also a **property** .. twin affordances
- Devices with both internal and external control (eg switches) have triple affordances. Action for control, event for manual operation and property for current state.
- Note: events are typically something of which history is tracked. Do properties have history?
    - The hiveot history service tracks events but not properties

TD Challenges - Affordances (4)

How to get the result of an action?

- What if the action has transient states? (such as valves and blinds)
- What if the action has no state (read data requests)

No minions were hurt in this action

HiveOT Solution - standardize Action Status object

- Action Status contains the operation, status, input, output, correlationID, …
- HiveOT protocol bindings map action responses to Action Status objects
- .. or hiveot sub-protocols uses it directly

Recommendation

- Standardize the Action Status for protocol bindings that use them (*).
- Recommend use of ActionStatus responses in protocol bindings where possible
    - regardless if they are synchronous or asynchronous
    - regardless if they have output or no output

(*) these are currently protocol dependent - this indicates there is a need to standardize this

TD Challenges - Affordances (5)

How to identify the actual ‘meaning’ of a property, event or action?

- Without a standard vocabulary it is up to the human to determine what something means. This stands in the way of automated integration.
- For example, when using 100 different environmental sensors from different manufacturers, every sensor has to be manually identified for their ingestion into a BI system.

Workaround

- For now, define a ‘hiveot:’ namespace vocabulary in @context and use these in @type fields.

Ideal solution

- Adopt a W3C vocabulary standard for units, environmental and controls naming

TD Challenges - Affordances (6)

Support for enum titles?

- Enums are just a list of keys in the TD. When using these in the TD, how to provide a proper presentation for them? Eg, a title?

Workaround - OnOff

- Don’t use enums for human consumption. Use ‘OnOff’ if possible, which supports a const as value and Title for consumption.
- This makes enums mostly useless.
- OneOff is overkill for this problem and quite elaborate.

TD Challenges - Affordances (7)

How to add multi-line descriptions?

- Documenting properties, inputs or outputs can sometimes take more than a single line. However the TD only accepts a single string and JSON doesn't do multi-line strings.

Workaround

- Accept an ‘array’ of strings here. The awesome TD playground validator will complain.
  It is tough to be a validator.

TD Challenges - Affordances (8)

How to support multi-line comment fields

- Same problem as with multi-line ‘description’ fields.

Workaround

- Accept an array of strings here. Same as previous point.

TD Challenges - Affordances (9)

How to describe a map of objects in the data schema?

Official solution (uhm, I think ..)

- Define an object schema with a properties and only a single empty key

```
“type” : “object”,
“properties”: {
      “”: {
         “title”:”ugh”,
         “type”
         “schema”: … yet another schema
      }
}
```

Workaround

- Don't use maps, use arrays.

Recommendation

- Add support for a ‘map’ data type.

TD Challenges - Affordances (10)

How to read the latest event value?

- A UI that displays events might want to show the last known event. However, the TD has no operations for ‘readevent’ and ‘readallevents’.

Workaround

- HiveOT adds these operations in the “hiveot:” namespace.
- The digital twin store also stores the latest event values.
- The history services persists historical events

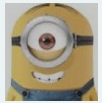
TD Challenges - Affordances (11)

How to describe basic vs advanced properties, events and actions in the TD?

- Use case: human consumers might be interested in some properties, events or actions but not all of them. The human consumer should be able to just look at the essential data without being overwhelmed with more advanced properties.
- Some zwave devices have close to 100 properties of which only a handful are useful to the regular consumer. How to differentiate them?

Options

- 1: use the hiveot vocabulary to indicate basic properties. This is not compatible with anything.
- 2: don’t put advanced properties in the TD. Uhm, right ….
- 3: convince the WoT group to add a ‘advanced’ property affordance field to TD 2.0



TD Challenges - Affordances (12)

How to define global/shared constants within and between things?

