ETSI TS 103 780 V1.1.1 (2022-08)

SmartM2M;
SAREF: oneM2M usage guidelines

Reference

DTS/SmartM2M-103780

Keywords

interoperability, IoT, IoT platforms, oneM2M, SAREF, semantic, smart lift

Contents

Intellectual Property Rights

Foreword

This Technical Specification (TS) has been produced by ETSI Technical Committee Smart Machine-to-Machine communications (SmartM2M).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

1 Scope

The present document has the objective to provide guidelines for the usage of SAREF over oneM2M (also including the SDT interoperability) for vertical industry sectors.

The present document also provides a simple use case for guiding application developers to model physical devices in oneM2M and adding semantic annotations that will enable interoperability of devices that are modelled in oneM2M:

- Description of a physical device that is to be modelled in oneM2M.
- Description of methods that can be used to model the device using oneM2M resources and procedures.
- The semantic annotation of the devices using the oneM2M base ontology.
- The semantic queries that can be used to discover device capabilities and enable interoperability.
- The call flows for implementation of the use case with a focus on the semantic aspects.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long-term validity.

The following referenced documents are necessary for the application of the present document.

Not Applicable.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI TS 118 112: "oneM2M; Base Ontology (oneM2M TS-0012)".

[i.2] ETSI TS 118 130: "oneM2M; Ontology based Interworking (oneM2M TS-0030)".

[i.3] ETSI TS 118 104: "oneM2M; Service Layer Core Protocol (oneM2M TS-0004)".

[i.4] onem2m-jupyter-notebooks.

NOTE: Available at https://github.com/ankraft/onem2m-jupyter-notebooks.

[i.5] ETSI TS 118 123: "oneM2M; Home Appliances Information Model and Mapping (oneM2M TS-0023)".

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

AE	Application Entity
API	Application Programming Interface
CSE	Common Service Entity
HTTP	Hypertext Transfer Protocol
IoT	Internet Of Things
IPE	Interworking Proxy Element
JSON	JavaScript Object Notation
M2M	Machine to Machine
NoDN	Non-oneM2M Device Node
RDF	Resource Description Framework
SAREF	Smart Applications REFerence ontology
SDT	Smart Device Template
SPARQL	SPARQL Protocol and RDF Query Language
URI	Uniform Resource Identifier
X-M2M-CTS	oneM2M HTTP protocol custom header for "Content Status"
X-M2M-RI	oneM2M HTTP protocol custom header for "Request Identifier"
X-M2M-RSC	oneM2M HTTP protocol custom header for "Response Status Code"
X-M2M-RVI	oneM2M HTTP protocol custom header for "Release Version Indicator"
XML	eXtensible Markup Language

4 Motivation

The assumption of many existing oneM2M applications is that they interact with other oneM2M applications through known resource structures. They either create the resources themselves or are configured to use specific resources. Information is typically stored in containers, sometimes as base64-encoded content instances, with the implicit assumption that applications have a-priori knowledge of the syntax and semantics of this information.

Depending on a-priori knowledge of the structures and data works well for small-scale vertical deployments of IoT devices. When the deployment evolves to include new devices, the existing applications change to reflect the new additions. However, in larger systems of IoT devices where the IoT devices may be a part of a legacy deployment or more than a single vertical solution, changes to all existing applications may become impractical. To enable growth and diversity of IoT devices in large heterogenous settings, applications need to be able to **discover** the structure and meaning of data from devices and how to use the services of the devices. In oneM2M Release 1 support for discovery of resources based on specific attribute values and the use of labels was defined. The agreement of a fixed set of labels (using a-priori knowledge) can be a viable solution for small deployments.

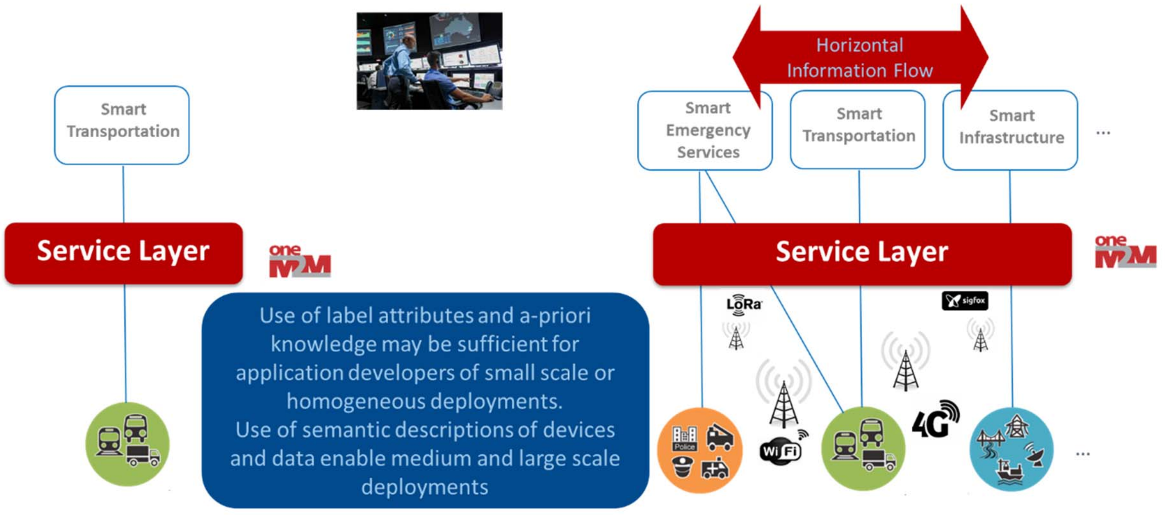

Horizontal Information Flow
Smart Transportation
Smart Emergency Services
Smart Transportation
Smart Infrastructure
...
Service Layer
Service Layer
Use of label attributes and a-priori knowledge may be sufficient for application developers of small scale or homogeneous deployments.
Use of semantic descriptions of devices and data enable medium and large scale deployments
LoRa
sigfox
4G
Wi Fi
Police
...

Figure 1: Semantic understanding of device and data in IoT deployments

For medium or large deployments of heterogeneous IoT devices a more expressive approach for describing and discovering IoT devices is provided by oneM2M. Each type of device in a heterogeneous deployment can model services and data in the oneM2M Service Layer using different structures and syntax of data. For example, temperature sensors may report measurements using different units such as Celsius, Fahrenheit and Kelvin. Additionally, those IoT devices may measure different aspects, such as indoor temperature, outdoor temperature, refrigerator temperature, etc. and the representation of the measurement may differ as well.

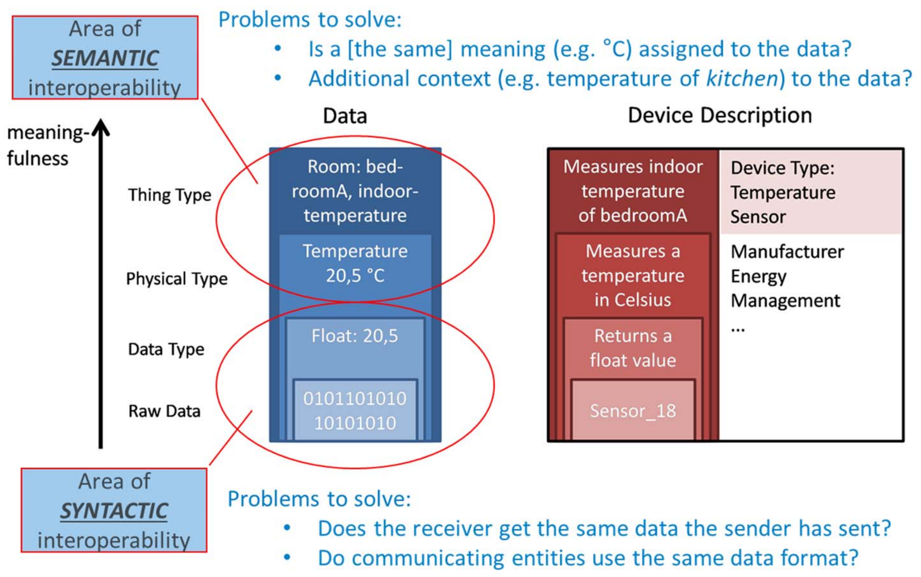

Area of SEMANTIC interoperability
Problems to solve:
• Is a [the same] meaning (e.g. °C) assigned to the data?
• Additional context (e.g. temperature of kitchen) to the data?
Data
Device Description
meaning-fulness
Thing Type
Physical Type
Data Type
Raw Data
Room: bed-roomA, indoor-temperature
Temperature 20,5 °C
Float: 20,5
0101101010 10101010
Measures indoor temperature of bedroomA
Measures a temperature in Celsius
Returns a float value
Sensor_18
Device Type: Temperature Sensor
Manufacturer Energy Management
...
Area of SYNTACTIC interoperability
Problems to solve:
• Does the receiver get the same data the sender has sent?
• Do communicating entities use the same data format?

Figure 2: Meaningfulness of data from IoT devices

With semantic annotations in oneM2M, all the different aspects of IoT devices can be described using RDF triples, which is a standard semantic format. The vocabulary used for a semantic description can be defined according to an ontology such as SAREF. With semantic discovery, applications can describe precisely what information they need or can deal with. This is powered by specifying a semantic filter using the SPARQL query language. The SPARQL filter is matched against the respective semantic annotations of each resource within the discovery scope. This feature in oneM2M helps applications to properly handle the data from the IoT devices.

Besides differences in the data from an IoT device in oneM2M the information model of devices can be modelled in a variety of ways. As with most IoT platforms, oneM2M supports custom information models that are defined for a specific use case and work well in small scale or single vertical scenarios. Another method that oneM2M defines to model devices is based on the semantic description of a device that is mapped to a resource structure (see ETSI TS 118 130 [i.2]). A third approach to modelling devices in oneM2M is the use of Smart Device Templates (see ETSI TS 118 123 [i.5]).

With all these options available to model a device the ability to have a-priori knowledge of a device model becomes less likely as IoT deployments scale beyond small vertical use cases. The oneM2M Base Ontology addresses this and enables developers of these different models to make them interoperable if the appropriate semantic annotations are made and semantic filtering is used to discover the appropriate API for a model. The focus of the remainder of this developer guide is to demonstrate this process.

5 System Description

5.1 Use case

The example scenario describes a clothes-washing machine and an application to monitor and control the IoT enabled product. This clause will show three different oneM2M resource tree models of the clothes washing machine and the call flows to create those models. The logic and call flows necessary for a client application to control and monitor the status of the clothes-washing machine is also described. In the next clause the washing machine capabilities are described using the SAREF ontology so that the client application can discover the washing machines. Additionally, the oneM2M Base Ontology describes how to use the device and commands that these clothes washing machines offer so that the client application can control any of them without regard to which resource tree model represents them.

