POLITECNICO DI TORINO

Master's degree in Communication and Computer Networks Engineering

Politecnico di Torino

Master's degree thesis

Advanced Semantic Discovery and Query in oneM2M

Supervisors:
Prof. Claudio CASETTI
Prof. Luigi LIQUORI

Candidate:
Behrad SHIRMARD

July 2023

"Computers are incredibly fast, accurate, and stupid. Human beings are incredibly slow, inaccurate, and brilliant. Together they are powerful beyond imagination."

- Albert Einstein, physicist

POLITECNICO DI TORINO

Abstract

Department of Electronics and Telecommunications

Master's degree in Communication and Computer Networks Engineering

Advanced Semantic Discovery and Query in oneM2M

by Behrad SHIRMARD

Currently oneM2M in its service layer has native discovery capabilities which work properly only if the search is related to specific known sources of information (e.g. searching for the values of a known set of containers) or if the discovery is well scoped and designed (e.g. the lights in a house). When oneM2M is used to discover wide sets of data or unknown sets of data, the functionality is typically integrated by ad-hoc applications that are expanding the oneM2M functionality. This means that this core service layer function (discovery) may be implemented with different flavors and this is not optimal when it comes to interworking and interoperability.

The objective of the present research is the study, development, and simulation of Advanced Semantic Discovery and Query capabilities for oneM2M and its contribution to the oneM2M standard.

The main aim is to enable an easy and efficient discovery of information and a proper interworking with external source/consumers of information (e.g. a distributed data base in a smart city or in a firm), or directly to search information in the oneM2M system for big data purposes. Following Research is done in INRIA labs for ETSI standards.

Contents

List of Figures

List of Abbreviations

ACK	**ACK**nowledge message
ACP	**A**ccess **C**ontrol **P**olicy
AE	**A**pplication **E**ntity
API	**A**pplication **P**rogram **I**nterface
AQL	**A**rango **Q**uery **L**anguage
AS	**A**utonomous **S**ystem
ASD	**A**dvanced **S**emantic **D**iscovery
ASDQ	**A**dvanced **S**emantic **D**iscovery **Q**uery
ASDQL	**A**dvanced **S**emantic **D**iscovery **Q**uery **L**anguage
ASN	**A**pplication **S**ervice **N**ode
BGP	**B**order **G**ateway **P**rotocol
BGP4	**B**order **G**ateway **P**rotocol **4**
C2P	**C**ustomer-to-**P**rovider
CAIDA	**C**enter for **A**pplied **I**nternet **D**ata **A**nalysis
CBOR	**C**oncise **B**inary **O**bject **R**epresentation
CC	**C**reative **C**ommons
CMDH	**C**ommunication **M**anagement and **D**elivery **H**andling
CNF	**C**onjunctive **N**ormal **F**orm
CRUD	**C**reate **R**etrieve**U**pdate **D**elete
CSE	**C**ommon **S**ervices **E**ntity
CSF	**C**ommon **S**ervice **F**unction
CSV	**C**ertified **S**erver **V**alidation
DCL	**D**ata **C**ontrol **L**anguage
DDL	**D**ata **D**efinition **L**anguage
DIS	**Dis**covery
DNF	**D**isjunctive **N**ormal **F**orm
DNS	**D**omain **N**ame **S**ystem
drt	**D**esired identifier **R**esult
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute
FQDN	**F**ully **Q**ualified **D**omain **N**ame
GraphQL	**G**raph **Q**uery **L**anguage
HTTP	**H**ypertext **T**ransfer **P**rotocol
IN	**I**nfrastructure **N**ode
IN-CSE	**I**nfrastructure **N**ode - **C**ommon **S**ervices **E**ntity
IoT	**I**nternet **o**f **T**hings
IP	**I**nternet **P**rotocol
IRR	**I**nternet level **R**outing **R**egistries
ISP	**I**nternet **S**ervice **P**rovider
JSON	**J**avaScript **O**bject **N**otation
KETI	**K**orean **E**lectronics **T**echnology **I**nstitute
lbl	**l**a**b**e**l**
M2M	**M**achine-to-**M**achine

Mca	Reference Point for M2M Communication with AE
Mcc	Reference Point for M2M Communication with CSE
MN	**M**iddle **N**ode
MN-CSE	**M**iddle **N**ode - **C**ommon **S**ervices **E**ntity
NoSQL	**N**ot **o**nly **SQL**
OA	**O**ptional **A**nnounced
OLAP	**O**nline **A**nalytical **P**rocessing
OLTP	**O**nline **T**ransactional **P**rocessing
OSPF	**O**pen **S**hortest **P**ath **F**irst
OWL	**W**eb **O**ntology **L**anguage
P2P	**P**eer-to-**P**eer
PDP	**P**olicy **D**ecision **P**oint
PEP	**P**olicy **E**nforcement **P**oint
PHP	**H**ypertext **P**reprocessor (originally: "Personal Home Page Tools")
PIP	**P**olicy **I**nformation **P**oint
PPM	**P**rivacy **P**olicy **M**anager
PRP	**P**olicy **R**etrieval **P**oint
QoS	**Q**uality **o**f **S**ervice
RBAC	**R**ole **B**ased **A**ccess **C**ontrol
RC	**R**emote **C**ontrol
RDBMS	**R**elational **D**atabase **M**anagement **S**ystem
RDF	**R**esource **D**escription **F**ramework
REST	**R**epresentational **S**tate **T**ransfer
RS	**R**outing **R**ecommendation **S**ystem
S2S	**S**ibling-to-**S**ibling
SAREF	**S**mart **A**pplications **REF**erence ontology
SDA	**S**emantic **D**iscovery **A**greement
SDPR	**S**emantic **D**iscovery **R**outing **P**rotocol
SDQ	**S**emantic **D**iscovery **Q**uery
SDR	**S**emantic **D**iscovery **R**outing
SDRM	**S**emantic **D**iscovery **R**outing **M**echanism
SLA	**S**ervice **L**evel **A**greement
SP	**S**ervice **P**rovider
SPARQL	**S**imple **P**rotocol **A**nd **R**DF **Q**uery **L**anguage
SQL	**S**tructured **Q**uery **L**anguage
SQR	**S**emantic **Q**uery **R**esolution
SQRM	**S**emantic **R**esolution **Q**uery **M**echanism
SQRS	**S**emantic **Q**uery **R**esolution **S**ystem
SR	**S**emantic **R**ecommendation system
SRT	**S**emantic **R**outing **T**able
SSN	**S**emantic **S**ensor **N**etwork
TD	**T**hing **D**escription
TR	**T**echnical **R**eport
TTL	**T**ime **T**o **L**ive
UPM	**U**niversidad **P**olitecnica de **M**adrid
URI	**U**niform **R**esource **I**dentifier
W3C	**W**orld **W**ide **W**eb **C**onsortium
WoT	**W**eb **o**f **T**hings
XML	e**X**tensible **M**arkup **L**anguage

Chapter 1

Introduction to oneM2M the IoT Standard [1]

1.1 oneM2M Basics

oneM2M is similar to a distributed Operating System for the Internet of Things. It takes the form of a middleware service layer consisting of a suite of common service functions (CSFs). The middleware service layer sits between applications and connectivity transport. oneM2M's common service functions (CSFs) are exposed to applications and to IoT devices via RESTful APIs.

A oneM2M service layer and/or applications can reside in field devices and sensors, on gateways and in back-end or cloud applications. This supports cooperative intelligence in distributed IoT systems.

This section covers the following topics:

1. Service Layer
2. Functional Architecture
3. Common Service Functions (CSFs)
4. Data Management
5. Horizontal Platform

1.1.1 Service Layer

oneM2M's Service Layer is typically implemented as a software layer. It sits between IoT applications and processing or communication hardware corresponding to the connectivity layer. Data storage, processing and transport in the connectivity layer normally rides on top of IP. However, oneM2M also supports non-IP transports via interworking proxies.

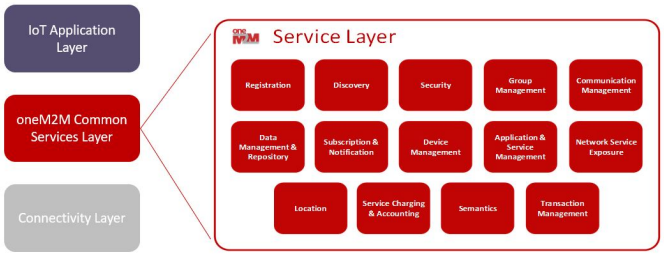

FIGURE 1.1: Service Layer in oneM2M.

The oneM2M Service Layer provides commonly needed functions for IoT applications. To date, oneM2M specifications cover fourteen such functions. Developers can use these functions progressively for their applications, beginning with the most frequently required ones such as device management, registration and security. More complex applications can incorporate features to support semantic interoperability and location services, for example.

oneM2M follows a modular standardisation roadmap. This allows for future IoT requirements and new common service functions.

1.1.2 Functional Architecture

oneM2M standards comprise a horizontal architecture in the form of a three-layer model comprising applications, middleware services and networks.

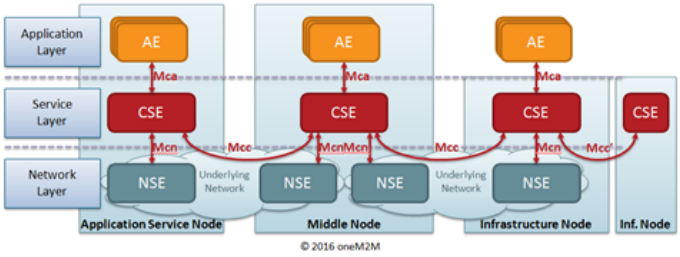

FIGURE 1.2: Horizontal Architecture of oneM2M

The oneM2M functional architecture comprises the following entities:

- Application Entity (AE): The Application Entity is an entity in the application layer that implements application service logic. Examples of AEs include an instance of a fleet tracking application, a remote blood sugar measuring application, a power metering application or a pump controlling application. Each application service logic can be resident in a number of nodes and/or more than once on a single node. Each execution instance of an application service logic is termed an "Application Entity" (AE) and is identified with a unique AE-ID.

- Common Services Entity (CSE): A Common Services Entity represents an instantiation of a set of "common service functions" of the oneM2M Service Layer. A CSE is actually the entity that contains the collection of oneM2M-specified common service functions that AEs are able to use. Examples of service functions offered by a CSE include: data storage and sharing with access control and authorization, event detection and notification, group communication, scheduling of data exchanges, device management, and location services. Each CSE is identified with a unique CSE-ID.

- Underlying Network Services Entity (NSE): A Network Services Entity provides services from the underlying network to the CSEs. Examples of such services include location services, device triggering, certain sleep modes like PSM in 3GPP based networks or long sleep cycles.

Application Entities (AEs) are hosted on nodes (e.g., enterprise server, device). These nodes may be virtualized or physical instances. AEs communicate with each other by sending requests to a Common Service Entity (CSE) that, in turn routes the request to the

target AE while providing services based on the request. CSEs are hosted on virtualized or physical nodes. An AE and a CSE can share the same node (e.g., device).

It is possible to enable bi-directional communications between oneM2M and non-oneM2M systems via an IPE (Interworking Proxy Entity). This is a specialized AE (Application Entity) that allows the oneM2M system to interact with any non-oneM2M system, in a seamless way, through the Mca interface. It relies on the capability to remap non-oneM2M data models to oneM2M resources (<AE>, <container>, <flexContainer>, etc.).

oneM2M Nodes

oneM2M has defined a set of Nodes that are logical entities identifiable in the oneM2M System. oneM2M Nodes typically contain CSEs and/or AEs. For the definition of Node types, oneM2M distinguishes between Nodes in the "Field Domain" – i.e. the domain in which sensors / actuators / aggregators / gateways are deployed – and the "Infrastructure Domain" – i.e. the domain in which servers and applications on larger computers reside.

Nodes can be of the following types:

- Application Service Node (ASN): a Node that contains one CSE and contains at least one Application Entity (AE), located in the Field Domain. An ASN could be implemented on a range of different devices ranging from resource constrained devices up to more powerful hardware. Examples of devices that could be represented by ASNs include data collection devices, more capable sensors and actuators including simple server functions.

- Application Dedicated Node (ADN): a Node that contains at least one AE and does not contain a CSE. It is located in the Field Domain. An ADN would typically be implemented on a resource constrained device that may not have access to ample storage or processing resources and – therefore – may be limited to only host a oneM2M AE and not a CSE. Examples for devices that could be represented by ADNs include simple sensor or actuator devices.

- Middle Node (MN): a Node that contains one CSE and could also contain AEs. MNs are located in the Field Domain. There could be several MNs in the Field Domain of the oneM2M System. Typically an MN would reside in an M2M Gateway. MNs would be used to establish a logical tree structure of oneM2M nodes, e.g. to hierarchically aggregate data of buildings / neighborhoods / cities / counties / states etc.

- Infrastructure Node (IN): a Node that contains one CSE and could also contain AEs. There is exactly one IN in the Infrastructure Domain per oneM2M Service Provider.

- Non-oneM2M Node (NoDN): A Node that does not contain oneM2M Entities (neither AEs nor CSEs). Typically such Nodes would host some non-oneM2M IoT implementations or legacy technology which can be connected to the oneM2M system via interworking proxies.

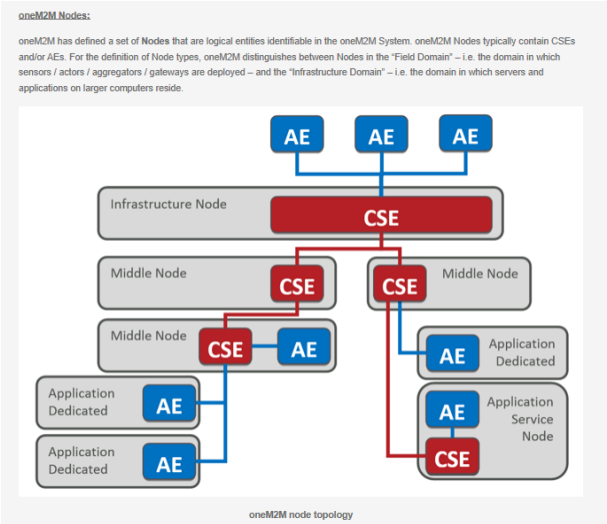

FIGURE 1.3: oneM2M node topology.

oneM2M Reference Points

The oneM2M functional architecture defines the following reference points:

- Mca: Reference point for the communication flows between an Application Entity (AE) and a Common Services Entity (CSE). These flows enable the AE to use the services supported by the CSE, and for the CSE to communicate with the AE. The AE and the CSE may or may not be co-located within the same Node.
- Mcc: Reference point for the communication flows between two Common Services Entities (CSEs). These flows enable a CSE to use the services supported by another CSE.
- Mcn: Reference point for the communication flows between a Common Services Entity (CSE) and the Network Services Entity (NSE). These flows enable a CSE to use the supported services provided by the NSE. While the oneM2M Service Layer is, usually independent of the underlying network – as long as it supports IP transport – it leverages specific M2M/IoT optimization such as 3GPP's eMTC features (e.g. device triggering, power saving mode, long sleep cycles, etc).
- Mcc': Reference point for the communication flows between two Common Services Entities (CSEs) in Infrastructure Nodes (IN) that are oneM2M compliant and that reside in different M2M Service Provider domains.

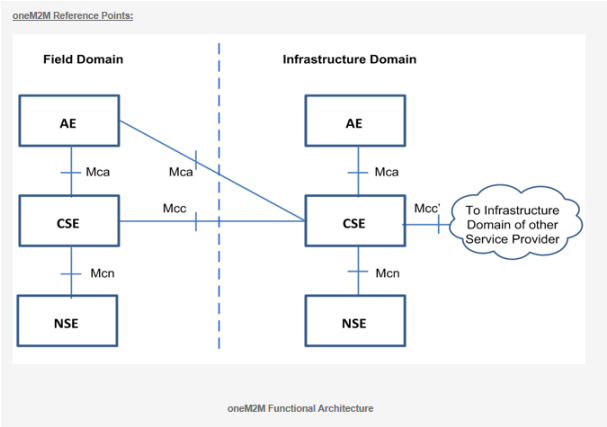

FIGURE 1.4: oneM2M functional architecture.

1.1.3 Common Service Functions (CSFs)

oneM2M defines a set of common service functions that apply to all the IoT domains. Think of these functions as items in a large toolbox to solve a number of IoT problems across many different domains. For example, a screwdriver can be used to fasten screws in a car as well as in a plane. In the same way, oneM2M CSFs, such as device management or security, are applicable to different IoT use cases and across different industry domains.

oneM2M members analysed a large number of IoT use cases to identify a set of common requirements in the initial phase of standardisation. This process resulted in the design of a set of Common Service Functions. Furthermore, oneM2M standardized how these functions are executed, i.e. it defined uniform APIs to access these functions.

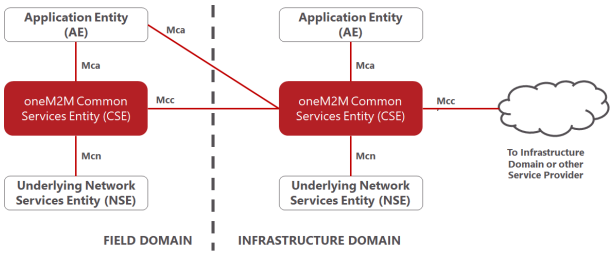

FIGURE 1.5: Reference points in oneM2M.

CSFs are general purpose services and are not specific to any IoT domain in particular. This enables each domain to build on top of this service layer and focus on its specific industrial needs. This is similar to functions of a generic operating system (OS) exposed to applications running on that OS. For instance, many applications read and write to files. File I/O is typically provided by the OS. oneM2M's Service Layer provides similar functions in a generic way to many different IoT Applications.

The common services functions reside within a CSE They provide services to the AEs via the Mca reference point and to other CSEs via the Mcc reference point.

1.1.4 Data Management

Within oneM2M, data storage relies on a Container type of resource. A Container is an entity that is associated with an object within an IoT system. This could be a specific sensor or a specific location, such as a room in a building, for example.

Containers are defined by the publisher of the data. This requires a function to discover and understand the data that an application wishes to consume. The scope of data management within oneM2M encompasses a set of CSFs that allow developers to manage Access Control Policies, Subscription & Notification functions and Discovery.

1.1.5 Horizontal Platform

The practice of building single-purpose and "vertical" domain applications leads to isolated silos. This makes it difficult to exchange data and to address cross-domain use cases. In contrast, a "horizontal" architecture allows the provision of a seamless interaction between applications and devices, even across silos and "verticals".

Using the example of a smart building use case, a security application can detect when nobody is in the building. It could then trigger lights to be switched off and for the air conditioning system to operate on a reduced setting.

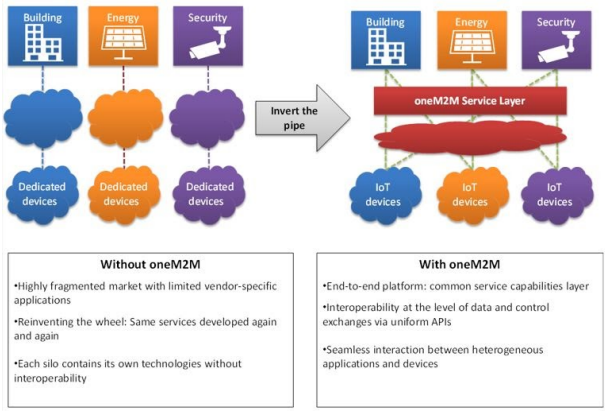

FIGURE 1.6: Horizontal Platforms.

1.2 oneM2M API

The oneM2M API is used by CSEs and AEs to communicate with one another as well as for data retrieval services. oneM2M REST APIs handle CRUD+N (Create, Retrieve, Update, Delete and Notification) operations.

The oneM2M API includes the following components:

- Primitives
- Resources + Attributes
- Data Types
- Protocol Bindings
- Procedures (CRUD+N).

Primitives are used to perform CRUD+N operations on resources hosted by CSEs or to send notifications to AEs. Each CRUD+N operation consists of a pair of Request and Response primitives.

Communication can originate from an AE or a CSE depending on the operation. Communication occurs via the exchange of oneM2M primitives across three reference points (Mca/Mcc/Mcc') defined in the oneM2M standard. Access and manipulation of the resources is subject to access control privileges.

