ETSI SmartM2M Technical Report 103716; oneM2M Discovery and Query solution(s) simulation and performance evaluation

Luigi Liquori, Marie-Agnès Peraldi-Frati, Andrea Cimmino, Raúl García Castro, Abdul Qadir Khan, Sara El Khatab, Oleksii Khramov, Enrico Scarrone, Joachim Koss

ETSI TR 103 716 V1.1.1 (2021-04)

SmartM2M;
oneM2M Discovery and Query solution(s)
simulation and performance evaluation

Reference
DTR/SmartM2M-103716

Keywords
interoperability, IoT, oneM2M, SAREF, semantic, simulation

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org/).

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™** and **LTE™** are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM®** and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

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

Modal verbs terminology

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

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

Introduction

oneM2M has currently native discovery capabilities that 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 function may be implemented with different flavours and this is not optimal for interworking and interoperability.

The objective of the present document [i.3] in conjunction with three other ones ETSI TR 103 714 [i.1], ETSI TR 103 715 [i.2] and ETSI TR 103 717 [i.4] is the study and development of semantic Discovery and Query capabilities for oneM2M and its contribution to the oneM2M standard.

The goal 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 to directly search information in the oneM2M system for big data purposes.

1 Scope

1.1 Context for the present document

In order to enhance the semantic capabilities of the oneM2M architecture by providing solid contributions to the oneM2M standards, four Technical Reports have been developed. Each of them is the outcome of a special study phase.

The study and development of semantic Discovery and Query capabilities for oneM2M and its contribution to the oneM2M standard is composed of four phases:

1) A **requirements** phase where requirements and use cases are formally identified and defined. As a minimum, this work includes discovery of specific information and of aggregated information, and interaction with external sources of data and queries. The oneM2M architecture ETSI TS 118 101 [i.6], the oneM2M semantic approach oneM2M TS-0034 [i.7], the current oneM2M capabilities and SAREF ETSI TS 103 264 [i.5], ETSI TS 118 101 [i.6], oneM2M TS-0034 [i.7], ETSI TS 103 410-1 [i.19], ETSI TS 103 410-2 [i.20], ETSI TS 103 410-3 [i.21], ETSI TS 103 410-4 [i.22], ETSI TS 103 410-5 [i.23], ETSI TS 103 410-6 [i.24], ETSI TS 103 410-7 [i.25], ETSI TS 103 410-8 [i.26], ETSI TS 103 410-9 [i.27], ETSI TS 103 410-10 [i.28] are at the basis of these use cases and requirements. This work is documented in ETSI TR 103 714 [i.1].

2) A **study** phase where possible approaches (existing and new ones) to a discovery and data aggregation solution are analysed with respect to the use cases and requirements. In particular, the need to plug in the solution on the oneM2M standard drives the solution analysis, to determine the best approach to be followed. The present document also looks to the query and discovery mechanisms already available, starting from the ones defined by ETSI (e.g. the one included in NGSI-LD ETSI GS CIM 009 [i.14]) to extract (and potentially adapt) the applicable components and to assure a smooth interworking with non-oneM2M solutions. This is documented in ETSI TR 103 715 [i.2].

3) A **simulation** phase is conducted in parallel and "circular" feedback with respect to the study phase, with the goal to provide a proof of concept, run suitable scenarios provided by previous phases and a performance evaluation to support the selection/development of the Discovery and Query solution. The simulator and the simulation results are documented in ETSI TR 103 716 [i.3] (the present document). An extract of the simulation results is included ETSI TR 103 715 [i.2] and ETSI TR 103 717 [i.4]. A selection of the use cases includes a set of oneM2M relevant configurations scenarios to be considered for the simulation activity described below.

4) A **standardization** phase where the Discovery and Query solution is specified and documented in ETSI TR 103 717 [i.4].

The present document covers the third of the four phases and provides some input for other documents listed below:

- ETSI TR 103 714: "SmartM2M; Study for oneM2M Discovery and Query use cases and requirements" [i.1];
- ETSI TR 103 715: "SmartM2M; Study for oneM2M Discovery and Query solutions analysis & selection" [i.2];
- ETSI TR 103 716: "SmartM2M; oneM2M Discovery and Query solution(s) simulation and performance evaluation" [i.3] (the present document);
- ETSI TR 103 717: "SmartM2M; Study for oneM2M Discovery and Query specification development" [i.4].