This simplified clothes-washing machine has enough features to demonstrate the difference between the different modelling approaches supported in oneM2M. The concepts shown here can be applied to a full featured clothes-washing machine or any other IoT enabled device for that matter. The features and capabilities that are modelled are:

- The washing machine has been produced by manufacturer **XYZ.**
- XYZ describes this type of washing machine as "`Very cool Washing Machine`".
- The model of the type of washing machine is **XYZ_Cool.**
- The state of the washing machine can take the values "WASHING" or "STOPPED" or "ERROR".
- The washing machine supports three commands: **ON, OFF, Toggle.**
- The washing machine is in **My_Bathroom.**

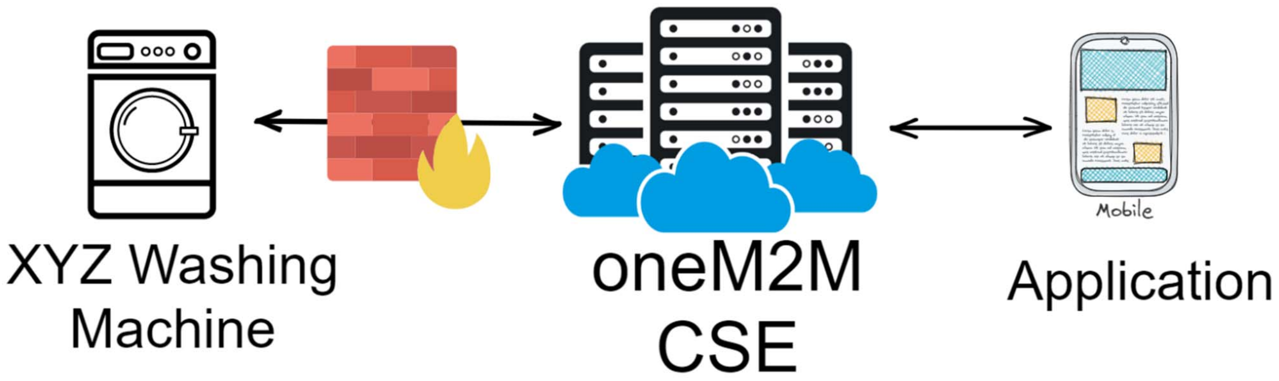
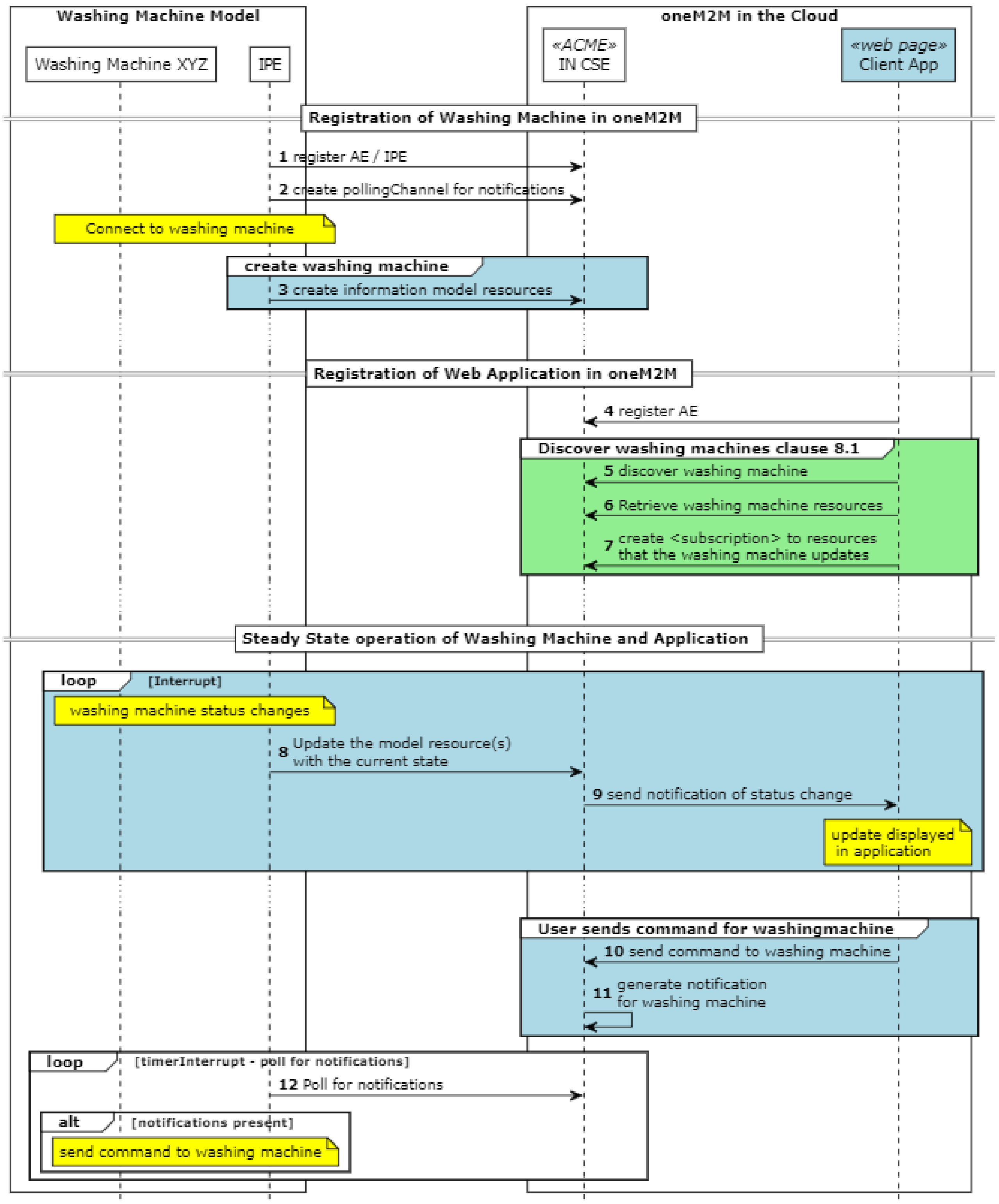


Figure 3: Functional Architecture for Smart Clothes Washing Machine

The clothes-washing machine is modelled as a non-oneM2M device (NoDN) for all three models. However, everything in this guide applies equally if these were modelled as native oneM2M devices. There is no difference in the model or the call flows for everything to the right of the Interworking Proxy Element (IPE) shown in the figure below. Figure 4 shows a generic set of oneM2M call flows for the clothes washing machine (and the IPE) and the client application communicating through the oneM2M CSE. The level of detail provided here applies to all the different modelling approaches for the clothes washing machine. Differences in the call flows that are dependent on the model used, shown in blue shading, are further detailed where the specific models are described.



Washing Machine Model
Washing Machine XYZ
IPE
oneM2M in the Cloud
«ACME» IN CSE
«web page» Client App
Registration of Washing Machine in oneM2M
1 register AE / IPE
2 create pollingChannel for notifications
Connect to washing machine
create washing machine
3 create information model resources
Registration of Web Application in oneM2M
4 register AE
Discover washing machines clause 8.1
5 discover washing machine
6 Retrieve washing machine resources
7 create <subscription> to resources that the washing machine updates
Steady State operation of Washing Machine and Application
loop [Interrupt]
washing machine status changes
8 Update the model resource(s) with the current state
9 send notification of status change
update displayed in application
User sends command for washingmachine
10 send command to washing machine
11 generate notification for washing machine
loop [timerInterrupt - poll for notifications]
12 Poll for notifications
alt [notifications present]
send command to washing machine

Figure 4: Generic oneM2M Call Flows

The messages shown in Figure 4 are further described here:

1. Register the AE/IPE: In oneM2M an IPE is a type of AE that is intended to communicate with NoDNs. The IPE is responsible for registering itself and creating the appropriate resources in a oneM2M CSE to model the NoDN as if it were a oneM2M device. The result is that a washing machine that is native oneM2M and a washing machine that is non-oneM2M can be modelled the same way and the client applications cannot tell the difference.

2. Create Polling Channel: A <pollingChannel> resource is used by applications or devices that are not reachable from the CSE that need to receive notification requests. This happens when, for example, the device is in a home with a firewall that prevents direct requests to the device from outside the local network in the home. (It is also appropriate for IoT devices that communicate using cellular networks.)

3. Create Information Model: The IPE creates all the resources needed to provide the status and enable control of the clothes washing machines. These messages (in almost all cases multiple resources are used) will be described with the details relevant to the specific model in later sub-clauses. This includes creating subscriptions to the resources that are used to enable the application to control the clothes washing machines.

4. Register Client application AE: Client applications are also modelled as <AE> resources and register in the oneM2M CSE.

5. Discover Washing Machine: An application designed to control the clothes washing machines produced by manufacturer XYZ will be able to discover them using a-priori knowledge of labels that are used to identify those washing machines. Later it has been shown how using the semantics capabilities of oneM2M and the SAREF ontology the same application can discover and control clothes washing machines from any manufacturer.

6. Retrieve clothes washing machine resources: The client application generally has a user interface to show the status and allow control of the clothes washing machine. The client application will retrieve the specific resources that it needs to provide that capability. The application may have more features than a given washing machine model supports or, similarly the clothes washing machine model may expose more features than the client application needs. This step will use SPARQL queries to dynamically determine what resources are needed by the client application.

7. Subscribe to resources: The client application is made aware of changes in the state of a clothes washing machine by receiving notifications of the changes. The client application first subscribes to the resources that contain information that it needs.

8. Update the model resources: When the state of the clothes washing machine changes, the change in state will be reflected in the oneM2M CSE.

9. Notification of state changes: When resources in the oneM2M CSE are created or updated the CSE will send notifications to applications that are subscribed to the resources. A client application that receives a notification can present this information to users or take some other actions.

10. Send commands to clothes washing machine: The client application exposes to a user features or capabilities of the clothes washing machine. The client application sends the appropriate oneM2M primitives, based on the model, to use those features or capabilities.

11. Generate notification for clothes washing machine: When the client application sends a oneM2M primitive to a resource that controls the clothes washing machine, a notification is generated (assuming notifications were created). In our scenario, since the clothes washing machine and the IPE are behind a firewall and therefore not reachable, the notification for the IPE are stored in the CSE and made available to the IPE via the long polling process.

12. Poll for notifications. Because the IPE cannot receive notifications directly, it shall use the long polling procedure to retrieve its notifications from the CSE. The IPE processes notifications by sending commands to the clothes washing machine using the API of the clothes washing machine.

5.2 Custom Model

Using oneM2M to represent devices allows for unlimited flexibility. A device model can be customized to support the needs of the manufacturer or system architecture. The resource tree structure shown here represents a custom model that has a single container for reading the status of the washing machine and a separate container to set or command the washing machine.