1.2.1 oneM2M REST Architecture

Representational State Transfer (REST) is a software architectural style that defines a set of constraints to be used for creating web services. As REST is an architecture style, it can be mapped to multiple protocols such as HTTP, CoAP, etc.

RESTful services allow requesting systems to access and manipulate textual representations of resources by using a uniform and predefined set of stateless operations. A stateless protocol operation does not require the server to retain session information or status about each communicating partner for the duration of multiple requests.

REST is not a protocol. It is about manipulating resources, uniquely identified by URIs. A resource is stateful and may contain links pointing to another resource. All the actions on resources are done through a Uniform Interface.

Six guiding constraints define a RESTful system. These constraints restrict the ways in which a server can process and respond to client requests:

- Client-server: separation of concerns is the principle behind the client-server constraints.
- Stateless server: request from client to server contains all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.
- Cache: the client can reuse response data, sent by the server, by storing it in a local cache.
- Layered system: allows an architecture to be composed of hierarchical layers. It enables the addition of features like a gateway, a load balancer, or a firewall to accommodate system scaling.
- Code-on-demand: (optional) REST allows client functionality to be extended by downloading and executing code in the form of scripts (e.g. JavaScript).
- Uniform interface
  - dentification of resources: resource identifier enables the identification of the particular resource involved in an interaction between components.
  - Manipulation of resources through representations: resource representations are the state of a resource that is transferred between components.
  - Self-descriptive messages: contain metadata to describe the meaning of the message.
  - Hypermedia as the engine of application state (HATEOAS): Clients find their way through the API by following links available in the resource representations

1.2.2 oneM2M Security Mechanisms

oneM2M defines a security framework for IoT deployments consisting of the following security centric capabilities:

- Enrolment
- Aecurity association establishment
- Authorization
- End-to-end security
- Privacy management

Enrolment

IoT applications or devices go through an initialization or joining process to connect securely to a system and become trusted. This requires provisioning and configuration of identifiers and security credentials. oneM2M supports the capability to remotely bootstrap and provision IoT devices and applications with the necessary identifiers and credentials. The oneM2M entity responsible for enrolment is called the M2M Enrolment Function (MEF).

Security association establishment

To services offered by the oneM2M Service Layer securely, devices/applications need to be mutually authenticated and establish what is known as a oneM2M security association with the Service Layer. oneM2M supports security associations based on pairwise symmetric keys, public key certificates and mutual third-party authentication. The security association results in a TLS or DTLS session being established between the oneM2M entities.

Authorization

oneM2M supports several options for authorizing the access to services and information stored within the Service Layer. The owner of the service or information can define policy rules to grant access to other oneM2M entities. Many possibilities are defined for the access rules by the Service Layer: static through an access list, dynamic with token generation, or distributed authorization system.

End-to-end security

Besides having security associations between oneM2M entities, the standard makes it possible to have end-to-end secure communication between source and destination endpoints. Hence, oneM2M supports the capability to end-to-end protect the confidentiality and integrity of oneM2M messages that flow through the Service Layer via intermediate oneM2M nodes.

Privacy management

oneM2M supports user privacy protection via a personal data management framework which converts a user's privacy preferences into access control information that protects a user's Personally Identifiable Information. oneM2M offers users the possibility to set up their privacy preferences.

1.2.3 oneM2M Primitives

Primitives are service layer messages transmitted over the Mca/Mcc/Mcc' reference points between Originators and Receivers. An Originator and Receiver can be an AE or a CSE. Each CRUD+N operation consists of one request and one response primitive.

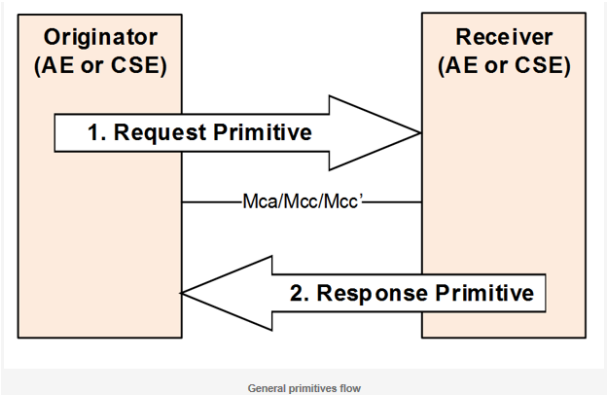

FIGURE 1.7: General primitive flow.

Primitives are binded to underlying transport layer protocols such as HTTP, CoAP , MQTT or WebSocket. Primitives are generic with respect to underlying network transport protocols. The binding for each primitive can be to zero or more messages in the transport layer.

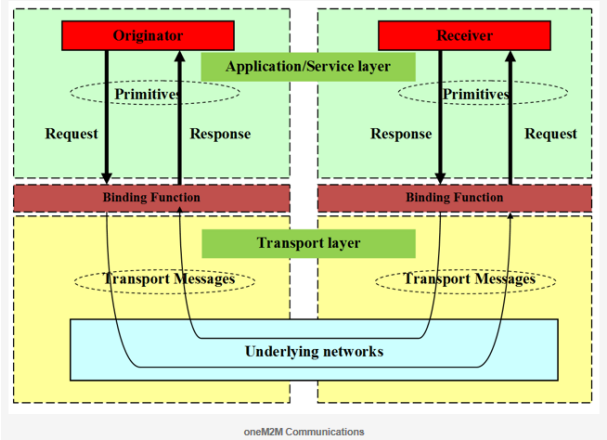

FIGURE 1.8: oneM2M communications.

Primitive structure

A primitive consists of two parts; control and content.

- **The control part** contains parameters required for the processing of the primitive itself (e.g. request or response parameters).
- **The content part** is optional based on the type of primitive and contains the representation of the resource consisting of all or a subset of the resource attributes.

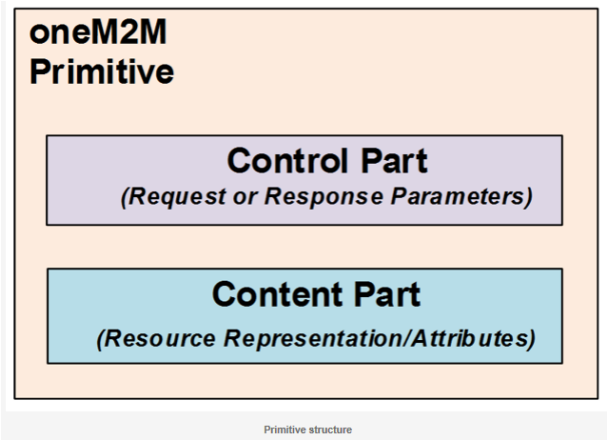

FIGURE 1.9: Primitive structure.

Primitives are encoded and serialized based on the particular oneM2M protocol binding being used. The originator and receiver of each primitive use the same binding, and thus use compatible forms of encoding/ decoding and serialization/de-serialization. During transfer, the control part is encoded based on the protocol binding being used and the content portion is serialized using XML, JSON or CBOR.

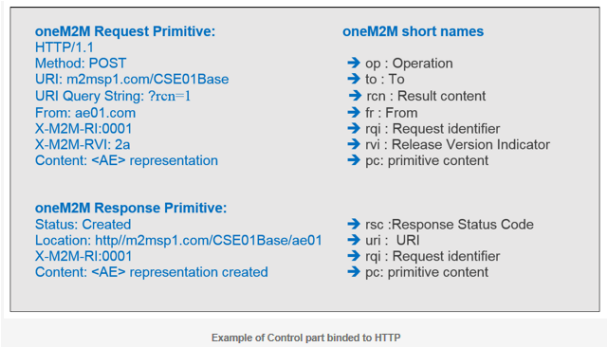

FIGURE 1.10: Example of control part bound to HTTP.

The Content part of a primitive contains a serialized representation of a resource. Here is an example of a oneM2M <container>resource representation in JSON format.

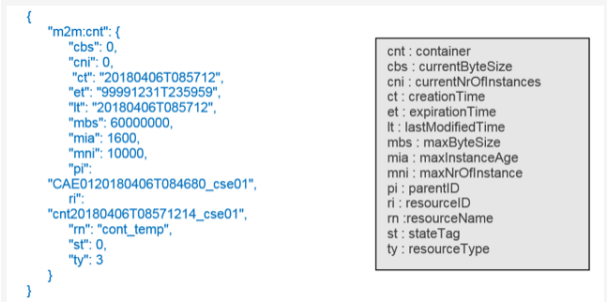

FIGURE 1.11: oneM2M <container>resource representation in JSON format.

Here is an example of a oneM2M<container>resource representation in XML format.

```
<?xml version="1.0" encoding="UTF-8"?>
<m2m:cnt xmlns:m2m="http://www.onem2m.org/xml/protocols" rn="cont_temp">
   <ty>3</ty>
   <ri>server/cnt-2951972863155866584</ri>
   <pi>server</pi>
   <ct>20181114T145000</ct>
   <lt>20181114T145000</lt>
   <et>20181114T145000</et>
   <st>0</st>
   <mni>10000</mni>
   <mbs>0</mbs>
   <mia>0</mia>
   <cni>0</cni>
   <cbs>0</cbs>
</m2m:cnt>
```

FIGURE 1.12: oneM2M<container>resource representation in XML format.

1.2.4 oneM2M Resources

This section describes the parameters that define oneM2M Resources and how they are used.

Resource Template

All information in the oneM2M System corresponding to AEs, CSEs, application data representing sensors, commands, etc. is represented as resources in the CSE. Each resource has its own specific type. Each resource type has a defined set of mandatory and optional attributes as well as child resources.

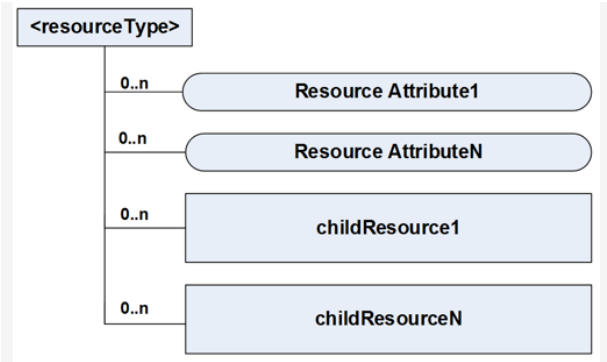

FIGURE 1.13: Resource types and its attributes.

Each resource is addressable and can be the target of CRUD operations specified in oneM2M primitives.

Resource Structure

The root of the oneM2M resource structure is <CSEBase>. The <CSEBase>resource is assigned an absolute address. All other child resources are addressed relative to <CSEBase>. Depending on the type of child resource it is instantiated 0..n times.

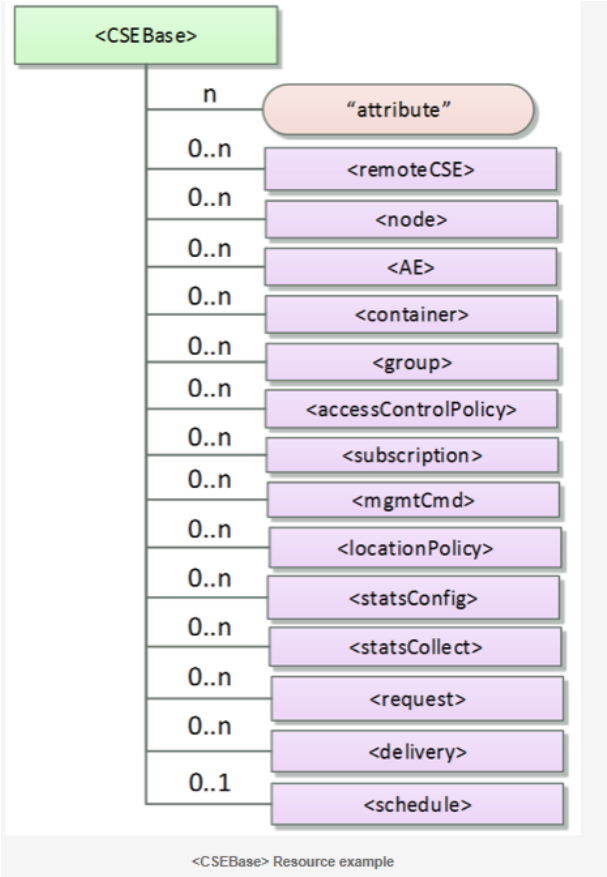

FIGURE 1.14: CSEBase resource example.

Resource Attributes

Each resource contains attributes that store information pertaining to the resource itself. The attributes are:

- Universal Attributes: which appear in all resources
- Common Attributes: which appear in more than one resource and have the same meaning whenever they do appear.
- Resource-specific attributes

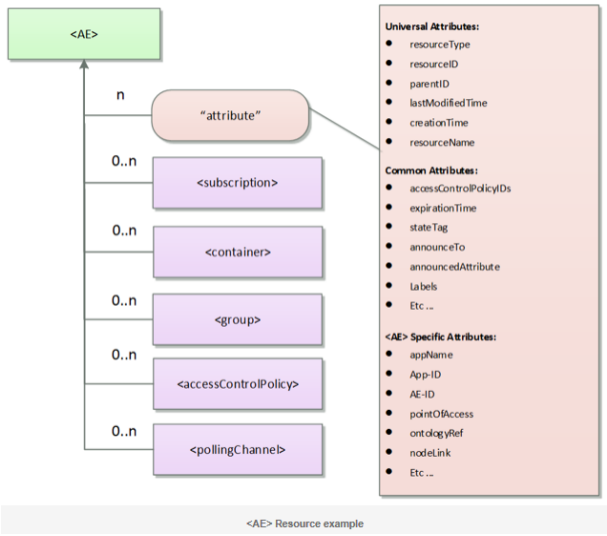

FIGURE 1.15: AE resource example.

Resource Schema

oneM2M defines XML, JSON and CBOR schemas which define the attributes of each resource type. Schemas bind oneM2M attributes to well-known data types defined by XML Schema definitions (e.g. xs:string, xs:anyURI, etc ...). Schemas also bind oneM2M attributes to oneM2M defined data types (e.g. m2m:id, m2m:stringList, etc ...).

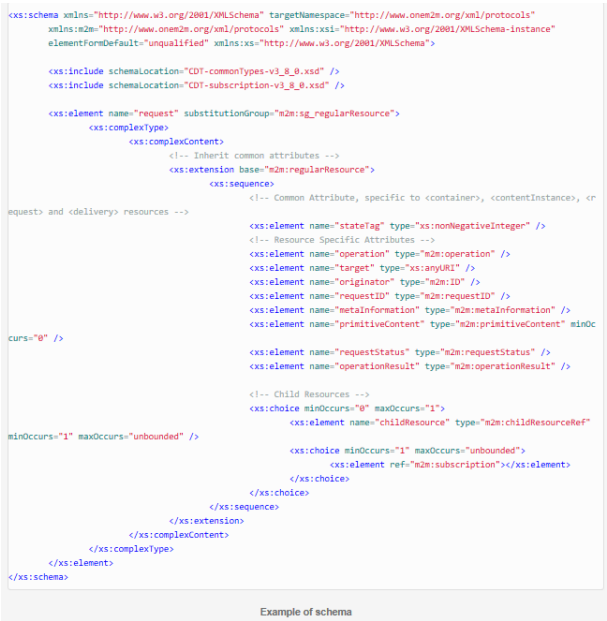

FIGURE 1.16: Example of resource schema.

1.2.5 oneM2M Procedures

oneM2M defines a set of procedures for developers to access resources in local and remote CSEs. Developers can then execute CREATE, RETRIEVE, UPDATE, DELETE and NOTIFY operations.

Access Resources in Local CSE

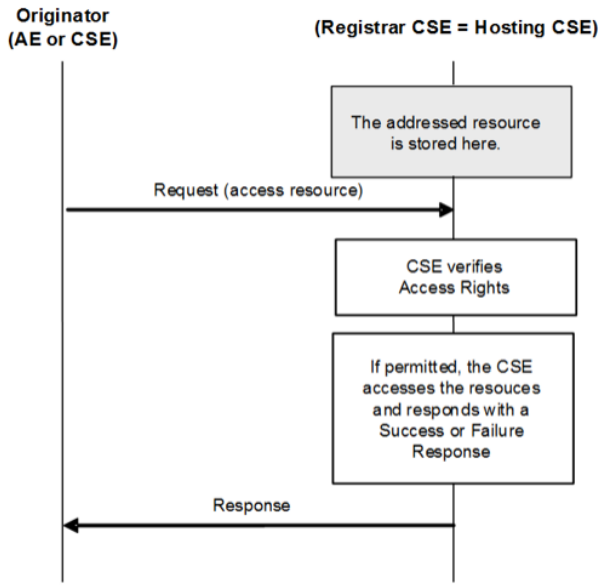

FIGURE 1.17: Access Resources in Local CSE.

Access Resources in Remote CSE

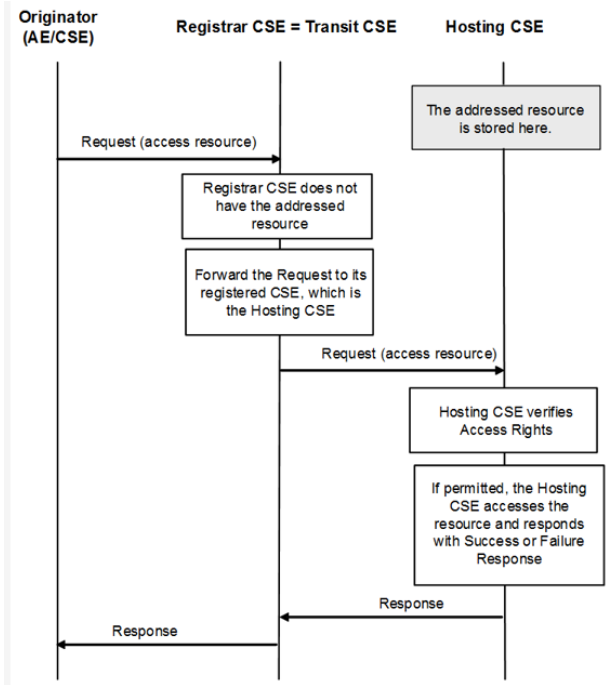

FIGURE 1.18: Access Resources in Remote CSE.

CREATE operation

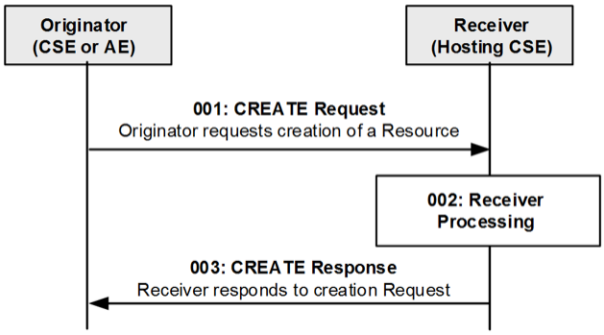

FIGURE 1.19: CREATE operation.

RETRIEVE Operation

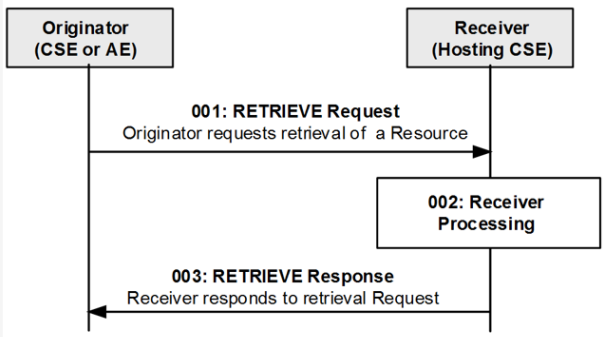

FIGURE 1.20: RETRIEVE Operation.

UPDATE Operation

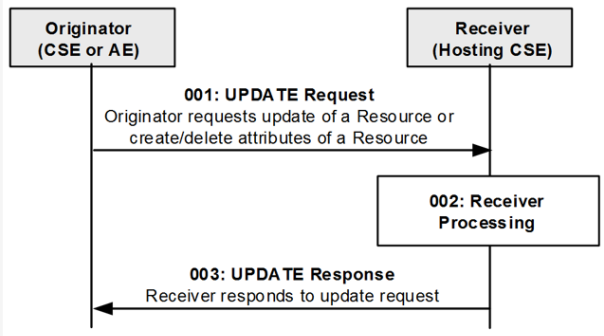

FIGURE 1.21: UPDATE Operation.

