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
- 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

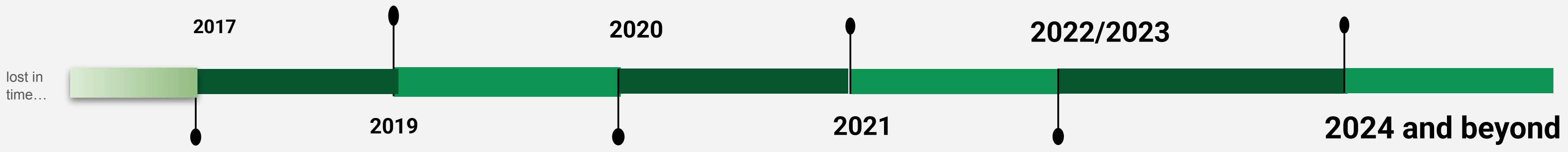
Introduction - Why HiveOT?

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



Iot train

Connecting IoT devices and services using MQTT.

myzone

MyZone provides a 'it just works' approach to IoT data collection and automation using IoT sensors, devices and services.

iotdomain

Simple standard for M2M discovery and exchange of information from IoT devices and related services.

NGSI-LD based

API based variant based on NGSI.

gRPC / Cap'nProto

RPC based variants to explore RPC instead of message bus approach.

HiveOT

Redesign based on WoT TD-1.1 web of things. Multi-protocol.

(Because its fun)

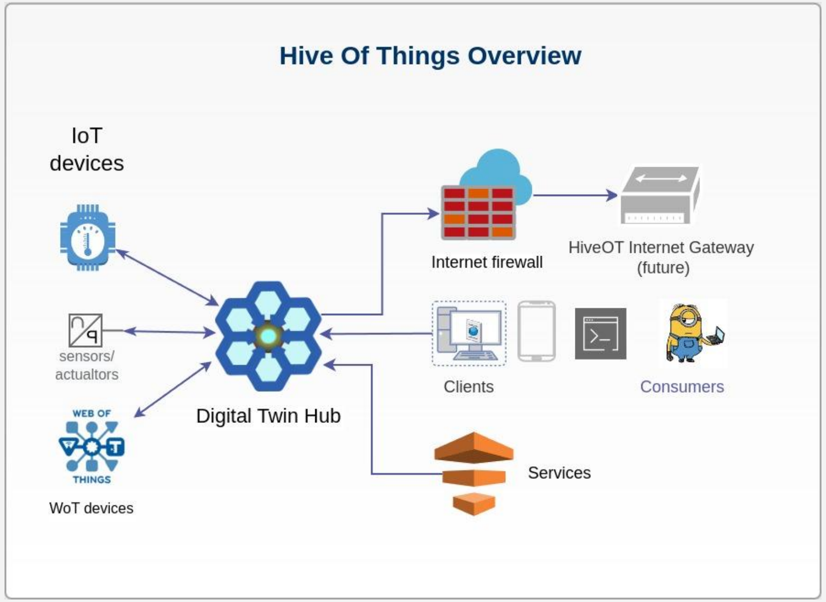
Introduction - What is HiveOT

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*

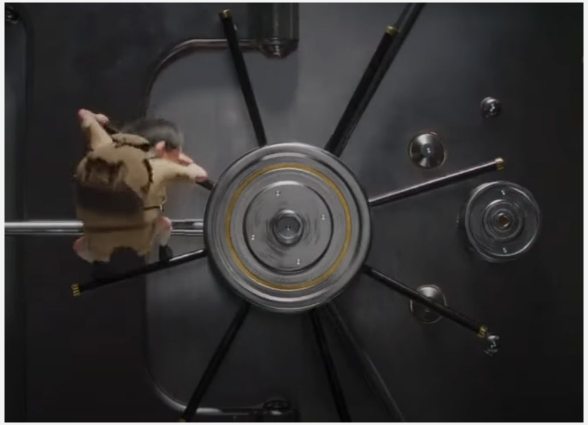
Security of Things - HiveOT

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.