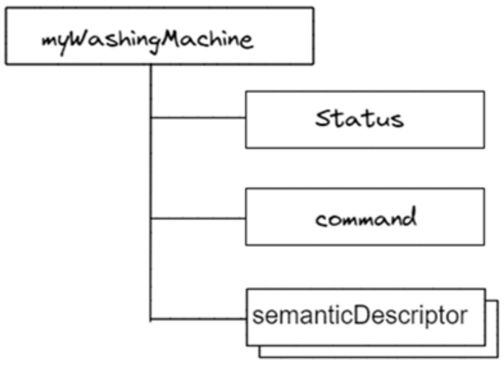


Figure 5: Custom washing machine model

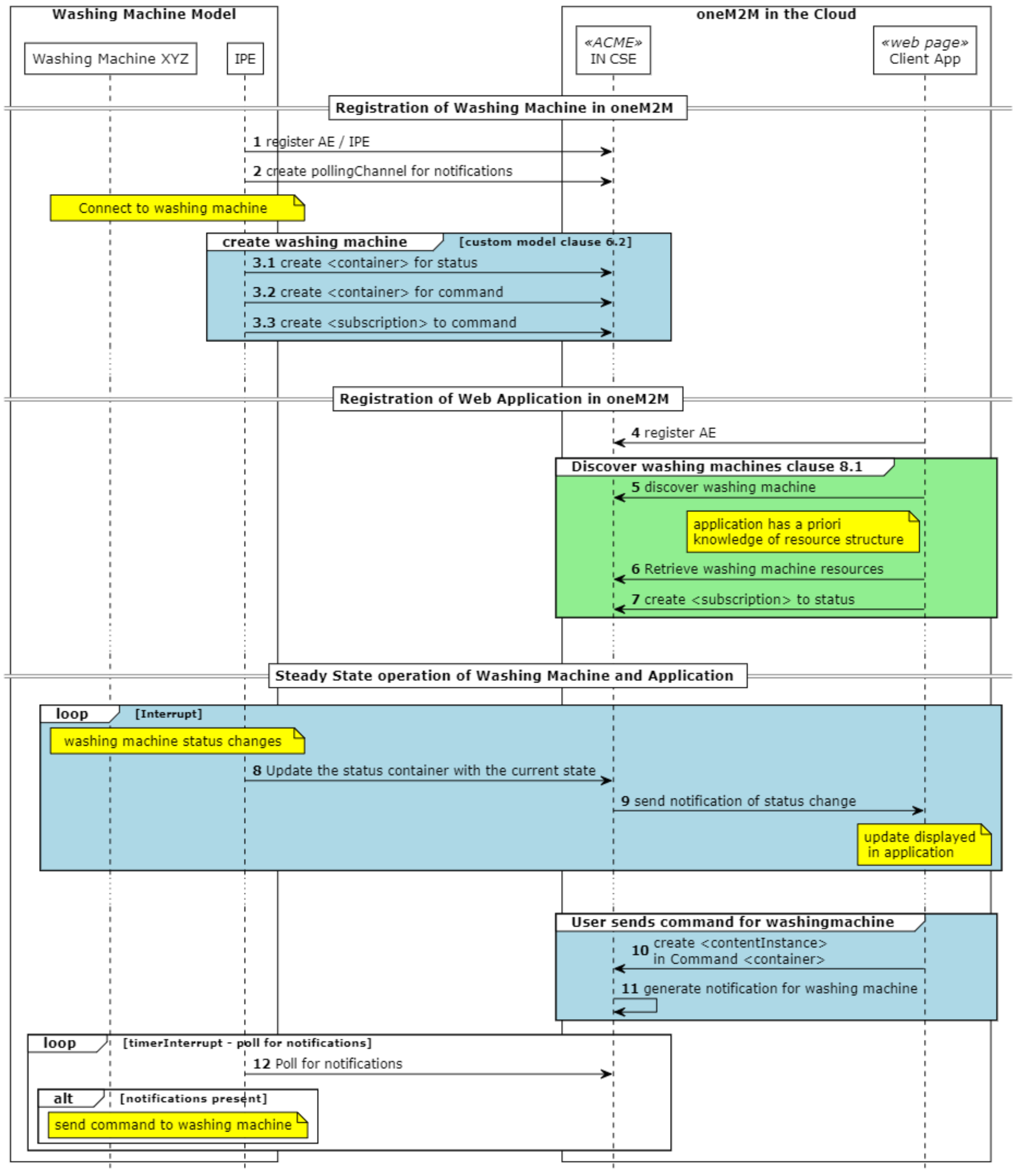

Washing Machine Model
Washing Machine XYZ
IPE
oneM2M in the Cloud
«ACME» IN CSE
«web page» Client App
Registration of Washing Machine in oneM2M
1 register AE / IPE
2 create pollingChannel for notifications
Connect to washing machine
create washing machine
[custom model clause 6.2]
3.1 create <container> for status
3.2 create <container> for command
3.3 create <subscription> to command
Registration of Web Application in oneM2M
4 register AE
Discover washing machines clause 8.1
5 discover washing machine
application has a priori knowledge of resource structure
6 Retrieve washing machine resources
7 create <subscription> to status
Steady State operation of Washing Machine and Application
loop
[Interrupt]
washing machine status changes
8 Update the status container with the current state
9 send notification of status change
update displayed in application
User sends command for washingmachine
10 create <contentInstance> in Command <container>
11 generate notification for washing machine
loop
[timerInterrupt - poll for notifications]
12 Poll for notifications
alt
[notifications present]
send command to washing machine

Figure 6: Custom Model oneM2M Call Flows

Only the messages highlighted in light blue are described here as the rest of the messages are the same as in the general call flow described in clause 5.1.

Create Information Model (Figure 6): The IPE creates all the resources needed to for the clothes washing machine that it knows how to model a priori. This IPE is developed with awareness of the clothes washing machine interface and the model that it is creating in the oneM2M CSE.

A <container> resource is created for the Status information of the clothes washing machine. The IPE creates <contentInstance> resources that have the following content when there are any changes in the status of the clothes washing machine:

```
{
    "WashingMachineStatus ": "WASHING", // Or "STOPPED", "ERROR"
}
```

A <container> resource is created for the command and control of the clothes washing machine. When the client application is setting the state of the device the following payload can be provided in a <contentInstance> resource:

```
{
    "state": "ON", // Or "OFF", "Toggle"
}
```

A <subscription> resource is created as a child of the command <container> resource by the IPE. This will cause a notification to be sent to the IPE when a new command is made by an application.

5.3 Semantic Modelling

A SAREF description of the washing machine is mapped to the resource structure shown in Figure 7 using the rules described in ETSI TS 118 130 [i.2] and ETSI TS 118 112 [i.1]. A complete derivation of this example is shown in ETSI TS 118 112 [i.1], clause B.1.3.3.

The description of our (simplified) washing machine using SAREF ontology is expanded upon here:

- The state of the washing machine is given by SAREF:state: **WashingMachineStatus** that can take the values "WASHING" or "STOPPED" or "ERROR".
- The washing machine has an actuating function: **StartStopFunction** which has three commands:
  - **ON_Command**
  - **OFF_Command**
  - **Toggle_Command**
- The washing machine has also a metering function: **MonitoringFunction** that sets the WashingMachineStatus.
- The washing machine is located at **My_Bathroom.**

Later it is shown that the description here has triples that are intended to help define the resource tree structure according to the rules described in ETSI TS 118 130 [i.2] and ETSI TS 118 112 [i.1]. However, when the description of the clothes-washing machine is put into a <semanticdescriptor> some are removed because they do not offer information useful for SPARQL queries.

```
@prefix rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs:   <http://www.w3.org/2000/01/rdf-schema#> .
@prefix oneM2M: <http://www.onem2m.org/ontology/Base_Ontology/> .
@prefix saref: <https://saref.etsi.org/core/> .
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/> .
@prefix sn:    <http://www.XYZ.com/WashingMachines/SerialNumbers/> .

sn:WASH_XYZ
        a                       <http://www.XYZ.com/WashingMachines#XYZ_Cool> ;
        rdfs:comment            "Very cool Washing Machine" ;
        saref:hasFunction       sn:WASH_XYZ-MonitoringFunction , sn:WASH_XYZ-StartStopFunction ;
        saref:hasManufacturer   "XYZ" ;
        saref:hasService        sn:WASH_XYZ-MonitorService , sn:WASH_XYZ-SwitchOnService ;
        saref:hasState          sn:WASH_XYZ-WashingMachineStatus ;
        s4bldg:isContainedIn    sn:My_Bathroom .


sn:WASH_XYZ-StartStopFunction-OFF_Command      a       saref:OffCommand .

sn:WASH_XYZ-StartStopFunction-Toggle_Command   a       saref:ToggleCommand .


sn:WASH_XYZ-StartStopFunction-ON_Command                          a       saref:OnCommand .

sn:WASH_XYZ-MonitoringFunction                a       saref:SensingFunction ;
                saref:hasCommand  sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus .

sn:WASH_XYZ-StartStopFunction                 a       saref:ActuatingFunction ;
                saref:hasCommand   sn:WASH_XYZ-StartStopFunction-Toggle_Command ,
                                                    sn:WASH_XYZ-StartStopFunction-OFF_Command ,
                                                    sn:WASH_XYZ-StartStopFunction-ON_Command .
```

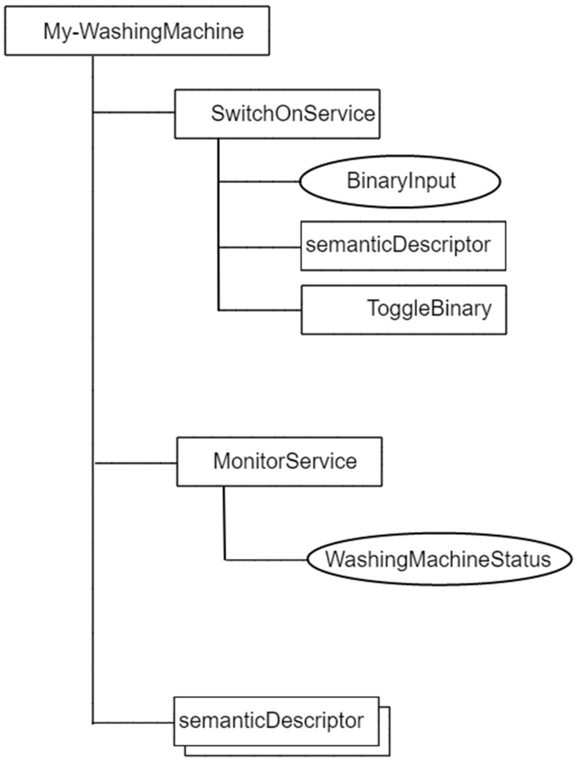


Figure 7: SAREF Washing Machine Model

The procedure defined in ETSI TS 118 112 [i.1] requires the IPE to parse the semantic description to generate a total of three custom <flexContainer> definitions to support the structure shown in Figure 7. The schemas generated are added as the content of a <contentInstance> resource under a container for these custom definitions. The locations of these schemas are referenced in the container definition attribute of the respective <flexContainer>:

- two *<flexContainer>* child-resources for Services and their <semanticDescriptor>s are used for modelling the services SwitchOnService and MonitorService;

- the SwitchOnService in turn has a child resource of type <*flexContainer*> for Operations which exposes the Toggle_Command;
- one *customAttribute* of the SwitchOnService <*flexContainer*> is used for holding the values for InputDataPoint: BinaryInput;
- one *customAttribute* of the MonitorService <*flexContainer*> is used for holding the values for OutputDataPoint: WashingMachineStatus.