DELETE Operation

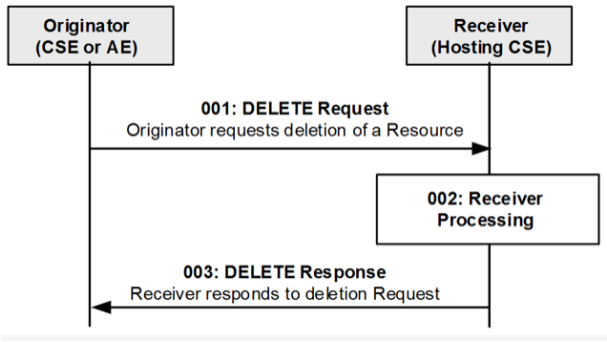

FIGURE 1.22: DELETE Operation.

NOTIFY Operation

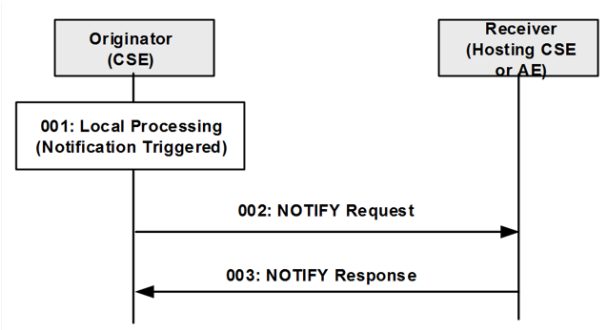

FIGURE 1.23: NOTIFY Operation.

1.3 oneM2M REST RESOURCES

1.3.1 Common Services Entity (CSE) Base

A <CSEBase>resource represents a CSE and serves as the root resource for all resources that are residing in the CSE.

1.3.2 Application Entity (AE)

An <AE>resource represents an Application Entity that is registered to a CSE. An <AE>resource supports attributes such as identifiers, contact information, status and capabilities of the Application Entity. An <AE>resource also serves as the root resource for all child resources of the Application Entity. For example, <container>, <flexContainer>, <subscription>and <group>resources of an Application Entity.

1.3.3 Container

Containers describe attributes of the data and child resources which are useful for representing hierarchical data structures. They can be created and organised in a similar

manner to folders on a computer. Each Container is allocated a unique ID and property fields that hold information about that container's contents. A container can contain additional containers which are analogous to subfolders on a computer.

A Container for data instances is represented by the <container>resource. It stores data that is used to share information with other entities and to track data. A <container>resource does not store actual values in the resource itself. Actual values from a sensor, for example, are stored in child-resources of <container>, i.e. <contentInstance>or further <container>resources. A single <contentInstance>contains a single piece of data sent by the publishing application.

<container>is the only resource allowed to have recursive child resources i.s. A <container>resource can have other <container>resources as child resources. This property is useful for representing hierarchical data structures.

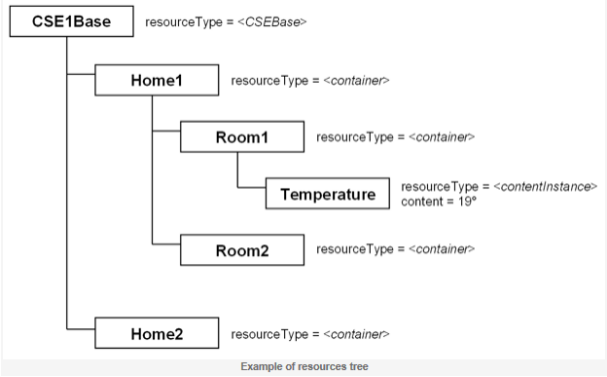

FIGURE 1.24: Example of resources tree.

Access Control Policy

Access Control Policies (ACPs) are used by the CSE to control access to resources. Resources are always linked to Access Control Policies. ACPs are shared between several resources.

Access Control Policies contain the rules (Privileges) defining:

- WHO can access the Resource (e.g. Identifiers of authorized AE/CSE).
- For WHAT operation (CREATE / RETRIEVE / UPDATE / DELETE...).
- Under WHICH contextual circumstances (Time, Location, IP address).

ACPs are represented by <accessControlPolicy>resources.These consist of privilege and selfPrivilege attributes that implement Access Control Rules (acr) through Access Control Originator (acor) and Access Control Operation (acop) procedures.

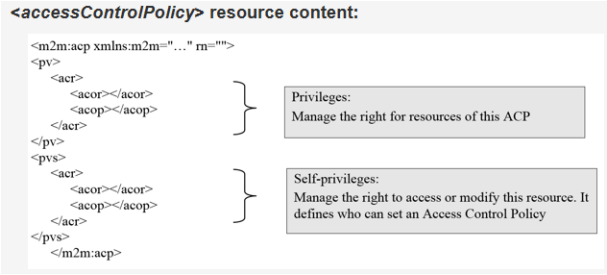

FIGURE 1.25: <accessControlPolicy>resources.

Access Control Operations are invoked through a value that is set on the basis of designated operation codes. The value '5', for example, corresponds to "CREATE and UPDATE" operations.

Operation Codes:

- CREATE 1
- RETRIEVE 2
- UPDATE 4
- DELETE 8
- NOTIFY 16
- DISCOVER 32

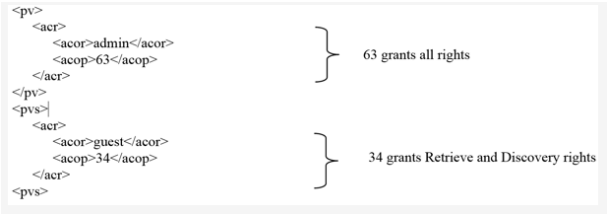

FIGURE 1.26: Access Control Operations.

Common attribute accessControlPolicyIDs link resources that are not <accessControlPolicy>resources to <accessControlPolicy>resources. All resources are accessible only if the privileges from the ACP grants it. All resources have an associated accessControlPolicyIDs attribute, either explicitly or implicitly.

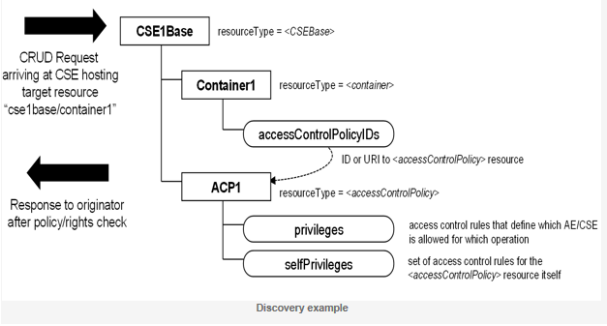

FIGURE 1.27: Discovery example of oneM2M.

1.3.4 Subscription & Notification

Events generated by resources can be received using the <subscription>resource. The <subscription>resource contains subscription information for its "subscribed-to" resource. The <subscription>resource is a child resource of the "subscribed-to" resource.

The originator (resource subscriber) has RETRIEVE privileges to the "subscribed-to" resource in order to create the <subscription>resource.

Notification policies specified in the attributes can be applied to the <subscription>. These specify which, when, and how notifications are sent. An example is the batchNotify attribute which indicates the receipt of notifications in batches rather than one at a time.

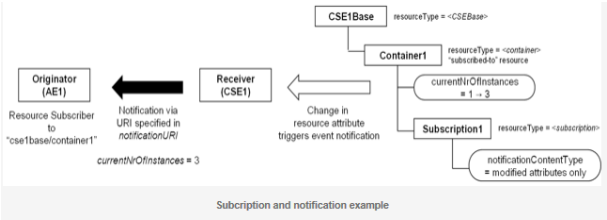

FIGURE 1.28: Subscription and notification example.

1.3.5 Discovery

Resource Discovery can be accomplished using the RETRIEVE operation by an Originator.

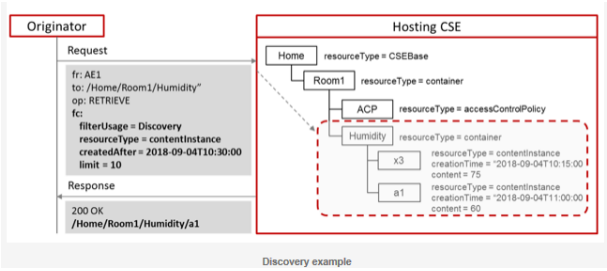

FIGURE 1.29: Discovery example in oneM2M.

Use of the filterCriteria parameter allows developers to limit the scope of results through parameters such as Type, Labels, Content Size.

1.3.6 Groups

A <group>resource represents a group of resources. A <group>resource can be used to do bulk manipulations on the member resources of the group. The <group>resource contains an attribute (i.e. memberIDs) that represents the members of the group and a <fanOutPoint>virtual resource that enables operations to be performed on all these member resources in either a unicast or multicast fashion.

1.3.7 flexContainer

A <flexContainer>is a customizable container for data instances. Like a <container>resource, <flexContainer>resources are used to share information with other entities and potentially to track the data. While the <container>resource includes data to be made accessible to oneM2M entities inside <contentInstance>child resources, a <flexContainer>resource supports content directly inside the <flexContainer>resource by means of one or more custom attribute(s). The attribute name and attribute data type of each custom attribute are defined explicitly for each customized <flexContainer>, i.e. the specific set of attribute name and type are defined in a corresponding XSD-file.

Chapter 2

Advanced Discovery and Query Use Cases And Requirements

2.1 Definition of terms

Advanced Semantic Discovery (ASD): an extension of the present oneM2M semantic discovery across a network of CSEs statically connected among them in a tree-like topology inside a single or multiple Service Provider (SP), including non oneM2M ones and in a mesh-like topology between the root of the different SPs

Advanced Semantic Discovery Query (ASDQ): word in the Advanced Semantic Discovery Query Language (ASDQL) according to the Theory of Formal Languages

Advanced Semantic Discovery Query Language (ASDQL): extension of the actual oneM2M Semantic Discovery Query Language (SDQL), which has to be suitable enough to describe queries that will be resolved in a cooperative way by a distributed network of CSEs

Note: Each CSE involved in the resolution participates in resolving subqueries and aggregating results by coordinating and cooperating among each other's.

Semantic Discovery Agreement (SDA): aims at adding a semantic registering information for the cooperation between CSEs

Note1: With ananalogy with the Border Gateway Protocol 2 kinds of cooperations are set:

- CSE1 to CSE2 meaning that CSE1 takes advantage of the infrastructure, MN-CSEs, and AEs registered in CSE2, and also shares security policies of CSE2. CSE1 is a CUSTOMER and CSE2 is a PROVIDER.
- CSE1 to CSE2 means that CSE1 and CSE2 mutually share infrastructure, MN-CSEs, and AEs and common security policies. CSE1 and CSE2 are PEERS.

NOTE 2: CUSTOMER and PROVIDER are roles that conform an asymmetric relationship, while PEER is a role conforming a symmetric relationship.

NOTE 3: Inside a single Service Provider, the SDA is not mandatory since it can be considered as PEER.

Semantic Discovery Routing (SDR): CSEs support a distributed Semantic Discovery Routing that listens for Advanced Semantic Discovery Query (ASDQ) and

1. reduces the Advanced Semantic Discovery Query (ASDQ) by means of the Semantic Query Resolution (SQR);
2. solves and forwards in a distributed way the queries;
3. reconstructs the partial results, sending back to the originator of the Advanced Semantic Discovery Query (ASDQ).

NOTE: Generally, two kinds of routing are discriminated, namely:

1. "Exhaustive". As an example, in the case that a semantic resource exists somewhere in the CSEs network, then the system will explore the entire distributed network until it will found it.

2. "Non-exhaustive". As an example, even in the case a semantic resource that exists somewhere in the CSEs network, the system will explore part of the distributed network until it will be stopped.

Semantic Query Resolution (SQR): each CSE contains a Semantic Query Resolution capability that takes as input an Advanced Semantic Discovery Query (ASDQ) and:

1. as output, produces a normalized Advanced Semantic Discovery Query (ASDQ);

2. produces a set of ordinary oneM2M Semantic Discovery Query (SDQ) from the normalized Advanced Semantic Discovery Query (ASDQ) one.

Semantic Recommendation (SR): capability in the CSE that takes routing decisions for forwarding a received Advanced Semantic Discovery Query (ASDQ)

NOTE: This capability uses the Semantic Routing Tables (SRT) and the Semantics Discovery Agreement (SDA).

Semantic Routing Table (SRT): contained in each CSE provides suitable routes to propagate the discovery queries according to the SDA

2.2 Semantic discovery in presence of a network of M2M Service Providers (M2MSP)

2.2.1 Description

The oneM2M system has implemented basic native discovery capabilities. This use case could be considered "parametric use-case" for Advanced Semantic Discovery (ASD) and it can be instantiated in many domain specific cases and illustrates the needs for an ASD within distributed network of CSEs belonging a single Service Provider and across different IoT Service Providers. It shows the importance of formalizing:

- an Advanced Semantic Discovery Query Language (ASDQL) able to write Advanced Semantic Discovery Query (ASDQ);

- a Semantic Discovery Routing Protocol (SDRP) to route an Advanced Semantic Discovery Query (ASDQ) between different CSEs;

- a Semantic Discovery Agreement (SDA), to state some communication agreements between CSEs;

- a Semantic Query Resolution functionality (SQR) allowing to locally resolve an Advanced Semantic Discovery Query (ASDQ) into some elementary standard oneM2M Semantic Discovery Queries (SDQ).

2.2.2 Actors

- Application Entities (AE) X of type T1, Y of type T2, Z of type T3, V of type T4, and W of type T5.

- 2 Middle Node Common Service Entities (MN-CSE) P, and Q:

  - A MN-CSE has a local database containing information on their registered AE. The local database includes location information (where each device is presently located), the device type, etc. Assuming P and Q have some Semantic Discovery Agreement (SDA) with A. Semantic Discovery Agreement (SDA) can be relaxed inside a single Service Provider.

- 4 Infrastructure Node Common Service Entities (IN-CSE) A, B, C, and D:
  - An IN-CSE has a local database containing information on their registered MN-CSE and AE. The local database includes location information (where each device is currently located), the device type, etc. Assuming A, B, C, and D have some "Semantic Discovery Agreement" (SDA) among each other.

2.2.3 Pre-Conditions

Consider the following topology:

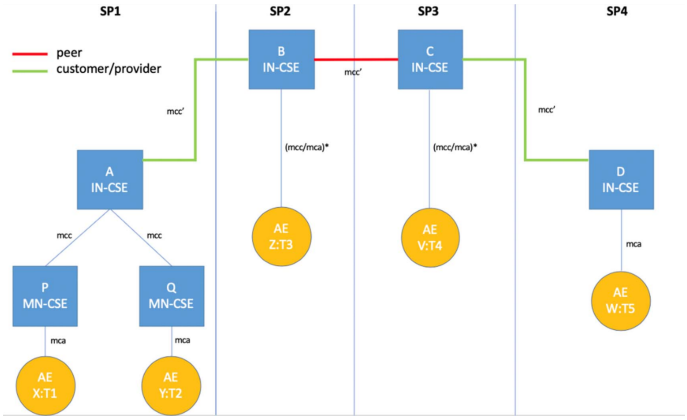

FIGURE 2.1: Pre-Condition topology.

2.2.4 Triggers

This clause presents, informally, three examples of the Advanced Semantic Discovery Query Language (ASDQL). Assuming AND, OR, NOT be non-terminals and ?T means a meta-variable of type T to be resolved.
EXAMPLE 1: X:T1 send to MN-CSE P:
ASDQ1 = ?T2 | FC2 AND ?T3 | FC3 AND ?T4 | FC4 AND ?T5 | FC5
The query can be intuitively read as follows: X is looking for:
some AE of type T2 registered in any CSE satisfying the filter criteria FC2, AND
some AE of type T3 registered in any CSE satisfying the filter criteria FC3, AND
some AE of type T4 registered in any CSE satisfying the filter criteria FC4, AND
some AE of type T5 registered in any CSE satisfying the filter criteria FC5
EXAMPLE 2: X:T1 send to MN-CSE P
ASDQ = ?T2 | FC2 OR ?T3 | FC3 OR ?T4 | FC4 OR ?T5 | FC5
The query can be intuitively read as follows: X is looking for:
some AE of type T2 registered in any CSE satisfying the filter criteria FC2, OR
some AE of type T3 registered in any CSE satisfying the filter criteria FC3, OR
some AE of type T4 registered in any CSE satisfying the filter criteria FC4, OR
some AE of type T5 registered in any CSE satisfying the filter criteria FC5
EXAMPLE 3: X:T1 send to MN-CSE P

ASDQ = (?T2 | FC2 OR ?T3 | FC3) AND (?T4 | FC4 OR ?T5 | FC5)
EXAMPLE 4: X:T1 send to MN-CSE P
ASDQ = (?T2 | FC2 AND ?T3 | FC3) OR (?T4 | FC4 AND ?T5 | FC5)
EXAMPLE 5: X:T1 send to MN-CSE P
ASDQ = (?T2 | FC2 AND ?T3 | FC3) OR (?T4 | FC4 AND (NOT ?T5 | FC5))
It is also possible to consider other non-terminals, such as (list not exhaustive):
ANY = search in all CSE databases;
CURRENT = search in the CSE local database;
CUSTOMER[N] = search in the databases of N CUSTOMER CSE;
PROVIDER[N] = search in the databases of N PROVIDER CSE;
PEER[N] = search start on the databases of N PEER CSE;
BETWEEN_TIME[SEC] = search should return in SEC;
BETWEEN_SPACE[METER] = search should give results in METER;
OF_BRAND[NAME] = search should give results of brand NAME.

2.2.5 Normal Flow

A "trace" of the Semantic Discovery Routing (SDR) generated by Example 1 is presented, the other examples can be easily traced following the same logic. This trace is inspired to a semantic discovery routing as follows:

- X sends an Advanced Semantic Discovery Query (ASDQ1) to P.
- P verifies the integrity of ASDQ1 and forwards the ASDQ1 to A that starts the Semantic Discovery Routing Protocol (SDPR) into the network of CSE.
- ASDQ1 is resolved using the Semantic Query Resolution System (SQRS) locally in A into four subqueries, namely ASDQ2, ASDQ3, ASDQ4, and ASDQ5, where:
  - ASDQ2 = ?T2 | FC2
  - ASDQ3 = ?T3 | FC3
  - ASDQ4 = ?T4 | FC4
  - ASDQ5 = ?T5 | FC5

1. A starts lookups in its local database, trying to solve ASDQ 2,3,4,5 but fail.
2. A down-forwards ASDQ1 to Q via an mcc pointer.
3. Q solve the subquery ASDQ2 ?T2 | FC2 in its local database returning Y to A.
4. A send backY to P and X
5. A up-forwards ASDQ3 and ASDQ4 and ASDQ5 to B via an mcc' pointer.
6. B solves the ASDQ3 ?T3 | FC3 in its local database returning Z to A (and back to P and X).
7. B side-forwards ASDQ4 and ASDQ5 to C.
8. C solves the ASDQ4 ?T4 | FC2 in its local database returning V to B (and back to A, P and X).
9. C down-forwards ASDQ5 to D.

10. D solves the ASDQ5 ?T5 | FC5 in its local database returning W to C (and back to B, A, P and X).

NOTE: When A up-forwards to B, it follows that A respect the CUSTOMER-PROVIDER SDA with B (e.g. A respects the SDA directives of B). When B side-forwards to C, it follows that B and C respect the PEER- PEER SDA directives. When C down-forwards to D, it follows that D respects the PROVIDER- CUSTOMER SDA with C (e.g. D respects the SDA directives of C).

The moral is: B and C should be "acknowledged" for their "routing job".

2.2.6 Alternative Flow

In the following alternative topology the CUSTOMER-PROVIDER SDA are reversed.

