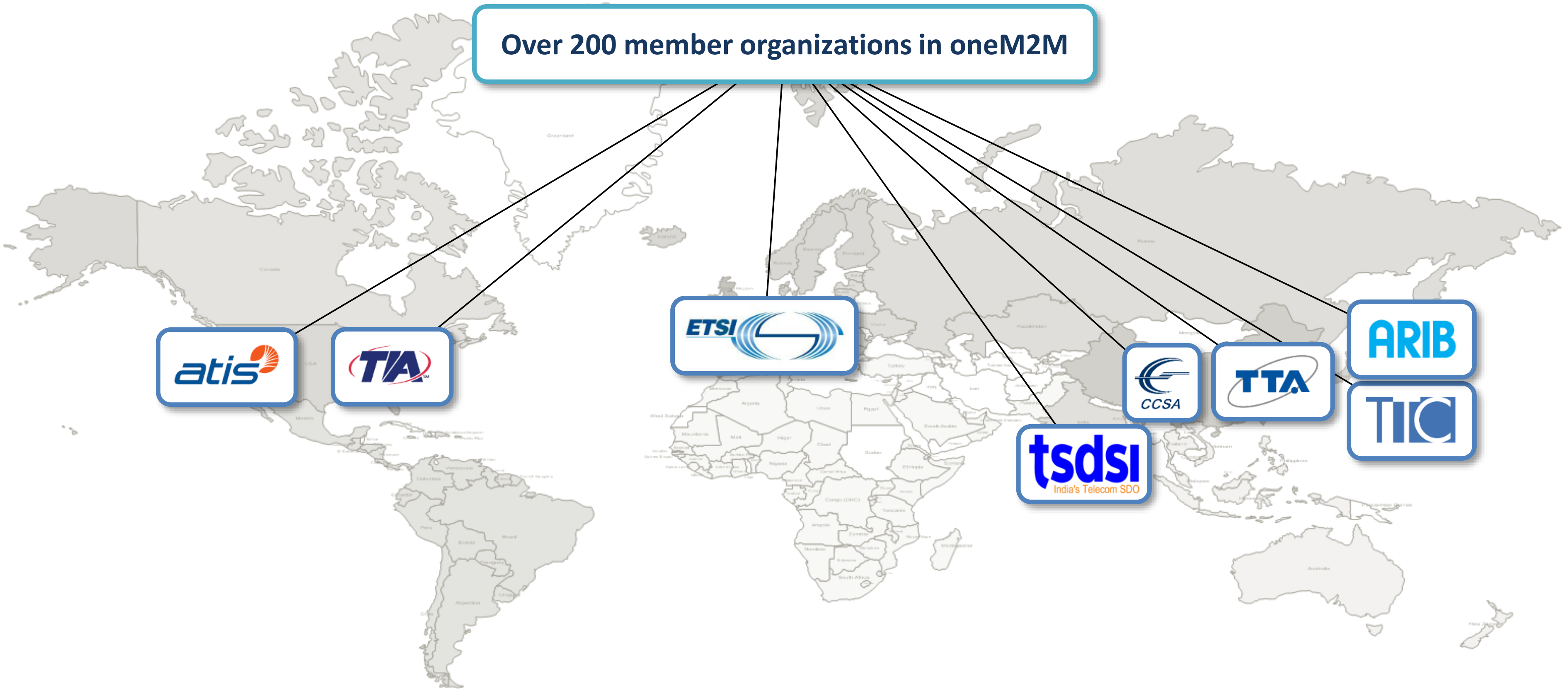
oneM2M Work on IoT Semantic and Data Model Interoperability

Group Name: IRTF
Source: Tim Carey, oneM2M MAS Vice-chair, timothy.carey@nokia.com
Meeting Date: July, 2017

oneM2M Partnership Project

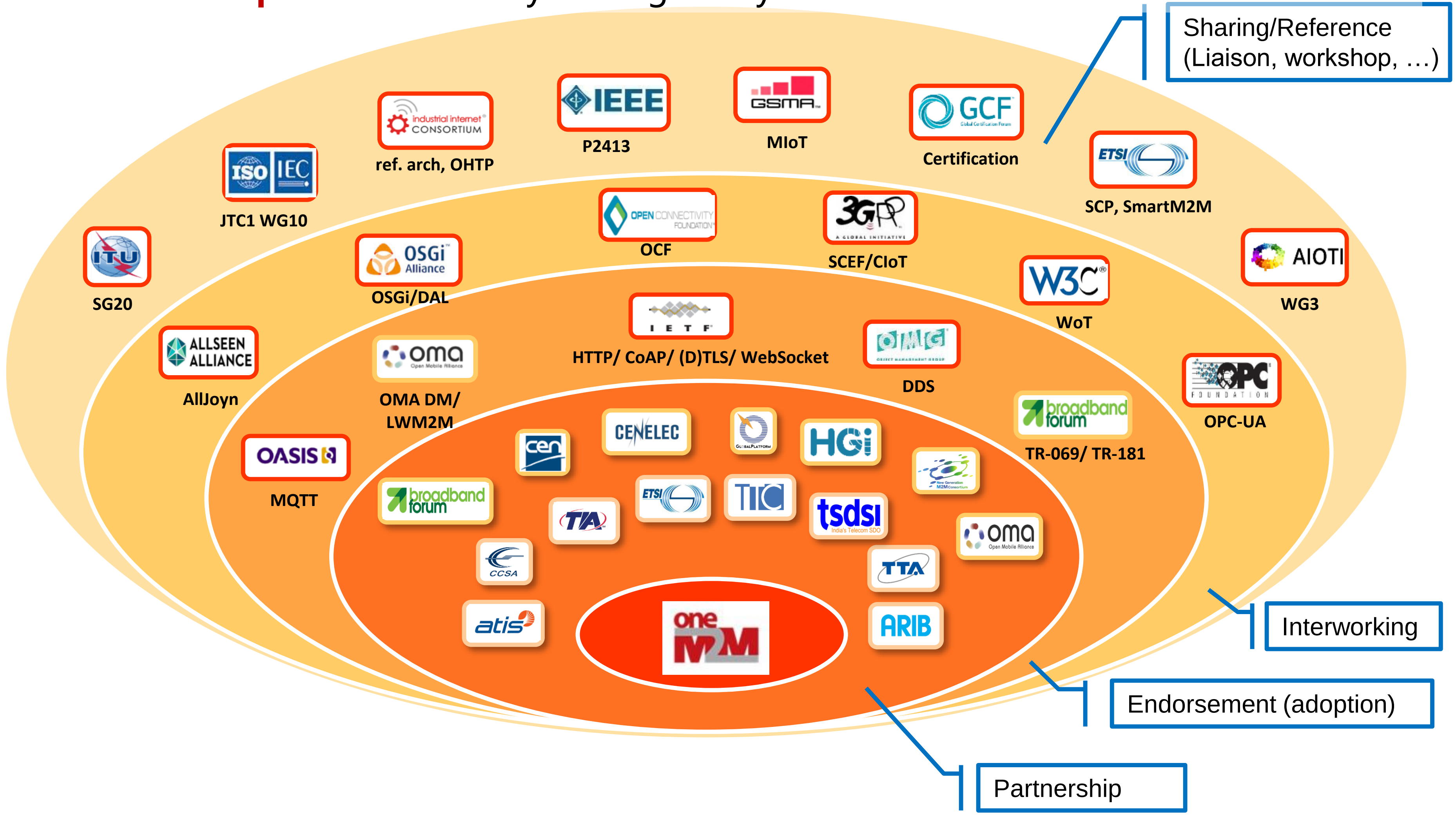


www.oneM2M.org

All document are publically available

Ongoing Collaborations

- Collaboration is important to reach common understanding, avoid overlap and build **interoperable** IoT ecosystems globally.