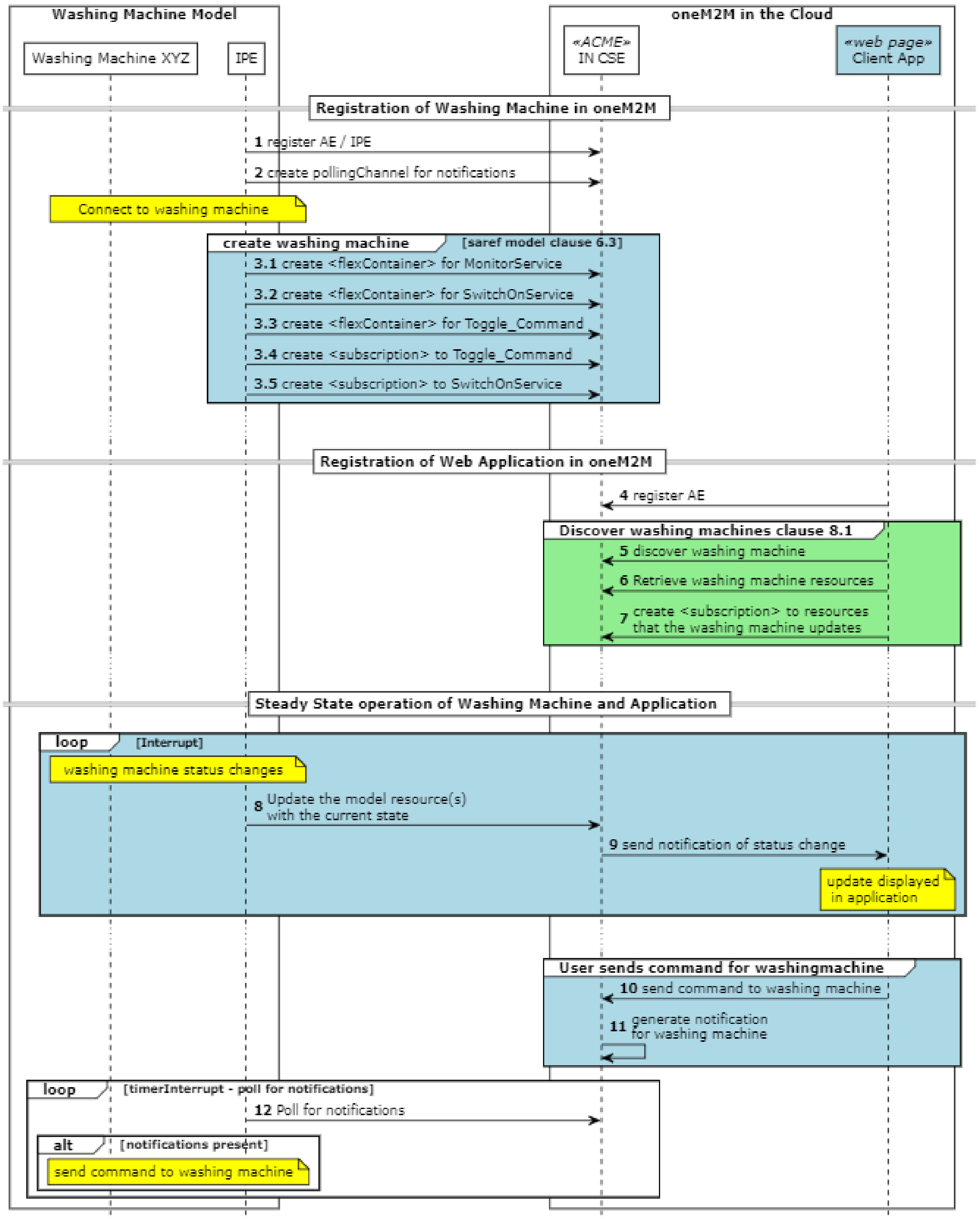

Washing Machine Model
Washing Machine XYZ
IPE
oneM2M in the Cloud
«ACME» IN CSE
«web page» Client App
Registration of Washing Machine in oneM2M
1 register AE / IPE
2 create pollingChannel for notifications
Connect to washing machine
create washing machine
[saref model clause 6.3]
3.1 create <flexContainer> for MonitorService
3.2 create <flexContainer> for SwitchOnService
3.3 create <flexContainer> for Toggle_Command
3.4 create <subscription> to Toggle_Command
3.5 create <subscription> to SwitchOnService
Registration of Web Application in oneM2M
4 register AE
Discover washing machines clause 8.1
5 discover washing machine
6 Retrieve washing machine resources
7 create <subscription> to resources that the washing machine updates
Steady State operation of Washing Machine and Application
loop
[Interrupt]
washing machine status changes
8 Update the model resource(s) with the current state
9 send notification of status change
update displayed in application
User sends command for washingmachine
10 send command to washing machine
11 generate notification for washing machine
loop
[timerInterrupt - poll for notifications]
12 Poll for notifications
alt
[notifications present]
send command to washing machine

Figure 8: SAREF Model oneM2M Call Flows

Only the messages highlighted in light blue are described here as the rest of the messages are the same as in the general call flow described in clause 5.1.

Create Information Model (Figure 8): The IPE creates all the resources needed to for the clothes washing machine that it generates from parsing the semantic description. This IPE is developed with awareness of the clothes washing machine interface but without awareness of the model that it is creating in the oneM2M CSE. This requires extra logic to parse the RDF triples to generate custom container definitions, which is not included in this example as only the output of the parsing process is shown.

> A <*flexContainer*> resource is created for the MonitorService with a single custom attribute washingMachineStatus. The IPE updates this resource with the following content when there are any changes in the status of the clothes washing machine:

```
{
    "WashingMachineStatus ": "WASHING", // Or "STOPPED", "ERROR"
}
```

> A <flexcontainer> resource is created for the SwitchOnService that allows command and control of the clothes washing machine. The client application sets the state of the device by updating the resource with the following payload:

```
{
    "BinaryInput": False, // Or True
}
```

A <flexcontainer> resource is created for the Toggle command as a child of the SwitchOnService. This action is used to change the current state of the clothes washing machine. The client application toggles the state of the device by sending an update request to the resource with an empty payload.

A <subscription> resource is created as a child of the Toggle command <flexContainer> resource by the IPE. This will cause a notification to be sent to the IPE when a new command is made by an application.

A <subscription> resource is created as a child of the SwitchOnService <flexContainer> resource by the IPE. This will cause a notification to be sent to the IPE when a new command is made by an application

5.4 Smart Device Modelling

ETSI TS 118 123 [i.5] defines a framework for developing common standardized models of devices. There are multiple device specific domains defined and new models and domains are added in each release of oneM2M. The Home Domain contains a deviceClothesWasher model that aligns with the device that has been modeled. The resource tree structure of the deviceClothesWasher is shown in Figure 9. There are many more potential services exposed in this model than our example simplified washing machine provides. The elements in bold are required for a compliant SDT model. Services in the model that are not supported by our simple clothes-washing machine are not implemented unless they are required. It should be noted that using an SDT model is the only model that can be certified by a certification authority.

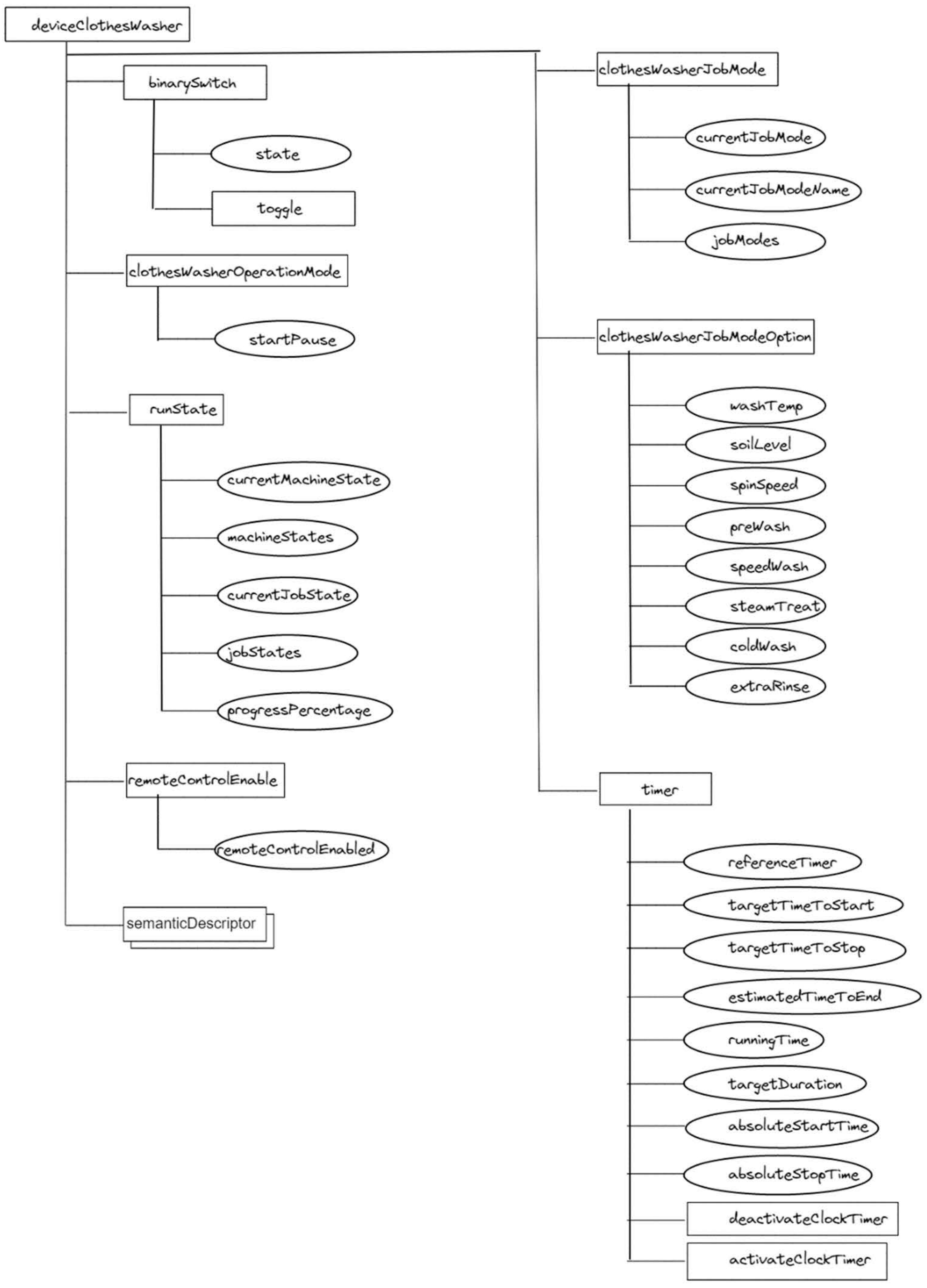


Figure 9: SDT washing machine model

When using a SDT model from ETSI TS 118 123 [i.5] to represent a physical device it is necessary to map the functionality of the device to be modelled with the existing modules defined for the SDT device.

Table 1

Meta-Data	Device Value	SDT modelling
Manufacturer	XYZ	The SDT model captures this information in a dmDeviceInfo ModuleClass
Manufacturer description	"Very cool Washing Machine"	The SDT model captures this information in a dmDeviceInfo ModuleClass
Model Type	XYZ_Cool	The SDT model captures this information in a dmDeviceInfo ModuleClass

Meta-Data	Device Value	SDT modelling
Supported Commands	ON OFF Toggle	The SDT model enables the ON and OFF commands using the *state* attribute of the *binarySwitch* ModuleClass. The Toggle command is supported by the *toggle* ActionModule
State	"WASHING" "STOPPED" "ERROR"	The SDT model offers runState ModuleClass which supports more enumerations than indicated by our product
Location	My_Bathroom	The SDT model does not have an attribute specifically for Location