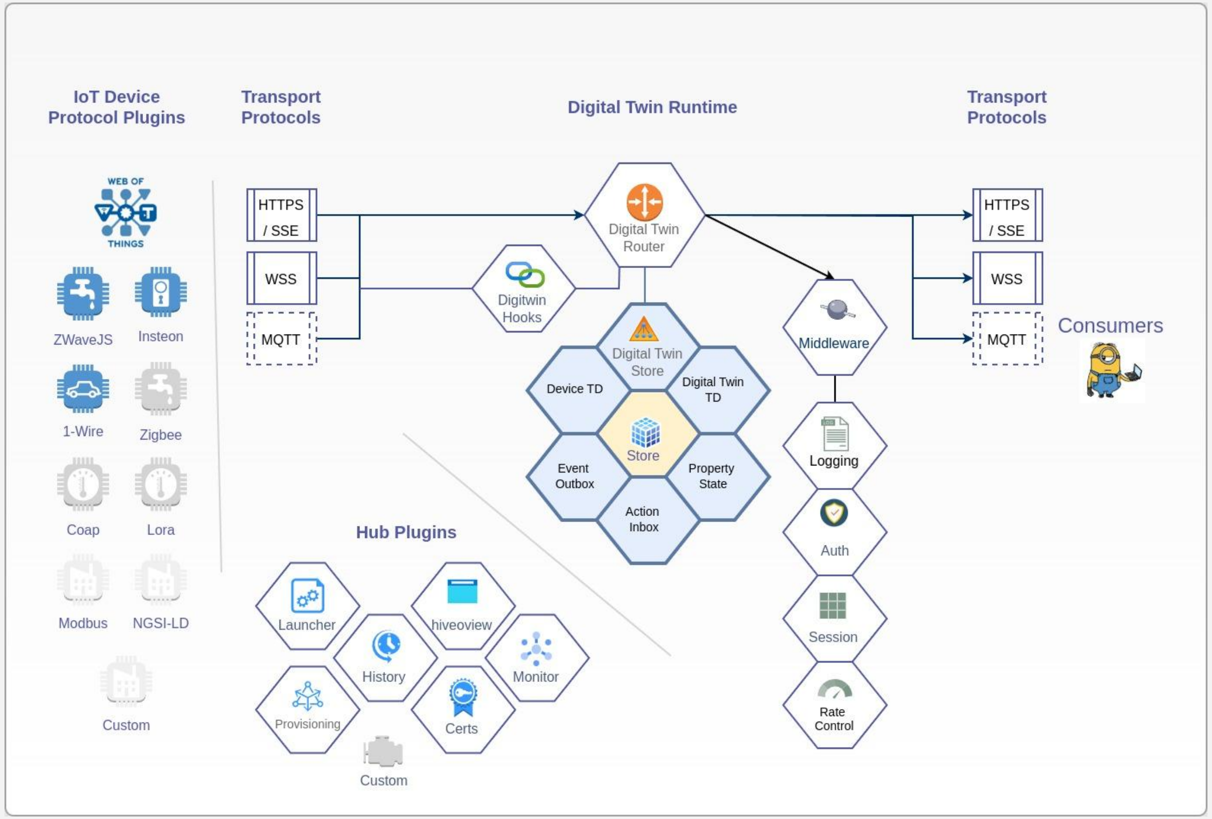
Benefits of using a Hub. Provide centralized:

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



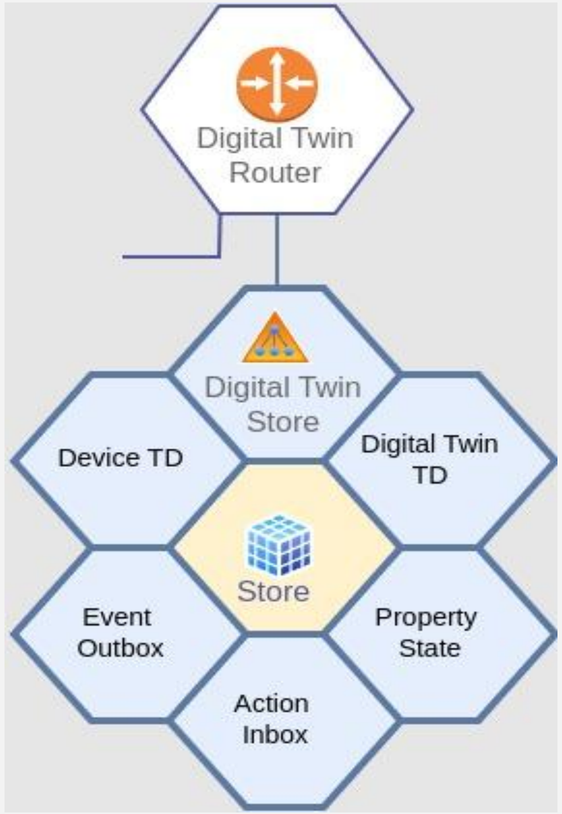
HiveOT Hub - An Edge Device

- 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

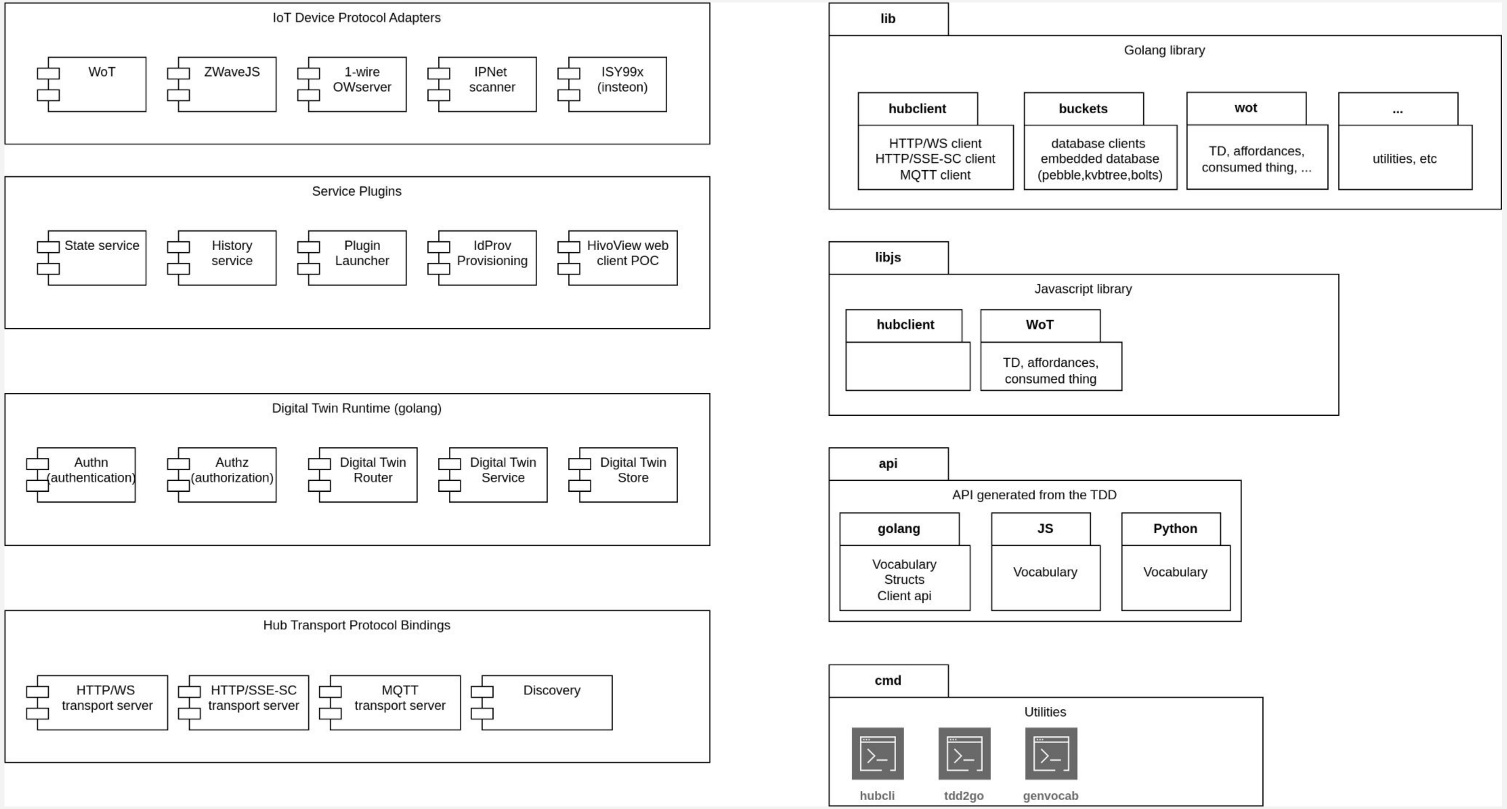


Digital Twin Runtime

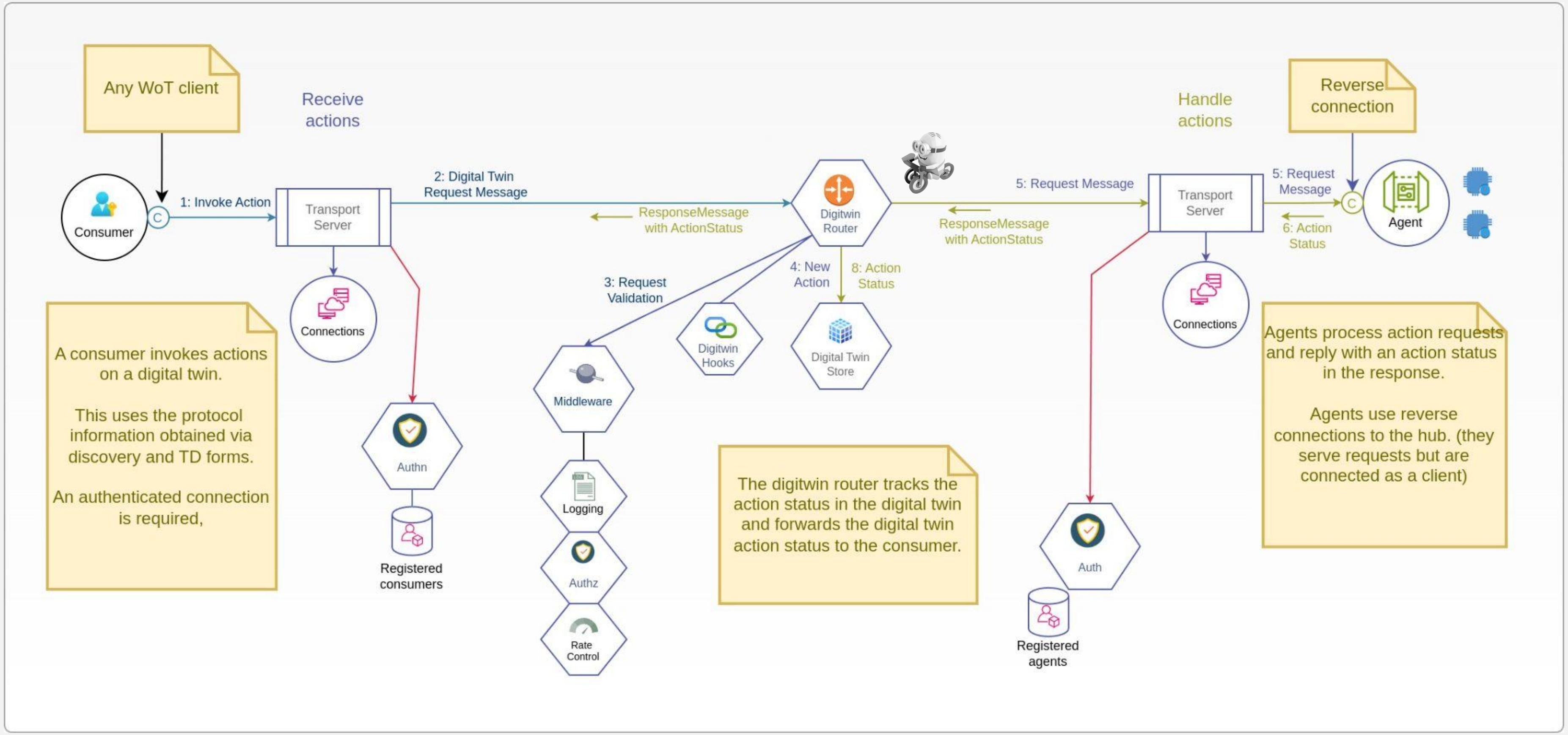
- HiveOT uses two TD's per Digital Twin:
  - Device TD
    - Sent by device agents
  - Digital Twin TD
    - Served to consumers
    - Forms updated with with Hub endpoints
    - Identify as digital-twin Thing-ID: dtw:{agentID}:{thingID}
- Consumers interact with the digital twin
  - Requests are routed 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