Strong implementation base

oneM2M

Industry-driven Open source implementations

LAAS-CNRS
Fraunhofer FOKUS
KETI
CISCO
OM2M Connecting things
open mtc
OCEAN
OPEN DAYLIGHT
IotDM

***Examples* of Commercial implementations /demos**

LG
सेंटर फॉर डेवलपमेंट ऑफ टेलीमैटिक्स
भारत सरकार का दूरसंचार प्रौद्योगिकी केन्द्र
सी-डॉट C-DOT Centre for Development of Telematics
Telecom Technology Centre of Govt. Of India
PILOT THINGS
SIERRA WIRELESS™
InterDigital®
QUALCOMM®
kt
HUAWEI
SK telecom
sensinov Global IoT Platform
MODACOM
LG U+
Hewlett Packard Enterprise

5 interop. events so far

M2M Common Service Layer in a Nutshell

A software "framework"

Located between the M2M applications and communication HW/SW that provide connectivity

Provides functions that M2M applications across different industry segments commonly need (eg. data transport, security/encryption, remote software update...)

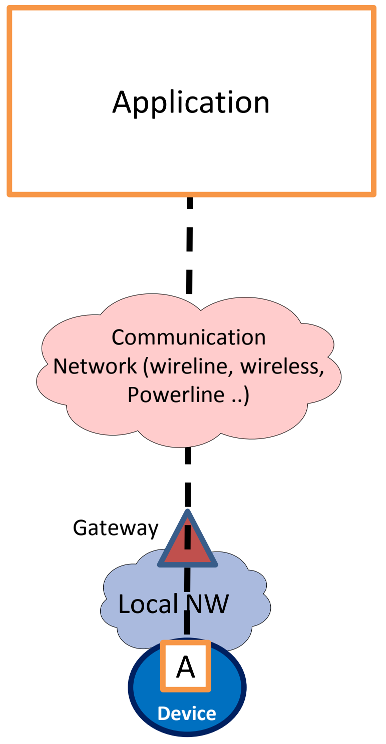
Provides these functions for the Internet of Things sitting both on the field devices/sensors and in servers

And it is a standard – not controlled by a single private company

oneM2M Architecture Approach

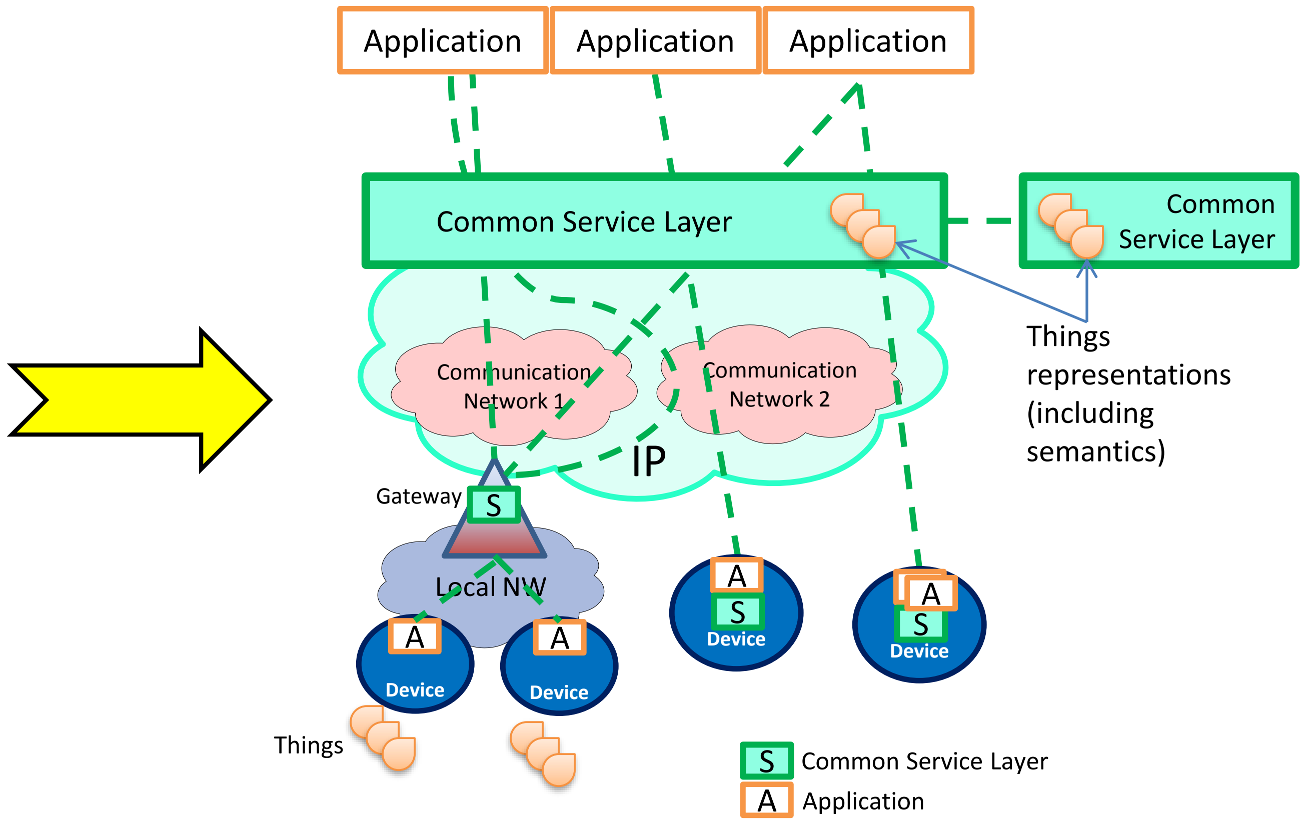
Pipe (vertical):

1 Application, 1 NW,
1 (or few) type of Device
Point to point communications



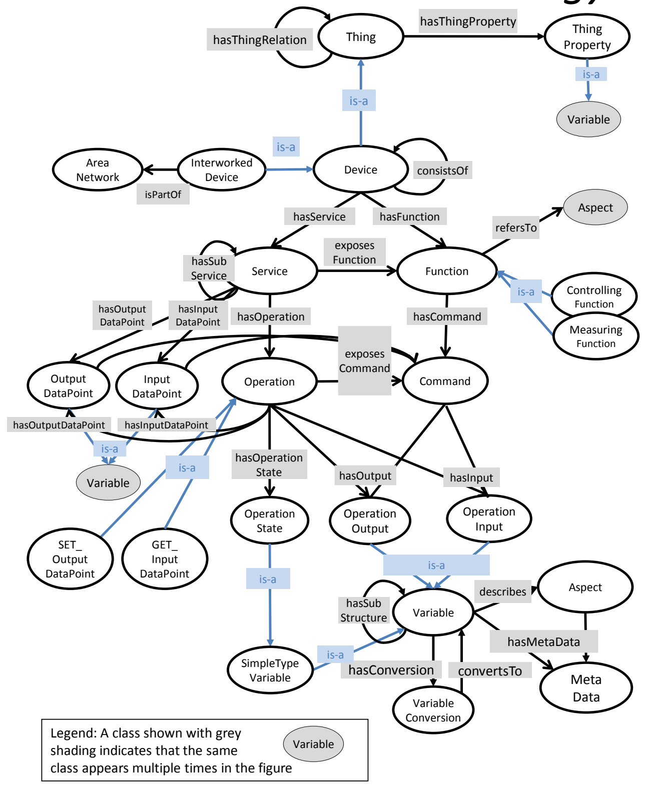
Horizontal (based on common Layer)

Applications share common service and network infrastructure
Multipoint communications



