4. Technical Requirements

The previous section delineated the Web of Things abstract architecture by showing various use cases and enumerating patterns for combining architectural components. This section describes technical requirements derived from the abstract architecture.

Components consisting Web of Things and the Web of Things architecture

Observation of the use cases shows that basic components such as devices, applications that access and control those devices, proxies (e.g. gateways and platforms) that sit between devices and applications to link them together, directories that provides device search functionality are required.

Those components are connected to the internet or field networks in offices, factories or other facilities. Note that all components involved may be connected to a single network in some cases, however, components can be deployed across multiple networks in general.

Devices

Access to devices are made based on descriptions of their functions and interfaces (i.e. Thing Description). Those include general description about devices, information models representing functions, transport protocol description for operating on information models, and security information, and so on.

General descriptions are about device identifiers (URI) and other human readable information.

Information models defines device attributes, and represent device’s internal settings, control functionality and notification functionality. Devices that have the same functionality have the same information model regardless of the transport protocols used.

Because many systems based on Web of Things architecture are crossing system Domains, vocabularies and meta data (e.g. ontologies) used in information models should be commonly understood by involved parties.

In addition to REST transports, PubSub transports are also supported.

Security information includes descriptions about authentication, authorization and secure communications.

Devices are required to put TDs either inside them or at locations external to the devices, and to make TDs accessible so that other components can find and access them.

Applications

Applications need to generate and utilize program interfaces internally based on device descriptions (i.e. TD).

Applications have to be able to obtain device descriptions (i.e. TD) through the network, therefore, need to be able to conduct search operation and acquire the necessary TD over the network.

Proxies

Proxies need to generate program interfaces internally based on device descriptions (i.e. TD), and to represent virtual devices by using those program interfaces. A proxy then has to produce a TD for the virtual device and make it externally available.

Identifiers of virtual devices need to be newly assigned, therefore, are different from the original devices. This makes sure that virtual devices and the original devices are clearly recognized as separate entities. Transport and security mechanisms and settings of the virtual devices can be different from original devices if necessary.

Virtual devices are required to have TDs put either inside the proxies or at locations external to the proxies, and to make the TDs accessible so that other components can find and access them.

Directory

For TDs of devices and virtual devices to be accessible from devices, applications and proxies, there needs to be a common way to share TDs. Directories can serve this requirement by providing functionalities to allow devices and proxies themselves automatically or the users to manually register the descriptions.

Descriptions of the devices and virtual devices need to be searchable by external entities. Directories have to be able to process search operations with search keys such as keywords from the general description in the device description or information models.

Security Function

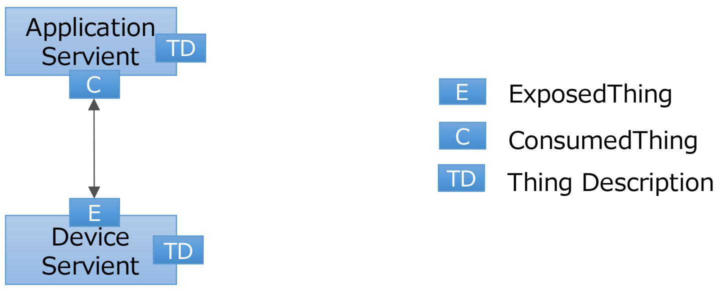
Security information related to devices and virtual devices needs to be described in device descriptions. This includes information for authentication/authorization and payload encryptions.

5. WoT servient architecture

This section defines components that make a Web of Things entity, clarifies each component's functions, and then shows how a web of Things as a whole works. The architecture described here is derived from the Web of Things use cases as well as technical requirements extracted from them.