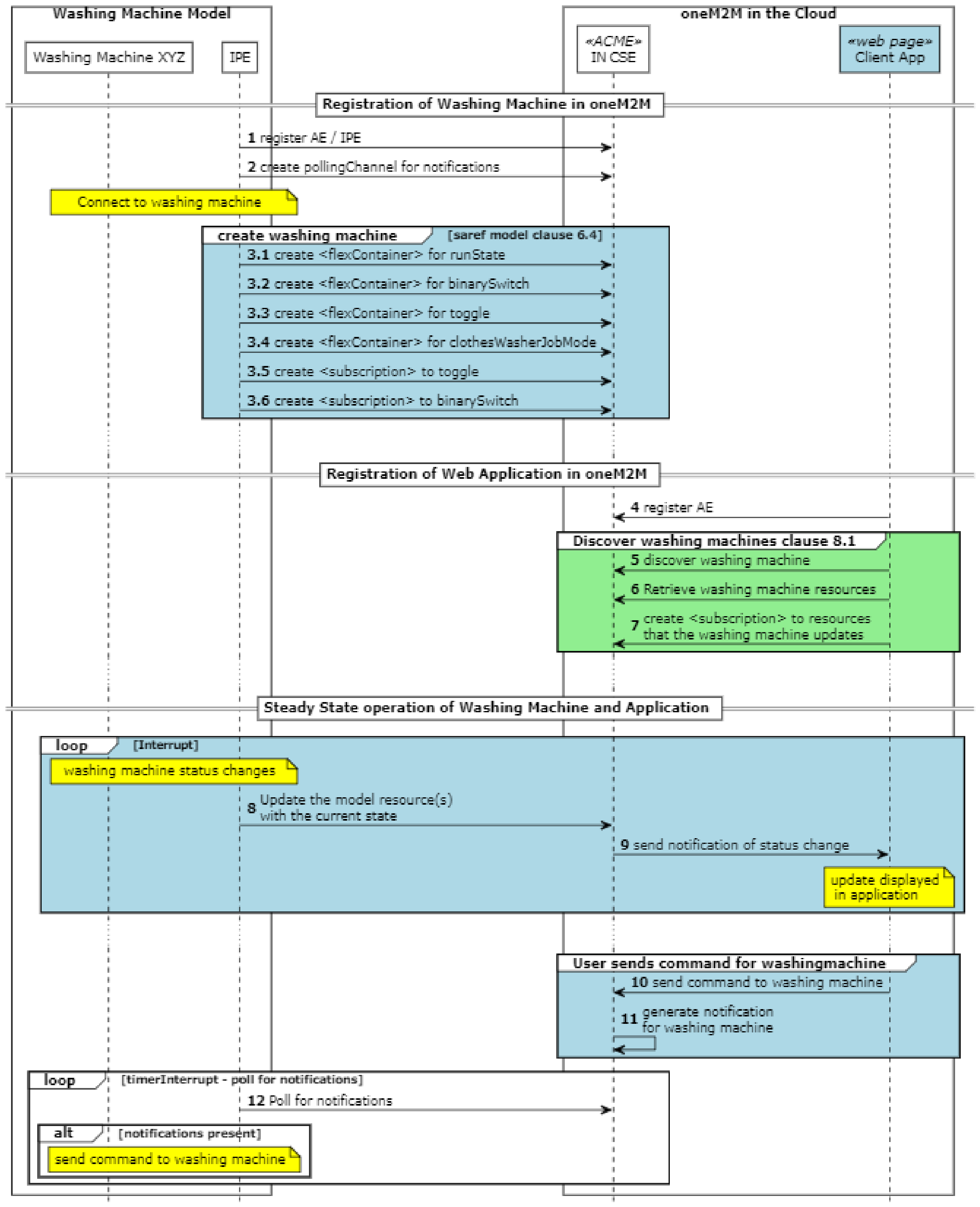

Washing Machine Model
Washing Machine XYZ
IPE
oneM2M in the Cloud
«ACME» IN CSE
«web page» Client App
Registration of Washing Machine in oneM2M
1 register AE / IPE
2 create pollingChannel for notifications
Connect to washing machine
create washing machine
[saref model clause 6.4]
3.1 create <flexContainer> for runState
3.2 create <flexContainer> for binarySwitch
3.3 create <flexContainer> for toggle
3.4 create <flexContainer> for clothesWasherJobMode
3.5 create <subscription> to toggle
3.6 create <subscription> to binarySwitch
Registration of Web Application in oneM2M
4 register AE
Discover washing machines clause 8.1
5 discover washing machine
6 Retrieve washing machine resources
7 create <subscription> to resources that the washing machine updates
Steady State operation of Washing Machine and Application
loop
[Interrupt]
washing machine status changes
8 Update the model resource(s) with the current state
9 send notification of status change
update displayed in application
User sends command for washingmachine
10 send command to washing machine
11 generate notification for washing machine
loop
[timerInterrupt - poll for notifications]
12 Poll for notifications
alt
[notifications present]
send command to washing machine

Figure 10: SDT model oneM2M Call Flows

Only the messages highlighted in light blue are described here as the rest of the messages are the same as in the general call flow described in clause 5.1.

Create Information Model (Figure 10): The IPE creates all the resources needed to for the clothes washing machine that it knows how to model a priori using SDT. This IPE is developed with awareness of the clothes washing machine interface and the model that it is creating in the oneM2M CSE.

A <*flexContainer*> resource is created for the runState with the custom attribute currentMachineState and MachineStates. The IPE updates this resource with the following content when there are any changes in the status of the clothes washing machine:

```
{
    "CurrentMachineState ": 3, // Or [1,3,5,6]
}
```

A <flexcontainer> resource is created for the binarySwitch module that allows command and control of the clothes washing machine. The client application sets the state of the device by updating the resource with the following payload:

```
{
    "state": false, // Or true
}
```

A <flexcontainer> resource is created for the Toggle command as a child of the binarySwitch. This action is used to change the current state of the clothes washing machine. The client application toggles the state of the device by sending an update request to the resource with an empty payload.

A <flexcontainer> resource is created for the clothesWasherJobMode with custom attributes currentJobMode and jobModes. This resource is mandatory for the deviceClothesWasher SDT model, but the IPE will set the states and never modify them.

A <subscription> resource is created as a child of the toggle <flexContainer> resource by the IPE. This will cause a notification to be sent to the IPE when a new command is made by an application.

A <subscription> resource is created as a child of the binarySwitch command <flexContainer> resource by the IPE. This will cause a notification to be sent to the IPE when a new command is made by an application.

6 Semantic Annotation in oneM2M

6.1 Semantic description of services using SAREF Ontology

The Smart Applications REFerence ontology (SAREF) is intended to enable interoperability between solutions from different providers and among various activity sectors in the Internet of Things (IoT), thus contributing to the development of the global digital market. SAREF explicitly specifies the recurring core concepts in the Smart Applications domain, the main relationships between these concepts, and axioms to constrain the usage of these concepts and relationships. SAREF is based on the fundamental principles of reuse and alignment of concepts and relationships that are defined in existing assets, modularity to allow separation and recombination of different parts of the ontology depending on specific needs, extensibility to allow further growth of the ontology, and maintainability to facilitate the process of identifying and correcting defects, accommodate new requirements, and cope with changes in (parts of) SAREF.

SAREF ontology has been used to describe the services of the washing machine and the oneM2M Base Ontology to describe the oneM2M interface for the services.

The services of any clothes washing machine are fundamentally the same regardless of which model is used. Especially in this case where the same clothes washing machine is described. The following RDF triples describe the services and functions of our clothes washing machine.

```
@prefix saref: <https://saref.etsi.org/core/>.
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sn: <http://www.XYZ.com/WashingMachines#XYZ_Cool/>.
@prefix m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>.

sn:WASH_XYZ_RESOURCE_ID a <http://www.XYZ.com/WashingMachines#XYZ_Cool>;
    rdfs:comment "Very cool Washing Machine";
    saref:hasFunction sn:WASH_XYZ-MonitoringFunction, sn:WASH_XYZ-StartStopFunction;
    saref:hasManufacturer  "XYZ";
    saref:hasService       sn:WASH_XYZ-MonitorService , sn:WASH_XYZ-SwitchOnService;
    saref:hasState         sn:WASH_XYZ-WashingMachineStatus;
    s4bldg:isContainedIn   sn:My_Bathroom ;
    m2m:oneM2MTargetURI    "RESOURCE_ID";
    m2m:hasOperation sn:WASH_XYZ-SwitchOnService_RESOURCE_ID,
                     sn:WASH_XYZ-StartStopFunction-ON_Command_RESOURCE_ID,
                     sn:WASH_XYZ-StartStopFunction-OFF_Command_RESOURCE_ID,
                     sn:WASH_XYZ-StartStopFunction-TOGGLE_Command_RESOURCE_ID,
                     sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus_RESOURCE_ID.
```

These triples will be placed into a <semanticDescription> resource in each of the models.

NOTE: These triples have a token "RESOURCE_ID" in several places that will be replaced at execution time with a resource identifier or resource address related to the parent of this particular <semanticDescriptor>.

6.2 Describing oneM2M APIs with the oneM2M Base Ontology

6.2.1 Clothes Washing Machine APIs using oneM2M

Because the resource tree structure for each of the models is different the oneM2M primitives needed to access the services of the clothes washing machine will also be different. However, the goal for interworking device models is to allow a user to issue the same command to perform an operation regardless of which model is used. This can be approximated in a dynamic manner using the oneM2M base ontology to describe each of the services offered by the device and the resources that provide access to those services. For example, the washing machine that has been described offers the following operations:

1) TURN ON WASHING MACHINE

2) TURN OFF WASHING MACHINE

3) TOGGLE THE WASHING MACHINE STATUS

4) GET STATUS OF WASHING MACHINE

The oneM2M primitives to execute these operations are dependent on the resource tree structure used to model the washing machine. For example, to determine the status of the washing machine for each model the following oneM2M requests and responses are used.

Table 2

Model	Request	Response
SDT	RETRIEVE /cseBaseName/IPE_ROOT/deviceclothesWasher/runState	{ " currentMachineState ": 3 "machineStates": [1,3,5,6] "currentJobState": 6 "jobStates":[2,3,4,5,6] "progressPercentage":95.0 }
SAREF	RETRIEVE /cseBaseName/IPE_ROOT/My-WashingMachine/sarefWashingMachine/MonitorService	{ "WashingMachineStatus":"WASHING" }
Custom	RETRIEVE /cseBaseName/IPE_ROOT/myWashingMachine/Status/la	{ "WashingMachineStatus":"WASHING" }

Similarly, to command the washing machine to STOP the following oneM2M primitives are sent.

Table 3

Model	Request

SDT	UPDATE /cseBaseName/IPE_ROOT/deviceClothesWasher/binarySwitch
	{"state": false }

SAREF	UPDATE /cseBaseName/IPE_ROOT/My-WashingMachine/SwitchOnService
	{"BinaryInput": false}

Custom	CREATE /cseBaseName/IPE_ROOT/myWashingMachine/Command
	{"OFF"}

6.2.2 Custom Model API semantic description

The specific primitive requests defined in clause 6.2.1 are described in RDF triples using the oneM2M base ontology. The properties of interest in the oneM2M base ontology are:

- m2m:oneM2MTargetURI
- m2m:hasDataRestriction
- m2m:oneM2Mmethod
- m2m:oneM2Mattribute

```
@prefix saref: <https://saref.etsi.org/core/>.
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sn: <http://www.XYZ.com/WashingMachines#XYZ_Cool/>.
@prefix m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>.

sn:WASH_XYZ-StartStopFunction-ON_Command_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/OnCommand>;
    m2m:oneM2MTargetURI "/myWashingMachine/command";
    m2m:hasDataRestriction "ON";
    m2m:oneM2Mmethod "CREATE".

sn:WASH_XYZ-StartStopFunction-OFF_Command_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/OffCommand>;
    m2m:oneM2MTargetURI "/myWashingMachine/command";
    m2m:hasDataRestriction "OFF";
     m2m:oneM2Mmethod "CREATE".

sn:WASH_XYZ-StartStopFunction-TOGGLE_Command_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/ToggleCommand>;
    m2m:oneM2MTargetURI "/myWashingMachine/command";
    m2m:hasDataRestriction "TOGGLE";
    m2m:oneM2Mmethod "CREATE".

sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/GetCommand>;
    m2m:oneM2MTargetURI "/myWashingMachine/status";
    m2m:oneM2Mmethod "RETRIEVE".
```

6.2.3 Semantic Model annotation

The specific primitive requests described in clause 6.2.1 are described in RDF triples using the oneM2M base ontology. The classes of interest in the oneM2M base ontology are:

- m2m:oneM2MTargetURI
- m2m:hasDataRestriction
- m2m:oneM2MMethod
- m2m:oneM2Mattribute

```
@prefix saref: <https://saref.etsi.org/core/>.
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sn: <http://www.XYZ.com/WashingMachines#XYZ_Cool/>.
@prefix m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>.

sn:WASH_XYZ-StartStopFunction-ON_Command_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/OnCommand>;
    m2m:oneM2MTargetURI "/My-WashingMachine/SwitchOnService";
    m2m:oneM2Mattribute  "BinaryInput";
    m2m:oneM2MMethod "UPDATE";
    m2m:hasDataRestriction "true".

sn:WASH_XYZ-StartStopFunction-OFF_Command_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/OffCommand>;
    m2m:oneM2MTargetURI "/My-WashingMachine/SwitchOnService";
    m2m:oneM2Mattribute  "BinaryInput";
    m2m:oneM2MMethod "UPDATE";
    m2m:hasDataRestriction  "false".

sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus_RESOURCE_ID a m2m:Operation,
<https://saref.etsi.org/core/GetCommand>;
    m2m:oneM2MTargetURI "/My-WashingMachine/sarefWashingMachine";
    m2m:oneM2MMethod    "RETRIEVE";
    m2m:oneM2Mattribute  "WashingMachineStatus".
```

6.2.4 SDT model annotation

The specific primitive requests described in clause 6.2.1 are described in RDF triples using the oneM2M base ontology. The classes of interest in the oneM2M base ontology are:

- m2m:oneM2MTargetURI
- m2m:hasDataRestriction
- m2m:oneM2MMethod
- m2m:oneM2Mattribute

```
@prefix saref: <https://saref.etsi.org/core/>.
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sn: <http://www.XYZ.com/WashingMachines#XYZ_Cool/>.
@prefix m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>.

sn:WASH_XYZ-StartStopFunction-ON_Command  a  m2m:Operation,
<https://saref.etsi.org/core/OnCommand>;
    m2m:oneM2MTargetURI "/deviceClothesWasher/binarySwitch";
    m2m:oneM2Mattribute  "powerState";
    m2m:hasDataRestriction "true";
    m2m:oneM2MMethod "UPDATE".

sn:WASH_XYZ-StartStopFunction-OFF_Command a m2m:Operation,
<https://saref.etsi.org/core/OffCommand>;
    m2m:oneM2MTargetURI "/deviceClothesWasher/binarySwitch";
    m2m:oneM2Mattribute  "powerState";
    m2m:hasDataRestriction "false";
    m2m:oneM2MMethod "UPDATE".

sn:WASH_XYZ-StartStopFunction-TOGGLE_Command a m2m:Operation,
<https://saref.etsi.org/core/ToggleCommand>;
    m2m:oneM2MTargetURI "/deviceClothesWasher/binarySwitch/toggle";
    m2m:oneM2MMethod "UPDATE".

sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus a m2m:Operation;
    m2m:oneM2MTargetURI "/deviceClothesWasher/runState";
    m2m:oneM2Mattribute "currentMachineState";
    m2m:oneM2MMethod "RETRIEVE".
```

7 Semantic Queries

7.1 Foreword

This clause describes the semantic queries and how the responses can be used to achieve interoperability. Generally, these queries are executed by an application that is designed to use the IoT devices.