Work on Semantics - Ontologies

The oneM2M Base Ontology

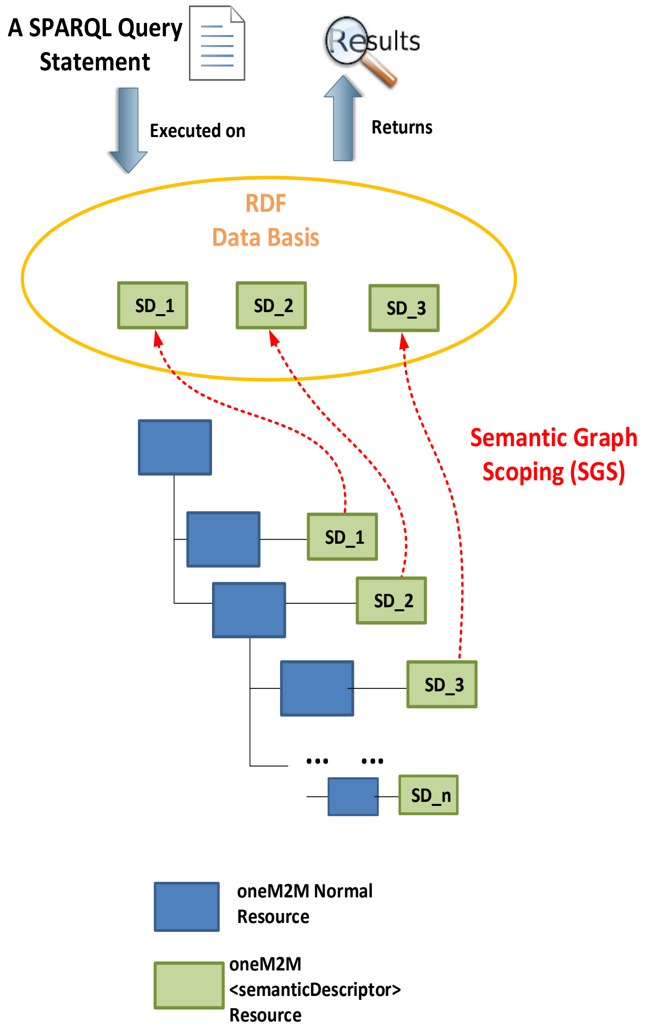


- oneM2M allows to ***annotate*** application specific resources (M2M data) with semantic description.
  - Uses a specialized resource type *<semanticDescriptor>*
  - Can contain proprietary semantics

  or
  - Semantics according to a published ontology
- The oneM2M ***base ontology*** is a top-level ontology that allows to create sub-classes (or equivalence classes) for application-level ontologies
  - Example: Smart Appliances Reference Ontology (SAREF)
- Ontologies can be used in oneM2M to describe the application specific data model of an external system for the purpose of interworking.
  - oneM2M ***Generic Interworking*** uses such an ontology to enable interworking of oneM2M entities with devices of the external system

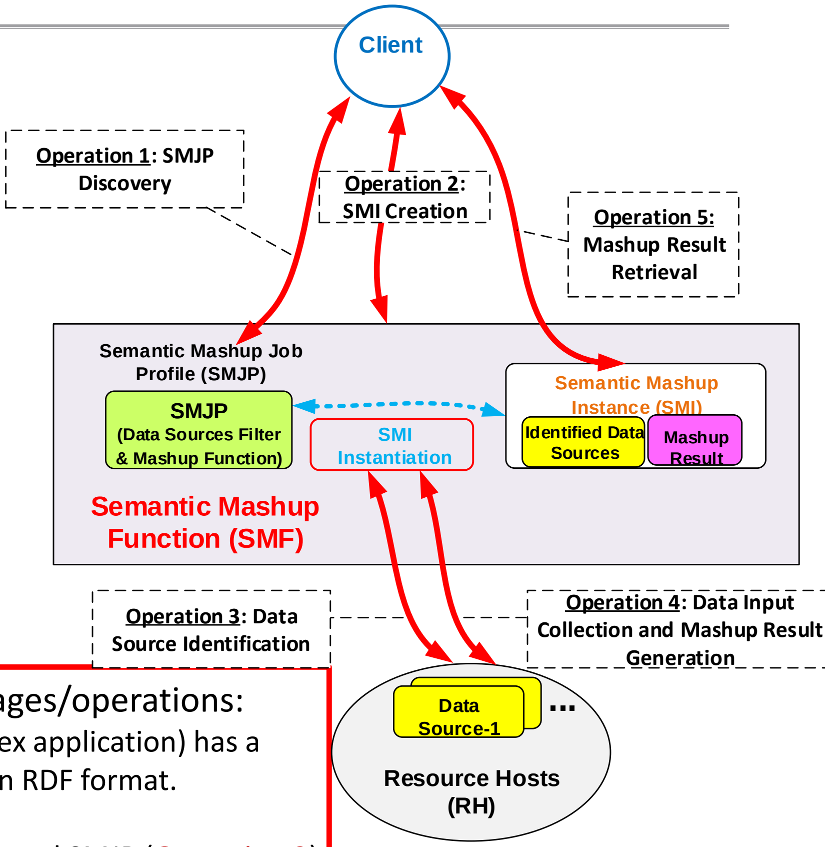
Work on Semantics Query

- oneM2M includes a semantic query feature that includes both discovery and query capabilities
  - Semantic resource discovery is used to discover resources: Give me the resources that represent the temperature sensors located in Room 1.
  - Semantic query is used to extract “useful knowledge” (to answer the query) over a set of “RDF data basis”. What is the manufacture name and production year of the temperature sensors located in Room 1?
- To successfully execute a semantic query requires appropriate semantic graph scoping and extra information represented in RDF triples
  - Semantic Graph Scoping: How to collect RDF triples from semantic descriptors (distributed in the resource tree) to construct a RDF data basis for a given semantic query.
  - Representing Extra Information in RDF Triples: This is for how to query information that was originally not stored as RDF triples, such as data stored in *<contentInstance>* resource (or other oneM2M attributes such as *expirationTime*, etc.).



Work on Semantics Mashup

- oneM2M supports semantic mashup, which fully leverages semantic-related technologies and is defined as a process to discover and collect data from more than one source as inputs, conduct business logic-related mashup function over the collected data, and eventually generate meaningful mashup results.
- Example: Users/clients are interested in a metric called "weather comfortability index", which cannot be directly provided by physical sensors, and in fact can be calculated based on the original sensory data collected from multiple types of sensors (e.g. temperature and humility sensors).
- A complete semantic mashup process consists of multiple stages/operations:
  - ➢ Each specific mashup application (e.g. weather comfortability index application) has a corresponding Semantic Mashup Job Profile (SMJP) represented in RDF format.
  - ➢ A client first discovers a SMJP based on her needs (Operation 1).
  - ➢ A Semantic Mashup Instance (SMI) is created based on the discovered SMJP (Operation 2).
  - ➢ The data sources are identified for the created SMI (e.g. through semantic resource discovery) based on the guideline as specified in the SMJP (Operation 3).
  - ➢ The data is collected from the identified data sources and mashup result is generated and may be periodically refreshed based on the mashup function as specified in the SMJP (Operation 4). The mashup result can be retrieved by the client (Operation 5).


Client
Operation 1: SMJP Discovery
Operation 2: SMI Creation
Operation 5: Mashup Result Retrieval
Semantic Mashup Job Profile (SMJP)
SMJP (Data Sources Filter & Mashup Function)
SMI Instantiation
Semantic Mashup Instance (SMI)
Identified Data Sources
Mashup Result
Semantic Mashup Function (SMF)
Operation 3: Data Source Identification
Operation 4: Data Input Collection and Mashup Result Generation
Data Source-1
...
Resource Hosts (RH)

Work on Semantics Access Control

- In oneM2M, access to resources needs to follow Access Control Policies (as defined in ACP resources).
- An unique issue in oneM2M for semantic query is that for a given semantic query, ACPs still need to be enforced in the sense that only the RDF triples stored in certain <*semanticDescriptor*> resources (i.e., allowed by ACP) can serve as the RDF data basis for this query.
- In oneM2M, two solutions are identified to solve this ACP-related issue when processing a semantic query:
  - 1. ACPs are still kept in their original form and stored in the resource tree. For each received semantic query, the Hosting CSE directly decides which <*semanticDescriptor*> resources are allowed by the ACP to form the RDF data basis for this query.
    - **Operations at the Hosting CSE:** Receive a semantic query -> semantic graph scoping with ACP (i.e. search <*semanticDescriptor>s* which are allowed by ACP) -> Generate RDF basis -> Execute the semantic query over RDF basis.
  - 2. ACPs stored in the resource tree are cloned and re-represented as RDF triples and directly stored in the Triple Store so that the semantic query with access control can be fully processed in the Triple Store.
    - **Operations at the Hosting CSE:** Clone ACP in RDF triples and store them in the Triple Store -> Receive a semantic query -> Modify the semantic query by adding ACP constraints in the query statement -> Execute the modified semantic query over all RDF triples in the Triple Store.