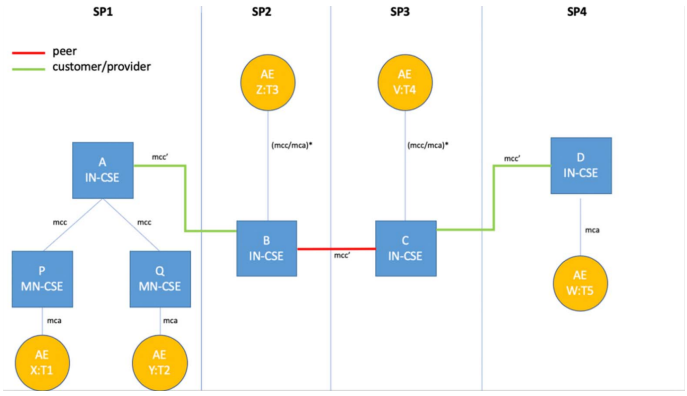

FIGURE 2.2: Pre-Condition topology for alternative flow.

A possible "trace" of the SDRM is expecting for the following caveat.

Caveat. When A down-forwards to B, it expects that B should respect the provider-customer SDA with A (e.g. B should acknowledge A. **This is not intuitive** since B does a favour to A and acknowledge A). When B side-forwards to C, it expects that B and C have a common SDA agreement and, as such, they do not acknowledge it each other. When C up-forwards to D, it expects that C and D have a common SDA agreement (e.g. C should acknowledge D. **This is not intuitive** since C do a favour to D and acknowledges D).

The moral is: B and C do a job for their providers and, moreover, they have to acknowledge for their "routing job".

Alternative traces happen in practice. Because of the distributed nature of the SDR, it is beneficial to try to incentivize routing respecting the SDA, and, as such, avoid routing not respecting the SDA. Those situations are not new in Internet and are referred as VALLEY ROUTING by Gao [6]. "Good routing" should guarantee that routing is always "valley preserving" (or "no valley").

2.2.7 Post-Conditions

X can start to interact with Y, Z, V, and W.

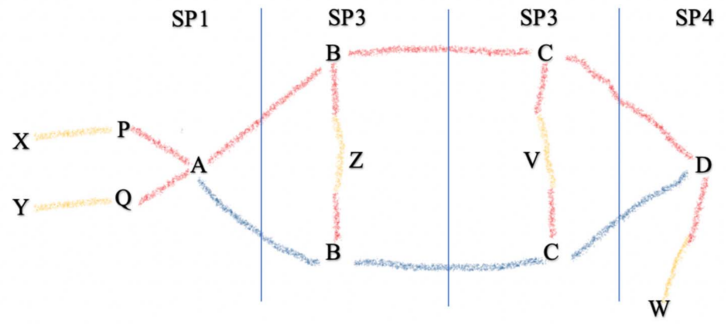

FIGURE 2.3: High-level illustration for multiple service providers semantic discovery use case.

2.2.8 Potential Requirements for the oneM2M system

1. The oneM2M system should provide mechanisms for Advanced Semantic Discovery (ASD) across a distributed network of IoT nodes within a single oneM2M Service Provider and across different IoT Service Providers.

2. A CSE receiving an Advanced Semantic Discovery Query (ASDQ) should extract the Semantic Discovery Query (SDQ), embedded in the packet payload, and shall resolve the query with respect to the locally available information and shall forward to other suitable CSEs the Advanced Semantic Discovery Query (ASDQ) to complete the discovery.

3. More specifically, the M2M system should provide:

    - an Advanced Semantic Discovery Query Language (ASDQL) that the ability to write Advanced Semantic Discovery Query (ASDQ);
    - a Semantic Discovery Agreement (SDA) to state some communication agreements between CSE;
    - a Semantic Query Resolution (SQR) that allows to locally translate an Advanced Semantic Discovery Query (ASDQ) into some elementary oneM2M Semantic Discovery Queries (SDQ);
    - a Semantic Discovery Routing (SDR) to route an Advanced Semantic Discovery Query (ASDQ) between different CSEs.

2.3 Semantic recommendation in CSEs for discovery

2.3.1 Description

This use case is built upon a cross-domain scenario in which a hospital has a large number of IoT devices which are in charge of performing different tasks. The IoT devices can be classified into the following categories: energy devices (load consumption, flexibility monitoring, energy switch, etc.), building devices (lights, door sensors, occupancy sensors, etc.) , personal devices (smart bands, smartphones, etc.), and devices related to health (hearth rate sensor, glucose monitor, etc.). These IoT devices are connected across the hospital network, but they do not necessarily belong to the same oneM2M Service Provider.

In this scenario, several actors need to discover and use IoT devices that are allocated outside their oneM2M Service Provider. For instance, if the energy devices detect an incoming negative peak of energy, entailing that a large number of devices should be switched off to avoid the whole hospital to run out of energy (losing critical systems for the patients). Then, the energy devices (or an application in charge) should be able to discover all the sensors in the building that are related to energy (like light bulbs or air condition) and switch them off. Also, the same actor (the energy devices or an application in charge) should detect the critical eHealth IoT devices for the patients and ensure that they are switched on. In the case that one of the eHealth IoT devices would run off, then the energy devices, or an application in charge, should perform a discovery task over the personal devices in order to find relevant people, i.e. doctors, nearby the critical eHealth IoT devices that are running out of energy in order to assist the patients.

This use case assumes that there is an interoperability platform (oneM2M) that allows monitoring and controlling the different IoT devices regardless their vendor. Also, the platform should ensure a secure and private environment so no unauthorized third party could access the network. This platform should ensure a sufficient rich discovery in order to meet the previous example.

- It is fully distributed in order for the Advanced Semantic Discovery (ASD) to reach from one oneM2M Service Provider to others that may contain relevant infrastructures.
- The Advanced Semantic Discovery Query (ASDQ) is expressed using an Advanced Semantic Discovery Query Language (ASDQL) so specific semantic terms from domains like energy or eHealth can be used.
- The Advanced Semantic Discovery (ASD) needs to happen in quasi real-time and therefore the communication mechanism across infrastructures that belongs to different Service Providers should not be blind; instead it should be guided by a Semantic Recommendation (SR).

This use case can be generalized to other domains in which IoT devices are spitted in different Service Providers and discovery needs be performed across them avoiding flooding the infrastructures, i.e. relying on a guided Semantic Recommendation System (SRS) to which infrastructures perform the Advanced Semantic Discovery (ASD), and to which discard, leveraging the network load.

2.3.2 Source

- oneM2M TS-0012: "oneM2M Base Ontology"
- oneM2M TR-0045: "Developer Guide Implementing Semantics"
- oneM2M TS-0001: "Functional Architecture"

2.3.3 Actors

- M2M Applications, M2M Service providers.
- IN-CSE and MN-CSE.
- IoT devices from the different domains.
- Medical staff, building staff, or technicians.

2.3.4 Pre-Conditions

network infrastructure distributed across different Service Providers. Intuitively, network infrastructure of oneM2M Service Providers is a set of M2M devices and Application Entities (AE) that have been installed and registered to their corresponding MN-CSE (Middle Node - Common Services Entity). The MN-CSEs have in turn been registered to the corresponding IN-CSE (Infrastructure Node - Common Service Entity).

All the different CSEs have Semantic Discovery Agreements (SDA) with each other, resulting in a tree-like network topology. In such a topology, the CSEs should rely as a Semantic Recommendation (SR) in order to assist the advanced semantic discovery resolution task performed by the CSEs involved in. As smarter the Semantic Recommendation (SR) is, as efficient will be the discovery in terms of time, CSEs visited, and number of query forwarded, among others. The discovery should allow expressing some network directives to address efficient routing across CSEs.

Both the registering and the discovery should be expressed according to the oneM2M described in oneM2M TS-0012 [i.20]. Nevertheless, due to tailored-domain terms required in the use case, the registering and the discovery should be also expressed with specific domain ontologies like the different extensions of SAREF.

2.3.5 Triggers

The IoT energy devices, or a technician, send first an Advanced Semantic Discovery Query (ASDQ) to find all the non- critical IoT devices allocated in the building. Then, a second ASDQ is issued to find all the critical IoT devices from the eHealth domain, and if required, a third ASDQ is issued to find relevant medical staff that could be near critical devices. The different ASDQs will rely on specific semantics, the first will contain information about devices and if they consume energy, the second about eHealth devices that are critical and cannot be switched off, and the third about the people, their roles in hospital, and their location.

2.3.6 Normal Flow

Following the first discovery task is showcased:

1. An IoT energy device, or a technician, sends an ordinary oneM2M Semantic Discovery Query (SDQ) to its CSE, written in SPARQL. The SPARQL query will contain terms about sensors that consume energy, are not from the eHealth domain, and are located in the b

2. The CSE verifies the integrity of the SDQ, and it tries to answer. If the CSE is not able to reply, it builds an ASDQ wrapping the SDQ.

3. Then the CSE forwards the ASDQ to other CSEs that may be located in the same oneM2M Service Provider, or in a different oneM2M Service Providers, or even sent to a non oneM2M IoT Service Provider.

4. Relying on the SR, the CSE selects and queries the relevant CSEs.

5. The CSE of the building will receive the ASDQ and will try to solve the embedded SDQ. The building contains suitable IoT devices so that the CSE will be able to produce and forward back a positive answer.

6. Finally the IoT energy device, or a technician, will receive the answer and the semantic discovery terminates successfully.

2.3.7 Post-Conditions

The query is answered if a resource that fulfills the discovery criteria is present in the network and "reasonably" reachable. For example, the discovery task needs be completed in a given threshold time even when crossing different IoT Service Providers is required.

2.3.8 High-Level illustration

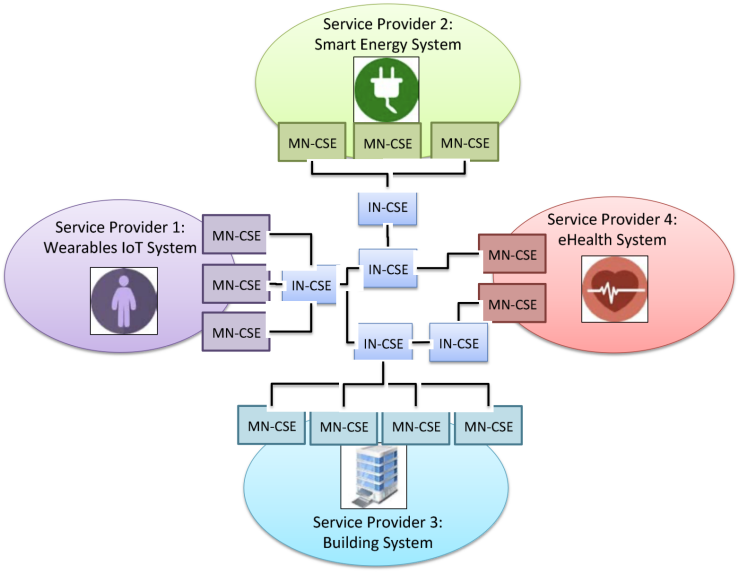

FIGURE 2.4: Semantic Recommendation in CSEs for Discovery.

2.3.9 Potential Requirements for the oneM2M system

1. The oneM2M system should integrate already standardized ontology extensions to the current oneM2M ontology to cope with new specific domains (e.g. SAREF core and its extensions)

2. Based on semantic information, the oneM2M system shall take routing decisions for forwarding a received ASDQ. The semantic information will allow the oneM2M system to maximize and to accelerate the semantic discovery process.

2.4 Facility management of a supermarket chain

2.4.1 Description

Building and facility managers need a helicopter view of the facilities management processes, regardless of existing building installations in order to make better-informed decisions and to enforce cross building policies. Building managers are faced with heterogeneous and vendor-specific installations. Centralized management of buildings oftentimes forces the owners to go through costly replacements to adopt mono-vendor solutions. Installation of new equipment requires costly system integration because devices are often designed to communicate with specific applications only.

This use case assumes a facility manager working for a supermarket chain and responsible of dozens of buildings. It is supposed that there is an interoperability platform

(oneM2M) that offers a standard interface to monitor and control all the buildings regardless of vendor. The facility manager could apply energy efficiency strategies to all buildings on large scale. He could for example, compare buildings to detect leaks, adjust the heat and the lighting according to forecast or predictive models, and compliant with applicable regulations.

The exposure of the huge amounts of data through modern APIs allows proliferation of new building services such as situational awareness, energy efficiency, intrusion detection, preventive maintenance and smart data.

Through further APIs, wider integration of the buildings with the outside world is achieved to give rise to fully integrated cities. The buildings start to interwork with energy grids (smart and micro grids), smart parking, electrical vehicle charging, waste management, etc., the ultimate goal for buildings to be considered really smart.

Assuring interoperability between all the data producing, data storing, and data processing components, semantic discovery and query mechanisms across and between sensors, devices, APIs and even IoT platforms are essential.

This use case is similar to the use case "Semantics query for device discovery across M2M Service Providers in clause 12.9 of oneM2M TR-0001-Use_Cases_Collection-V4_3_0. However, it extends the requirements with a focus on the discovery and query capabilities, introducing a direct relation with the semantic aspects and enabling more sophisticated semantic queries.

2.4.2 Source

- ETSI SR 003 680: "SmartM2M; Guidelines for Security, Privacy and Interoperability in IoT System Definition; A Concrete Approach"

2.4.3 Actors

- M2M devices as e.g. energy meters, temperature sensors, fire detectors, leak detectors, lightning controls, heat and air condition controls, surveillance cameras , cash boxes, inventory controls.
- Facility manager.
- M2M Service providers.
- M2M Applications e.g. data analytics, fault detection, energy efficiency, hypervision.

2.4.4 Pre-Conditions

M2M devices in the super markets have been installed and registered to their corresponding MN-CSE (Middle Node - Common Services Entity). The MN-CSEs have been registered to the corresponding IN-CSE (Infrastructure Node - Common Services Entity).

The M2M Application Provider 1 has contractual relationships with the M2M Service Providers 2, 3 and 4. The M2M Service Providers 1 and 2 have databases that contain information on the devices located in the supermarkets of the supermarket chain.

The facility manager wants to make use of the devices within his supermarkets and of the API "Facility management" in order to apply energy efficiency strategies to all buildings on large scale and to compare buildings to detect leaks, adjust the heat and the lighting according to forecast or predictive models, and compliant with applicable regulations. Assessing the warehouses stocks enables to refill it in time and a centralized fault detection ensures to take countermeasures.

The M2M Service Provider 3 wants to access data from energy consuming/measuring devices and/or respective databases of the M2M Service Providers 1 and 2 in order to optimize his energy providing balance.

The M2M Service Provider 4 wants to access data from parking lot sensors, from charging stations for electrical vehicles, data about the product range and warehouse stocks of the M2M Service Providers 1 and 2 in order to provide relevant services to the city inhabitants.

2.4.5 Triggers

The facility manager, the API "Facility management", the M2M Service Provider 3 or 4 (further on called "REQUESTER") sends an Advanced Semantic Discovery Query to the M2M Service Provider 1 or 2 (further on called "REQUEST RECEIVER"). The request contains information about the device to be discovered, e.g. a device type, a localization and other filters criteria.

2.4.6 Normal Flow

Following, one example of a typical scenario is described:

1. Via a device (e.g. user terminal), which is connected to the API "Facility management", the facility manager initiates an Advanced Semantic Discovery Query within the domain of the M2M Service Provider 1 to the smart meters of a specific area of a special supermarket, which enquires information about its energy consumption.

2. The API "Facility management" verifies the integrity of the Advanced Semantic Discovery Query and sends a semantic discovery request to the MN-CSE of the supermarket.

3. The MN-CSE searches for the specific requested type of devices whether they are connected t or not.

4. If the requested type of devices are connected to the MN-CSE, then it returns the requested information of the devices to the M2M Application.

5. If the requested devices are not connected to the MN-CSE, then a negative acknowledge is sent back to the M2M Application.

6. The API "Facility management" processes, if necessary, the received information and forwards it to the requesting device of the facility manager.

2.4.7 Alternative Flow

Following, one example of an alternative scenario is described:

1. An M2M Application of the Service provider 4 (Smart Cities domain) initiates an Advanced Semantic Discovery Query within the domain of M2M Service Providers 1 and 2 to find and identify the sensors of their parking lots, which enquires information about free parking spaces.

2. The IN-CSE of the Service provider 1 verifies the integrity of the Advanced Semantic Discovery Query and distributes it to the MN-CSEs of the supermarkets.

3. The MN-CSEs search for the specific requested type of devices whether they are connected to not.

4. If the requested type of devices are connected to the MN-CSE, then it returns the requested information of the devices to the IN-CSE, which forwards it to the requesting Service Provider 4.

5. If the requested devices are not connected to the MN-CSE, then a negative acknowledge is sent back to the IN-CSE, which forwards it to the requesting Service Provider 4.

6. The requesting M2M Application of Service Provider 4 processes the data and provides them in an appropriate way to the users of the M2M Application (e.g. city inhabitants).

2.4.8 Post-Conditions

The facility manager, the API "Facility management", the M2M Service Provider 3 or 4 can start to employ the devices based on the Advanced Semantic Discovery Query sent to the M2M Service Provider 1 or 2.

2.4.9 High-Level illustration

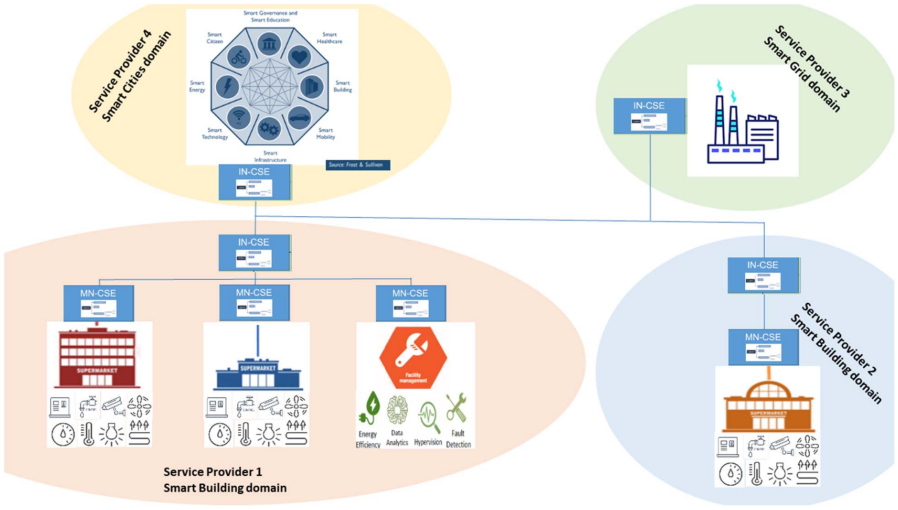

FIGURE 2.5: Facility management of a supermarket chain.

2.4.10 Potential Requirements for the oneM2M system

The following potential requirements are additional to the ones already identified in clauses 5 and 6.

1. Advanced Semantic Discovery shall support queries written with specific domain ontologies, e.g. SAREF.

2. Advanced Semantic Discovery shall support semantic reasoning between the baseline oneM2M ontology and the identified domain specific ontologies, e.g. SAREF.

As example, if a query is looking for a oneM2M device observing Celsius temperature, then the Advanced Semantic Discovery would potentially return a SAREF temperature sensor.

3. Advanced Semantic Discovery shall provide capabilities to identify multiple set of targets, and a multiplicity of searches (e.g. by setting parameters or filters).

4. The oneM2M Access Control Policy shall include discovery permissions to support Advanced Semantic Discovery. When an Advanced Semantic Discovery is performed by the oneM2M System, it shall operate according to the indications associated with the desired information.

It is also expected that:

- The solution would be based an evolution of the current oneM2M architecture and functionality and would reuse existing standard ontology mechanisms e.g. considering the SAREF standard developed in ETSI TC SmartM2M (which is also aligned with the W3C ontology approach). This intends to assure also a smooth interworking with relevant non-oneM2M solutions.

- The solution would be complete and will be a part of the oneM2M core functions, to avoid the need of ad hoc applications designed to expand the oneM2M functionality with the risk of being implemented with different flavours.

2.5 Healthcare network and clinical knowledge administration

2.5.1 Description

This use case looks at the semantic discovery requirements through a networking environment between people with disease (patients), the elderly, who want to live an independent life while remaining in their homes, special invalid people with a high risk of falling in their homes, doctors/care taking people, people practicing fitness exercises to improve their health, and institutions/organizations, who manage a clinical knowledge and information data basis or analyses of patient data.