7.2 Discovery Queries

By using the oneM2M Base Ontology in the <semanticDescriptor> resources it is possible to send queries to the oneM2M CSE to find the services offered by a device and further query those services to discover the oneM2M primitives to access those services.

Here is a list of queries that usable for all three of the models:

Query 1: Find all washing machines of manufacturer XYZ.

```
PREFIX sn:<http://www.XYZ.com/WashingMachines#XYZ_Cool/>
PREFIX m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>
PREFIX saref: <https://saref.etsi.org/core/>

SELECT   ?res ?wm where {
    ?wm  a  sn:XYZ_Cool .
    ?wm m2m:oneM2MTargetURI ?res
}
```

This lists 3 washing machines.

Table 4

res	wm
"myWashingMachine"	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine
"My-WashingMachine"	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_My-WashingMachine
"deviceClothesWasher"	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_deviceClothesWasher

Query 2: List all the commands offered by a specific washing machine

```
PREFIX sn:<http://www.XYZ.com/WashingMachines#XYZ_Cool/>
PREFIX m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>
PREFIX saref: <https://saref.etsi.org/core/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?wm ?operation ?command
WHERE {
    ?wm m2m:hasOperation ?operation .
    ?operation a ?command .
    VALUES ?wm {<http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine>} .
    ?command rdfs:subClassOf saref:Command
}
```

This lists the operations and commands and functions associated with the commands.

Table 5

wm	operation	command
http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ-StartStopFunction-ON_Command_myWashingMachine	saref:OnCommand
http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ-StartStopFunction-OFF_Command_myWashingMachine	saref:OffCommand
http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ-StartStopFunction-TOGGLE_Command_myWashingMachine	saref:ToggleCommand
http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_myWashingMachine	http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ-MonitoringFunction-WashingMachineStatus_myWashingMachine	saref:GetCommand

7.3 Interoperability Queries

The following queries demonstrate how interoperability is achieved using semantics in oneM2M. Using the results of the queries above it is possible to issue the following types of queries to determine exactly how to use the services of the washing machines, without regard the way it was modelled. This query shows how to use the saref:GetCommand for the SDT model of the washing machine

Query 3: How do I use saref:GetCommand of the SDT washing machine?

```
SELECT ?sarefCommand ?method ?targetURI ?attr ?res
WHERE {
    ?wm m2m:hasOperation ?operation .
    ?operation a m2m:Operation .
    ?operation m2m:oneM2MMethod ?method .
    optional {?operation m2m:hasDataRestriction ?res} .
    optional {?operation m2m:oneM2Mattribute ?attr} .
    ?operation a ?sarefCommand .
    ?operation m2m:oneM2MTargetURI ?targetURI .
    VALUES ?wm {<http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_deviceClothesWasher>} .
    VALUES ?sarefCommand {saref:GetCommand}
}
```

The result of executing query 3 is:

sarefCommand	Method	targetURI	Attr	res
saref:GetCommand	"RETRIEVE"	"/deviceClothesWasher/runState"	"currentMachineState"	

Query 3 can be modified for demonstration purposes to show the response for all three washing machines by removing the line beginning with "VALUES ?wm". The result of this query can be compared with the expected responses described in clause 6.2.1.

sarefCommand	method	targetURI	attr	res
saref:GetCommand	"RETRIEVE"	"/deviceClothesWasher/runState"	"currentMachineState"	
saref:GetCommand	"RETRIEVE"	"/My-WashingMachine/sarefWashingMachine/MonitorService"	"WashingMachineStatus"	
saref:GetCommand	"RETRIEVE"	"/myWashingMachine/status/la"		

Query 4: How do I use all commands of the washing machine modelled with SDT.

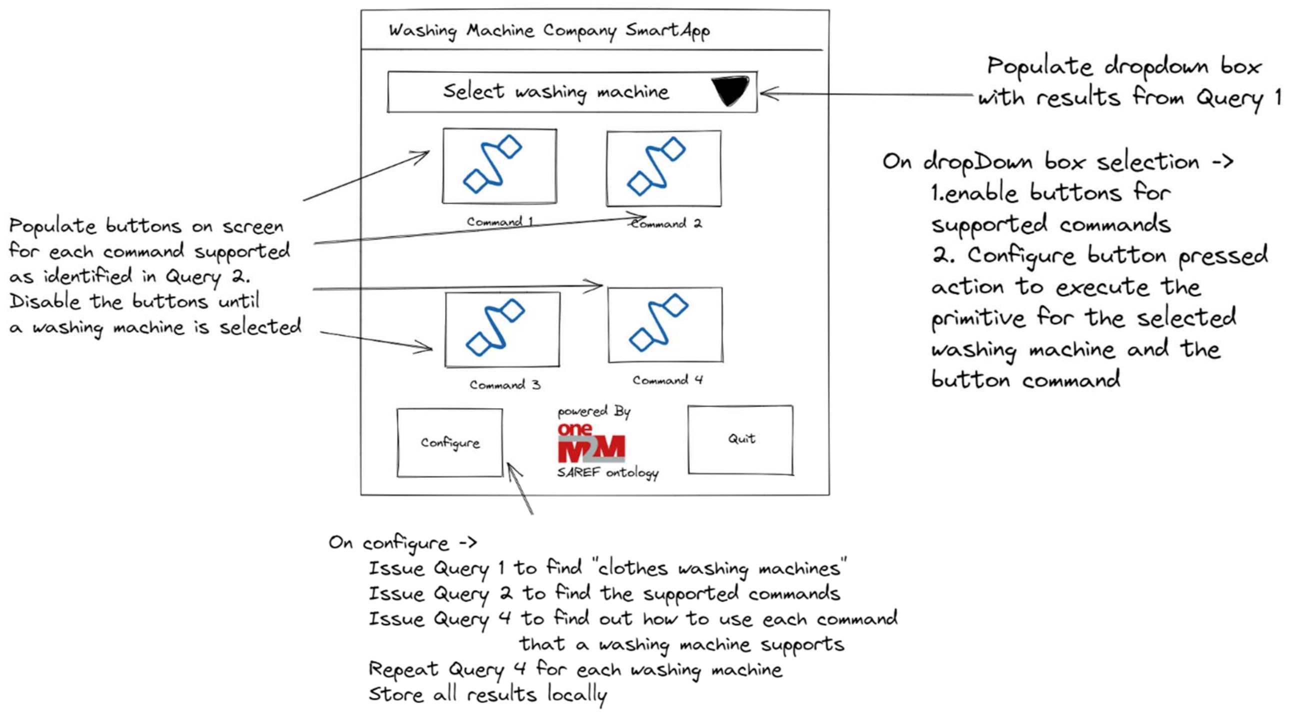
```
SELECT ?sarefCommand ?method ?targetURI ?attr ?res
WHERE {
    ?wm m2m:hasOperation ?operation .
    ?operation a m2m:Operation .
    ?operation m2m:oneM2MMethod ?method .
    optional {?operation m2m:hasDataRestriction ?res} .
    optional {?operation m2m:oneM2Mattribute ?attr} .
    ?operation a ?sarefCommand .
    ?operation m2m:oneM2MTargetURI ?targetURI .
    VALUES ?wm {<http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_deviceClothesWasher>} .
    VALUES ?sarefCommand {saref:GetCommand saref:OnCommand saref:OffCommand saref:ToggleCommand}
}
ORDER BY ?sarefCommand
```

This query can be issued after discovering the appropriate device to dynamically build the commands needed to perform operations on the device.

sarefCommand	method	targetURI	attr	res
saref:GetCommand	"RETRIEVE"	"/deviceClothesWasher/runState"	"currentMachineState"	
saref:OffCommand	"UPDATE"	"/deviceClothesWasher/binarySwitch"	"powerState"	"False"
saref:OnCommand	"UPDATE"	"/deviceClothesWasher/binarySwitch"	"powerState"	"True"
saref:ToggleCommand	"UPDATE"	"/deviceClothesWasher/binarySwitch/toggle"		

The SPARQL query that is used is a critical component of the ability to dynamically determine the API of the model. The tokens following the SELECT statement are variables that will be included in the response. For this use case it is necessary to know what oneM2M primitive to send to a CSE to perform the desired command. The method implies the type of resource that is at the targetURI. An UPDATE method implies that the targetURI is a <flexContainer> whereas if the method is CREATE then the resource type being created will be a <contentInstance>. In the case of a <flexContainer> "attr" specifies the custom attribute that needs to be updated and the "res" specifies the value to use in that attribute.

By designing a smartphone application to control the washing machine it might look like the wireframe shown in Figure 11.



Populate dropdown box with results from Query 1

On dropDown box selection ->
1.enable buttons for supported commands
2. Configure button pressed action to execute the primitive for the selected washing machine and the button command

Populate buttons on screen for each command supported as identified in Query 2.
Disable the buttons until a washing machine is selected

On configure ->
Issue Query 1 to find "clothes washing machines"
Issue Query 2 to find the supported commands
Issue Query 4 to find out how to use each command that a washing machine supports
Repeat Query 4 for each washing machine
Store all results locally

Figure 11: Wireframe

8 Procedures

8.1 Introduction

Previous clauses describe the use case, the call flows, the semantic description of our device and the SPARQL queries that can be used based on the created semantic descriptions. As it is possible to see in this clause, the values returned from a query are fully dependent on the query issued. The oneM2M CSE will pass the query results back in the format defined by SPARQL query results. This example receives a JSON response (the query is sent using XML just to highlight that this can be done).

This clause will show how to create the oneM2M primitives that implement the use case described above. This example will focus on the semantic description resources and the semantic queries. The specific primitives needed to create the <AE>, <container>, and <flexcontainer> resources can be found in oneM2M Developer Guides at https://wiki.onem2m.org/index.php?title=Developer_Guides.