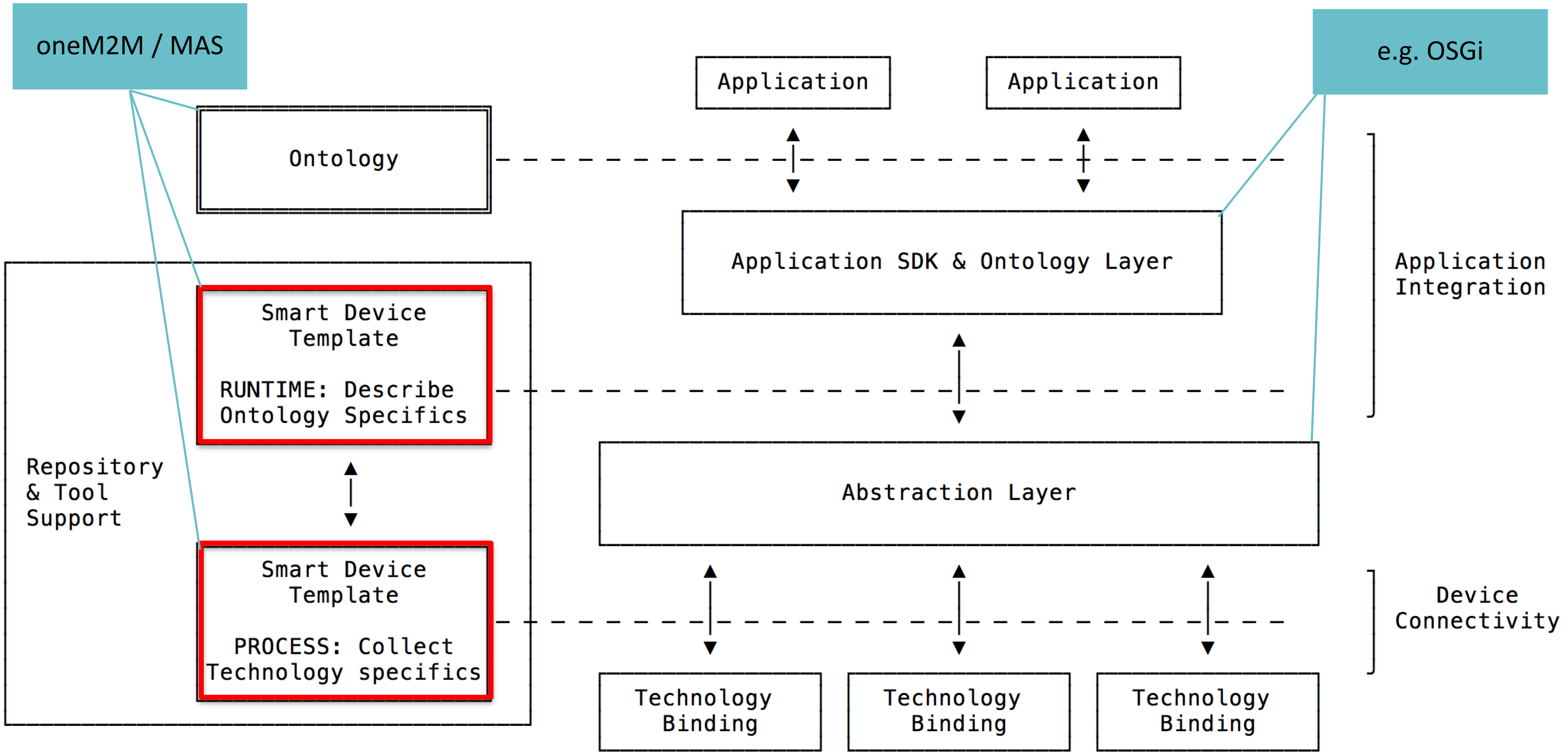
Work on Abstraction using SDT:Goals

Describe devices and device services in a way which is independent of the LAN technology in a format which is convenient and reliable for integration.

1. **Keep it simple**, especially for manufacturers to contribute
2. **Modularity** for functions and device types
3. Make it easy for developers to **create** unified APIs
4. Be **independent** of underlying home-area network technologies

Make it available under an **open license**.