1.2 Scope of the present document

The present document describes the proof of concepts developed through two complementary simulators for the Advanced Semantic Discovery (ASD). These simulators enable running suitable scenarios provided by previous study phases (see ETSI TR 103 714 [i.1] and ETSI TR 103 715 [i.2]), and a preliminary performance evaluation and routing complexity to support the selection and development of the Discovery and Query solution to be contributed to oneM2M. It documents the simulator and the simulation results. An extract of the simulation results will be included in ETSI TR 103 715 [i.2] and ETSI TR 103 717 [i.4], and will be used to support the discussion and the proposal with oneM2M.

Two simulators are presented in the present document. They have been developed in parallel and can be considered as complementary. The **first one** implements the **Advanced Semantic Discovery and their Components** in the OMNeT++ Discrete Network Event Simulation Tool [i.30]. This simulator focuses on new network resources and routing features needed for extending the existing discovery mechanism of the oneM2M system. The simulator demonstrates an implementation of the ASD in the context of different Service Providers, in an enhanced topology and associated with different Key Performance Indexes (KPIs).

The **second simulator** is the **Distributed Semantic Resource Directory (DSRD)** that focuses on the underlying format to represent, store, and query resources. The simulator results advocate that the existing Semantic Query mechanism from oneM2M, which is aligned with the W3C SPARQL protocol, is more efficient for discovery than the current semantic discovery oneM2M mechanism, which is also not aligned with the W3C Standard [i.13]. As a result, for the sake of efficiency and standard alignment the current Semantic Query mechanism could be upgraded to become the new oneM2M semantic discovery mechanism. The simulation defines performance indexes that provides quantitative values for comparing the different approaches and reach to such conclusions.

Both simulators are not currently interconnected but they demonstrate results on the routing mechanisms and resources storage for a common Advanced Semantic Discovery proposal in oneM2M.

The present document is structured as follows:

- Clauses 1 to 3 set the scene and provide references as well as definitions of terms, symbols and abbreviations, which are used in the present document.
- Clause 4 provides information about the OMNeT++ [i.13] discrete event simulation tool, and provides detailed information about the implementation of the Advanced Semantic Discovery protocol.
- Clause 5 presents simulation result of executing suitable scenarios in OMNeT ++.
- Clause 6 presents the Distributed Semantic Resource Directory Simulator (DSRDS).
- Annex A refers to some code of the simulators and extra material.

2 References

2.1 Normative references

Normative references are not applicable in the present document.

2.2 Informative references

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

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

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

[i.1] ETSI TR 103 714: "SmartM2M; Study for oneM2M Discovery and Query use cases and requirements".

[i.2] ETSI TR 103 715: "SmartM2M; Study for oneM2M Discovery and Query solutions analysis & selection".

[i.3] ETSI TR 103 716: "SmartM2M; oneM2M Discovery and Query solution(s) simulation and performance evaluation".

[i.4] ETSI TR 103 717: "SmartM2M; Study for oneM2M Discovery and Query specification development".

[i.5] ETSI TS 103 264: "SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping".

[i.6] ETSI TS 118 101: "oneM2M; Functional Architecture (oneM2M TS-0001)".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=31496.

[i.7] oneM2M TS-0034: "Semantics Support".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=31425.

[i.8] IETF RFC 6762 (2013): "Multicast DNS", S. Cheshire & M. Krochmal.

NOTE: Available at https://www.rfc-editor.org/rfc/rfc6762.txt.

[i.9] IETF RFC 6690 (August 2012): "Constrained RESTful environments (CoRE) link format", Z. Shelby, (p. 30).

[i.10] W3C Recommendation 9 April 2020: "Web of Things (WoT) Thing Description", Kaebisch S., Kamiya T., McCool M., Charpenay V., Kovatsch M.

[i.11] oneM2M TR-0057 (V0.2.0): "Getting Started with oneM2M".

NOTE: Available at https://ftp.onem2m.org/Work%20Programme/WI-0089/TR-0057-Getting_started_with_oneM2M-V0_2_0.DOC.