8.2 Implementation

8.2.1 Semantics Description Utilities

The requirements for the creation of a <semanticDescription> resource include base64 encoding the 'dsp' attribute. There are many libraries that do this operation and for this example a python library was used.

```
import base64

def smdEncode(description):
    msgAscii = description.encode('ascii')
    b64 = base64.b64encode(msgAscii)
    descriptionb64 = b64.decode('ascii')
    return descriptionb64

def smdDecode(message):
    b64d = message.encode('ascii')
    msgdAscii = base64.b64decode(b64d)
    return msgdAscii.decode('ascii')
```

8.2.2 Semantic Query Utilities

When sending a SPARQL query as a request parameter for a oneM2M primitive, the query shall be "url" encoded. There are many libraries that do this operation and for this example a python library was used.

```
import urllib.parse

def encodedSparqlQuery(query):
    return (urllib.parse.quote(query,safe='')) #safe is by default '/'.
```

8.3 Semantics representations and primitives

8.3.1 Introduction

In oneM2M the <semanticDescriptor> resource is used to provide semantic annotations, such as the ones in clause 7. Semantic annotations can use the oneM2M base ontology as well as external ontologies, such as SAREF. The oneM2M base ontology is primarily used to discover how to use the APIs for devices that are modelled in oneM2M. External Ontologies are used to describe the capabilities of the device being modeled or other features of data that is available in oneM2M, i.e. ontologies could describe the content of data or metadata. A <semanticDescriptor> resource can be a child of <AE>, <container>, <contentInstance>, <group>, <node>, <flexContainer>, <timeSeries>, <mgmtObj> resources.

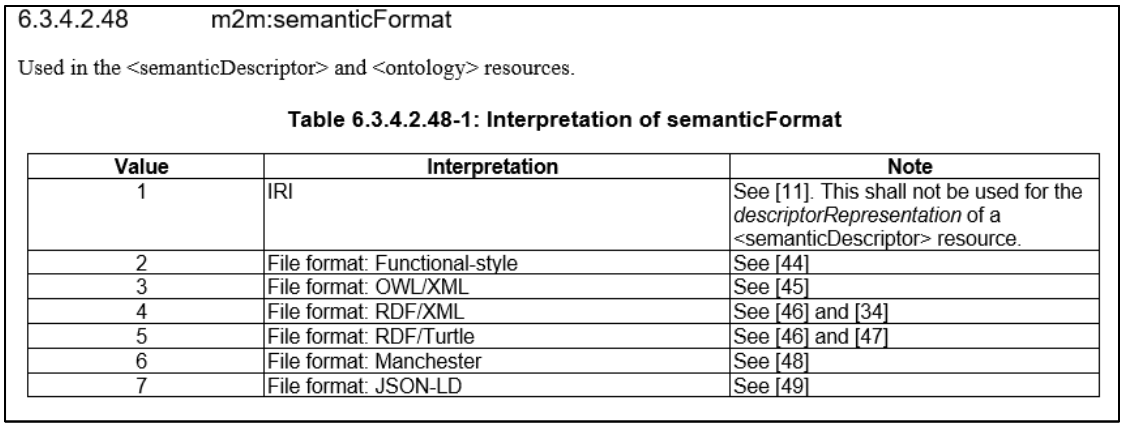
The semantic annotations in a <semanticDescriptor> can apply to the parent resource or other resources. There are two types of semantic searches that can be performed:

1) semantic discovery; and

2) semantic query.

A semantic discovery will find matching <semanticDescriptor> resources and provide the URI of the parent resource of the <semanticDescriptor> resources that match the query. A semantic query request will return the response to the SPARQL query in the format defined in the query. These differences may impact the decision regarding what parent resource to target for a <semanticDescriptor> resource.

The representation of a <semanticDescriptor> resource shall be in one of the semantic formats supported in oneM2M. The supported formats from ETSI TS 118 104 [i.3] are:



6.3.4.2.48 m2m:semanticFormat

Used in the <semanticDescriptor> and <ontology> resources.

Table 6.3.4.2.48-1: Interpretation of semanticFormat

Value	Interpretation	Note
1	IRI	See [11]. This shall not be used for the *descriptorRepresentation* of a <semanticDescriptor> resource.
2	File format: Functional-style	See [44]
3	File format: OWL/XML	See [45]
4	File format: RDF/XML	See [46] and [34]
5	File format: RDF/Turtle	See [46] and [47]
6	File format: Manchester	See [48]
7	File format: JSON-LD	See [49]

Figure 12: supported format from ETSI TS 118 104 [i.3]

In this use case RDF/XML is used in the primitives as that is supported by the test implementation. The semantic annotations shown in clause 6 are written in RDF/Turtle. A convenient utility to convert the RDF/Turtle to RDF/XML is available at https://www.easyrdf.org/converter or the python rdflib library.

Another requirement for the <semanticDescriptor> resource is that the *descriptor* attribute is set to the value of the semantic triples encoded using xsd:base64Binary.

And finally, when issuing a Semantic request, whether it is a discovery or query, the *semanticFilter* parameter of the request requires "percent-encoding" when using the HTTP protocol binding, as used in the examples in the present document.

Since the <semanticDescriptor> resource is separate from the resources that it describes there is considerable flexibility available to application developers. For example, if a product such as an IPE for a clothes-washing machine does not provide <semanticDescriptor> resources, it is possible for another application to provide the <semanticDescriptor> resources. This can support application development that continually expands its supported devices. In this clothes-washing machine use case the client application can deploy with any of the three models described above, but as the application developer becomes aware of other clothes-washing machines, they can create the <semanticDescriptor> resources for those devices and then applications that have been developed to use the original deployed devices will be interoperable with these new devices, without change to the application. This concept is one of the ways that oneM2M breaks down the silos of vertical deployments.

8.3.2 Create <semanticDescriptor>

The <semanticDescriptor> resource can be created by the entity that is creating the model or by a separate client entity, depending on the <accessControlPolicies> of the parent resource. This example shows the create <semanticDescriptor> for the Custom Model. The RDF triples used to semantically describe the washing machine were first converted to RDF/XML and then base64 encoded. The result of that encoding is used for the "dsp" attribute of the <semanticDescriptor>.

The following code is written using python and uses the utilities available at [i.4] to implement the CREATE and RETRIEVE functions. Other classes, such as Graph, smdxml, and smdEncode are defined in clause 8.2.

```
prefixes_io = '''@prefix saref: <https://saref.etsi.org/core/>.
@prefix s4bldg: <https://saref.etsi.org/saref4bldg/>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix sn: <http://www.XYZ.com/WashingMachines#XYZ_Cool/>.
@prefix m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>.
'''

smdBase = '''sn:WASH_XYZ_RESOURCE_ID a <http://www.XYZ.com/WashingMachines#XYZ_Cool>;
    rdfs:comment "Very cool Washing Machine";
    saref:hasFunction sn:WASH_XYZ-MonitoringFunction, sn:WASH_XYZ-StartStopFunction;
    saref:hasManufacturer  "XYZ";
    saref:hasService       sn:WASH_XYZ-MonitorService , sn:WASH_XYZ-SwitchOnService;
    saref:hasState         sn:WASH_XYZ-WashingMachineStatus;
    s4bldg:isContainedIn   sn:My_Bathroom;
    m2m:oneM2MTargetURI    "RESOURCE_ID";
    m2m:hasOperation sn:WASH_XYZ-SwitchOnService_RESOURCE_ID, sn:WASH_XYZ-StartStopFunction-
ON_Command_RESOURCE_ID, sn:WASH_XYZ-StartStopFunction-OFF_Command_RESOURCE_ID, sn:WASH_XYZ-
StartStopFunction-TOGGLE_Command_RESOURCE_ID,
                sn:WASH_XYZ-MonitoringFunction-WashingMachineStatus_RESOURCE_ID.
'''

smdfull = prefixes_io + smdBase
g = Graph().parse(data=smdfull, format='n3')
smdxml = g.serialize(format='xml', indent=4)

targetURI = 'myWashingMachine'
payload = smdxml.replace("RESOURCE_ID", targetURI)
smd2b64 = smdEncode(payload)

CREATE (
    'http://localhost:50000/oneM2M-semantics/' + targetURI,                   # This is the
Custom Washing Machine AE

    # Request Headers
    {
        'X-M2M-Origin' : originator1,               # Set the originator
        'X-M2M-RI'       : '123',                   # Request identifier
        'X-M2M-RVI'    : '3',                       # Release verson indicator
        'Accept'              : 'application/json', # Response shall be JSON
        'Content-Type'   : 'application/json;ty=24' # Content is JSON, and represents an
<semanticDescriptor> resource
    },

    # Request Body
    {
        'm2m:smd': {
            'rn':'smdCustomWasher',
            'dcrp':'application/rdf+xml:1',       # the RDF triples use RDF/XML format;
            'dsp': smd2b64                               # the base64 encode triples
        }
    }
)
```

The resulting oneM2M primitive request and response using the HTTP protocol binding and JSON payload binding is shown below.

Sending request to "http://localhost:50000/oneM2M-semantics/myWashingMachine"

Header Field	Value
X-M2M-Origin	Cipe1
X-M2M-RI	123
X-M2M-RVI	3
Accept	application/json
Content-Type	application/json;ty=24