- Values can be used in multiple properties, events and actions
- Values can be shared or re-used in multiple things

Workaround (partial)

- Add a Thing level dataschema with a enum constants to ‘schemaDefinitions’

TD Challenges - Forms (1)

Why is a Form required for every .. property .. event .. and .. action?

- Forms generate a huge amount of bloat especially for hubs and gateways where all Things can be reached using the same messaging protocol and connection endpoint
- Also, most devices only need a single connection to use the supported operations.

Proposal (currently implemented in hiveot)

1. Move the use of forms to the protocol binding specification.
    - Backwards compatible with existing http-basic and sse bindings
    - Protocol bindings can inject the forms they need into the TD before it is published.
    - New protocols can specify what they need to connect and exchange messages

TD Challenges - Forms (3)

How to include metadata (thingID, name, clientID) in SSE-SC messages?

(sse-sc is a variant of sse that uses a shared connection for all sse messages)

- Use-case: agent receives an action for a Thing via its SSE connection (agents connect to the Hub). The SSE data is the input data as per TD, but how to convey the thingID, action name and messageID?

Workaround (implemented)

- Create a new protocol binding called sse-sc that shares the connection for all operations
- Use the standard message envelopes for requests/response/notification.
- Currently re-using the draft websocket message envelopes

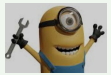
TD Challenges - Forms (3)

How to link one or more asynchronous responses to an action?

- Use-case: Consumer invokes action which will return one or more responses after some time. How to describe this?

Solution - Use a correlationID (Implemented in HiveOT)

- Include a correlationID in each request (invokeaction and writeProperty)
- Include the same correlationID in progress notifications and the response.
- Do not use protocol bindings that do not support this.



Recommendation

- Consider use of message envelopes that carries this information. This will enable the same behavior across all protocols that support the message envelopes. As a bonus this standardization simplifies testing.

TD Challenges - Forms (4)

Is this a transport, or application problem?

How to confirm writing a property?

- Use-case: Consumer writes a property value. A UI dialog closes when a confirmation is received or displays an error if failed. How to get confirmation or failure response?
- Note that property updates can take a while to be applied.

Solution (implemented in hiveot)

- writeProperty always returns a response on success or on failure.
- Synchronous protocol bindings, eg http-basic, can return a synchronous response after the request is sent or wait for completion. The latter is not good for gateways and hubs though.
- This is a similar problem to invoking actions.
- When the property has updated all observers are notified.

TD Challenges - Forms (5)

How does a client know if an output or alternative response is received?

- A client invokes an action and expects an output value. The TD describes a form with additionalResponses, for example to report an error. The client receives the response as per TD and needs to differentiate somehow between normal and additional responses.
  Section "5.3.4.2.2 Response-related Terms Usage" describes a response name-value pair that can be used, but where is it described?
- How does this fit in the response data?

Workaround 1

- Parse the expected output and on failure parse using the schemas from additionalResponses. The ‘try it until it works' approach is not really a good specification. (although Python users might like it ;)

Workaround 2

- Include a 'dataschema' field in the response that describes the dataschema used in the result. This might be a more elegant approach with nr 1 as a fallback. This is protocol dependent unfortunately and cannot be relied on.

TD Challenges - Forms (6a)

A case for standard message envelopes separate from the protocol bindings.

Currently:

- Thing operations are implemented in protocol binding API's. The application layer and transport layer get mixed. This complicates implementation and limits code re-use.
- Servients implement message formats for each operation and each protocol binding. Without standardization each binding ends up with different messages for operations like:
  - Read property, properties, all properties
  - Query action, actions, all actions,
  - Subscribe event, subscribe all events
  - Observe, ..
  - etc, etc
- As a result each protocol binding has to specify how to encode the payload of the operations, including operation, thingID, affordance name, errors, correlationID, timestamps.
- These fields are WoT application level concerns, not transport level concerns. Why does a transport have to specify how to encode them. A transport should only concern themselves with transporting a payload.