On one side the number of the elderly is increasing permanently, on the other side, the doctors' anterooms are overcrowded. Therefore, telecare and telehealth systems get more and more important. M2M applications for eHealth support the remote management of patient illnesses by e.g. tracking blood sugar levels, controlling insulin dosage, measuring blood pressure and heartbeat, record infrequent abnormal heart rhythms, etc.

M2M applications for eHealth can enable the elderly to live an independent life and remain in their homes in cases when normally assistance would be needed. Remote monitoring of patient vital signs (e.g. pulse, temperature, weight, and blood pressure) minimizes the number of required doctor office visits. Further caretaking measures ensure that patients are taking their medications according to the required schedule and to track the activity level of seniors (e.g. time spent in bed each day, amount of daily movement in their homes) as a way of inferring their overall health and detecting changes that may require a doctor's or some other person's attention.

Various studies have concluded that falls in the home are the most common cause of injury among the elderly population, and one of the leading causes of morbidity and mortality among this population. A so called 'long lie' is a fall, in which the person remains on the ground for 5 min or more before being able to get up without assistance, or help arriving, which could detrimentally affect both the psychological as well as physical

wellbeing of the individual. Automated fall detection systems are using worn fall detectors, which trigger an alarm, when both the orientation and acceleration forces of the person reach a pre-set threshold. In case of an emergency detection, an alarm will be sent immediately to the emergency service centre, possibly together with pictures or videos, which a camera being installed in the home has taken.

M2M applications for eHealth can be used to record health and fitness indicators such as heart and breathing rates, energy consumption, fat burning rate, etc. during exercise sessions, and to log the frequency and duration of workouts, the intensity of exercises, running distances, etc. When this information is uploaded to a back-end server, it can be used by the user's physician as part of their health profile, and by the user's personal trainer to provide feedback to the user on the progress of their exercise program. It allows adapting exercise programs or physiotherapy more precisely and more quickly to the needs of the patient/user.

This use case assumes that there is an interoperability platform (oneM2M) that offers a standard interface to monitor and control all the eHealth devices regardless of vendor. It is supposed that professional knowledge generation bodies (e.g. colleges/universities) get authorization to access the patient data. They can assist clinicians for the appropriate diagnosis and method of treatment by providing clinical (textual) guidelines and recommendations in order to reduce the risk of medical errors and to assist in decision-making processes.

Assuring interoperability between all the various data producing, data storing, and data processing components, semantic discovery and query mechanisms across and between sensors, devices, APIs and even IoT platforms are essential.

2.5.2 Source

- ETSI SR 003 680: "SmartM2M; Guidelines for Security, Privacy and Interoperability in IoT System Definition; A Concrete Approach"
- ETSI TR 102 732: "Machine-to-Machine Communications (M2M); Use Cases of M2M applications for eHealth"
- AIOTI Report: "IoT Relation and Impact on 5G, Release 3.0"

2.5.3 Actors

- M2M eHealth devices as e.g. wearable sensors, falling detectors, blood pressure meter.
- Emergency supervisor.
- Doctors and caring people.
- Knowledge generation bodies.
- M2M Service providers.
- M2M Applications e.g. data analytics.

2.5.4 Pre-Conditions

M2M devices in the patients/the elderly homes and fitness locations have been installed and registered to their corresponding M2M Service Provider. In this use case, the devices are represented by devices that are ADN registered to MN-CSEs.

The M2M Application Providers 1, 2 and 3 have relationships one with each other. The M2M Service Providers 1 and 2 host information on the devices located in the patients/the elderly homes and fitness locations.

The doctors and caring people want to make use of the devices in the patients/the elderly homes and fitness locations in order to check the health or behavioral data of the patients as a way of inferring their overall health and detecting changes that may require a doctor's or some other person's attention. The emergency supervisor is operating the Intelligent Emergency Response System and manages alarms.

The M2M Service Provider 3 wants to access data from of the M2M Service Providers 1 and 2 in order to manage its knowledge data basis and to update analysis results, recommendations and guidelines.

2.5.5 Triggers

The doctor or caring person, the API "Clinicians patient data analysis" or the M2M Service Provider 3 (further on called "REQUESTER") sends an Advanced Semantic Discovery Query to the M2M Service Provider 1 or 2 (further on called "REQUEST RECEIVER"). The request contains information about the device to be discovered, e.g. a device type, a localization and other filters criteria.

2.5.6 Normal Flow

Following, one example of a typical scenario is described:

1. Via a device (e.g. user terminal), which is connected to the API "Doctors or caring people", the doctor or caring person initiates an Advanced Semantic Discovery Query within the domain of the M2M Service Provider 1 to an eHealth device type of a specific group of patients, which enquires information about the pulse, temperature, weight or blood pressure.

2. The API "Doctors or caring people" verifies the integrity of the Advanced Semantic Discovery Query and sends a semantic discovery request to the MN-CSE of the specific group of patients.

3. The MN-CSE searches for the specific requested type of devices whether they are connected or not.

4. If the requested type of devices are connected to the MN-CSE, then it returns the requested information of the devices to the M2M Application.

5. If the requested devices are not connected to the MN-CSE, then a negative acknowledge is sent back to the M2M Application.

6. The API "Doctors or caring people" processes, if necessary, the received information and forwards it to the requesting device of the doctor or caring person.

2.5.7 Post-Condition

The REQUESTER (doctors or caring people, the API "Clinicians patient data analysis" or the M2M Service Provider 3) can start to employ the devices based on the Advanced Semantic Discovery Query sent to the M2M Service Provider 1 or 2.

2.5.8 High-Level illustration

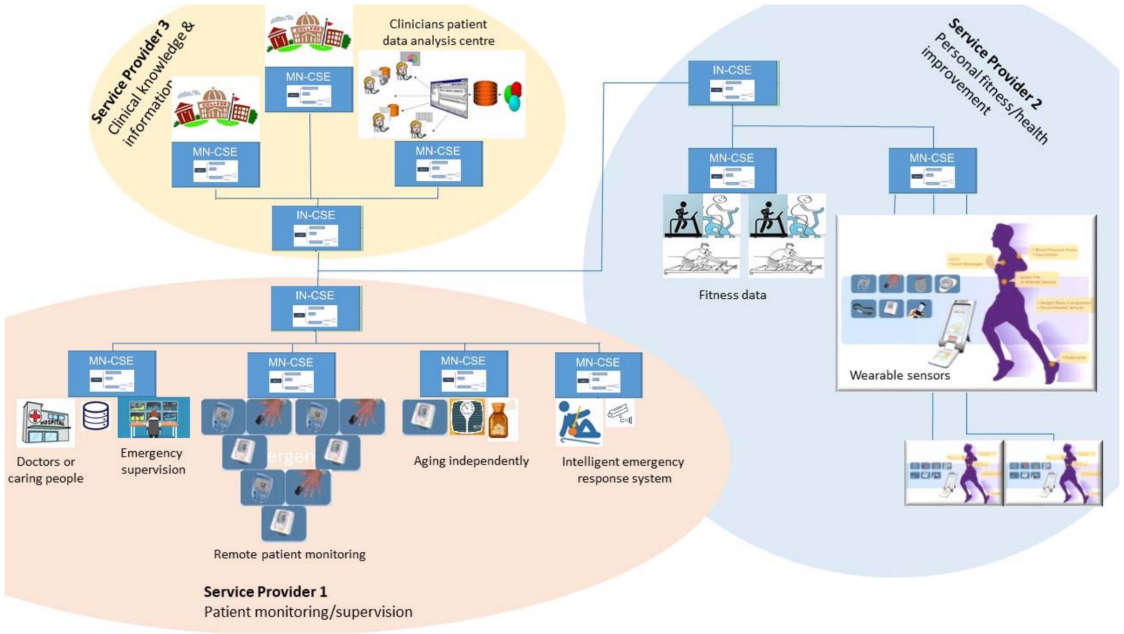

FIGURE 2.6: Healthcare network and clinical knowledge administration.

2.5.9 Potential Requirements for the oneM2M system

The following potential requirements are additional to the ones already identified in clause 5, 6 and 7:

1. Advanced Semantic Discovery shall prioritize queries, e.g. to ensure a quick response to urgent situations.

2. Advanced Semantic Discovery shall minimize complexity to avoid impacting negatively the oneM2M system performance.

Chapter 3

Advanced Semantic Discovery

3.1 Introduction

The present clause is based on the ETSI Technical Report ETSI TR 103 714 (4 use cases and 17 potential requirements concerning an Advanced Semantic Discovery related Work Item on oneM2M).

The present clause describes the Advanced Semantic Discovery functionalities and its impact in the oneM2M specification.

The clause gives a High-Level View of Advanced Semantic Discovery functionalities and a Detailed Forecast of the impact on the oneM2M specification (e.g. new feature, new resources, new parameters, modifications to the existing ones and on the oneM2M API).

Each clause of an Advanced Semantic Discovery functionality is composed by further clauses of representing one Functional feature or an enhancement of oneM2M resources or an API extension as necessary to achieve the Advanced Semantic Discovery solution.

It presents a description of the mechanism designed to support Advanced Semantic Discovery and the main building blocks that will be part of the Advanced Semantic Discovery. It may contain figures and examples. The information contained are the basis to develop the detailed contribution to oneM2M in ETSI TR 103 717 and such as could be written with precise references to the oneM2M elements and terminology.

3.2 Functional description of oneM2M Advanced Semantic Discovery

3.2.1 Semantic Discovery Agreements

The rationale of SDA is rather simple: Advanced Semantic Discovery induces some traffic flow: each CSE is considered as an Autonomous System in BGP. As an example, Figure 3.1, inspired by the oneM2M architecture (see oneM2M TS-0001 [i.5]), presents some CSEs with their respective abstract relationships.

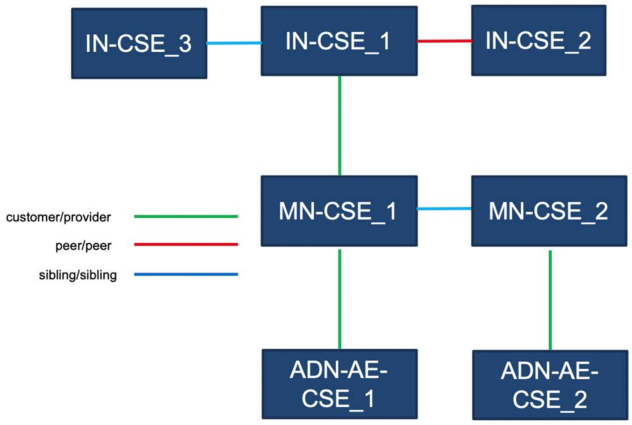

FIGURE 3.1: C2P and P2P and S2S CSE relationships.

CSE_A and CSE_B are in CUSTOMER-to-PROVIDER relationship when CSE_A takes advantage of the Infrastructure and AEs registered in CSE_B, and they also share the same Security Policies.

CSE_A and CSE_B are in a PEER-to-PEER relationship, when both CSEs mutually take advantage of the AEs and the infrastructure of each other and share compatible Security Policies.

CSE_A and CSE_B are in a SIBLING-to-SIBLING relationship, when both CSE belong to the same Trusted Domain or appear as a result of mergers and acquisitions.

When a CSE receives an advanced discovery, from downstream (i.e. from one of its customer) or from upstream (i.e. from one of it providers) or from side stream (e.g. from one of those peers) it can choose to forward the query through some (but not all) its customers, or peers or providers. The "strategy" to select the next hop heavily depends on the SDA. The same happens when a CSE generates an advanced discovery query. A simple rule of thumb, directly derived by BGP protocol is as follows: a semantic routing not respecting the SDA will produce routing that generate e.g. a routing path of the shape:

PROVIDER -> CUSTOMER -> PROVIDER

This path is unwelcome for the customer that pay the provider generating the query but also the provider where the query is destinated: in other words: it "pay" the two customers for an unwanted connectivity. As such, all no-valley routing paths are admitted during the Advanced Semantic Discovery routing. No-valley routing are therefore of the shape:

CUSTOMER->PEER->PEER->PROVIDER -> CUSTOMER

In other words, a valid path should have the following valid path pattern: zero or more CUSTOMER-to-PROVIDER links, followed by zero or one PEER-to-PEER link, followed by zero or more PROVIDER-to-CUSTOMER links. In addition, SIBLING-to-SIBLING links can appear in any number anywhere in the path.

3.2.2 Semantic Non-Functional Issues

ASD Query with Priorities

AEs can generate Advanced Semantic Queries looking for some discovery TYPES and adding special FILTER CRITERIA, such as TIME[SEC] or SPACE[METERS]. When a CSE

receives this query it should put it "on the top of the stack" and route before the others scheduled queries.

To give a simple intuition, the scheduler can be build (in C++ syntax) as a map like:

```
std::map<int,std::tuple<int,time\_t,int>> schedulerMap;
```

where:

- **Id** is the unique local identifier of the query, generated sequentially by the CSE;
- **GateIndex** indicates the CSE-to-CSE SDA relationship where the query is coming from, i.e. from a Customer CSE, or a Peer CSE or a Provider CSE or a Sibling CSE;
- **StartTime** denotes the (absolute) time the query was generated;
- **MaxTime** is the maximum time (in seconds) that the AE is supposed to wait before receiving an answer;
- **Direction** can be one of the following: up forward, down forward, or side forward, in compliance with the SDA and the no-valleys routing restrictions.

The scheduler will be in charge to rearrange the queue and reschedule queries that has not yet received a response.

Performance of ASD

Every CSE should be able to self-measure the local performance of the Advanced Semantic Discovery. The CSE will produce a LOG file listing the following information (list not exhaustive):

- the absolute number of queries received, aggregated by:
  - AE directly registered to the CSE;
  - Provider CSEs;
  - Customer CSEs;
  - Sibling CSEs;
  - Peers CSEs;
  - FEA TURE-TYPES;
- the response time of each query forwarder to a neighbor CSE, aggregated by:
  - Provider CSEs;
  - Customer CSEs;
  - Sibling CSEs;
  - Peers CSEs;
- the success and failure rate of queries generated by all the AE directly registered in the CSE itself;
- the success and failure rate of queries forwarded and aggregated by:
  - Provider CSEs;
  - Customer CSEs;
  - Sibling CSEs;
  - Peers CSEs.

This file should be sent to the first IN-CSE in the CSE registration chain on demand or every N seconds or every M queries received or generated or forwarded.

Searching Multiple set of Targets

According to the requirement elicitation, the Advanced Semantic Discovery Query (ASDQ) should be implemented according to the SPARQL 1.1 protocol, and the Advanced Semantic Discovery Query Language (ASDQL) should be implemented according to the SPARQL 1.1 Query Language. Following both SPARQL standards, the Advanced Semantic Discovery will be able of searching over multiple set of targets.

3.2.3 Semantic Discovery Query and Query Language

According to the requirement elicitation, the Advanced Semantic Discovery Query (ASDQ) needs to be implemented using the well-known and established W3C SPARQL protocol. Additionally, as for the Advanced Semantic Discovery Query Language the W3C SPARQL query language can be adopted. Both standards meet the requirements listed in the aforementioned clauses; nevertheless the SP ARQL protocol may be adjusted to meet the potential requirement number 4 for the oneM2M system.

3.2.4 Semantic Discovery Routing and Resolution Mechanism

A "trace" of the Semantic Discovery Routing (SDR) generated in Figure 3.2 (red lines represents request and reply Advanced Semantic routing paths from the AE that originates the complex query till the searched AE). The other examples in the same clause can be easily traced following the same logic. This trace is inspired to a semantic discovery routing as described in [i.24] and [i.25] and proceeds as follows:

- X sends an Advanced Semantic Discovery Query (ASDQ1) to P;
- P verifies the integrity of ASDQ1 and forwards the ASDQ1 to A that starts the Semantic Discovery Routing Protocol (SDPR) into the network of CSE;
- ASDQ1 is resolved using the Semantic Query Resolution System (SQRS) locally in A into four subqueries, namely ASDQ2, ASDQ3, ASDQ4, and ASDQ5, where:

ASDQ2 = ?T2 | FC2
ASDQ3 = ?T3 | FC3
ASDQ4 = ?T4 | FC4
ASDQ5 = ?T5 | FC5

1. A starts lookups in its local database, trying to solve ASDQ 2,3,4,5 but fails;
2. A down-forwards ASDQ1 to Q via an mcc pointer;
3. Q solve the subquery ASDQ2 ?T2 | FC2 in its local database returning Y to A;
4. A sends back Y to P and X;
5. A up-forwards ASDQ3 and ASDQ4 and ASDQ5 to B via an mcc' pointer;
6. B solves the ASDQ3 ?T3 | FC3 in its local database returning Z to A (and back to P and X);
7. B side-forwards ASDQ4 and ASDQ5 to C;
8. C solves the ASDQ4 ?T4 | FC2 in its local database returning V to B (and back to A, P and X);

9. C down-forwards ASDQ5 to D;

10. D solves the ASDQ5 ?T5 | FC5 in its local database returning W to C (and back to B, A, P and X).

NOTE 1: When A up-forwards to B, it follows that A respect the CUSTOMER-to-PROVIDER SDA with B (e.g. A respects the SDA directives of B). When B side-forwards to C, it follows that B and C respect the PEER- to-PEER SDA directives. When C down-forwards to D, it follows that D respects the PROVIDER-to- CUSTOMER SDA with C (e.g. D respects the SDA directives of C).

The moral is: B and C should be "acknowledged" for their "routing job".

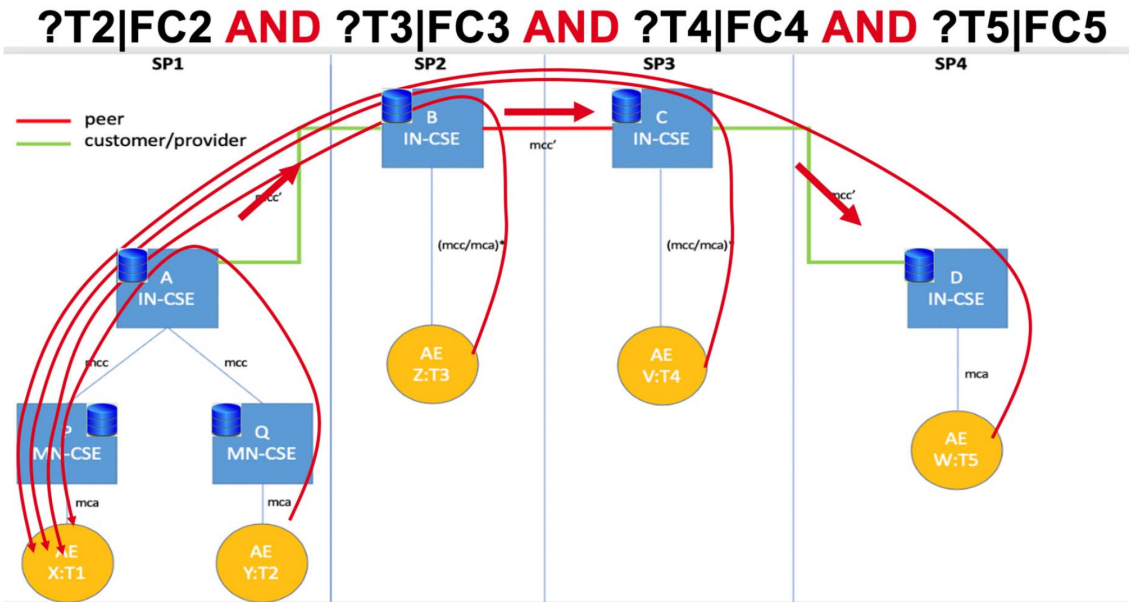

FIGURE 3.2: Advanced Semantic Discovery Routing Flow.

In Figure 3.3, an alternative flow is represented. In this alternative topology the CUSTOMER-to-PROVIDER SDA are reversed.