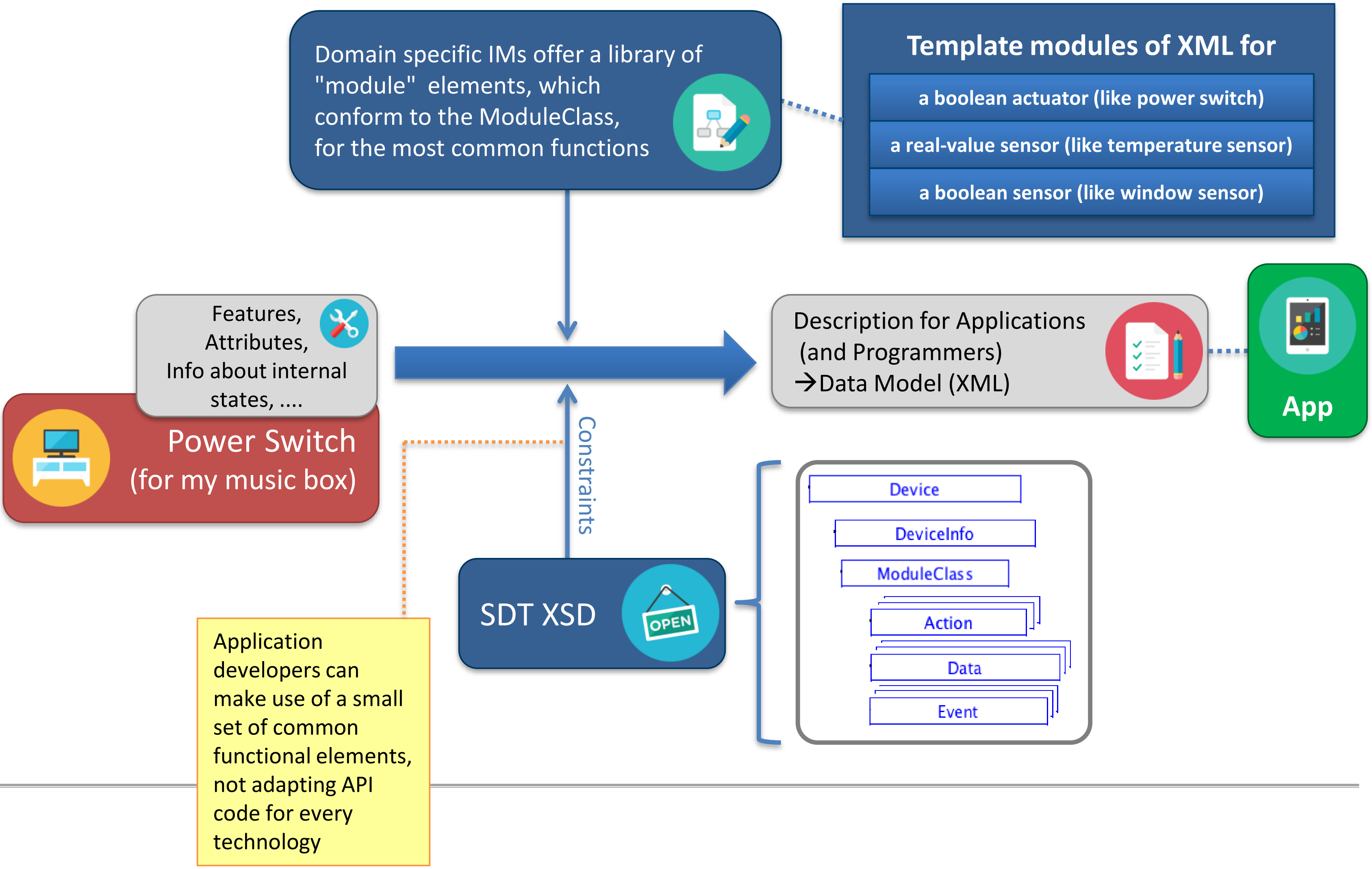
How Things Come Together



SDT: Basic Components

Domain	Unique name, or "wrapper" which acts like a namespace, set by the organization creating the SDT, allowing reference to a package of definitions for the contained ModuleClasses and device definitions. Can be referenced when extending ModuleClasses. It has two possible uses: to select the scope of a technology domain, or to set the scope of a use case domain (like Home, SmartGrid, etc)
Device & Sub-Device	Physical, addressable, identifiable appliance/sensor/actuator, that has one or more functionalities.
ModuleClass	Specification of a single service with one or more service methods, the involved abstracted data model and related events. The expectation is that each separate service which may be used in many kinds of Devices (like PowerON/OFF, Open/Close, ...) will be described by a ModuleClass which can be re-used in many Device definitions.
DataPoint	A DataPoint element represents an aspect of a device which can be read/written to, and forms part of a device's data model. Manipulating DataPoints is the most common way of controlling devices. Each DataPoint has an associated type which facilitates data integrity.
Action	Action elements are an efficient way of describing arbitrary sequences of operations/methods; these are very common in automation. Actions preserve transaction integrity by putting together all the parameters ("args", see next section) with the method which checks and executes them, in one step.
Property	Property elements are used to append to Devices and their ModuleClass elements with arbitrary additional information.

Re-usable XML Modules


Domain specific IMs offer a library of "module" elements, which conform to the ModuleClass, for the most common functions
Template modules of XML for
a boolean actuator (like power switch)
a real-value sensor (like temperature sensor)
a boolean sensor (like window sensor)
Features, Attributes, Info about internal states, ...
Power Switch (for my music box)
Description for Applications (and Programmers) →Data Model (XML)
App
Constraints
SDT XSD
OPEN
Device
DeviceInfo
ModuleClass
Action
Data
Event

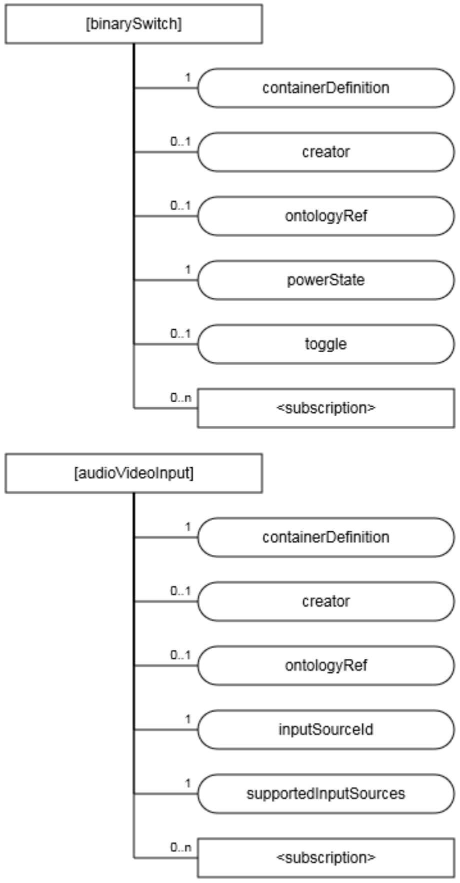
Application developers can make use of a small set of common functional elements, not adapting API code for every technology

Information Modelling in oneM2M

SDT Modeling

- **Property**
  - country
  - deviceID
  - deviceType
  - deviceName
  - deviceModelName
  - …
- **Module**
  - binarySwitch
  - audioVolume
  - televisionChannel
  - audioVideoInput
  - mediaSourceList

Resource Mapping



SDT Mapping

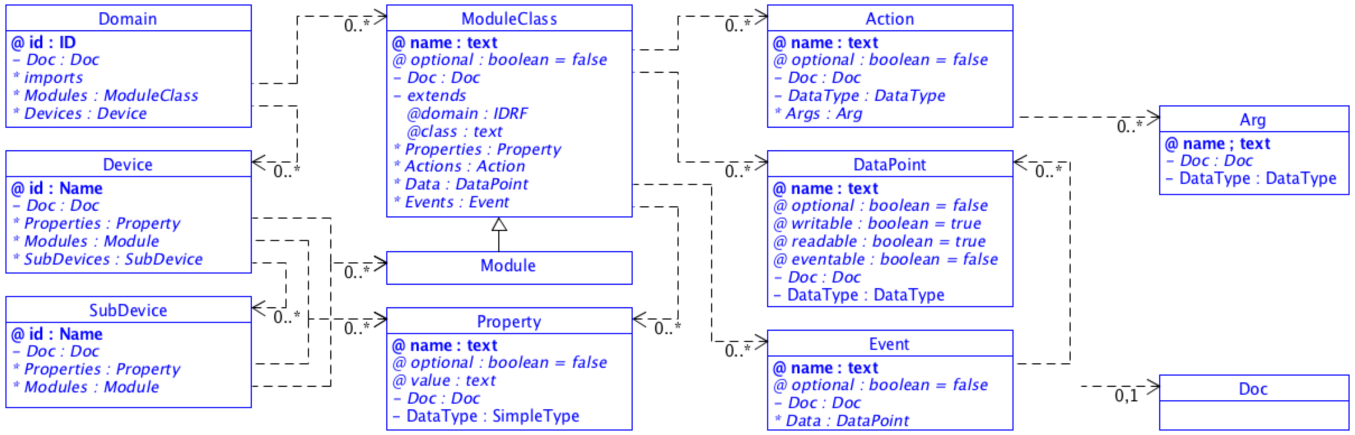
```
<ModuleClass name="binarySwitch">
    <Doc>This ModuleClass provides capabilities to control and monitor the state of
    power.</Doc>
    <Actions>
        <Action name="toggle" optional="true">
            <Doc>Toggle the switch.</Doc>
        </Action>
    </Actions>
    <Data>
        <DataPoint name="powerState" readable="true" writable="true" eventable="true"
        optional="false">
            <Doc>The current status of the binarySwitch. "True" indicates turned-on,
            and "False" indicates turned-off.</Doc>
            <DataType>
                <SimpleType type="boolean" />
            </DataType>
        </DataPoint>
    </Data>
</ModuleClass>
```