A consumer invokes actions on a digital twin.

This uses the protocol information obtained via discovery and TD forms.

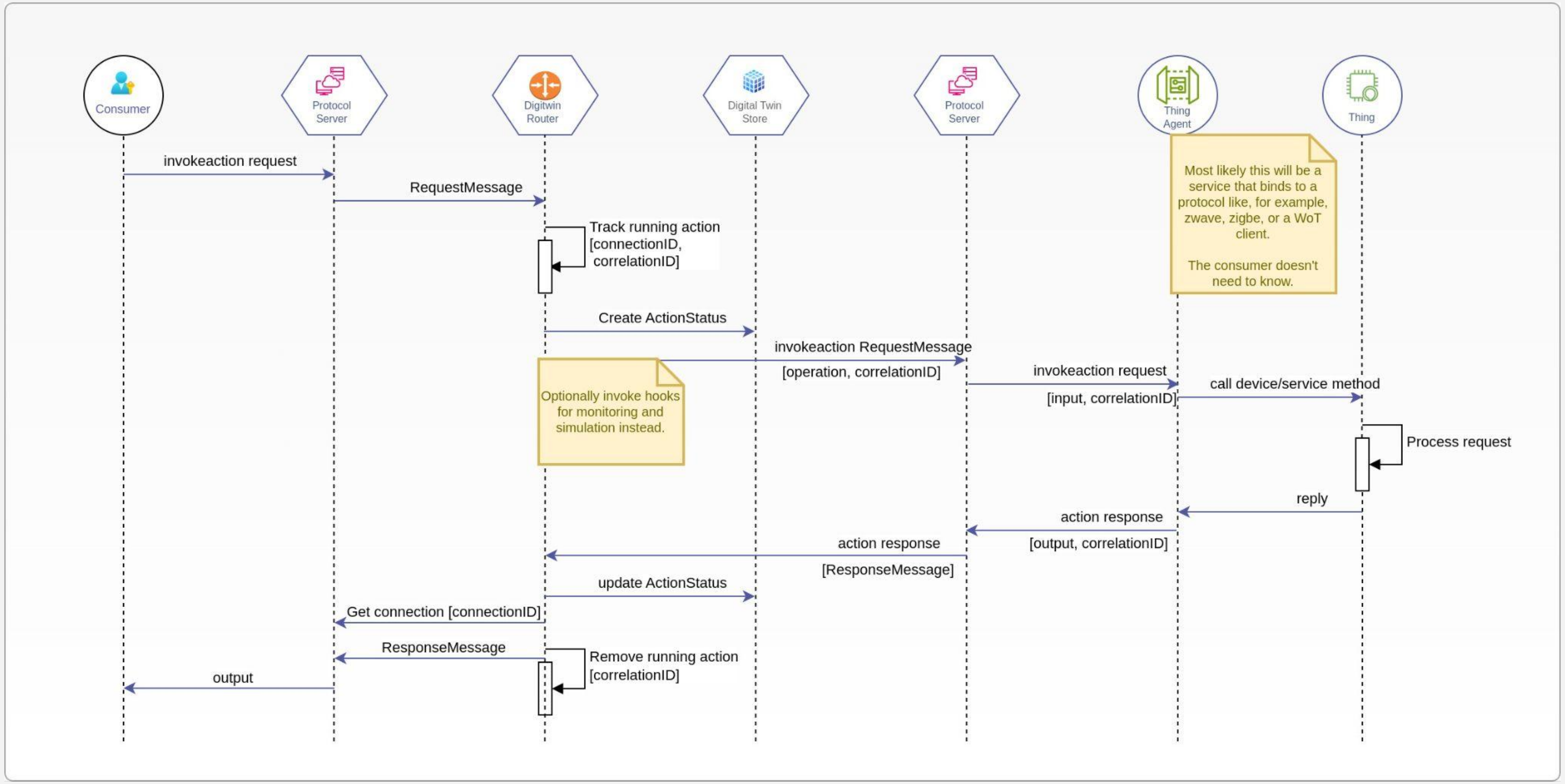
An authenticated connection is required,

The digitwin router tracks the action status in the digital twin and forwards the digital twin action status to the consumer.