[i.12] oneM2M TR-0045 (V0.3.1): "Developer Guide Implementing Semantics".

NOTE: Available at https://www.onem2m.org/component/rsfiles/download-file/files?path=Draft_TR%5CTR-0045-Developer_Guide_Implementing_Semantics-V0_3_1.docx.

[i.13] W3C Recommendation (2008): "SPARQL query language for RDF", Prud'hommeaux E. & Seaborne A.

[i.14] ETSI GS CIM 009: "Context Information Management (CIM); NGSI-LD API".

NOTE: Available at https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.04.01_60/gs_CIM009v010401p.pdf.

[i.15] IETF RFC 4271: "Border Gateway Protocol 4 (BGP4)", January 2006.

NOTE: Available at https://tools.ietf.org/html/rfc4271.

[i.16] Elmagarmid A.K., Ipeirotis P.G., Verykios V.S.: "Duplicate record detection: A survey". IEEE™ Trans. on Knowl. and Data Eng. 19(1), 1-16 (2007).

[i.17] Abdul Qadir Khan: "Semantic Resource Discovery and Security for Internet of Things (IoT)", Master Thesis, Université Côte d'Azur, December 2020.

[i.18] Sara El Khatab: "Semantic Resource Discovery and Security for Internet of Things (IoT)", Master Thesis Université Côte d'Azur, December 2020.

[i.19] ETSI TS 103 410-1: "SmartM2M; Extension to SAREF; Part 1: Energy Domain".

[i.20] ETSI TS 103 410-2: "SmartM2M; Extension to SAREF; Part 2: Environment Domain".

[i.21] ETSI TS 103 410-3: "SmartM2M; Extension to SAREF; Part 3: Building Domain".

[i.22] ETSI TS 103 410-4: "SmartM2M Extension to SAREF Part 4: Smart Cities Domain".

[i.23] ETSI TS 103 410-5: "SmartM2M; Extension to SAREF Part 5: Industry and Manufacturing Domains".

[i.24] ETSI TS 103 410-6: "SmartM2M; Extension to SAREF; Part 6: Smart Agriculture and Food Chain Domain".

[i.25] ETSI TS 103 410-7: "SmartM2M; Extension to SAREF; Part 7: Automotive Domain".

[i.26] ETSI TS 103 410-8: "SmartM2M; Extension to SAREF; Part 8: eHealth/Ageing-well Domain".

[i.27] ETSI TS 103 410-9: "SmartM2M; Extension to SAREF; Part 9: Wearables Domain".

[i.28] ETSI TS 103 410-10: "SmartM2M; Extension to SAREF; Part 10: Water Domain".

[i.29] Vandenbussche P. Y., Atemezing G. A., Poveda-Villalón M. & Vatant B. (2017): "Linked Open Vocabularies (LOV): a gateway to reusable semantic vocabularies on the Web". Semantic Web, 8(3), 437-452.

[i.30] OMNeT++ Simulation Manual.

NOTE: Available at https://doc.omnetpp.org/omnetpp/manual/.

[i.31] RDF 1.1 Primer. World Wide Web Consortium (2014).

[i.32] Luigi Liquori, Rossano Gaeta and Matteo Sereno: "A Network Aware Resource Discovery Service", EPEW 2019 - 16th European Performance Engineering Workshop, November 2019, Milano, Italy, Volume 12039 of Lecture Notes in Computer Science, Springer Verlag, pages 84-99, 2019.

3 Definition of terms, symbols and abbreviations

3.1 Terms

Void.

3.2 Symbols

Void.

3.3 Abbreviations

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

ADN	Application Dedicated Node
AE	Application Entity
AE-IN	Application Entity Infrastructure Node
API	Application Program Interface
ASD	Advanced Semantic Discovery
ASDQ	Advanced Semantic Discovery Query