TD Challenges - Forms (6b)

A case for standard message envelopes separate from the protocol bindings.

Proposal (with a bit of hand waving)

- Define the message envelope/data schema for each Thing level operation.
  - All messages includes the operation, thing-id, affordance name, correlationID, messageID, error info, and data fields.
  - A correlation field is used for linking requests to responses.
  - Use a RequestMessage envelope for write property and invoke action requests.
  - Use a ResponseMessage envelope for returning results.
  - Use a NotificationMessage envelope for asynchronous response such as event and property updates and action progress.
- Transport protocol bindings only concern themselves with passing the message envelopes.
  - Transport bindings can read the envelope to implement subscriptions
  - Transport protocol are added to the TD and defines how to connect to send and receive messages.

TD Challenges - Forms (6c)

A case for standard message envelopes separate from the protocol bindings.

Benefits:

- Defines (currently unspecified) Thing level operation payloads
- Simplifies protocol binding implementations
- Simplifies application development as all protocols use the same message envelopes
- Extensible with additional operations without changing the transport protocol bindings
- Opens the door to a standard testing framework for messaging of operation

Backwards compatible:

- The decision to use these message envelopes are part of the protocol binding specification
- Legacy devices can still use protocol binding that inject forms in the TD. eg http-basic.

TD Challenges - Other (1)

How to express authorization rules (eg required roles) in a TD?

- Example: An explosive detonator has the remote denotation action in the TD, available to the operator. It wouldn’t be good to allow the ‘detonate’ action to be used by tourists.
- The problem is that there is no authorization support in the TD.

Workaround 1

- Take a 'capabilities' approach. Split actions requiring different authorization into multiple Things. The service programmatically tells the authorization service which roles can use a Thing it publishes.

Workaround 2

- Add a custom 'allow' and 'deny' field to each action that lists which roles are allowed/denied invoking an action. Consumers only see allowed actions based on their role.

TD Challenges - Other (2)

How to implement affordances in golang without inheritance or unions?

- The TD describes a property affordance as a subclass of interaction affordance and a dataschema. A dataschema however has several subclasses: ArraySchema, BooleanSchema, NumberSchema, …
- How can the (compile time) inheritance tree depend on the value of the type field? How can a property affordance be a subclass of all these subclasses? Golang has no union support.

Option 1

- Implement DataSchema as a flattened union of all schemas, containing all fields.

Option 2

- Use a map[string]any and determine the expected data schema at runtime using reflection.
    - Wastes CPU cycles, especially on small devices.

Challenges - Other (3)

Use-case:

> Consumer needs to login to the Hub, obtain an auth token, and occasionally refresh the auth token. Oauth2 is not used. The token can be a JWT or paseto token.

How to describe a hub login with password, auth token, and refresh a token in a TD?

- Note: this hasn't been investigated properly and is on the todo list so it might have an easy answer.
- Does this depend on the protocol binding?
- What to do if the binding doesn't describe it?

Challenges - Other (4)

1. The TD defines fields that can contain different types (string vs array). How to implement this is a strictly typed language such as golang.
    a. Workaround. Use (*cough cough*) ‘any’ and provide a GetMethod that uses reflection to convert to a single type.
2. How to describe a RPC service API in the TD?
    a. This requires a correlation between request and response
3. SecurityScheme definition (5.3.3.1)
    a. The fourth paragraph: "Security schemes generally may require additional authentication parameters, such as a password or key. The location of this information is indicated by the value associated with the name in, often in combination with the value associated with name."
    b. I don’t understand this sentence. Is this an example of security through obscurity? ;)

Future

Improve WoT interoperability.

Websocket and Mqtt transport protocol implementations.

Lots .. LOTS .. more IoT protocol bindings (once things are stable).

A HiveOT bridge to connect Hub instances, building a real hive of hubs.

Monitoring and simulation for digital twins

… apparently a lot more work …