High-level architecture

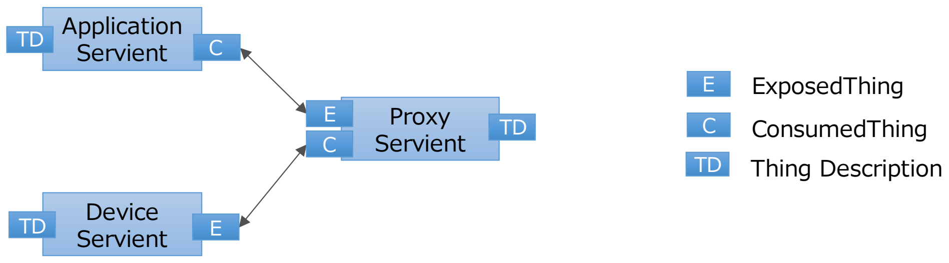
Overview of WoT component's behavior is explained using a couple of system configuration diagrams. First, shown below is a configuration that consists of a device and an application shown below.



Functions of a device is described by a Thing Description (TD). A TD describes, among other things, a device's identifier, functions and attributes of a device implemented internally, communication protocols (i.e. transport layer) information. Details of TD format are defined the Thing Description specification.
Each Web of Things device MUST have a corresponding TD. Applications can recognize a device by obtaining a TD of the device.
Conceptually, a device can be thought to have an ExposedThing that provides an interface that conforms to the TD. On the other hand, an application can be conceptually considered to have a ConsumedThing that provides an interface functionality for an application to utilize. An application can generate a ConsumedThing upon a receipt of a TD. Communication between an application and a device are realized by ConsumedThing and ExposedThing connect to each other

and exchange messages.

Next, in the below configuration, an application and a device connect to each other via a proxy.



A proxy contains both ExposedThing and ConsumedThing functionality, and relay messages that are exchanged between an application and a device. In a proxy, a ConsumedThing creates a shadow of the device, and an application can access the shadow device through the proxy's ExposedThing.
Proxy's ExposedThing and ConsumedThing can communicate in different protocols. For example, a device and an application can use separate protocols, CoAP and HTTP, respectively. Even when there are multiple devices and they use different protocols, an application can communicate with those devices using a single protocol through the proxy. The same is true for device authentication. An application only need to handle a single authentication method even when multiple devices connected to a proxy use different authentication methods.
A proxy creates a ConsumedThing based on a TD and generates another TD for a shadow device. The TD for a shadow device uses a new identifier different from the original device TD's, and changes communication protocols if necessary. A proxy then creates an ExposedThing that serves as the Thing for the TD. An application communicates with a device via a proxy through a ConsumedThing that works according to the TD for the shadow device.

Simple configuration of WoT servient

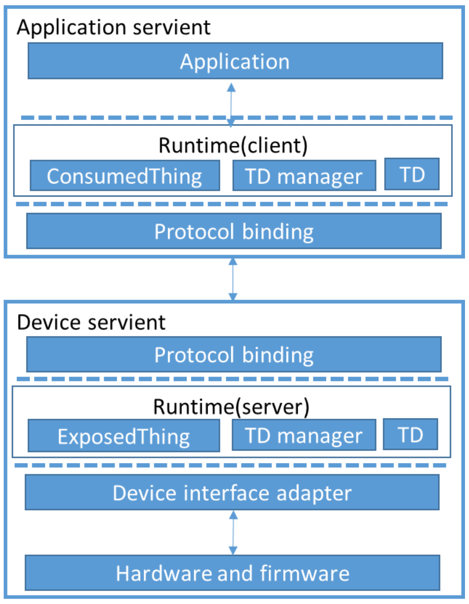
The remainder of this section discusses and clarifies the inner structure of a servient.

First, inner structure of a servient is explained based on the structure of an application servient and a device servient.