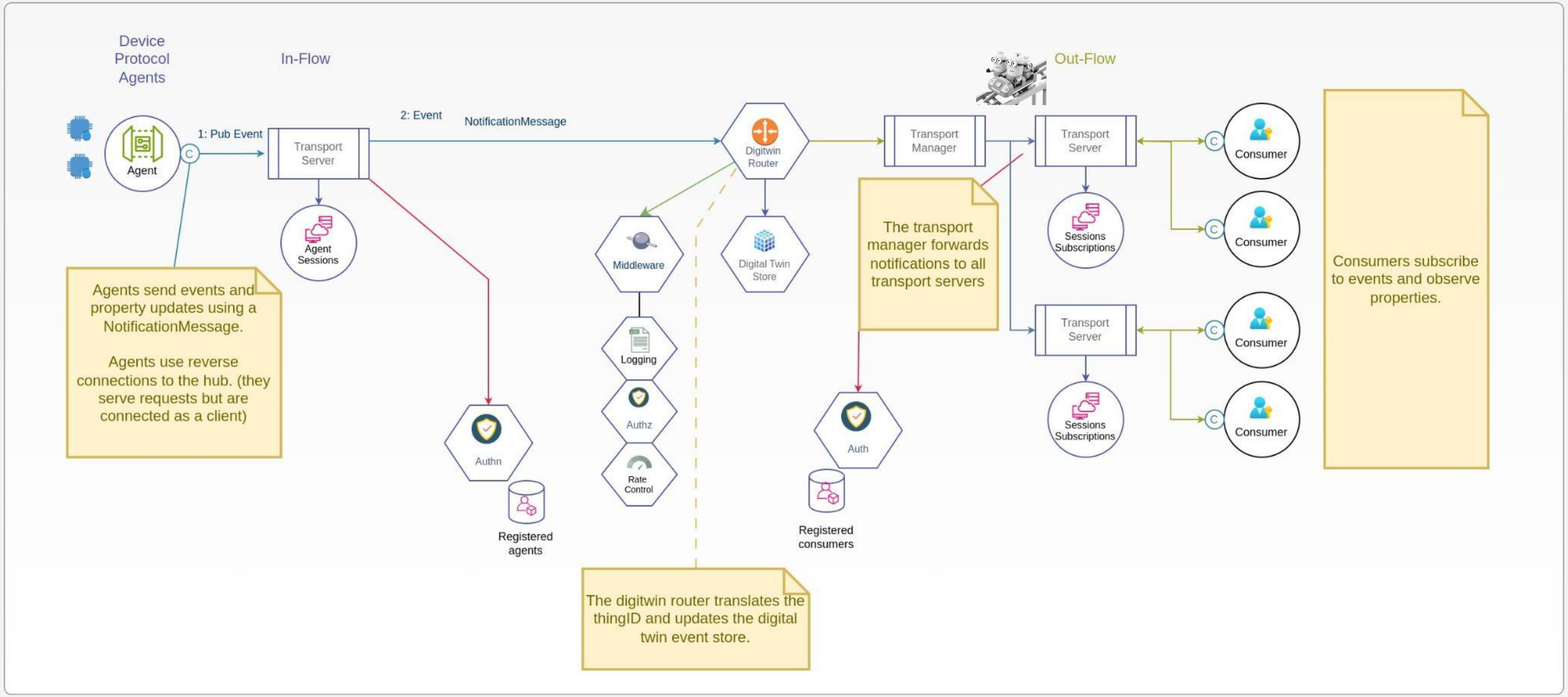
Agents process action requests and reply with an action status in the response.

Agents use reverse connections to the hub. (they serve requests but are connected as a client)

Hub Messaging - Action sequence


Consumer
Protocol Server
Digitwin Router
Digital Twin Store
Protocol Server
Thing Agent
Thing
invokeaction request
RequestMessage
Track running action [connectionID, correlationID]
Most likely this will be a service that binds to a protocol like, for example, zwave, zigbe, or a WoT client.
The consumer doesn't need to know.
Create ActionStatus
invokeaction RequestMessage
[operation, correlationID]
Optionally invoke hooks for monitoring and simulation instead.
invokeaction request
[input, correlationID]
call device/service method
Process request
reply
action response
[output, correlationID]
action response
[ResponseMessage]
update ActionStatus
Get connection [connectionID]
ResponseMessage
Remove running action [correlationID]
output

Hub Messaging - Event Flow



Agents send events and property updates using a NotificationMessage.