ASDQL	Advanced Semantic Discovery Query Language
ASRD	Advanced Semantic Resource Discovery
BGP	Border Gateway Protocol
C2P	CUSTOMER-to-PROVIDER
CAIDA	Centre for Applied Internet Data Analysis
CNF	Conjunctive Normal Form
CRUD	Create, Read, Update, Delete
CSE	Common Service Entity
DDoS	Distributed Denial of Service
DNF	Disjunctive Normal Form
DSRD	Distributed Semantic Resource Directory
DSRDS	Distributed Semantic Resource Directory Simulator
ETSI	European Telecommunications Standards Institute
ID	Identifier
IN-CSE	Infrastructure Node Common Service Entity
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Index
M2M	Machine-to-Machine
mDNS	multicast Domain Name System
MN-AE	Middle Node Application Entity
MN-CSE	Middle Node Common Service Entity
NED	NEtwork Description
OWL	Web Ontology Language
P2P	PEER-to-PEER
RDF	Resource Description Framework
RFID	Radio-Frequency IDentification
S2S	SIBLING-to-SIBLING
SAREF	Smart Applications REFerence ontology
SDA	Semantic Discovery Agreements
SDG	Semantic Descriptor Generator
SDQ	Semantic Discovery Query
SDQL	Semantic Discovery Query Language
SDR	Semantic Discovery Routing
SDRM	Semantic Discovery Routing Mechanism
SP	Service Provider
SPARQL	Simple Protocol and RDF Query Language
SQR	Semantic Query Resolution
SQRM	Semantic Query Resolution Mechanism
SQRS	Semantic Query Resolution System
SR	Semantic Recommendation
SRD	Semantic Resource Discovery
SRS	Semantic Recommendation System
SRT	Semantic Routing Table
TR	Technical Report
TTL	Time To Live
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
W3C	World Wide Web Consortium
WoT	Web of Things
XML	eXtensible Markup Language

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

4.1 OMNeT++ discrete network event simulator

OMNeT++ [i.30] is an object-oriented discrete event network simulation framework. It provides a user-friendly interface and script/procedural and OO languages for modelling and simulating network structure and behaviours. 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 modelling 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.
- Behaviour of modules are defined in a C++ code.
- Initialization file where the parameters and the scenarios for the simulation are set [i.17].

The modelling 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 behaviours 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) [i.18]. 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 behaviour 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 analysed. 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. The latter ones are described in clause 4.4.

- **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.

"Soft resources" called mechanisms exist among these elements that can be considered as "hard resources" which participate to the Advanced Semantic Discovery Routing and update the SRT.

4.2.2 OMNeT++ implementations of oneM2M components

The present clause illustrates the main phases of implementation of the simulator. At first the implementation of oneM2M components is considered.

As mentioned in the previous clause 4.2.1, AE and CSE are the core entities that intervene in a oneM2M topology.

- **Application Entity (AE) interface:** AEs are implemented through the module name "AE". It is a generic one with an inout gate that allows to send or receive queries. AEs are connected to CSEs.

```
simple AE
{
    parameters:
        @display ("i=misc/node, #0080FF,23; is=s");
    gates:
        inout cse[] @loose;
}
```