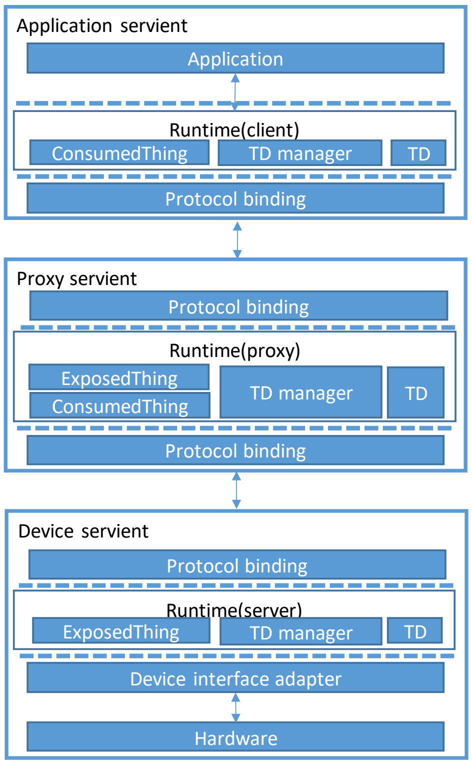
WoT runtime defines an abstract interface that contains methods such as READ, WRITE, INVOKE, SUBSCRIBE, NOTIFY through which to access devices. When a servient communicates over a network, those methods are bound to concrete protocols. For example, in the case of HTTP, HTTP methods such as PUT and GET are assigned to implement the abstract methods. WoT clients can retrieve and change device settings or device attributes defined in TDs through those abstract methods.

In a device servient, TD manager creates an ExposedThing based on TD. TD manager also advertises a TD to other servients, or provides other servients with a TD upon request. A device servient's functions are usually implemented as hardware, and those functions are accessible through its firmware. An ExposedThing's abstract interface is translated into hardware commands by a device interface adapter. A device interface adapter is a custom software developed for each specific type of hardware.

In an application servient, on the other hand, a TD manager obtains TDs by getting one from location advertised by other servients or requesting other servients of TDs, and creates a ConsumedThing based on the obtained TD. Functions of an application servient are usually implemented as an application. The abstract interface of a ConsumedThing is provided as a programming language (such as JavaScript) interface, and the application achieves its functions by using this interface.

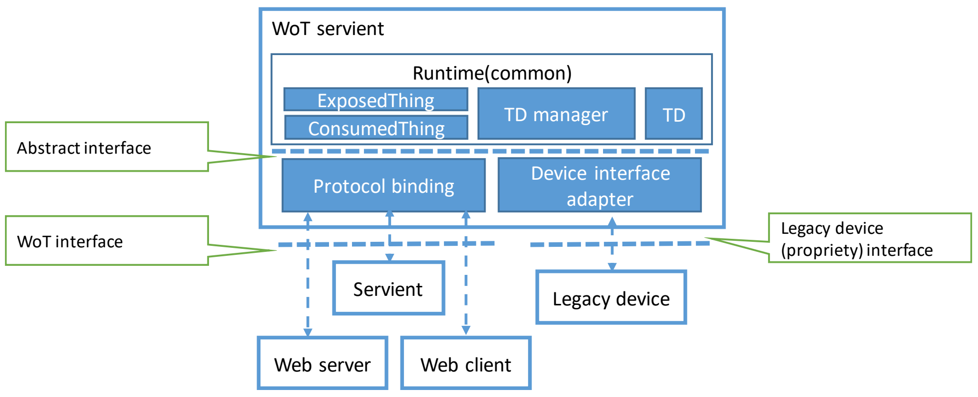


Next, inner structure of a servient is explained based on the structure of a proxy servient that connects a device servient and application servient together.



A runtime in a proxy servient has both ExposedThing and ConsumedThing functionality. A TD manager, similar to that of an application servient, obtains a TD of a device servient, and creates a ConsumedThing. At the same time, a TD manager creates a shadow of the device as well as a TD for the shadow device where the identifier is a new one and protocol bindings are appropriately described to serve for the application. An ExposedThing is created based on this TD, and a TD manager notifies other servients of the TD.

Below is a WoT architecture diagram that summarizes the inner structure of a WoT servient. A runtime creates ExposedThing or ConsumedThing based on TDs, provides an abstract interface that through protocol binding can interact with other servients. Note that the diagram also contains a legacy device that does not by itself support WoT abstract interface. A device interface adapter converts legacy interface into an abstract interface to allow them to interact with the runtime.


WoT servient
Runtime(common)
ExposedThing
ConsumedThing
TD manager
TD
Abstract interface
Protocol binding
Device interface adapter
WoT interface
Legacy device (propriety) interface
Servient
Legacy device
Web server
Web client