XSD Mapping

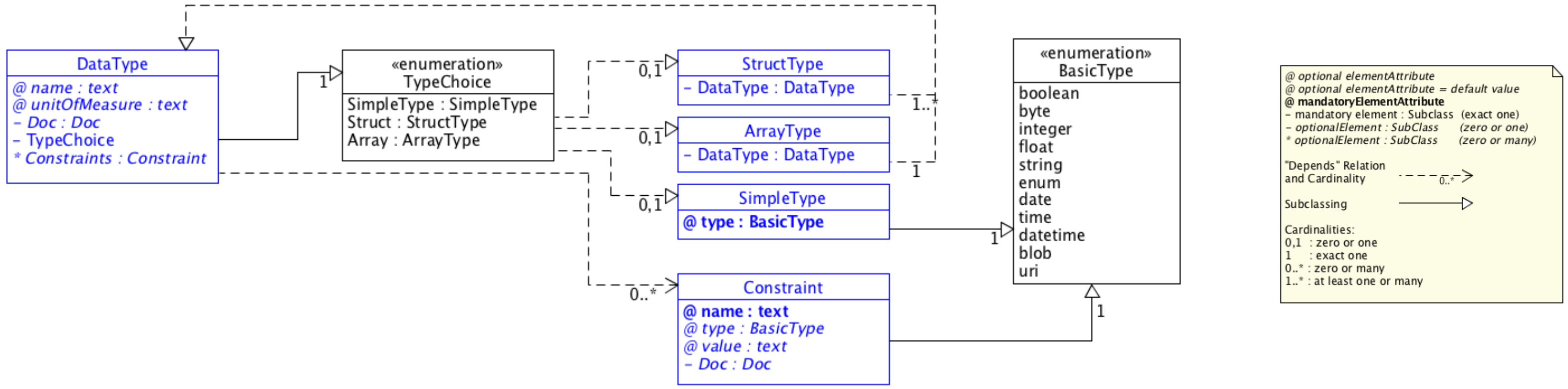
```
<xs:element name="binarySwitch" type="hd:binarySwitch" />
<xs:complexType name="binarySwitch">
    <xs:complexContent>
        <xs:extension base="m2m:flexContainerResource">
            <xs:sequence>
                <!-- Resource Specific Attributes -->
                <xs:element name="powerState" type="xs:boolean" />
                <!-- Child Resources -->
                <xs:choice minOccurs="0" maxOccurs="1">
                    <xs:element name="childResource" type="m2m:childResourceRef" minOccurs='
                    1" maxOccurs="unbounded" />
                    <xs:choice minOccurs="1" maxOccurs="unbounded">
                            <xs:element ref="hd:toggle" />
                        <xs:element ref="m2m:subscription"  />
                    </xs:choice>
                </xs:choice>
            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

SDT as UML

Components

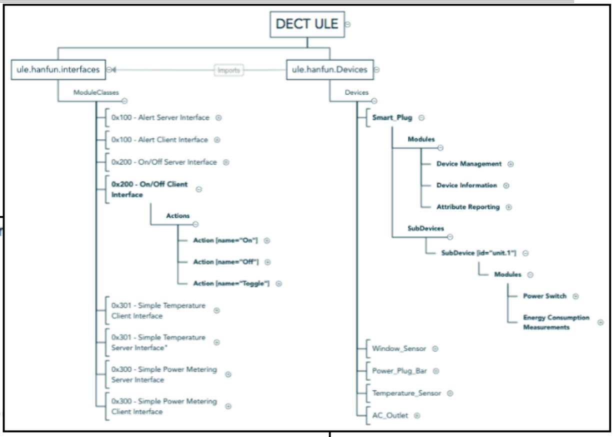


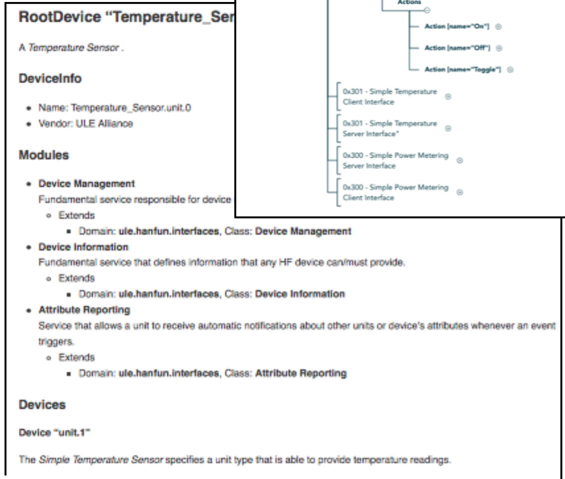
Data Types



SDT Tool

- Generate oneM2M XSD
- Generate documentation
  - Plain Text
  - Markdown
  - OPML (Mindmap)
  - SVG for oneM2M Resources
- Generate code templates
  - Java interfaces and classes
  - Swagger


DECT ULE
ule.hanfun.interfaces
Imports
ule.hanfun.Devices
ModuleClasses
0x100 - Alert Server Interface
0x100 - Alert Client Interface
0x200 - On/Off Server Interface
0x200 - On/Off Client Interface
Actions
Action [name="On"]
Action [name="Off"]
Action [name="Toggle"]
0x301 - Simple Temperature Client Interface
0x301 - Simple Temperature Server Interface"
0x300 - Simple Power Metering Server Interface
0x300 - Simple Power Metering Client Interface
Devices
Smart_Plug
Modules
Device Management
Device Information
Attribute Reporting
SubDevices
SubDevice [id="unit.1"]
Modules
Power Switch
Energy Consumption Measurements
Window_Sensor
Power_Plug_Bar
Temperature_Sensor
AC_Outlet



RootDevice "Temperature_Ser

A *Temperature Sensor* .

DeviceInfo

- Name: Temperature_Sensor.unit.0
- Vendor: ULE Alliance

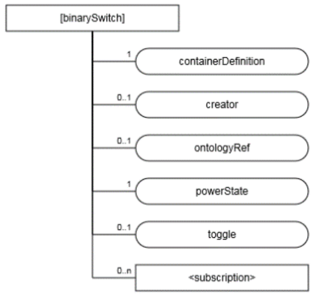
Modules

- **Device Management**
  Fundamental service responsible for device
  - Extends
    - Domain: **ule.hanfun.interfaces**, Class: **Device Management**
- **Device Information**
  Fundamental service that defines information that any HF device can/must provide.
  - Extends
    - Domain: **ule.hanfun.interfaces**, Class: **Device Information**
- **Attribute Reporting**
  Service that allows a unit to receive automatic notifications about other units or device's attributes whenever an event triggers.
  - Extends
    - Domain: **ule.hanfun.interfaces**, Class: **Attribute Reporting**

Devices

Device "unit.1"

The *Simple Temperature Sensor* specifies a unit type that is able to provide temperature readings.



```xml
<xs:element name="binarySwitch" type="hd:binarySwitch" />
<xs:complexType name="binarySwitch">
    <xs:complexContent>
        <!-- Inherit Common Attributes from data type "flexContainerResource" -->
        <xs:extension base="m2m:flexContainerResource">
            <xs:sequence>

                <!-- Resource Specific Attributes -->

                    <xs:element name="powerState" type="xs:boolean" />

                <!-- Child Resources -->

                <xs:choice minOccurs="0" maxOccurs="1">
                    <xs:element name="childResource" type="m2m:childResourceRef" minOccurs="1" maxOccurs="unbounded" />
                    <xs:choice minOccurs="1" maxOccurs="unbounded">

                            <xs:element ref="hd:toggle" />

                        <xs:element ref="m2m:subscription"  />
                    </xs:choice>
                </xs:choice>

            </xs:sequence>
        </xs:extension>
    </xs:complexContent>
</xs:complexType>
```

```java
package ule.hanfun.interfaces;

public interface AlertServerInterface {

    // Actions

    void status(Integer unitType, Integer stateAttribute);