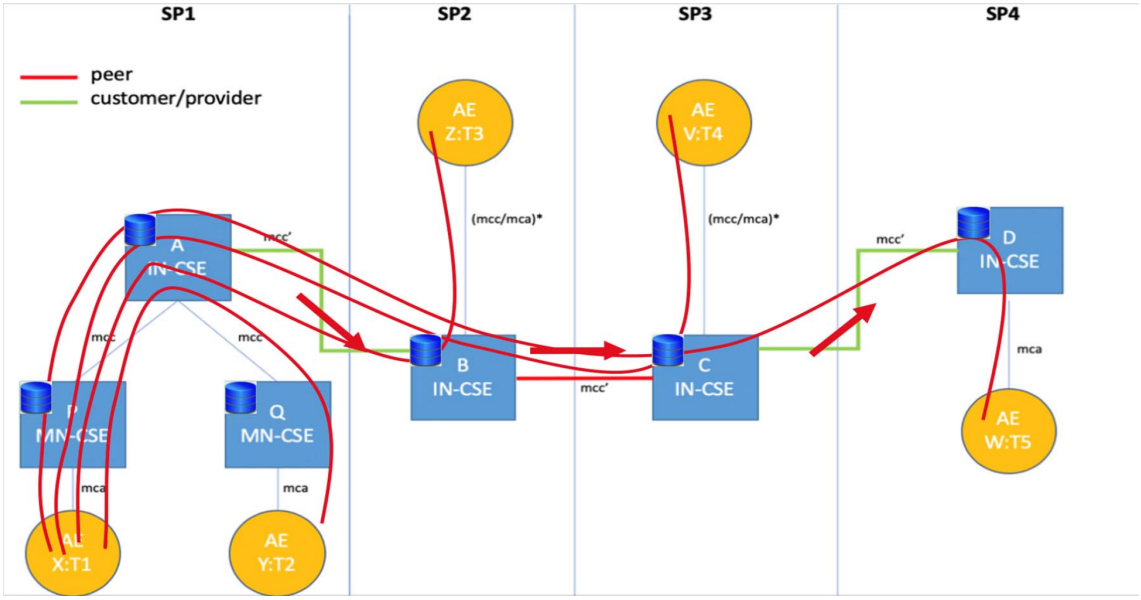

FIGURE 3.3: Alternative Advanced Semantic Discovery Routing Flow.

A possible "trace" of the SDRM, proceeds as in clause 5.5 of ETSI TR 103 714, excepting for the following caveat.

Caveat. When A down-forwards to B, it expects that B should respect the PROVIDER-to-CUSTOMER SDA with A (e.g. B should acknowledge A. This is not intuitive since B does a favour to A and acknowledge A). When B side- forwards to C, it expects that B and C have a common SDA agreement and, as such, they do not acknowledge it each other. When C up-forwards to D, it expects that C and D have a common SDA agreement (e.g. C should acknowledge D. This is not intuitive since C do a favor to D and acknowledges D).

The moral is: B and C do a job for their providers and, moreover, they have to acknowledge for their "routing job".

Alternative traces are possible but they should not be incentivized by the SDA in practice. Because of the distributed nature of the SDR, it is beneficial to try to incentivize routing respecting the SDA, and, as such, avoid routing not respecting the SDA. Those situations are not new in Internet and are referred as VALLEY ROUTING by Gao [6]. "Good Advanced Semantic Routing" in oneM2M should guarantee that routing is always "valley preserving" (or "no valley").

Algorithmic description of the routing

The CSEs support a distributed Semantic Discovery Routing (SDR) that listens for Advanced Semantic Discovery Query (ASDQ) and:

1. It extracts the Advanced Semantic Discovery Query (ASDQ).
2. It reduces the Advanced Semantic Discovery Query (ASDQ) into a set of **#m** Semantic Discovery Queries (SDQ) by means of the Semantic Query Resolution System (SQRS).
3. It solve as much Semantic Discovery Queries (SDQ) as it can, using resources registered in the CURRENT CSE, say **#n**.
4. It forwards the remaining **#(m-n)** Semantic Discovery Queries (SDQ) to α-CSE Customer (in "downstream"), taken from the CURRENT Semantic Routing Table (SRT), resolving say **#p**.

5. It forwards the remaining **#(m-n-p)** Semantic Discovery Queries (SDQ) to β-CSE PEER (in "side stream"), taken from the CURRENT Semantic Routing Table (SRT), resolving say **#q**.

6. It forwards the remaining **#(m-n-p-q)** Semantic Discovery Queries (SDQ) to γ-CSE PROVIDERS (in "upstream"), taken from the CURRENT Semantic Routing Table (SRT), resolving say **#r** $\leq$ **#(m-n-p-q)**.

7. It reconstructs in a reverse order the partial results, sending back to the originator of the Advanced Semantic Discovery Query (ASDQ).

Note2: α, β, and γ are protocol specific parameters that can be locally modified in each CSE.

Note3: if #r=#(m-n-p-q), then the query is exhaustive, and this is in contrast with non-exhaustive routing. Generally, two kinds of routing are discriminated, namely: "Exhaustive". As example, in the case that a semantic resource exists somewhere in the CSEs network, then the system will explore the entire distributed network until it will found it. "Non-exhaustive". As example, even in the case a semantic resource that exists somewhere in the CSEs network, then the system will explore part of the distributed network until it will be stopped.

3.2.5 Semantic Routing Tables

A simplified example of a local resource database hosted in a CSE can be represented as follows.

TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_1 ... URI_q	(CSE_1, #cu_1) ... (CSE_x, #cu_m)	(CSE_1, #pe_1) ... (CSE_y, #pe_m)	(CSE_1, #pr_1) ... (CSE_z, #pr_m)
WATER_VALVE	URI_1 ... URI_r	(CSE_1, #cu_1) ... (CSE_x, #cu_n)	(CSE_1, #pe_1) ... (CSE_y, #pe_n)	(CSE_1, #pr_1) ... (CSE_z, #pr_n)
AIR_POLLUTION_STATION	URI_1 ... URI_s	(CSE_1, #cu_1) ... (CSE_x, #cu_p)	(CSE_1, #pe_1) ... (CSE_y, #pe_p)	(CSE_1, #pr_1) ... (CSE_z, #pr_p)

FIGURE 3.4: Semantic Routing Table.

Each CSE has a SDA Routing Table listing (aka, PHYSICAL Routing Table) for each CSE the number of CSE physically connected with their corresponding SDA, see Figure 3.5 below.

CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
CSE_1 ... CSE_x	CSE_1 ... CSE_y	CSE_1 ... CSE_z

FIGURE 3.5: SDA-Table.

An alternative example of routing table hosted in a CSE is the following:

```
@prefix ads: <http://www.m2m.org/ads#> .
@prefix saref: <https://w3id.org/saref#> .

<http://cse1.com/001> a <http://www.m2m.org/ads#RoutingTable> ;
 ads:hasEntries [
  a ads:RoutingEntry ;
  ads:responseRatio "80%" ;
  ads:peer <http://cse1.com/002> ;
  ads:summaries [
   a <https://w3id.org/saref#Humidity> ;
   saref:measuresProperty [ a saref:Humidity ]
  ], [
   a saref:TemperatureSensor ;
   saref:measuresProperty [
    a saref:Temperature ;
    saref:hasMeasurement [ saref:isMeasuredIn om:degree_Celsius ]
   ] ;
   saref:isLocatedIn [ a saref:BuildingSpace ]
  ], [
   a saref:Sensor ;
   saref:measuresProperty [
    a saref:Temperature ;
    saref:hasMeasurement [ saref:isMeasuredIn om:degree_Celsius ]
   ]
  ]
 ], [
  a ads:RoutingEntry ;
  asd:responseRatio "20%" ;
  ads:customer <http://cse1.com/003> ;
  ads:summary [
   a saref:LightBulb ;
   saref:measuresProperty [ a saref:Status ]
  ]
 ] .
```

FIGURE 3.6: Routing table hosted in a CSE.

The previous Semantic Routing Table is implemented as an RDF document using an example ontology, i.e. ads, to describe its concepts. This routing table contains two entries, one for a peer CSE (http://cse1.com/002) and another one for a customer CSE (http://cse1.com/003). For those entries the table contains the response ratio of each of those CSEs, and also, for each, a summary of the resource description resisted in those target CSEs.

Considering that in either oneM2M and in the Advanced Semantic Discovery the query language required is SPARQL, having RDF summaries of the resource descriptions stored in other CSEs will improve the routing. On the one hand, since the summaries are RDF and the query language is SPARQL, choosing relevant CSEs depending on the terms specified in the query and relying on the terms of the summaries will be the same as solving partially the SP ARQL query over them. On the other hand, regardless the implementation of the semantic routing table they also store some attributes referring to the topology of the network (like if a CSE is a customer or a peer), or features like the response ratio, to provide routing capabilities considering also these physical features.

3.2.6 Semantic Routing Tables Upgrade and Propagation

Each time a resource register inside a CSE, this should send a SEMANTIC UPGRADE message to all neighbors CSEs. As example, when y news AE-THERMOMETER with URI_1 ... URI_y register on that CSE, the Semantic Routing Table will be upgraded (in red the upgrades) as in Figure 3.7.

C S E TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_x URI_1 ... URI_y	(CSEcust1,#_) (CSEcust2,#_)	(CSEpeer,#_)	(CSEprov,#_)

FIGURE 3.7: Upgrading a Semantic Routing Table with y new AE-THERMOMETERS.

And the following NOTIFICATION messages will be sent to all the adjacent CSEs:

NOTIFY CSEcust1 WITH #+y THERMOMETER
NOTIFY CSEcust2 WITH #+y THERMOMETER
NOTIFY CSEpeer WITH #+y THERMOMETER
NOTIFY CSEprov WITH #+y THERMOMETER

When an adjacent CSE will receive the NOTIFY message, its routing table will be modified shown in Figure 3.8 (in yellow the upgrades).

CSEcust1 TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_v, URI_w	...	...	(CSE, #_+y)

CSEcust2 TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_z	...	...	(CSE, #_+y)

CSEpeer TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_a	...	(CSE, #_+y)	...

CSEprov TYPE	URI	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	URI_b, URI_c, URI_d	(CSE, #_+y)	...	...

FIGURE 3.8: Upgrading the adjacent SRT CSEs with the new y AE-THERMOMETERS.

Analogously, when a resource unregisters on that CSE, a corresponding notify message:

NOTIFY URI{cust1,cust2,peer,prov} WITH #-1 THERMOMETER

Will be sent to all adjacent CSEs.

3.2.7 Semantic Recommendation System

Semantic Recommendation system (SR), a mechanism is meant to be embedded in each CSE, that, upon a reception of an Advanced Semantic Query from a registered AE or from another CSE, should be able to:

1. read the SDA-Table and the SRT;
2. take the best decision to found the next hops for the Advanced Semantic Query;
3. annotate all the CSE belonging to the SDA-Table that participate pro-actively to successful routing.

The solution proposed to implement is inspired to the Kademlia P2P overlay network.

TYPE	BUCKETS[TYPE]	CSE CUSTOMERS	CSE PEERS	CSE PROVIDERS
THERMOMETER	CSE_1 ... CSE_q	(CSE_1, #cu_1) ... (CSE_x, #cu_m)	(CSE_1, #pe_1) ... (CSE_y, #pe_m)	(CSE_1, #pr_1) ... (CSE_z, #pr_m)
WATER_VALVE	CSE_1 ... CSE_r	(CSE_1, #cu_1) ... (CSE_x, #cu_n)	(CSE_1, #pe_1) ... (CSE_y, #pe_n)	(CSE_1, #pr_1) ... (CSE_z, #pr_n)
AIR_POLLUTION_STATION	CSE_1 ... CSE_s	(CSE_1, #cu_1) ... (CSE_x, #cu_p)	(CSE_1, #pe_1) ... (CSE_y, #pe_p)	(CSE_1, #pr_1) ... (CSE_z, #pr_p)

FIGURE 3.9: SRT with Recommendation System.

In a nutshell: the new column CSE RECOMMENDED (aka BUCKETS[TYPE]) is added to the SRT. For each TYPE SRT[TYPE][CSE-RECOMMENDED] is set as an ordered LIST[100][CSE]. As in Kademlia, those lists are called BUCKETS and in oneM2M BUCKETS[TYPE]. At the beginning the CSE RECOMMENDED column is empty for all TYPE. As soon as the Advanced Semantic Routing proceeds the list of 100-best CSEs start to be filled.

A sketch of the algorithm to enter the BUCKET[TYPE] is the following:

1. IF the CSE answers successfully to an Advances Semantic Query for that TYPE and already is in the BUCKET[TYPE], then move it to the tail of the list;
2. OTHERWISE:
   (a) IF the BUCKET[TYPE] has fewer than 100 entries, then inserts the news CSE at the tail of the list;
3. OTHERWISE:
   (a) ping the least-recently seen CSE;
   (b) IF the least-recently seen CSE fails to respond, THEN it is evicted from BUCKETS[TYPE] and the new CSE is inserted at the tail;
4. OTHERWISE it is moved to the tail of the list, and the new CSE is discarded.

NOTE 1: BUCKETS[TYPE] will generally be kept constantly fresh, due to traffic of requests traveling through nodes.

NOTE2: When there is no traffic: each CSE picks a random CSE in one of their K BUCKETS[TYPE] and performs an Advanced Semantic Discovery node search for that TYPE.

Chapter 4

Implementing ASD and their Components in the OMNeT++ Discrete Network Event Simulation Tool

4.1 OMNeT++ discrete network event simulator

OMNeT++ is an object-oriented discrete event network simulation framework. It provides a user-friendly interface and script/procedural and OO languages for modeling and simulating network structure and behaviors. OMNeT++ is strongly considered as a component-oriented approach which promotes structured and reusable models, which communicate with message passing; using the simulation class library.

The modeling and simulation rely on different files and formats:

- Network infrastructure can be described either in the OMNeT++ graphical editor or with the NEtwork Description (NED) language. NED language allows the user to make or reuse some simple modules and assemble them into compound modules.
- Behavior of modules are defined in a C++ code.
- Initialization file where the parameters and the scenarios for the simulation are set

The modeling elements in OMNeT++ are simple and compound modules that can declare input, output, or bidirectional gates. Gates interface the modules: on the gates, messages are sent out through output gates and arrive through input gates. OMNeT++ allows to reuse the predefined messages templates or the definition of new ones with specific payload. Input and output gates of two different modules are linked with channels. Channels are created either in simple modules or module hierarchy: within a compound module, corresponding gates of two submodules, or a gate of one submodule and a gate of the compound module can be connected. The NED language has several features which let it scale well to large projects.

The behaviors of modules are defined in a source file in C++. Modules are interconnected through channels that originate and end on gates. OMNeT++ proposes a large set of libraries that cover the main protocols of the internet with a large set of modules that can be reused. As in the current experimentation, it is also possible to design a specific protocol and to that aim, OMNeT++ proposes efficient libraries for managing the classical data structures used in the Net (routing table, data of protocols). The definition of new messages with specific payload are set in an .msg file. The message is linked with the source file and then sent between the modules using gates.

In OMNeT++, simulation models are parameterized and configured for execution using configuration files called INI files. Such a file is linked to the network defined in the NED file and allows to initialize the parameters of different modules and the network topology. OMNeT++ provides a source file format and also a configuration form to specify the parameters and network. The INI File Editor is a very useful feature because it contains the detailed parameters of the simulation model.

- The logic of simulation modules is implemented in the C++ source files. Every module declared in the NED file has its behavior encoded in a C++ source file. There are two specific methods assigned to a module: initialize and handleMessage where the logic should be programmed. The simulator first executes an initialization phase where the initialize method of each instance of modules is run. A phase follows, in which the handleMessage methods of each module are run in sequence. This method checks the presence of message at the gates. Depending on this status, the simulator either runs the sequential code of the module or waits for the next simulation step.

At a fine grain, various policies for simulation exist, which can be chosen: event-by-event, normal, fast, express, run until (a scheduled event, any event in a module, or given simulation time).

Once the simulation has been run, the result can be analyzed. The Analysis file (.ANF extension), a trace of simulation, has been stored either in a vector or scalar form depending on the format defined in the NED and initialization file. The statistical methods can be applied on the results to extract the relevant information and to plot the final result.

Concerning the technological aspects the simulation can be run from:

- **Command line (Cmdenv):** adequate for big and batch simulation, it does not provide any graphical or visual view of the simulation.
- **Tkenv:** a graphical user interface used in the development stage of the simulation and also for presentations. Tkenv allows to debug and trace the simulation at any point of the execution and to provide a batch simulation.
- **Qtenv:** supports interactive simulation results, animation, inspection, tracing and debugging. In addition to model development and verification, it allows a detailed picture of the simulation state and it logs its execution.

4.2 oneM2M Components

4.2.1 oneM2M components to be simulated

This clause provides an overview of the components involved in the simulator of the Advanced Semantic Resource Discovery (ASRD). Some of them are standardized oneM2M components, some others are additional elements needed as additional resources or data structures for an ASRD protocol.

- Application Entity (AE): Application Entities are closely related to the end-user part; they can be sensors or actuators or they can be associated to some mobile agents controlling it, like, e.g. power metering application, or a controlling application. They constitute the end leaf of a tree made of several CSEs interconnected. AEs are the end resources of the oneM2M infrastructure. An AE can be registered or simply announced in one or multiple CSEs. This entity is the starting point for the

resource discovery as they originate the query. An AE is referenced with a unique identifier called URI.

- Common Service Entity (CSE): in a oneM2M architecture, a CSE manages multiple services such as the registration, the notification of AEs and the inter-networking services for the connexion between CSEs. A CSE plays the role of a gateway for storing resources. Each CSE has a local resource database containing information on the registered AEs and CSEs. It also plays a role of a gateway for forwarding certain queries such as those for the resource discovery. IN-CSE and MN-CSE have not been distinguished in the simulation. More or less both play the same role. A CSE is referenced with a unique identifier called URI.

4.3 Semantic Discovery Agreements between CSEs

4.3.1 Introducing new oneM2M roles for CSE and Agreements between CSEs

The Semantic Discovery Agreement (SDA) aims at adding a semantic registration information for the cooperation between CSEs. Three kinds of abstract relationships between CSEs need to be settled: CUSTOMER-to-PROVIDER (C2P), PEER-to-PEER (P2P) and SIBLING-to-SIBLING (S2S) Those relations are useful to set up Routing Paths Weight intra and interTrusted Domains and Service Providers (SP) and to define valid and invalid Advanced Semantic Discovery routing paths.

Intuitively, a Path Weight corresponds to a kind of "cost of communication", related to bandwidth, or any other efficiency parameters or even property of the link. This cost is also related to the direction of the communication and, therefore, can be either positive or negative. Thanks to the path weight, it could be possible to establish a local and global cost of communication for a given query.

The "strategy" to select the next hop heavily depends on the SDA. A simple rule of thumb, directly derived by the BGP protocol IETF RFC 4271, is as follows: a semantic routing not respecting the SDA could generate e.g. a routing path of the shape:

PROVIDER -> CUSTOMER -> PROVIDER

The associated path weight is negative. In BGP [i.15], this routing path is called a VALLEY ROUTING by Gao [6] and it is usually unwelcome for the customer that "pays" the provider generating the query but also the provider where the query is destined, in other words: it "pays" the two CSEs for an unwanted connectivity (a service that is given to the two providers instead of being payed because offering that connectivity).

Following the same approach, only no-valley routing paths are admitted in Advanced Semantic Discovery routing (which is equivalent to say that valley-routing are forbidden). Therefore, a valid path should have the following pattern: "zero or more CUSTOMER-to-PROVIDER links, followed by zero or one PEER-to-PEER link, followed by zero or more PROVIDER-to-CUSTOMER links. In addition, SIBLING-to-SIBLING links can appear in any number anywhere in the path".

4.3.2 OMNeT++ implementation of CSEs semantic discovery agreements

The semantic discovery agreements are implemented by defining roles for gates. The interconnection of gates of different types characterizes the relationships between CSEs.

```
simple CSE
{
    gates:
        inout provider[] @loose;
        inout customer[] @loose;
        inout sibling[]  @loose;
        inout peer[]     @loose;
        inout ae[]       @loose;
}
```

FIGURE 4.1: Gates SDA definition in OMNeT++

Figure 4.1 shows the definition of gates in a CSE module. CSE module may have potentially five different gates that interface the module with others CSE. One gate can be used to connect with AE.

The valley-free property of the modules is ensured by using the gates definition and properties. The weight of the associated path is managed by applying a unique and constant cost (0, 1 or -1) on paths in the source file of the CSE. The SDA table shows the Semantic Discovery Agreements between the CSEs. In the OMNeT++ implementation those CSEs discovery agreements can be inferred by using the gates keyword.

