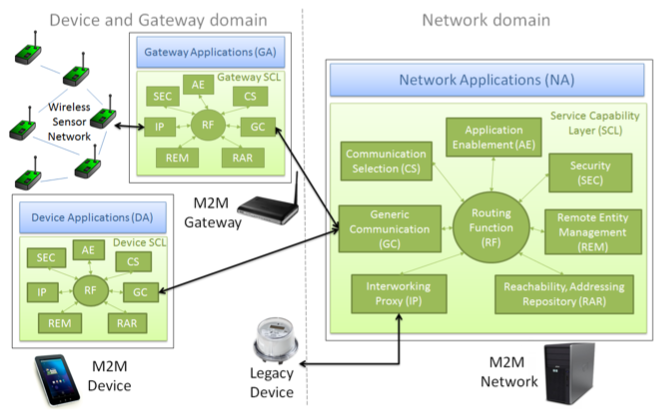
# OM2M

## Smart M2M standard (ETSI M2M)

* Provide a horizontal M2M service platform with a generic set of service capabilities to enable M2M interoperability in terms of communication and data.
* Define a Service Capability Layer (SCL) on top of connectivity layers:
* SCL deployed in network server, gateways and devices.
* SCL enables discovery, registration, authentication, data-transfer using containers, publish/subscribe, groups, access rights, security, etc.
* Interface with existing technologies:
* Multiple communication protocol binding.
* Re-use existing remote device management.
* Interwork with vendor- specific and legacy devices.
* Facilitate innovation across industries by exposing data and providing services.



* OM2M implements a RESTful API.
* All M2M communications are performed based on simple primitive procedures.
* Each SCL data model is structured using a standardized resource tree.
* The resources can be simply triggered using basic CRUD requests.
* OM2M is a java platform running on top of an OSGi Equinox runtime which make it highly extensible via plugins.
* Each SCL includes required plugins and is build as an Eclipse product using maven and Tycho.

# openHAB

## What is openHAB?

openHAB is a software for integrating different home automation systems and technologies into one single solution that allows over-arching automation rules and that offers uniform user interfaces.

This means that openHAB:

1. Is designed to be absolutely vendor-neutral as well as hardware/protocol-agnostic.
2. Can run on any device that is capable of running a JVM (Linux, Mac, Windows).
3. Lets you integrate an abundance of different home automation technologies into one.
4. Has a powerful rule engine to fulfill all your automation needs.
5. Comes with different web-based UIs as well as native UIs for iOS and Android
6. Is fully open source
7. Is maintained by a passionate and growing community
8. Is easily extensible to integrate with new systems and devices
9. Provides APIs for being integrated in other systems

## Architectural Principles

openHAB does not try to replace existing solutions, but rather wants to enhance them - it can thus be considered as a system of systems. It therefore assumes that the sub-systems are setup and configured independently of openHAB as this is often a very specific and complex matter (including “pairing” processes, direct device links etc.). Instead, openHAB focuses on the “daily use” side of things and abstracts from the devices themselves.

A core concept for openHAB is the notion of an **“item”**. An item is a data-centric functional atomic building block - you can think of it as an **“capability”**. openHAB does not care whether an item (e.g. a temperature value) is related to a physical device or some “virtual” source like a web service or an calculation result. All features offered by openHAB are using this “item” abstraction, which means that you will not find any reference to device specific things (like IP addresses, IDs etc.) in automation rules, UI definitions and so on. This makes it perfectly easy to replace one technology by another without doing any changes to rules and Uis.

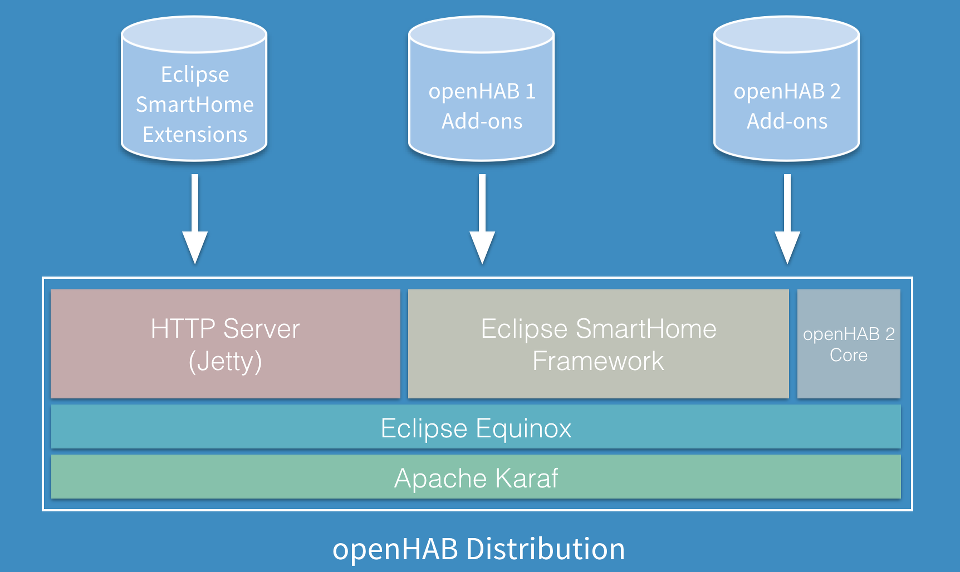
A very important aspect of openHAB’s architecture is its **modular design**. It is very easy to add new features (like the integration with yet another system through a “binding”) and you can add and remove such features at runtime. This modular approach has been a huge enabler for the active community around openHAB with many engaged contributors.