    // DataPoint getters/setters

    Integer _getState();
    void _setEnable(Integer enable);
    Integer _getEnable();
}
```

Work on Information Models

- Soon after the initial launch of release 1.0 of oneM2M, member companies requested to develop information models that can be interworked from various IoT technologies and represented to applications with a consistent fashion.
  - The work of the Proximal IoT Interworking specification (TS-0033) that incorporates various interworking technologies (e.g., LwM2M, OCF, AllJoyn, DDS, OPC-UA, WoT, Modbus)
  - The work of the generic interworking mechanism (TS-0030) is to specify a procedure where devices do not necessarily require a specific representation (e.g., KNX).
  - The first domain requesting development of information models was the home domain – this led to a Study of the Home Domain (TR-0013) and follow-on device model specifications (TS-0023).
- Based on the successful work with the Home Appliance models, member companies are looking toward developing information models in the Industrial (TR-0018), Vehicular (TR-0026) and Smart City (TR-0046) Domains. But we will only do these in collaboration with experts in these domains.

Home Appliances Models

- Based on SDT, oneM2M defines **Information Models for Home Appliances** in TS-0023 and its latest version is v.3.4.0.
- These appliance/device models were contributed by device manufacturers, service providers as well as adaptations from other standards (e.g., ECHONET, OMA Device WebAPIs)
- This version currently includes 42 device models and 70 module classes
- TS-0023 is still evolving and has more device models and functionality for Rel-3
- Television device model example

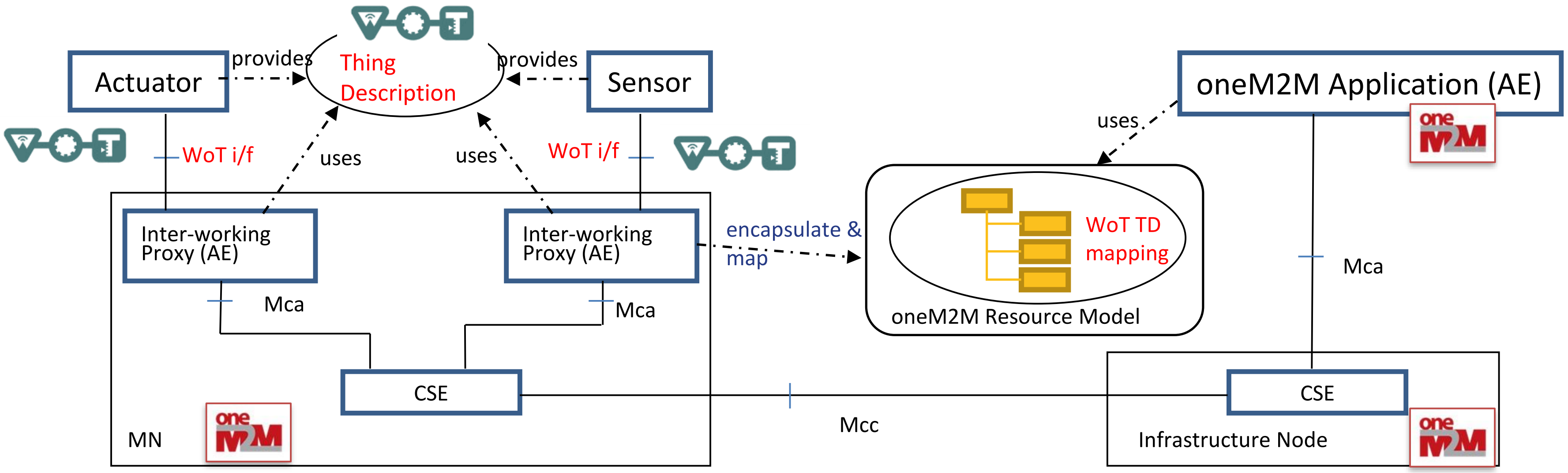
	Properties (Optionality)	**Module Class** (Optionality)	**Actions** (All optional) **& Data Points** (Optionality)
TV	manufacturer(M)	binarySwitch(M)	toggle(), powerState(O)
	modelID(M)	audioVolume(O)	upVolume(), downVolume(), volumePercentage(M), stepValue(O), muteEnbled(M),…
	deviceType(M)	televisionChannel(O)	upChannel(), downChannel(), chNumber(M), availableCh(O), perviousCh(O), chName(O)
	deviceName(O)	playerControl(O)	nextTrack(), previousTrack(), currentPlyerMode(M), modeName(O), modes(M), speed(O)
	country(O)	mediaInput(O)	mediaID, supportedMediaSources, mediaName, status, mediaType