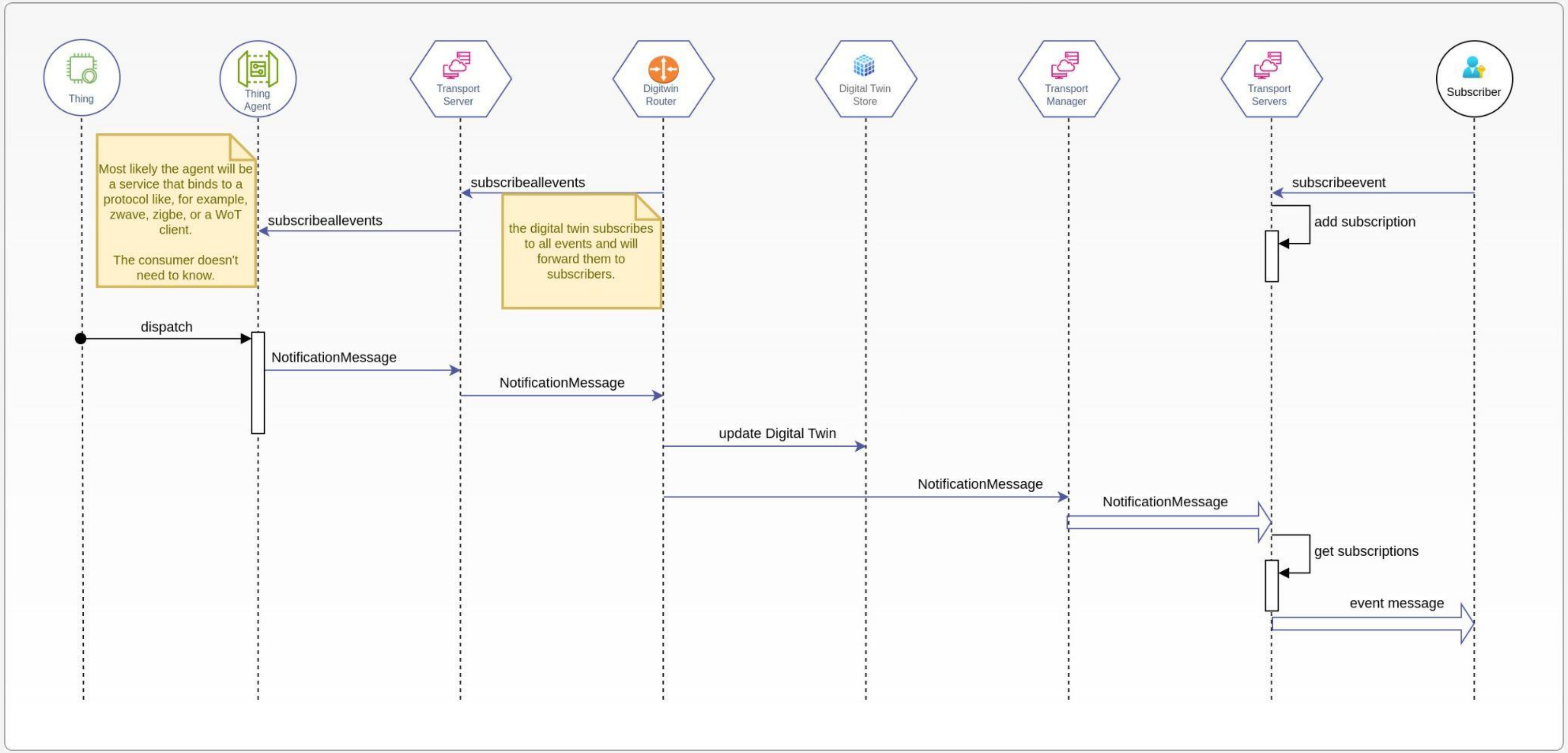
Agents use reverse connections to the hub. (they serve requests but are connected as a client)

The digitwin router translates the thingID and updates the digital twin event store.

The transport manager forwards notifications to all transport servers

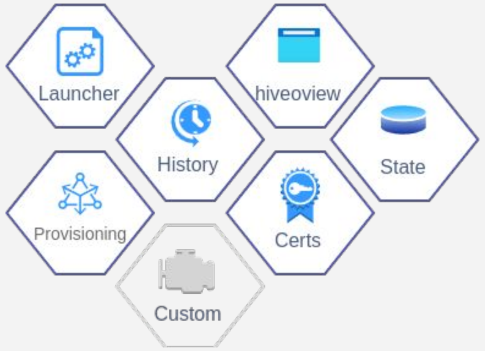
Consumers subscribe to events and observe properties.

Hub Messaging - Event Sequence



Hub Services - Batteries Included

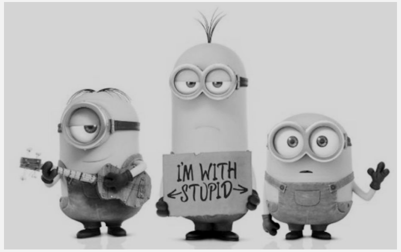
- Launcher - starting and stopping services
- History - event history
- Hiveoview - example web ui
- Certificates - server certificate management
- Provisioning - provisioning of devices
- Monitor - monitor Thing status and interactions *(*)*
- Custom - easily extensible