4.4 Semantic Routing Table (SRT)

4.4.1 oneM2M semantic routing table embedded in every CSE

The Semantic Routing Table (SRT) is a data structure, embedded in each CSE, that contains information to route an Advanced Semantic Query received by a CSE and routed to another CSE. The data structure should contain at least the following information:

1. Semantic Discovery Agreement (SDA)-table listing the URI of all the CSEs connected and the associated relationship (Customer-2-Provider, Peer-2-Peer and Sibling-2-Sibling);
2. FEATURE_TYPE-table listing the URI of all AEs, having the FEATURE_TYPE and directly registered (or announced) in the current CSE;
3. AE_NUMEROSITY-table: for each FEATURE_TYPE, the SRT should indicate the "number" of AEs of that type grouped by the SDA relationships.

Feature_Type	Local Database	CSE Customers	CSE Peers	CSE Siblings	CSE Providers
THERMOMETER	(URI, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)
WATER_VALVE	(URI, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)	(cse_1, #)… (cse_n, #)

FIGURE 4.2: Example of Semantic Routing Table.

Figure 4.2 shows a simple routing table, as implemented in OMNeT++. The SDA relationships are implemented in the routing table. The URI is the ID of the directly connected AE of the type. The "#" symbol in the tables means the number of the type of

resources connected via Customer, Peer, Sibling and Provider. The CSEs will send notify messages to their first neighbors to inform them about new added resources. The node has to send a notification message that a new resource of this type is added in the local database. The protocol provides the possibility to send notification messages at Level Array [customer, sibling, peer, provider]. Setting the Level Array at [1,1,1,1] means, that notifications have to be sent to all first neighbors; a Level Array of [1,1,1,2] means the notify will be forwarded to first neighbors at Customer and Peer and 2 levels towards Provider, and so on.

The Level Array is used to improve the efficiency of the routing and the view of network resources. Providers will improve visibility and availability of resources. Peers and Siblings will improve inter domain routing. Customers are helpful in routing in multiple providers.

Semantic Routing Tables are implemented in a C++ source file associated to a CSE module. The implementation relies on data structures called map which are associative containers that store elements in a mapped fashion with a unique key value to identify elements (the mapped value). Maps come with basic and efficient functions to write and search data.

In the case of routing tables, the map key is the feature type and the value is a structure named Routing Entry:

```
std::map<std::string feature_type, RoutingEntry> SemanticRoutingTable
```

The Routing Entry structure, given in Figure 4.3, composes itself with five maps. The routing entries contain, for each feature type, the URI list of customers, peers, siblings, provider and local database and the number of the resources of that feature type.

The key is an URI for these maps and the value is the number of the resources. For example (1,30) means that there are 30 resources with an URI equal to 1 of that feature type. The Routing Entry structure is shown below:

```
struct RoutingEntryStruct {
    std::map<URI,int> database;
    std::map<URI,int> CSECustomer;
    std::map<URI,int> CSEProvider;
    std::map<URI,int> CSESibling;
    std::map<URI,int> CSEPeer;
}
```

FIGURE 4.3: Semantic Routing Table data structure.

Data structure in Figure 4.3 is updated dynamically. Each time a CSE receives the registration message from the AE, it registers this AE in the local database and notifies neighbours about it. On reception of a notify message, the CSE updates its routing table accordingly. The sending of notify and registration messages helps keeping the routing table updated.

4.5 Updating Semantic Routing Tables

4.5.1 oneM2M registration and ASD notification

In oneM2M an AE registers to a CSE. The resource type or FEATURE_TYPE is stored in the local database. Every time a resource is added or deleted, the AE informs the CSE by sending a registration message which allow the CSE to keep its local database updated.

The ASD protocol uses the notification service, presented on Figure 4.4. A CSE sends a notification message to inform the neighbor CSEs about the new added/modified resources. The notify message will help the CSEs to keep their Semantic Routing Tables updated. Whenever the CSE registers some resource it will inform the neighbors about it and also in case when the resource is unregistered or deleted.

The notify message contains a condensed information defining the feature type and the number variation, e.g. :

```
NOTIFY CSEcust     WITH #+1 Feature_type
NOTIFY CSEpeer     WITH #+1 Feature_type
NOTIFY CSEsibling  WITH #+1 Feature_type
NOTIFY CSEprovider WITH #+1 Feature_type
```

FIGURE 4.4: ASD notification message.

The receiving CSE will modify its routing table accordingly. For example:

```
NOTIFY CSEcust WITH #+1 thermometer
```

After receiving this notification message from the CSEprovider, CSEcust will update its table accordingly.

Analogously, when a resource unregisters on that CSE, a corresponding notify message will be sent to all neighbor CSEs:

```
NOTIFY URI {cust1,cust2,peer,sibling,prov} WITH #-1 thermometer
```

4.5.2 OMNeT++ implementation of registration and ASD notification

In OMNeT++, the notification and registration are implemented as messages. Registration messages are sent by all AEs to the CSE and can be sent by CSEs, as shown in Figure 4.5.

```
case REGISTRATION:
        {
            AEMessage *regMsg = new AEMessage("REGISTRATION");
            // set the message fields
            regMsg->setURI(URI);
            if(ran_number <= 50)
            {
                int random = intuniform(0, feature_types.size()-2);
                feature_type = feature_types[random];
                regMsg->setFeature_type(feature_type.c_str());
            }
            else {
                int random1 = intuniform(feature_types.size()-2,
                                         feature_types.size()-1);
                feature_type = feature_types[random1];
                regMsg->setFeature_type(feature_type.c_str());
            }
            regMsg->setData(data);
            regMsg->setOp_code(REGISTRATION);
            //send to the output gate of the AE $o as output and 0
              is the number
            send(regMsg,"cse$o",0);
            break;

          }
```

FIGURE 4.5: Registration of AE.

During the registration the URI and the feature_type are set. Those information is the minimal one, needed to proceed with an ASD further. The ASD notification is a service called by a CSE to notify any modification of the routing table to the other CSEs.

Notify the neighbors is implemented as in Figure ⁞refNMG.

```
/*
* notifyCSE is used to create and broadcast notification message to the
neighbours.
*/
void CSE::notifyCSE(std::string feature_type, int delta) {

    EV << "inside notify\n";
//assemble message
    auto msg = generateMessage(NOTIFY);
    msg->setFeature_type(feature_type.c_str());
    msg->setDelta(delta);

// send to CSEs
    notifyNeighbors(msg);
}

/*
* notifyNeighbors is used to broadcast notification to all neighbours,
* excluding the neighbour that sent the message to the current CSE.
* Also, populates gate vector of the message with the arrival gate.
*/
void CSE::notifyNeighbors(ASDMessage *msg) {
    std::vector<cGate*> gateVector = msg->getGateVector();
    //You update the discoveryMessage with this object
    msg->setGateVector(gateVector);

    if (msg->getArrivalGate() != nullptr) {
        gateVector.push_back(  msg->getArrivalGate()->getOtherHalf());
                               msg->setGateVector(gateVector);
    }

    EV << "sending messages to downstream\n";
    multicast ("customer", msg);

    EV << "sending messages to sidestream\n";
    multicast("peer", msg);
    multicast("sibling", msg);

    EV << "sending messages to upstream\n";
    multicast("provider", msg);
    delete msg;
}
```

FIGURE 4.6: Notification message generation.

4.6 Semantic Discovery Routing (SDR)

4.6.1 oneM2M unicast routing: efficiently forwards a query to the next CSE hop by inspecting the routing table

The Semantic Discovery Routing Mechanism (SDRM) is a software entity embedded in each CSE, that listens for an Advanced Semantic Discovery Query (ASDQ) and:

1. Reduces and fragments an Advanced Semantic Discovery Query compound query into unitary queries.

2. Solves and forwards the queries in a distributed way.

3. Reconstructs the partial results, sending back to the originator of the Advanced Semantic Discovery Query (ASDQ).

4. In case of Advanced Semantic Discovery Routing, the next hop for query is chosen based on the contents of Semantic Routing Table (SRT). The records are inspected and searched in the map with the FEATURE_TYPE key. Then, columns of the record are inspected by the relationship type between CSEs - Customer, Sibling, Peer, Provider.

The order of prevalence is chosen by using the following list:

1. Customer
2. Sibling
3. Peer
4. Provider

This order has been pre-chosen based on potential weight of communication and type of the relationship between CSEs. Similarly to the BGP protocol [i.15], valley-free routing has to be achieved programmatically. In other words, if record for sought resource is present in Customer column, the Semantic Routing Table (SRT) lookup will be stopped, and next hop will be deduced from the Customer column.

4.6.2 oneM2M multicast routing: efficiently chooses and forwards a query to many CSE by inspecting the routing table

Multicast routing

Multicast strategy is used when a Semantic Routing Table (SRT) does not have any indication of records that satisfy the query. Multicast routing is parameterized with 4 values:
Alpha: maximal number of Customers to reroute query to;
Beta: maximal number of Providers to reroute query to;
Gamma: maximal number of Siblings to reroute query to;
Delta: maximal number of Peers to reroute query to.
These four independent parameters are used to optimize the routing. These parameters are based on the current topology, the relationships between CSEs and the network size. Multicast routing still complies with the valley-free routing.

The multicast can unfortunately generate loops during the forwarding. In the current case these loops are detected and the routing is organized in a loop-free manner. Loop-free routing is implemented to ensure the most efficient way of querying the system, e.g. to avoid same node processing query more than once. Loop-free property is achieved using two distinct approaches:

- Buffer of queries processed, e.g., redirected, by the CSE;
- Query path memorization.

4.6.3 OMNeT++ implementation of unicast and multicast Semantic Discovery Routing

The semantic routing determines the best match for the query based on records in the semantic routing table. It will return the best fit for the first match, with respect to link hierarchy.

Figure 4.7 presents some OMNeT++ code of the routing. First, the routing lookup in the local database. If no resources can be found, the code looks in the routing table data structure to check for the resource, starting with the customers, then the Siblings, the Peers and finally the Providers.

```
std::vector<URI> CSE::routeQuery(discoveryMessage *msg) {
    std::string feature_type = msg->getFeature_type();
    std::vector<URI> URI_Found;
    auto it = this->SemanticRoutingTable.find(feature_type);
    if (it == this->SemanticRoutingTable.end()) {
        EV << "feature type does not exist" << "\n";
        return URI_Found;
    }

    if (it->second.CSECustomer.size() > 0) {
        for (auto cit = it->second.CSECustomer.begin();
                cit != it->second.CSECustomer.end(); cit++) {
            URI_Found.push_back(cit->first);
        }
        return URI_Found;
    }
    if (it->second.CSESibling.size() > 0) {
        for (auto sit = it->second.CSESibling.begin();
                sit != it->second.CSESibling.end(); sit++) {
            URI_Found.push_back(sit->first);
        }
        return URI_Found;
    }
    if (it->second.CSEPeer.size() > 0) {
        for (auto sit = it->second.CSEPeer.begin();
                sit != it->second.CSEPeer.end(); sit++) {
            URI_Found.push_back(sit->first);
        }
        return URI_Found;
    }
    if (it->second.CSEProvider.size() > 0) {
        for (auto pit = it->second.CSEProvider.begin();
                pit != it->second.CSEProvider.end(); pit++) {
            URI_Found.push_back(pit->first);
        }
        return URI_Found;
    }

    return URI_Found;
}
```

FIGURE 4.7: Checking the routing table for solving the query.

If no resources are found and the resulting vector is empty, it means that SRT does not have any records satisfying the query. In that case, a multicast routing is launched. Figure 4.8 shows the code for multicasting the query.

```
/*
* fallbackRouteQuery is used when semantic routing fails
* (i.e. semantic routing table lookup returns no results)
* It multicasts query with coefficients.
* It routes query in valley-free manner.
*/
void CSE::fallbackRouteQuery(discoveryMessage *msg) {
    int D = msg->getDirection();

    bool successful = false;

    /*
     * We need to send response only if all of the broadcasts have failed
     *
     * Thus, we are performing logical AND between all invocations of broadcast
     *
     * If all of them fail - we will send response
     * */
    switch (D) {
    case DOWN: {
        successful = multicast("customer", msg, this->multicastAlpha);
        successful =
                !successful ?
                        multicast("sibling", msg, this->multicastGamma) : true;
        break;
    }
    case SIDE_SIBLING: {
        successful = multicast("sibling", msg, this->multicastGamma);
        successful &= multicast("customer", msg, this->multicastAlpha);
        break;
    }
    case SIDE_PEER: {
        break;
    }
    case UP: {
        successful = multicast("provider", msg, this->multicastBeta);
        successful =
                !successful ?
                        multicast("sibling", msg, this->multicastGamma) : true;
        successful &= multicast("customer", msg, this->multicastDelta);
        break;
    }
    default:
        break;
    }

    if (!successful) {
        generateResponseMessage(msg, ResultCode::NOT_FOUND);
    }
}
```

FIGURE 4.8: Propagating the query on multicast for solving the query.

Processed queries buffer implementation in OMNeT++. The so called queryKey holds pair - URI of the sender of the query and sequential query ID. These two values are sufficient to uniquely identify the query. The processed query buffer is represented via triplets - URI of query sender, queryID, TTL (end of life for the query).

This approach allows for so-called Loop-Free Routing. Having a local buffer of processed queries allows to drop already seen queries, thus drastically reducing request flood caused by single query. The desired effect is achieved by numbering queries sent by nodes in increasing manner. The (URI, queryID) pair allows to uniquely identify a query sent by the node. A processed query buffer will be useful for buffering results and for

performing batch updates in the network - regarding notification about resource registration or cancellation, Simulation implementation imposes certain limitations, which may be absent in real world system. Nonetheless, this approach can be adopted in order to achieve the same effect, reducing overhead and relieving load imposed on the network. Loop-Free Routing is an essential part of the proposed Advanced Semantic Routing implementation:

```
typedef std::pair<URI, int> queryKey;
std::map<queryKey, int64_t> processedQueries;
```

Checking for query in local buffer is showed in Figure 4.9.

```
/*
* seenQuery is used to check whether the query being processed was previously
processed.
* It checks the local query buffer for the query ID.
* Also, performs cleanup of stale buffer records.
*/
bool CSE::seenQuery(discoveryMessage *msg) {
    std::map<queryKey,int64_t> newProcessed(this->processedQueries);
    for (auto record : newProcessed) {
        if (record.second > simTime().inUnit(SimTimeUnit::SIMTIME_S)) {
            this->processedQueries.erase(record.first);
        }
    }

    queryKey key;
    key.second = msg->getQueryID();
    key.first  = msg->getURI_init();

    if (this->processedQueries.find(key) != this->processedQueries.end()) {
        return true;
    }

    return false;
}
```

FIGURE 4.9: Sending the query to the appropriate CSE.

4.7 Semantic Recommendation System (SRS)

4.7.1 oneM2M Semantic Recommendation System embedded in each CSE

The Semantic Recommendation System (SRS) is a piece of software embedded in each CSE. The SRS grants an access to the SRT and can be seen as an observer of each query and a help to the CSE to set up the next hop to be taken. Intuitively, when a CSE receives an Advanced Semantic Query, the SRS will look up the SDA-table and Semantic Routing Table and it will take the best possible decision to find the next hop for the query. The process can be succinctly descripted in points as follows:

1. read the SRT (and the SDA-Table encoded in the SRT of the OMNeT++ implementation);
2. take the best decision to choose the next hop(s) for the Advanced Semantic Query;
3. annotate all the CSEs belonging to the SRT that participate pro-actively to successful routing.

The SRS will be empty, when the simulation starts and will be updated upon reception of each Semantic Query reply. The approach and the behavior of a SRS is inspired from the Kademlia P2P overlay network ETSI

One column in the SRT is added, namely CSE_BUCKET which will have the number of the CSEs which successfully participated in the routing before as shown in Figure 4.10 below.

For each FEATURE_TYPE, SRT[FEATURE_TYPE][CSE-RECOMMENDED] is set as an ordered LIST[100][CSE]. As in Kademlia, those lists are called BUCKETS and in oneM2M CSE_BUCKETS[FEATURE_TYPE].

The algorithm can be sketched as follows (see ETSI TR 103 715 [i.2]):

1. IF the CSE answers successfully to an Advances Semantic Query for that TYPE and already is in the:
   - BUCKET[FEATURE_TYPE], THEN move it to the tail of the list;
2. OTHERWISE:
   - IF the BUCKET[FEATURE_TYPE] has fewer than 100 entries, THEN inserts the news CSE at the tail of the list;
3. OTHERWISE:
   - ping the least-recently seen CSE;
   - IF the least-recently seen CSE fails to respond, THEN it is evicted from BUCKETS[FEATURE_TYPE] and the new CSE is inserted at the tail;
4. OTHERWISE: it is moved to the tail of the list, and the new CSE is discarded.

Feature_Type	Local Database	CSE_BUCKETS	CSE Customers	CSE Peers	CSE Siblings	CSE Providers
THERMOMETER	(URI, #)	CSE_1 … CSE_x	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)
WATER_VALVE	(URI, #)	CSE_1 … CSE_y	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)	(CSE_1, #)… (CSE_n, #)

FIGURE 4.10: Semantic Routing Table with the CSE_BUCKETS.

Chapter 5

ASDQ Performance Evaluation on OMNeT++

In this chapter the aim is to discuss about the performance evaluation of the Advanced Semantic Discovery and Query (ASDQ). First, the terminology will be discussed to be clarified what are the meaning of abbreviations which are used in simulation, and then, some explanation will be given about how simulation works alongside the final results related to the ASDQ.

5.1 Terminology

The list of definitions for some abbreviations are as follow:

Application Entity (AE): An IoT device in the network, it can be user or resource which is directly connected to the CSE.

Common Service Entity (CSE): An entity in the network that has specific functions and serving the AEs and CSEs which are connected to it. Routing the query and saving the AEs information are the examples of these functions.

Notification Depth (ND): When the AE is connecting to the network, first, it registers itself in the local CSE and then the CSE notifies the other CSEs which are its neighbors. Notification depth equal to 1 means that CSE only notifies about its AEs just to the connected neighbors as Customer or Provider or Peer or Sibling. Notification depth equal to 2, means, CSE notifies its resources to the neighbors of the neighbors as well.

Time to Live (ttl): Number of hops that the query is allowed to investigate the network.

Success Rate [%]: Percentage that defines the quantity of the resources which are discovered, out of total number of resources in the network.

Alpha (α): Downstream multicast parameter, provider to customer (p2c). The number of customer CSEs which are in possession of resources. For instance, if the alpha is 1, it means, the CSE can route the query just to one of its customer CSEs which have the highest number of resources. CSEs in routing table are sorted based on the number of resources they have, in decreasing manner.

Beta (β): Sidestream multicast parameter, sibling to sibling (s2s). The number of sibling CSEs which are in possession of resources. For instance, if the beta is 1, it means, the CSE can route the query just to one of its sibling CSEs which have the highest number of resources. CSEs in routing table are sorted based on the number of resources they have, in decreasing manner.

Gamma (γ): Upstream mlticast parameter, customer to provider (c2p). The number of provider CSEs which are in possession of resources. For instance, if the gamma is 1, it means, the CSE can route the query just to one of its provider CSEs which have the

highest number of resources. CSEs are sorted based on the number of resources they have, in decreasing manner.

Delta (δ): Sidestream multicast parameter, peer to peer (p2p). The number of peer CSEs which are in possession of resources. For instance, if the delta is 1, it means, the CSE can route the query just to one of its peer CSEs which have the highest number of resources. CSEs are sorted based on the number of resources they have, in decreasing manner.

BGP: Border Gate Protocol.

Objective Modular Network Testbed in C++ (OMNeT++): It is a Discrete Event Simulator for Networks.

5.2 Simulation and Results

In this section, the procedure of the ASDQ during the simulation on the OMNeT++ platform will be explained and the graphs will be presented as results/proof of the simulation. Figure 5.7 shows a sample network with small amount of entities in order to be visible in OMNeT++

5.2.1 Simulation procedure in brief

In reality, all of the IoT devices or AEs register themselves in the CSEs whenever they connect to the network regardless of either the queries are running in network or they are going to start after registration. In our simulation we took the second scenario as the starting point of the simulation, and therefore, the query starts after registering all the AEs or resources in the network. During the query process, registering or unregistering the AEs or resources is possible. In order to explain the results which are taken from the simulation, we start to explain from the beginning, that is, registering the AEs in CSEs.