Figure 4.2.2-1: AE module and interface definition in OMNeT++

1) **Common Service Entity (CSE) interface:** A CSE interconnects with other CSEs and AEs. Depending on the SDA between CSEs, as shown in Figure 4.2.2-2, roles are associated to gates such as provider, customer, peer, sibling and ae-gates gathering the relation into vectors. A Provider gate is used to connect to CSE-provider, a Sibling gate to connect CSE-siblings, a Peer gate to connect CSE-peers, a Customer gate to connect CSE-customers, and an AE gate to connect any CSE.

```
simple CSE
{
    parameters:
        @signal[packet_size] (type="long");
        @statistic[throughput]
(title="total_numb_of_packets";source="packet_size";record=vector,last);
        @display ("i=abstract/router, blue,9");
        int notification_depth = default (1);
        int alpha = default (1);
        int beta  = default (1);
        int gamma = default (2);
        int delta = default(1);
        int queryBufferTTL = default(2);
        //volatile double sendInterval @unit(s);
        volatile double delayTime @unit(s);

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

Figure 4.2.2-2: CSE module and interface definition in OMNeT++

The alpha, beta, gamma, and delta parameters are explained in clause 4.8 and are used during the **multicast routing** phase. Additional performance parameters, called Weight Paths, can be added in the module to characterize the routing performance.

4.3 oneM2M network topologies considered in the simulator

4.3.1 Description of the oneM2M network topologies

As represented in Figure 4.3.1-1, the topologies considered are essentially mesh networks. The topology used for the simulation is the one specified in ETSI TR 103 714 [i.1] in which the CSEs are connected in a mesh network shape.

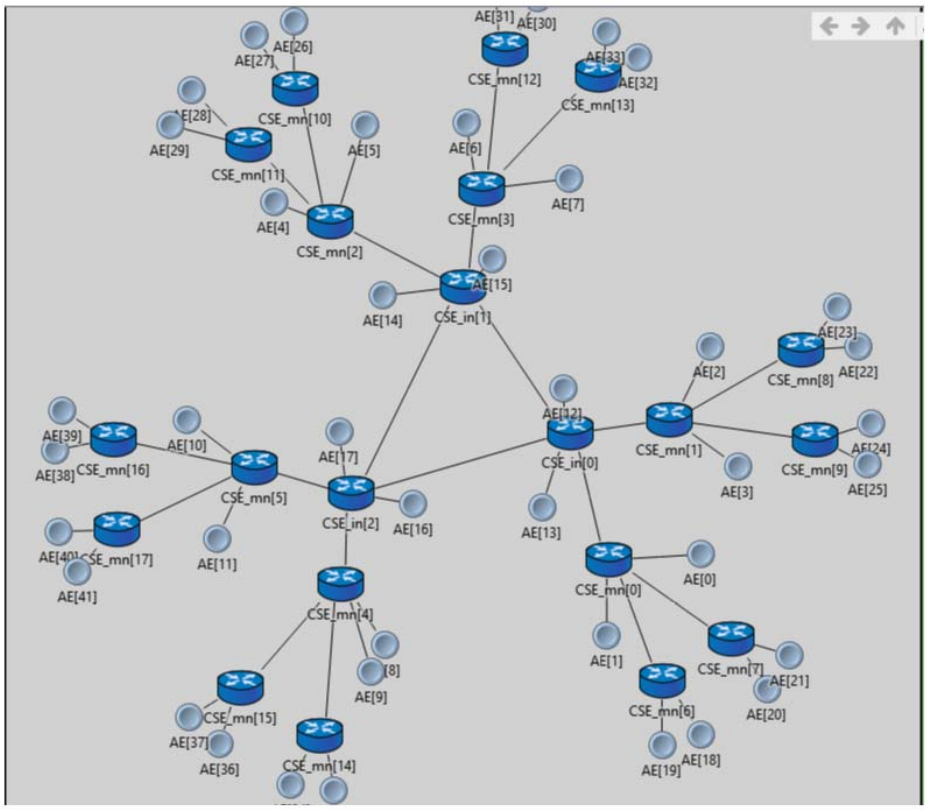

Figure 4.3.1-1: Graphical topology representation in OMNeT++

There are four types of agreements between the CSEs: **Customer**, **Provider**, **Peer** and **Sibling**. There are no peer and sibling links between CSE_MNs. The peer and sibling relations exist only at upper level of the topology. SDAs between CSEs are not clearly marked on Figure 4.3.1-1 due to lack of space, but they are defined in the NED file, which specifies the interconnecting gates. Here the topology is statically fixed, but the dynamic registration of CSEs has to be considered.

4.3.2 OMNeT++ topologies implementation

oneM2M topologies are implemented in OMNeT++ using the NED language. Gates are used as relationship between CSEs. The connection allows to connect simple modules to form a network. NED also allows to select the channel, can form a new or reuse the previous implemented channels. The NED file for the simulation is shown in Figure 4.3.2-1.

```
network SmallNetwork
{
    types:
       channel extends ned.DelayChannel {
            delay = 100ms;
        }
    submodules:
        CSE_in[3]: CSE;
        CSE_mn[6]: CSE;
          AE [12]: AE;

}
```

Figure 4.3.2-1: NED file with submodule declaration

Figure 4.3.2-1 shows a small network with three (3) CSE_INs, six (6) CSE_MNs and twelve (12) AEs. Channels encapsulate parameters and behaviour associated with connections. Channels are like simple modules, in the sense that there are C++ classes behind them. The rules for finding the C++ class for a NED channel type is the same as with simple modules: the default class name is the NED type name unless there is a `@class` property (`@namespace` is also recognized), and the C++ class is inherited when the channel is subclassed. See [i.30]. Here, channel is used in order to introduce configurable transmission delay.

Channels are defined in the Network Definition file, its usage can be seen on Figure 4.3.2-2.