Body

```
{
    "m2m:smd": {
        "rn": "smdCustomWasher",
        "dcrp": "application/rdf+xml:1",
        "dsp":
"PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0idXRmLTgiPz4KPHJkZjpSREYKICAgeG1sbnM6bTJtPSJodHRwczovL2dpd
C5vbmVtMm0ub3JnL01BUy9CYXNlT250b2xvZ3kvcmF3L21hc3Rlci9iYXNlX29udG9sb2d5Lm93bCMiCiAgIHhtbG5zOnJkZj0
iaHR0cDovL3d3dy53My5vcmcvMTk5OS8wMi8yMi1yZGYtc3ludGF4LW5zIyIKICAgeG1sbnM6cmRmcz0iaHR0cDovL3d3dy53M
y5vcmcvMjAwMC8wMS9yZGYtc2NoZW1hIyIKICAgeG1sbnM6czRibGRnPSJodHRwczovL3NhcmVmLmV0c2kub3JnL3NhcmVmNGJ
sZGcvIgogICB4bWxuczpzYXJlZj0iaHR0cHM6Ly9zYXJlZi5ldHNpLm9yZy9jb3JlLyIKPgogIDxyZGY6RGVzY3JpcHRpb24gc
mRmOmFib3V0PSJodHRwOi8vd3d3LlhZWi5jb20vV2FzaGluZ01hY2hpbmVzI1hZWl9Db29sV0FTSF9YWVpfbXlXYXNoaW5nTWF
jaGluZSI+CiAgICA8cmRmOnR5cGUgcmRmOnJlc291cmNlPSJodHRwOi8vd3d3LlhZWi5jb20vV2FzaGluZ01hY2hpbmVzI1hZW
l9Db29sIi8+CiAgICA8cmRmczpjb21tZW50PlZlcnkgY29vbCBXYXNoaW5nIE1hY2hpbmU8L3JkZnM6Y29tbWVudD4KICAgIDx
zYXJlZjpoYXNGdW5jdGlvbiByZGY6cmVzb3VyY2U9Imh0dHA6Ly93d3cuWFlaLmNvbS9XYXNoaW5nTWFjaGluZXMjWFlaX0Nvb
2xXQVNIX1hZWi1Nb25pdG9yaW5nRnVuY3Rpb24iLz4KICAgIDxzYXJlZjpoYXNGdW5jdGlvbiByZGY6cmVzb3VyY2U9Imh0dHA
6Ly93d3cuWFlaLmNvbS9XYXNoaW5nTWFjaGluZXMjWFlaX0Nvb2xXQVNIX1hZWi1TdGFydFN0b3BGdW5jdGlvbiIvPgogICAgP
HNhcmVmOmhhc01hbnVmYWN0dXJlcj5YWVo8L3NhcmVmOmhhc01hbnVmYWN0dXJlcj4KICAgIDxzYXJlZjpoYXNTZXJ2aWNlIHJ
kZjpyZXNvdXJjZT0iaHR0cDovL3d3dy5YWVouY29tL1dhc2hpbmdNYWNoaW5lcyNYWVpfQ29vbFdBU0hfWFlaLU1vbml0b3JTZ
XJ2aWNlIi8+CiAgICA8c2FyZWY6aGFzU2VydmljZSByZGY6cmVzb3VyY2U9Imh0dHA6Ly93d3cuWFlaLmNvbS9XYXNoaW5nTWF
jaGluZXMjWFlaX0Nvb2xXQVNIX1hZWi1Td2l0Y2hPblNlcnZpY2UiLz4KICAgIDxzYXJlZjpoYXNTdGF0ZSByZGY6cmVzb3VyY
2U9Imh0dHA6Ly93d3cuWFlaLmNvbS9XYXNoaW5nTWFjaGluZXMjWFlaX0Nvb2xXQVNIX1hZWi1XYXNoaW5nTWFjaGluZVN0YXR
1cyIvPgogICAgPHM0YmxkZzppc0NvbnRhaW5lZEluIHJkZjpyZXNvdXJjZT0iaHR0cDovL3d3dy5YWVouY29tL1dhc2hpbmdNY
WNoaW5lcyNYWVpfQ29vbE15X0JhdGhyb29tIi8+CiAgICA8bTJtOm9uZU0yTVRhcmdldFVSST5teVdhc2hpbmdNYWNoaW5lPC9
tMm06b25lTTJNVGFyZ2V0VVJJPgogICAgPG0ybTpoYXNPcGVyYXRpb24gcmRmOnJlc291cmNlPSJodHRwOi8vd3d3LlhZWi5jb
20vV2FzaGluZ01hY2hpbmVzI1hZWl9Db29sV0FTSF9YWVotU3dpdGNoT25TZXJ2aWNlX215V2FzaGluZ01hY2hpbmUiLz4KICA
gIDxtMm06aGFzT3BlcmF0aW9uIHJkZjpyZXNvdXJjZT0iaHR0cDovL3d3dy5YWVouY29tL1dhc2hpbmdNYWNoaW5lcyNYWVpfQ
29vbFdBU0hfWFlaLVN0YXJ0U3RvcEZ1bmN0aW9uLU9OX0NvbW1hbmRfbXlXYXNoaW5nTWFjaGluZSIvPgogICAgPG0ybTpoYXN
PcGVyYXRpb24gcmRmOnJlc291cmNlPSJodHRwOi8vd3d3LlhZWi5jb20vV2FzaGluZ01hY2hpbmVzI1hZWl9Db29sV0FTSF9YW
VotU3RhcnRTdG9wRnVuY3Rpb24tT0ZGX0NvbW1hbmRfbXlXYXNoaW5nTWFjaGluZSIvPgogICAgPG0ybTpoYXNPcGVyYXRpb24
gcmRmOnJlc291cmNlPSJodHRwOi8vd3d3LlhZWi5jb20vV2FzaGluZ01hY2hpbmVzI1hZWl9Db29sV0FTSF9YWVotU3RhcnRTd
G9wRnVuY3Rpb24tVE9HR0xFX0NvbW1hbmRfbXlXYXNoaW5nTWFjaGluZSIvPgogICAgPG0ybTpoYXNPcGVyYXRpb24gcmRmOnJ
lc291cmNlPSJodHRwOi8vd3d3LlhZWi5jb20vV2FzaGluZ01hY2hpbmVzI1hZWl9Db29sV0FTSF9YWVotTW9uaXRvcmluZ0Z1b
mN0aW9uLVdhc2hpbmdNYWNoaW5lU3RhdHVzX215V2FzaGluZ01hY2hpbmUiLz4KICA8L3JkZjpEZXNjcmlwdGlvbj4KPC9yZGY
6UkRGPgo="
    }
}
```

Response

Header Field	Value
Content-Type	application/vnd.onem2m-res+json
X-M2M-RI	123
X-M2M-RSC	**2001**
Content-Location	/cse_01/smd165779801715710114cse01
Content-Length	2952
X-M2M-RVI	3

Body

```
{
    "m2m:smd": {
        "ct": "20220714T112657",
        "et": "99991231T235959",
        "lt": "20220714T112657",
        "pi": "Cipe1",
        "ri": "smd165779801715710114cse01"
           }
}
```

The code above is repeated for each <semanticDescriptor> resource that is created. For this use case, there are six <semanticdescriptor> resources created. Three <semanticDescriptor> resources that describe the capabilities of the washing machine are identical except for the "RESOURCE_ID" token that is replaced with the appropriate value. The other three <semanticDescriptor> resources describe the API of the model and therefore have different values for the oneM2M baseOntology classes.

8.3.3 Semantic Query

When a SPARQL query is created, it can be passed in the "semanticFilter" request parameter, shown below with the shortname form of the request parameter of "smf". The query shall first be ascii encoded. Using python, the following code will execute query 4 from above to dynamically determine all four primitives needed to use the SDT model of the washing machine. Additionally, the "Semantic Query Indicator", "sqi", is set to "1" to distinguish this query request from a semantic discovery request. A semantic discovery request will return a list of URIs that match the query rather than a query response.

```
query = '''PREFIX sn:<http://www.XYZ.com/WashingMachines#>
PREFIX m2m: <https://git.onem2m.org/MAS/BaseOntology/raw/master/base_ontology.owl#>
PREFIX saref: <https://saref.etsi.org/core/>

SELECT ?sarefCommand ?method ?targetURI ?attr ?res
WHERE {
    ?wm m2m:hasOperation ?operation.
    ?operation a m2m:Operation.
    ?operation m2m:oneM2MMethod ?method.
    optional {?operation m2m:hasDataRestriction ?res}.
    optional {?operation m2m:oneM2Mattribute ?attr}.
    ?operation a ?sarefCommand.
    ?operation m2m:oneM2MTargetURI ?targetURI.
    VALUES ?wm {<http://www.XYZ.com/WashingMachines#XYZ_CoolWASH_XYZ_deviceClothesWasher>}.
    VALUES ?sarefCommand {saref:GetCommand saref:OnCommand saref:OffCommand saref:ToggleCommand}

}
ORDER BY ?sarefCommand
'''
encSMQ = encodedSparqlQuery(query)

RETRIEVE (
    'http://localhost:50000/oneM2M-semantics?fu=1&sqi=1&smf='+ encSMQ,

    # Request Headers
    {
        'X-M2M-Origin' : originator2,                 # Set the originator
        'X-M2M-RI'       : 'semQ1',                     # Unique request identifier
        'X-M2M-RVI'    : '3',                               # Release verson indicator
        'Accept'              : 'application/json'        # Response shall be JSON
    }
)
```

The resulting oneM2M primitive request and response using the HTTP protocol binding and JSON payload binding is shown below. It is important to note the JSON response to the query. An application will have to parse the response from the query to get the desired information.

```
Sending request to:
http://localhost:50000/oneM2M-semantics?fu=1&sqi=1&smf=PREFIX
sn%3A<http%3A%2F%2Fwww.XYZ.com%2FWashingMachines%23> PREFIX m2m%3A
<https%3A%2F%2Fgit.onem2m.org%2FMAS%2FBaseOntology%2Fraw%2Fmaster%2Fbase_ontology.owl%23> PREFIX
saref%3A <https%3A%2F%2Fsaref.etsi.org%2Fcore%2F> SELECT %3FsarefCommand %3Fmethod %3FtargetURI
%3Fattr %3Fres WHERE { %3Fwm m2m%3AhasOperation %3Foperation . %3Foperation a m2m%3AOperation .
%3Foperation m2m%3AoneM2MMethod %3Fmethod . optional {%3Foperation m2m%3AhasDataRestriction
%3Fres} . optional {%3Foperation m2m%3AoneM2Mattribute %3Fattr} . %3Foperation a %3FsarefCommand .
%3Foperation m2m%3AoneM2MTargetURI %3FtargetURI . VALUES %3Fwm
{<http%3A%2F%2Fwww.XYZ.com%2FWashingMachines%23XYZ_CoolWASH_XYZ_deviceClothesWasher>} . VALUES
%3FsarefCommand {saref%3AGetCommand saref%3AOnCommand saref%3AOffCommand saref%3AToggleCommand} }
ORDER BY %3FsarefCommand
```

Headers

Header Field	Value
X-M2M-Origin	Cipe1
X-M2M-RI	semQ1
X-M2M-RVI	3
Accept	application/json

Response

Header Field	Value
Content-Type	application/vnd.onem2m-res+json
X-M2M-RI	semQ1
X-M2M-RSC	2000
Content-Location	/cse_01/smd1657798017157101114cse01
Content-Length	1043
X-M2M-RVI	3
X-M2M-CTS	2

Body

```
[
    {
        "attr": {
            "type": "literal",
            "value": "currentMachineState"
        },
        "method": {
            "type": "literal",
            "value": "RETRIEVE"
        },
        "sarefCommand": {
            "type": "uri",
            "value": "https://saref.etsi.org/core/GetCommand"
        },
        "targetURI": {
            "type": "literal",
            "value": "/deviceClothesWasher/runState"
        }
    },
    {
        "attr": {
            "type": "literal",
            "value": "powerState"
        },
        "method": {
            "type": "literal",
            "value": "UPDATE"
        },
        "res": {
            "type": "literal",
            "value": "False"
        },
        "sarefCommand": {
            "type": "uri",
            "value": "https://saref.etsi.org/core/OffCommand"
        },
        "targetURI": {
            "type": "literal",
            "value": "/deviceClothesWasher/binarySwitch"
        }
    },
    {
        "attr": {
            "type": "literal",
            "value": "powerState"
        },
        "method": {
            "type": "literal",
            "value": "UPDATE"
        },
        "res": {
            "type": "literal",
            "value": "True"
        },
        "sarefCommand": {
            "type": "uri",
            "value": "https://saref.etsi.org/core/OnCommand"
        },
        "targetURI": {
            "type": "literal",
            "value": "/deviceClothesWasher/binarySwitch"
        }
    },
    {
        "method": {
            "type": "literal",
            "value": "UPDATE"
        },
        "sarefCommand": {
            "type": "uri",
            "value": "https://saref.etsi.org/core/ToggleCommand"
        },
        "targetURI": {
            "type": "literal",
            "value": "/deviceClothesWasher/binarySwitch/toggle"
        }
    }
]
```

9 Conclusion

The Internet of Things has led to the requirement to model physical devices. In oneM2M there are three methods available to build a model of a device:

- Create a custom model using the variety of data sharing resources available in oneM2M to represent the information and services that are exposed by a real-world device.
- Create an ontology-based model, where the first step is to semantically describe the device in a manner that can then be procedurally (or programmatically) used to create the resources needed to model the device.
- Create a Smart Device Template based model by selecting a standardized pre-existing model that is best aligned to the device.

The present document presented a use case where each of the three methods were used to model a clothes-washing machine.

The goal of the present document is to show how interoperability can be realized for applications that are working with physical devices but do not have a priori knowledge of how the device is modelled. The use case showed how the oneM2M primitives for the same device operation are different depending on which method is used to model the device. The different models would require different custom application programming to support all three models. Additionally, the custom model and the ontology-based model can result in many different variations. This leads to an unmanageable scalability problem for application developers as they attempt to control the clothes-washing machine with any number of model variations.

oneM2M addressed this interoperability beginning in Release 2 with the ability to semantically describe the device model structure using the oneM2M Base Ontology. Combining the ability to describe the model structure with the use of SAREF ontologies, and other foreign ontologies, oneM2M provides the ability to dynamically discover and use devices that are modelled differently.

Annex A (informative): Bibliography

- oneM2M Drafting Rules.

NOTE: Available at: http://www.onem2m.org/images/files/oneM2M-Drafting-Rules.pdf.

- ETSI TS 118 111: "oneM2M; Common Terminology (oneM2M TS-0011)".
- ETSI TR 118 525: "oneM2M; Application Developer Guide".
- ETSI TS 118 101: "oneM2M; Functional Architecture (oneM2M TS-0001)".
- ETSI TS 118 109: "oneM2M; HTTP Protocol Binding (oneM2M TS-0009)".
- ETSI TR 103 783: "SAREF: SDT interoperability and oneM2M base ontology alignment".

History

Document history		
V1.1.1	August 2022	Publication