As it is mentioned before, first, AEs register themselves to the CSEs which are connected to them, after registering, every CSE notifies its resources based on Notification Depth to the neighbors which are as customers, providers, siblings, and peers. After all of these processes the query starts from the random AE or AEs in the network as an user requesting for resources, this request can be asking for either all resources in the network (exhaustive) or defined number of resources in the network(non-exhaustive). In our case, since the aim is to show the strength of the protocol, therefore, all results are based on **exhaustive** discovery. By starting the discovery and query, AE asks its parent CSE for the resources with defined type, then, the parent CSE reroutes the query to its neighbors based on α, β, γ, and δ parameters.

The ASDQ protocol is the BGP-Aware protocol in which the mentioned parameters (α, β, γ, δ) are adjusting the ASDQ protocol to be compatible with the **No-Valley** routing based on BGP.

5.2.2 An example

For instance, in figure 5.1 notification depth (**ND**) is equal to 1 and there are four different curves with different parameters in percentage. The violet curve has just [α] parameter equal to 10%. It means every CSE routes the query just to 10% of its customer CSEs which are its neighbors.

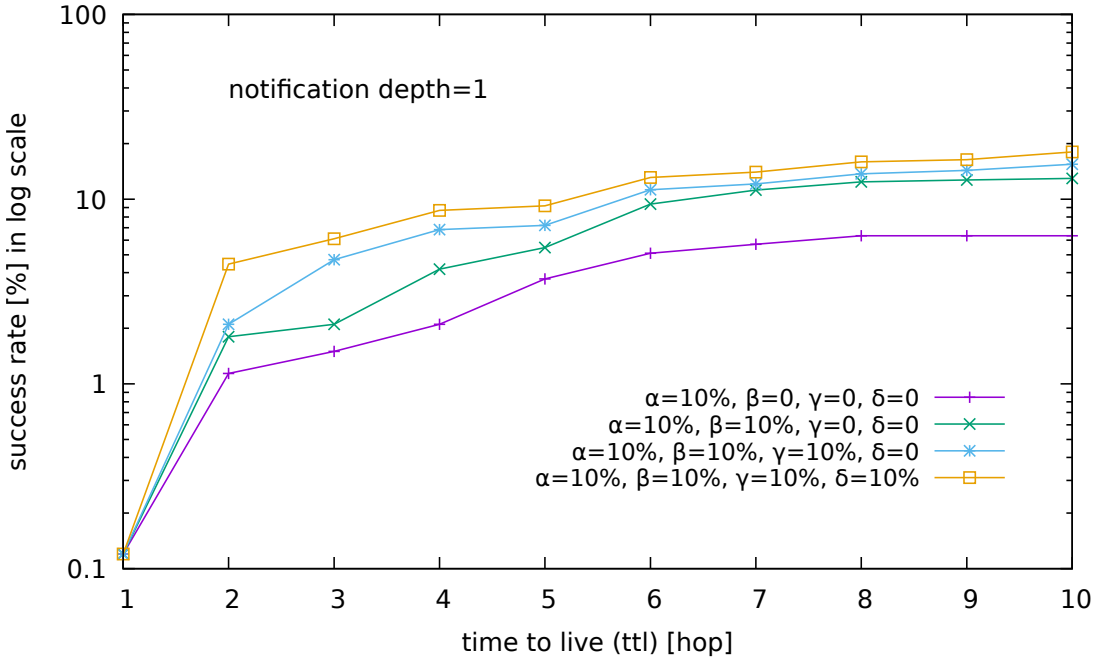

FIGURE 5.1: success rate vs. time to live, ND=1

5.3 Conclusion

It is plane to understand that in figures 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6 the success rate is increases by increasing the routing parameters and especially **time to live (ttl)** in every trend (curve) which is very important in routing the query to far distances even to other **Service Providers** to retrieve the resources information.

Figures 5.3, 5.4, 5.5, and 5.6 are on the next pages.

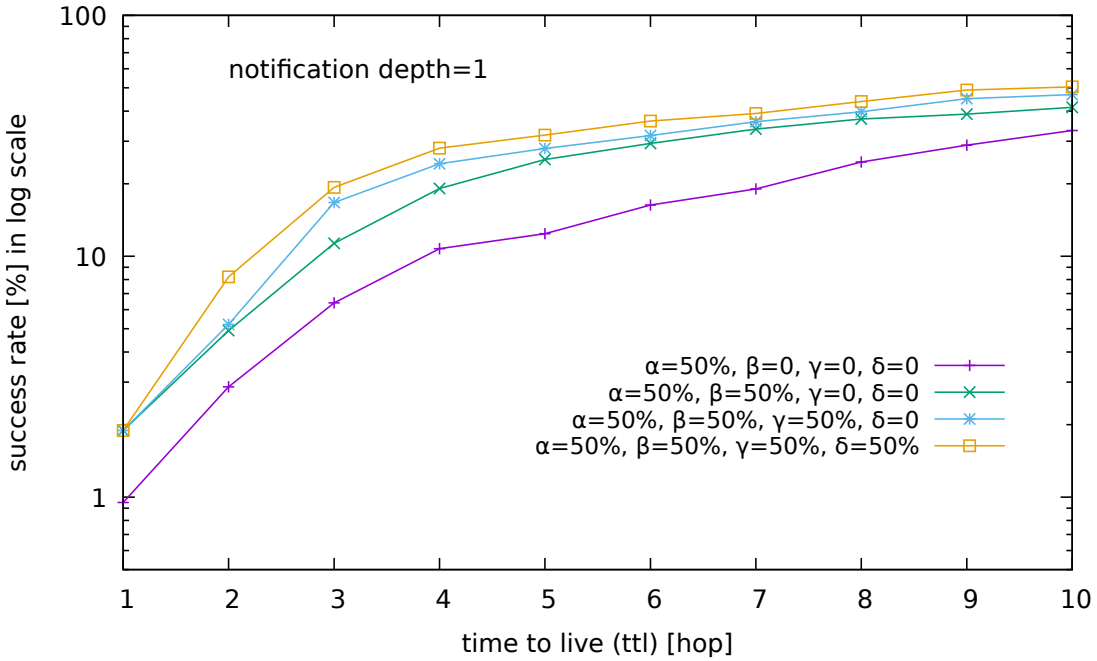

FIGURE 5.2: success rate vs. time to live, ND=1

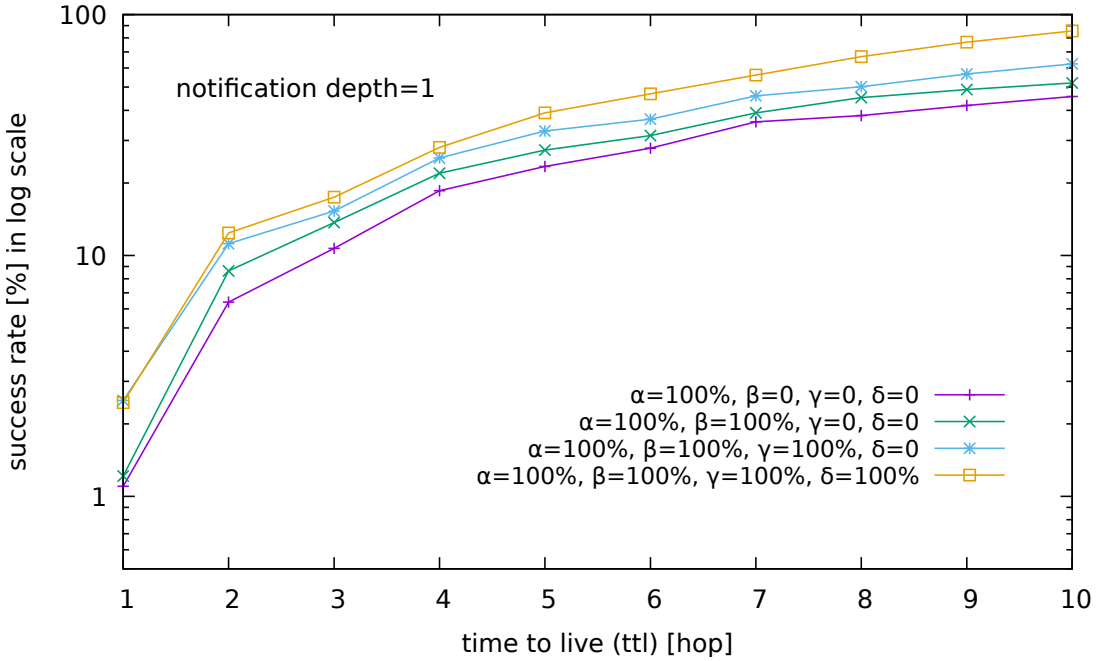

FIGURE 5.3: success rate vs. time to live, ND=1

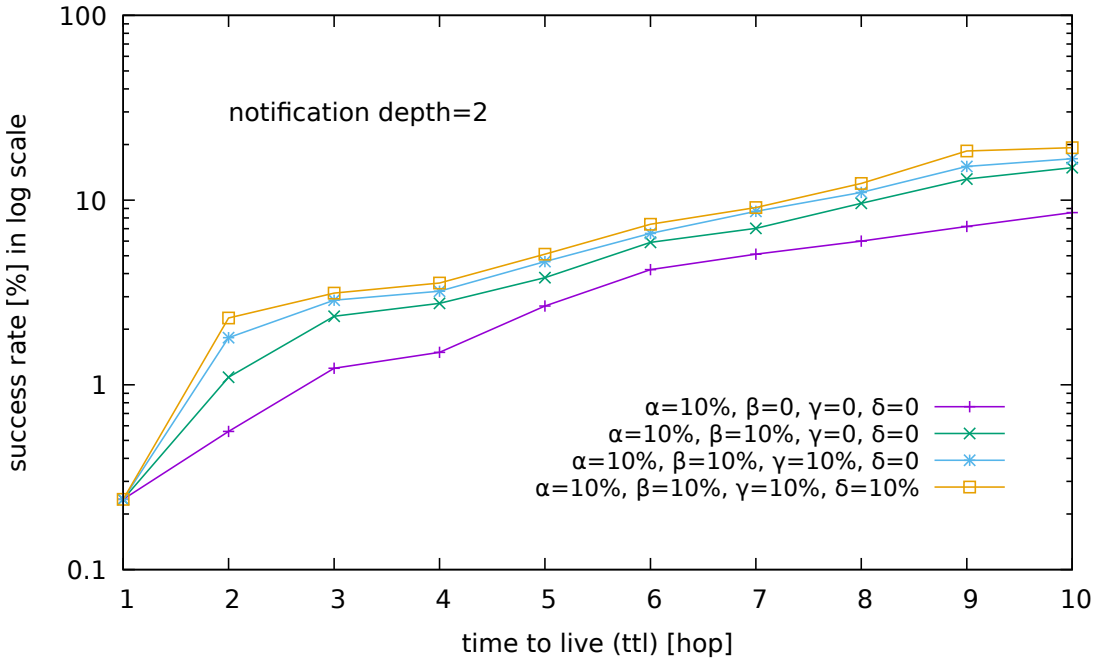

FIGURE 5.4: success rate vs. time to live, ND=2

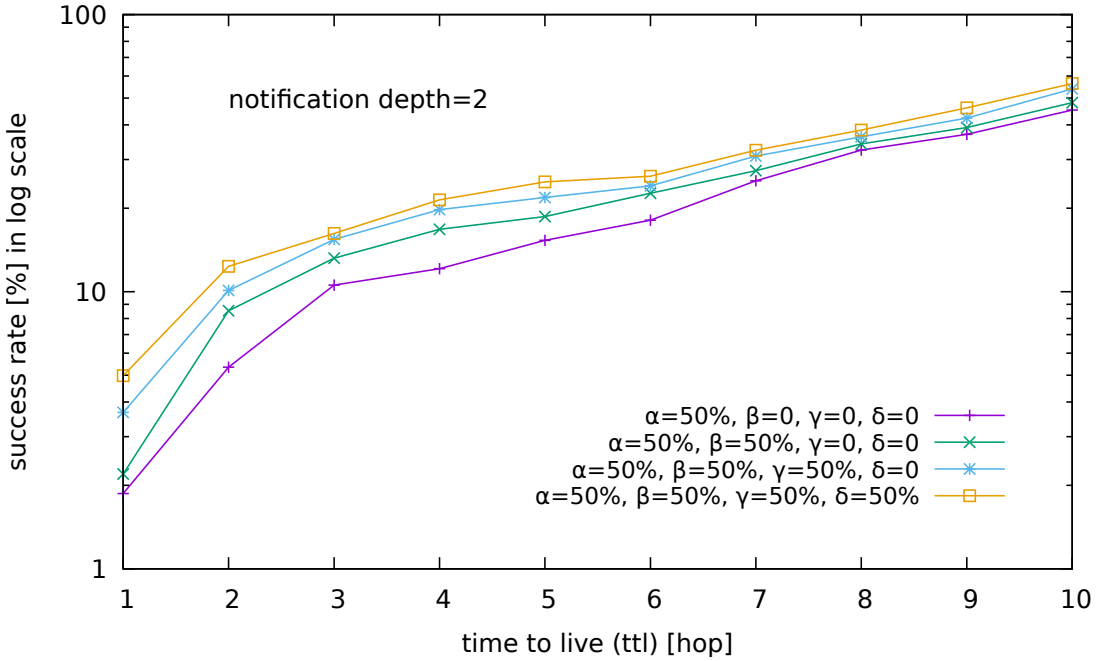

FIGURE 5.5: success rate vs. time to live, ND=2

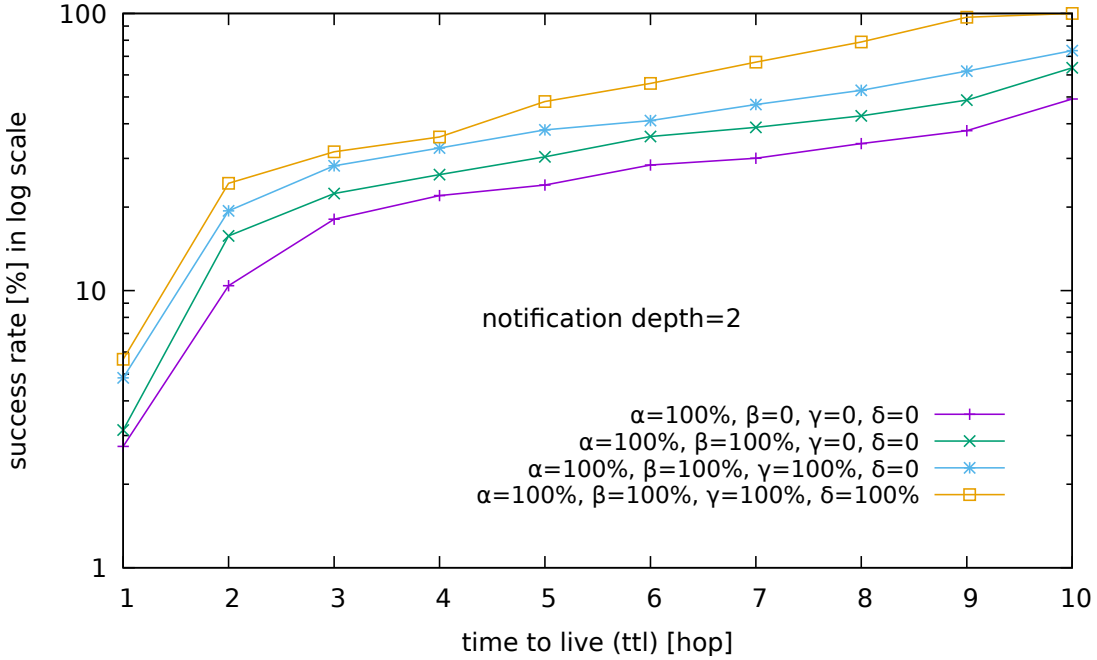

FIGURE 5.6: success rate vs. time to live, ND=2

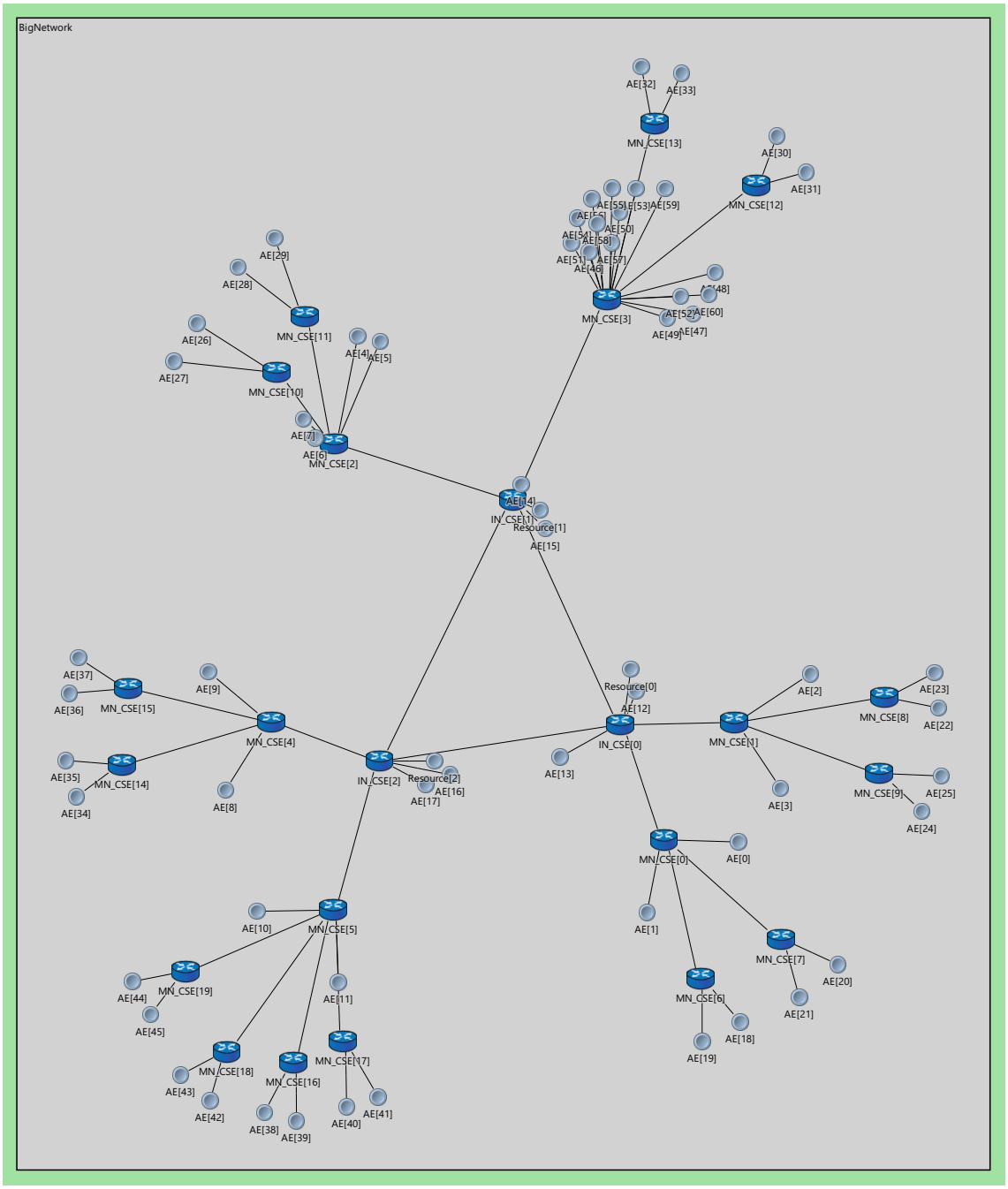

FIGURE 5.7: sample network, this network has less entities in order to be visible.

References

[1] URL: https://www.onem2m.org/.

[2] *ETSI TR 103 714*.

[3] *ETSI TR 103 715*.

[4] *ETSI TR 103 716*.

[5] *ETSI TR 103 717*.

[6] Lixin Gao. “"On inferring autonomous system relationships in the internet. IEEE/ACM Trans.” In: *Netw.9(6)* 733-745 (2001).