```
connections allowunconnected :
        for i=0..2 {
            CSE_in[i].sibling++ <--> Channel <--> CSE_in[(i+1)%3].sibling++;
        }
        for j=0..1 {
            CSE_mn[j].provider++ <--> Channel <--> CSE_in[0].customer++;
        }
        for j=2..3 {
            CSE_mn[j].provider++ <--> Channel <--> CSE_in[1].customer++;
        }
        for j=4..5 {
            CSE_mn[j].provider++<--> Channel <--> CSE_in[2].customer++;
        }
        for j=0..1 {
            CSE_mn[0].ae++ <--> Channel <--> AE[j].cse++;
        }
        for j=2..3 {
            CSE_mn[1].ae++ <--> Channel <--> AE[j].cse++;
        }
        for j=4..5 {
            CSE_mn[2].ae++ <--> Channel <--> AE[j].cse++;
        }
        for j=6..7 {
            CSE_mn[3].ae++ <--> Channel <--> AE[j].cse++;
        }
        for j=8..9 {
            CSE_mn[4].ae++ <--> Channel <--> AE[j].cse++;
        }
        for j=10..11 {
            CSE_mn[5].ae++ <--> Channel <--> AE[j].cse++;
        }
}
```

Figure 4.3.2-2: NED modules inter-connections

NOTE: NED has some imperative constructs which enable parametric topologies with declarative constructs (loops and conditionals). With parametric topologies, NED holds a real advantage.

The key word `allowunconnected` is used when inout gates are used. It allows to run the simulation even if the *`in`* or the `out` part of the gates are not connected.

At the end of the file, the connection of the AE can be done in the same way by specifying the gate and the sub-module. E.g. in Figure 4.3.2-2 the CSE_MN is connected to a CSE_IN via a provider gate and the CSE_IN is connected to a CSE_MN via a customer gate, which defines the relationship. The result topology is shown in Figure 4.3.1-1.

4.4 Semantic Discovery Agreements between CSEs

4.4.1 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.

As an example, Figure 4.4.1-1, inspired by the oneM2M architecture, illustrates some CSEs with their respective abstract relationships.

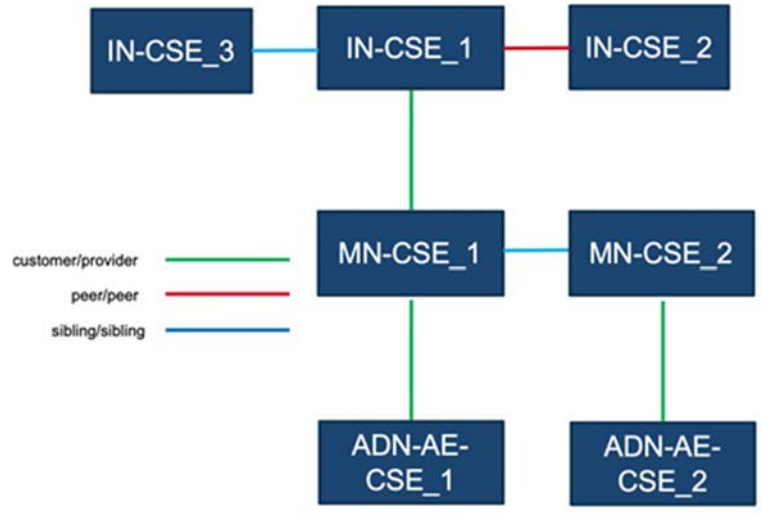

Figure 4.4.1-1: Abstract relationships between CSEs

Figure 4.4.1-1 provides the following relations:

- **CUSTOMER-to-PROVIDER (C2P)** relationship holds between the following CSEs: IN-CSE_1 <C2P> MN-CSE_1, and MN-CSE_1 <C2P> ADN-AE-CSE_1, and MN-CSE_2 <C2P> ADN-AE-CSE_2.
- **PEER-to-PEER (P2P) non transitive** relationship holds between the following CSEs: IN-CSE_1 <P2P> IN-CSE_2.
- **SIBLING-to-SIBLING (S2S) transitive** relationship holds between the following CSEs: IN-CSE_3 <S2S> IN-CSE_1 and MN-CSE_1 (S2S) MN-CSE_2.

As example: a message from C2P has a positive path weight, while a message from P2C has a negative path weight. Messages from P2P and S2S have zero path weight. These relationships and associated path weights can influence the query routing policies and routes. When a CSE receives an advanced discovery, from downstream (e.g. from one of its customers) or from upstream (e.g. from one of its providers) or from sidestream (e.g. from one of those peers) it can choose to forward the query through some (but not all) of its customers, or siblings, or peers, or providers with some parameter according to the best choice of the path weight.

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 [i.15], 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 [i.29] and it is usually **unwelcomed** 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-SIGLING links can appear in any number anywhere in the path".* Citation of ETSI TR 103 715 [i.2].

4.4.2 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 4.4.2-1: Gates SDA definition in OMNeT++

Figure 4.4.2-1 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.

Figure 4.2.2-2 contains parameters like alpha, beta, gamma and delta which are used to forward the messages to customers, siblings, providers and peers respectively. This allows achieving the goal to forward to a few of the customers, peers, siblings and providers, but not to all. The value of these parameter can be set in the initialization file.

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.

4.5 Semantic Routing Table (SRT)

4.5.1 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 4.5.1-1: Example of Semantic Routing Table

Figure 4.5.1-1 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 neighbours 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 neighbours; a Level Array of [1,1,1,2] means the notify will be forwarded to first neighbours 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 interdomain routing. Customers are helpful in routing in multiple providers.

4.5.2 OMNeT++ implementation of Semantic Routing Tables

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 4.5.2-1, 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 4.5.2-1: Semantic Routing Table data structure

Table 4.5.2-1 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.

4.6 Updating semantic routing tables

4.6.1 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 4.6.1-1. A CSE sends a notification message to inform the neighbour 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 neighbours 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 4.6.1-1: 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: the updated routing table looks like in Figure 4.6.1-2.

Feature_Type	Local Database	CSE customer	CSE Peer	CSE Sibling	CSE Provider
THERMOMETER	(Uri, #)	...	..	...	**(CSE_prov, y)**

Figure 4.6.1-2: ASD update message results in modification of the routing table

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

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

4.6.2 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 4.6.2-1.