## openHAB structure

openHAB 2 is an open-source solution based on the **Eclipse SmartHome** framework. It is fully written in **Java** and uses **Apache Karaf** together with **Eclipse Equinox** as an **OSGi runtime** and bundles this with **Jetty** as an **HTTP server**.

openHAB is highly modular software, which means that **the base installation (the “runtime”) can be extended through different kinds of “add-ons”**, either to communicate with new home automation solution, or to offer a new kind of user interface.

The structure of openHAB can be summarized this way:



**Add-ons** can come from three different origins:

* Add-ons coming from the Eclipse SmartHome project, as Eclipse SmartHome is the base of openHAB 2,
* Add-ons coming from the openHAB version 1 package, as openHAB 2 includes a compatibility layer,
* Add-ons made for openHAB 2.

**Apache Karaf** → Karaf provides dual polymorphic container and application bootstrapping paradigms to the Enterprise. Focus on your business code and application, Karaf deals with the rest.

Karaf provides projects, answering the enterprise needs:

* Run your applications in Karaf Container
* Easily develop and bootstrap your applications with Karaf Boot
* Manage bunch of clustered instances with Karaf Cellar
* Manage your artifacts repositories with Karaf Cave
* Monitoring, alerting, and BAM with Karaf Decanter

**Eclipse Equinox** → From a code point of view, Equinox is an implementation of the OSGi core framework specification, a set of bundles that implement various optional OSGi services and other infrastructure for running OSGi-based systems. The Equinox OSGi core framework implementation is used as the reference implementation and as such it implements all the required features of the latest OSGi core framework specification.

# openHAB concepts

Eclipse SmartHome strictly differentiates between the physical view and the functional view on the system. While the physical view is required for setup, configuration, troubleshooting, etc., the functional side covers the information that matter to applications, such as user interfaces and automation logic.

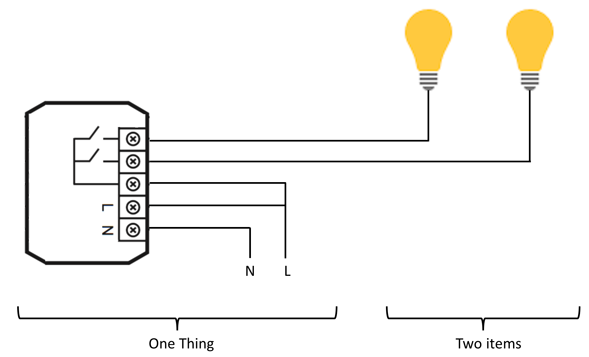
## Things, Channels, Items and Links

**Things** are the entities that can be physically added to a system and which can potentially provide many functionalities at once. It is important to note that things do not have to be devices, but they can also represent a web service or any other manageable source of information and functionality. Things provide their functionality through a set of **Channels**. Channels are “passive” and can be regarded as a declaration of a Thing, what it can offer. It is up to the individual setup, which of the Channels are actively used through Items (see below).

**Items** represent (fine-grained) functionality that is used by applications - as user interfaces or automation logic. Items have a state and they can receive commands.

The glue between Things and Items are **Links**. Links are associations between exactly one Thing Channel and one Item. If a Channel is linked to an Item, it is “enabled”, which means that the functionality that the Item represents is handled through the given Channel. Channels can be linked to multiple Items and Items can be linked to multiple Channels.

To illustrate these concepts, take a two-channel actuator that controls two lights:



The actuator is the *Thing*. This might be installed in the electrical cabinet, it has a physical address and needs to be setup and configured in order to be used. The user is instead interested in the two lights, which are located at different locations in his home. These lights are the desired functionality, thus the *Items* and they are linked to the *Channels* of the actuator. A *Link* can be regarded like a physical wire in this example.