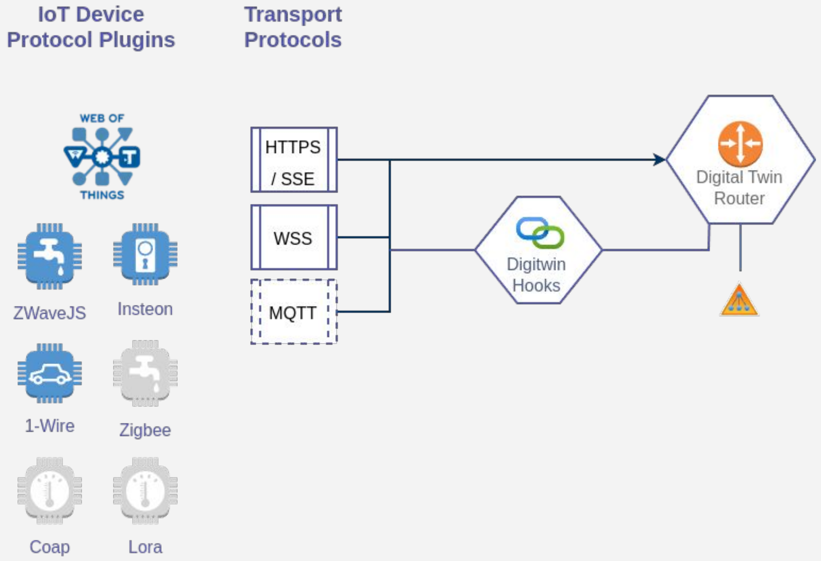


(*) future

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
    - …



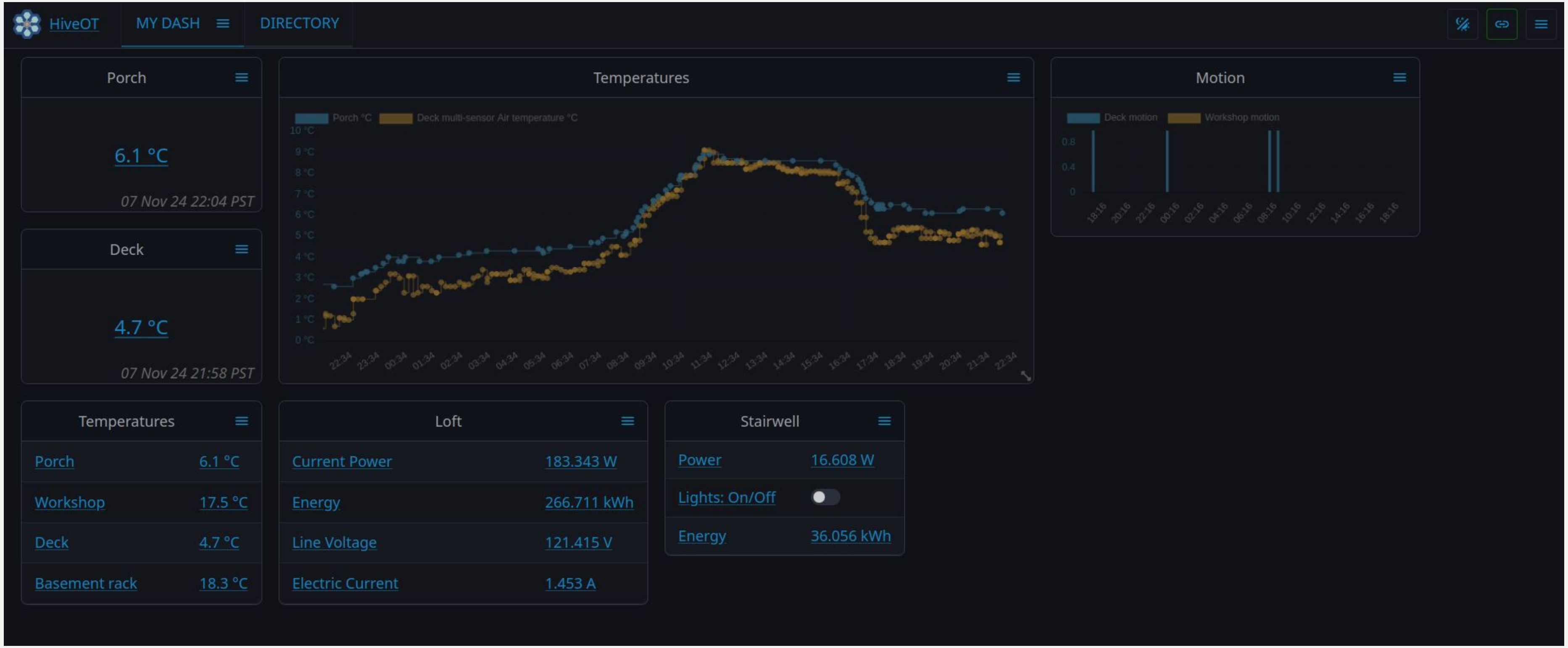


(*) future as there are currently few native WoT IoT devices

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

A Little Demo



Porch

6.1 °C

07 Nov 24 22:04 PST

Deck

4.7 °C

07 Nov 24 21:58 PST

Temperatures

Motion

Temperatures

Porch	6.1 °C
Workshop	17.5 °C
Deck	4.7 °C
Basement rack	18.3 °C

Loft

Current Power	183.343 W
Energy	266.711 kWh
Line Voltage	121.415 V
Electric Current	1.453 A

Stairwell

Power	16.608 W
Lights: On/Off	
Energy	36.056 kWh

(but, .. where is the pumphouse?)