```
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 4.6.2-1: 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 neighbours is implemented as in Figure 4.6.2-2.

```
/*
* 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 4.6.2-2: Notification message generation

4.7 Advanced Semantic Discovery Query Language (ASDQL)

4.7.1 oneM2M "basic" Advanced Semantic Query Language

Advanced Semantic Discovery Query (ASDQ) could be implemented in OMNeT++ using a simple query language that clearly defines the following information at least:

- FEATURE_TYPE of the AE, which is looked for;
- AND/OR/NOT and the **numerosity** of the AE, which is looked for, could be part of an Advanced Query Language that is actually left unimplemented in this OMNeT++ simulator. In the near future, a more complex logical query language could be implemented on basis of oneM2M expert information. However, an advanced query should be considered as satisfied if - and only if - the "logical formula" is satisfied completely.

NOTE: The W3C SPARQL Query Language [i.13] can be also considered as a potential candidate to implement the logical Advanced Semantic Discovery Query Language. This possibility is for further study by oneM2M experts.

4.7.2 oneM2M "compound" Advanced Semantic Query Language simulation in OMNeT++

Semantic Discovery Routing Mechanism (SDRM) can be seen in the OMNeT++ simulator as a parser of complex query compound one's made of **Conjunctive and Disjunctive Normal Forms**. Compound ASQL are actually out of the scope of the current OMNeT++ implementation because of their intrinsic complexity. Implementing a simple or a more complex query language in the oneM2M standard is for further study by oneM2M experts. As such, the simulation of the ASD has been tested on simple atomic queries. However, this is sufficient for a first quantitative and qualitative analysis.

4.7.3 OMNeT++ implementation of basic queries

In the current version of the OMNeT++ simulator, only basic, unitary queries are implemented which means asking for a single FEATURE_TYPE at the time. The requested FEATURE_TYPE can be selected manually or randomly from the list of FEATURE_TYPE. In this simple query implementation, single AEs having a desired feature type are looked for. The implementation of the query message in OMNeT++ happens by defining a message named `discoveryMessage` which has different fields as shown in Figure 4.7.3-1.