	swVersion(O)	mediaOutput(O)	mediaID, supportedMediaSources, mediaName, status, mediaType

Home Appliances Models: Semantic Interoperability

- Realizing the need for alignment of information/data models as a first step in interoperability, oneM2M is starting a process of aligning its models with other groups (e.g., OCF). Remember that the original models were, in part, based on work of other standards organizations.
- As the first target, oneM2M recently has started the process to align Home Appliance information models with OCF's version 1.0 data models.
  - 25 of 35 devices were only defined in OCF 1.0 while 10 were defined in both
  - 25 new devices are added to oneM2M, and 10 existing oneM2M models are modified to be aligned with OCF 1.0 data models
  - So now 35 devices in OCF 1.0 is all mappable with oneM2M devices
- Next step for OCF data model mapping
  - OCF may need to define new device models that are existing in only oneM2M to support full interoperability in the future

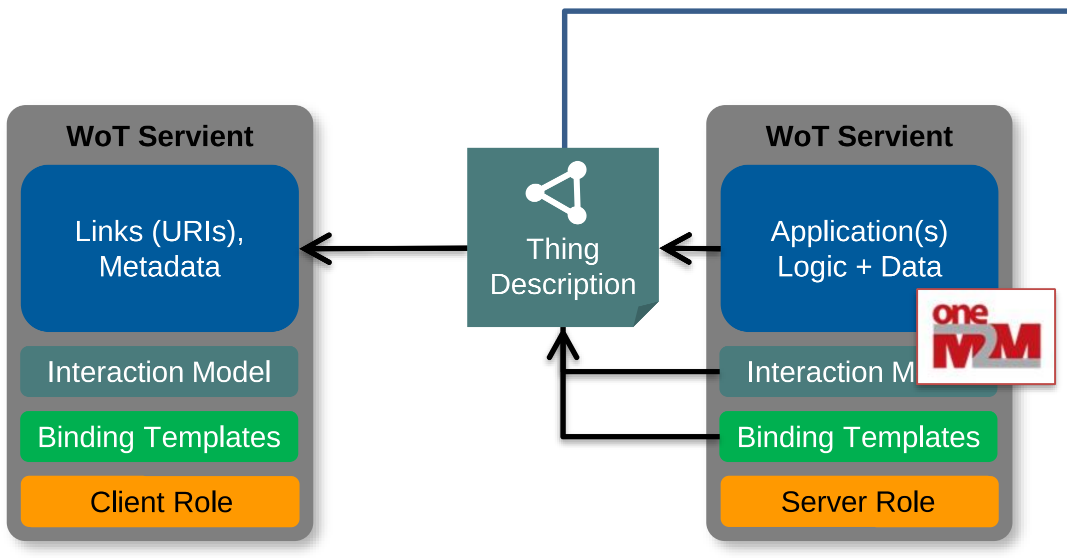
Example: WoT Interworking – WoT Interface

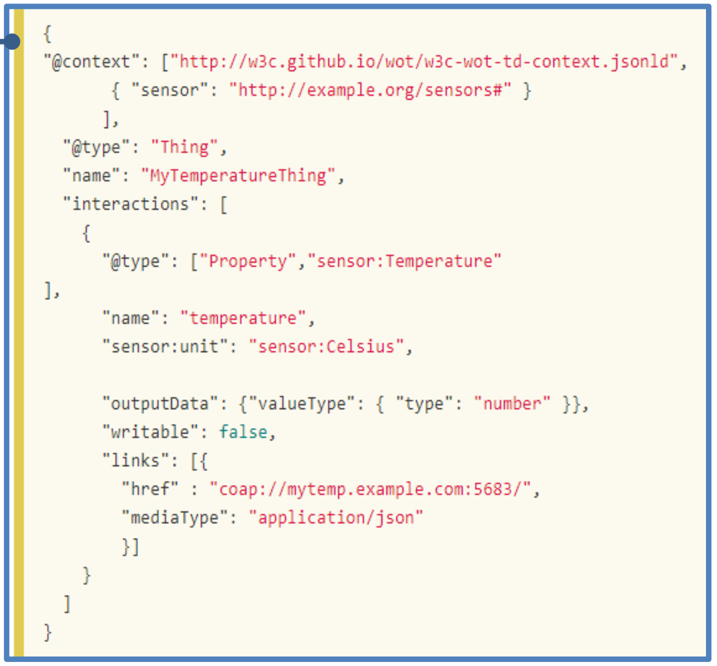
- Exposing the WoT interface (described in TD) to oneM2M systems
  - Benefit: WoT services/data can be consumed by oneM2M applications



Example: WoT Interworking – oneM2M Interface

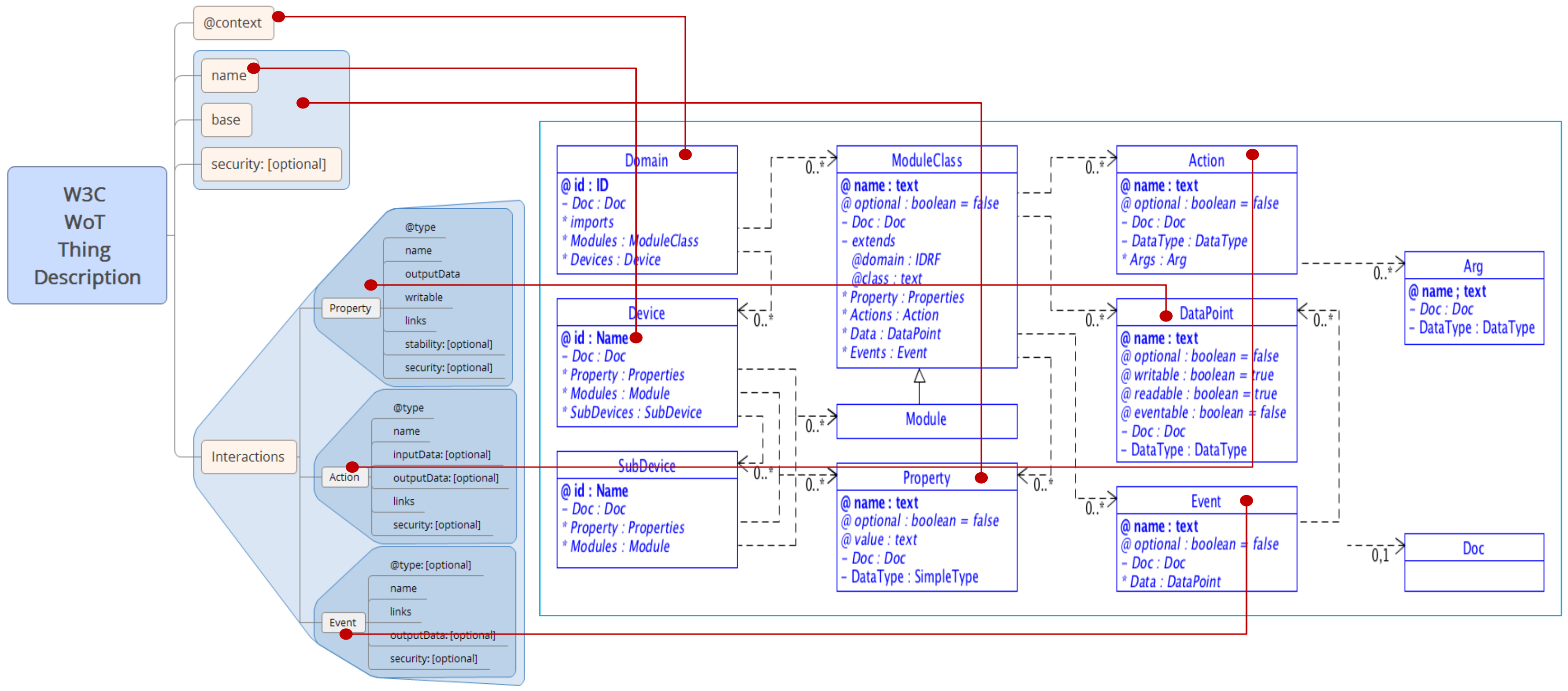
- Exposing oneM2M interfaces to WoT systems
  - Benefit: oneM2M services/data can be consumed by WoT Servients





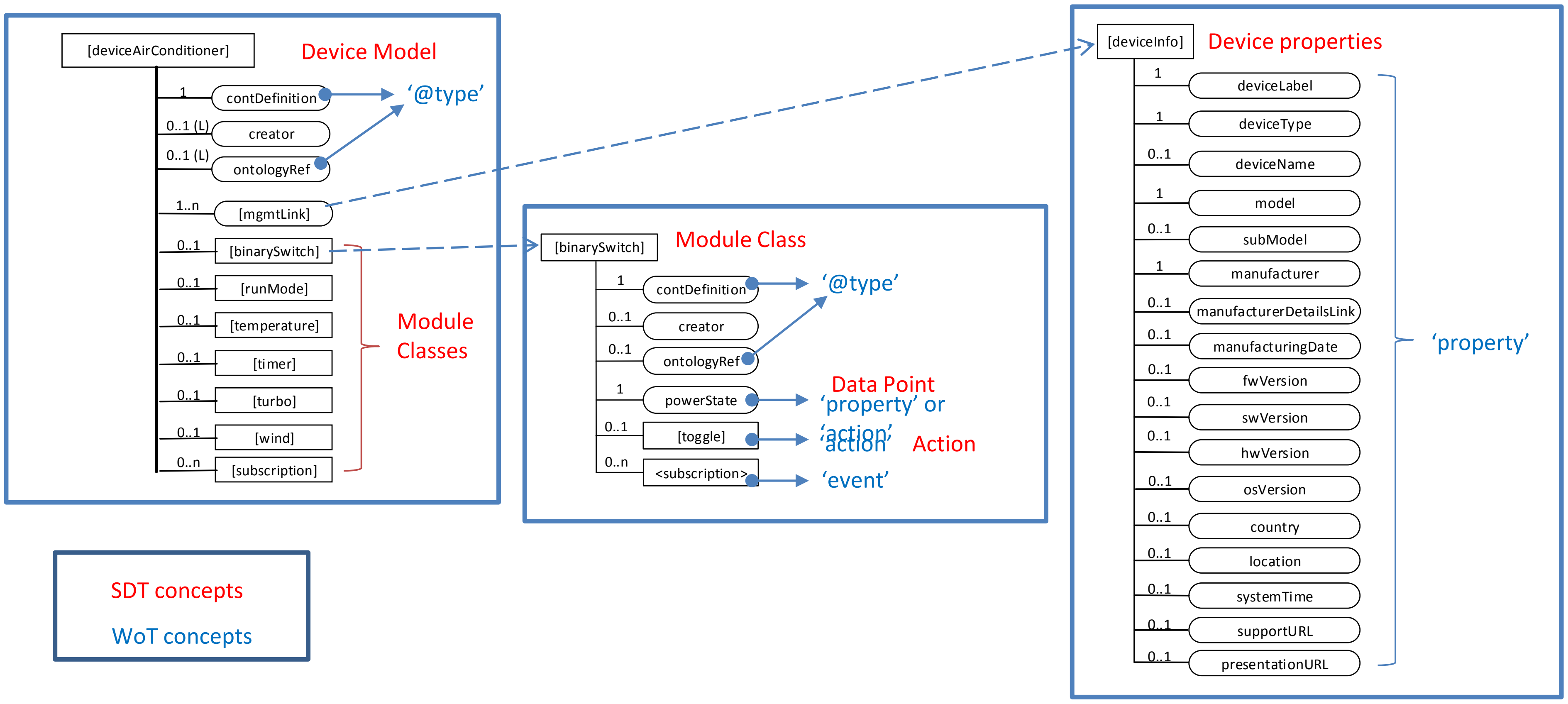
```
{
"@context": ["http://w3c.github.io/wot/w3c-wot-td-context.jsonld",
        { "sensor": "http://example.org/sensors#" }
       ],
  "@type": "Thing",
  "name": "MyTemperatureThing",
  "interactions": [
    {
      "@type": ["Property","sensor:Temperature"
],
      "name": "temperature",
      "sensor:unit": "sensor:Celsius",

      "outputData": {"valueType": { "type": "number" }},
      "writable": false,
      "links": [{
        "href" : "coap://mytemp.example.com:5683/",
        "mediaType": "application/json"
        }]
    }
  ]
}
```

Example: WoT Interworking - SDT Mapping



- Conceptual alignment observed between oneM2M/HGI SDT and W3C WoT information models
- It implies promising semantic interoperability.
- Technical study is progressing on both sides

Example: WoT Interworking – Home Automation with SDT



SDT concepts

WoT concepts