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 there is a button or switch for it 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 the result of an event affects internal state then it is **also** a property .. twin affordances
- Internal state with both internal and external control (eg switches) have triple affordances. Action for control, event for manual operation and property for current state.

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.

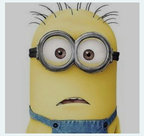
TD Challenges - Affordances (9)

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)

The TD describes consumer - device interaction. How to describe device - hub interaction?

- Devices publish a TD, events, and property updates
- Devices receive (or subscribe) actions and property writes.

Workaround

- Define as custom operations of the ‘digitwin’ service.

TD Challenges - Forms (2)

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

- Thing level operations can describe how to read and write properties using URI variables, no per-property form is needed.
- Same for events (subscribe) and actions
- Per-affordance Forms generates a lot of bloat

Workaround (to be discussed)

- Make Forms optional
- Specify payload for Thing level forms

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 1

- Create a new protocol binding called sse-sc that shares the connection
- Specify a message envelope for properties, events and actions as is done in the Websocket specification.
- Might be a good future path that is more extensible.

Workaround 2 (chosen workaround)

- Create a new protocol binding called sse-sc that shares the connection.
- Use the SSE ID field to add metadata: {thingID}/{name}/{messageID}.
- This is rather ugly as ID field is not intended for this purpose.

TD Challenges - Forms (4)

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?

Option 1 (chosen solution - non standard)

- Send an action status response message on each update and include a correlation ID in both request and response.
- This has to be specified in each protocol binding.

Option 2 (non standard)

- Send an event with the same name as the action and include a correlation ID.
- This makes handling it protocol independent for the application although the method of include the correlation ID is protocol dependent.

TD Challenges - Forms (5)

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.

Workaround 1

- Use an event with the same name as the property and include a correlation ID in the request and response messages. No idea how to describe this in the TD Forms.

Workaround 2

- Handle it the same way as an invoke action operation. (different for each protocol binding)
- Thing pushes a PropertyStatus message to the consumer.

TD Challenges - Forms (6)

Where does the encoding/decoding of input and output payloads take place?

- TD Forms define a "contentType" field that describes the encoding of the payload.

Answered

- This is clear now. Encoding is handled in the transport protocol. The forms in the TD contain the available transport protocols and its encoding, for every single property, event and action affordance. It just took some time to figure it out.

TD Challenges - Forms (7)

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 a transport header that describes the dataschema used in the result. (currently being tested). This might be a more elegant approach with nr 1 as a fallback.

TD Challenges - Forms (8a)

A case for a WoT application protocol separate from the transport protocol.

Currently:

- Thing level operations do not have a payload definition. They can be very useful though to reduce the amount of Forms needed.
- 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 are required to implement a different and unnecessary large API footprint for each protocol binding. Without standardization each binding has a different API.
    - Read property, properties, all properties
    - Query action, actions, all actions,
    - Subscribe event, subscribe all events
    - Observe, ..
    - etc, etc
- Protocol bindings will have to convert the operations and payload to a serializable message format and expand it on receiving it.
    - Why not standardize the message format instead? A transport protocol shouldn't have to concern itself with the payload.

TD Challenges - Forms (8b)

A case for a WoT application protocol separate from the transport protocol (cont.)

Proposal (with a bit of hand waving)

- Operations become part of a WoT application level protocol that is independent of the underlying transport protocol.
- Define the message envelope/data schema for each Thing level operation. There is quite a bit overlap so not necessarily 1:1.
  - All messages includes the operation, thing-id, affordance name, and data fields.
  - Include a correlation field for cases where a response is expected.
  - Potentially use a single message envelope, if this turns out to be possible.
  - Specify both request and response messages (eg read property request and read property response)
  - There are no forms per operation.
- Transport bindings are only used for passing the messages from consumer to agent and vice-versa, regardless of its content.
  - Transport bindings can read the envelope to implement subscriptions
  - A simple transport ‘Form’ in the TD defines the transport operation for publishing and subscribing messages
- Add Thing operations to publish events, property value updates as part of the WoT application protocol (These are application concerns, separate from the transport concern)

TD Challenges - Forms (8c)

A case for a WoT application protocol separate from the transport protocol (cont.)

Benefits:

- Fills a gap of missing Thing level operation payloads
- Greatly simplifies protocol binding implementations as a much smaller API footprint is needed
- Re-use the same application message handling regardless of the underlying transport protocol
- Extensible with additional operations without changing the transport protocol bindings

But wait, there is choice:

- In the name of freedom of choice and backwards compatibility, a protocol binding specification can specify if the message envelope is used or whether the ‘legacy’ forms method is used.

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. Size of the TD
2. 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.
3. How to describe a RPC service API in the TD?
   a. This requires a correlation between request and response
4. The TD doesn’t do X, how to do it?
   a. Just because a TD can be used to describe a toaster that doesn’t mean it can be used as a toaster. Can’t do everything.
5. 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 …