```
message discoveryMessage
{
    // ID assigned to QUERY
    int queryID;
    // this is the identifier of the very first AE generating Discovery QUERY
    int URI_init;
    // this is identifier of in route CSE, sending the message to another CSE
    int URI_route;
    // this is type of Resource CSE is looking for. it can be
       "waterValve","thermometer","airStation","ATM","smartLock"
    string feature_type;

//this can be Registration, update, cancellation, Query, Response, Notify
  int op_code;
  // Indicates result of query
  int returnCode;
  // Used with depth of Notify
  int delta;
  // this value lists the number of remaining hops before end of forwarding QUERY
    int hopCount;
    // this will be UP (customer to provider) or DOWN (Provider to Customer)
    // or SIDE (Sibling to Sibling and Peer to Peer). direction where
       the message has been sent
    int direction;
```

```
    // this is the index of the gate from which the request has been sent
       originator gate.
    int initialGateIndex;
    //This is the result obtained when looking in the local database
    DBresult dbResult;
    // this is the list that contains the discovery path (list of CSE that
       forward the query)
    GateVector gateVector;
}
```

Figure 4.7.3-1: Discovery Message structure

4.8 Semantic Discovery Routing (SDR)

4.8.1 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.

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

4.8.2.1 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 presented in clause 4.4.

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.

4.8.2.2 Processed queries buffer

Queries are uniquely identified by URI of the sender and sequential `queryID`. If a CSE receives a query, and its ID is present in its local buffer, processing of the query will be stopped, no response will be sent.

This approach allows to correctly handle routing loops. If the query took the shortest path and already traversed through said node, then there is no need to process and redirect it again, so CSE node just omits it.

The `TTL` of buffer record is calculated based on the parameter `queryBufferTTL` - time in seconds, during which the record will be valid.

4.8.2.3 Query path memorization

In order to efficiently route a query response to the origin Application Entity (AE), the path is saved in a query message. If at any point a message is to be sent to the same CSE, it has already visited, sending of the message to the visited CSE will be stopped. If other potential receivers of the message exist, the message will be redirected to those CSEs with respect to multicast parameters.

4.8.3 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 4.8.3-1 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 4.8.3-1: 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 4.8.3-2 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 4.8.3-2: 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 4.8.3-3.

```
/*
* 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 4.8.3-3: Sending the query to the appropriate CSE

4.9 Semantic Recommendation System (SRS)

4.9.1 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 succintly 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 behaviour of a SRS is inspired from the Kademlia P2P overlay network ETSI TS 103 410-2 [i.20].

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 4.9.1-1 below.

For each `TFEATURE_TYPE`, `SRT[FEATURE_TYPE][CSE-RECCOMENDED]` 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:**

   a) **IF** the `BUCKET[FEATURE_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[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.

NOTE 1: `CSE_BUCKETS[FEATURE_TYPE]` will generally be kept constantly fresh, due to traffic of requests travelling through nodes.

NOTE 2: When there is no traffic: each CSE picks a random CSE in one of their `CSE_BUCKETS[FEATURE_TYPE]` and performs an Advanced Semantic Discovery node search for that `FEATURE_TYPE`.

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 4.9.1-1: Semantic Routing Table with the CSE_BUCKETS

NOTE 3: In the actual OMNeT++ implementation V3.7.5 [i.30], the SRS is not yet implemented. Implementing a prototype of the above pseudocode is for further study. It might greatly improve the success rate of Advanced Semantic Queries.

5 Simulating Network Scenarios in OMNeT++

5.1 Simulation Scenarios

Initialization phase of a simulation run: simulations have prefixed parameters, specific to each run, that are settled in OMNeT++ init files (`omnetpp.ini`). These parameters state the topology of the network and parameters related to the routing execution. These parameters can be `delayTime, notification_depth, alpha, beta, gamma, maxHops, TTL`.

The main simulation scenario can be described using the following pseudocode:

0. Initialization of the network with parameters.
1. Select randomly one application entity - AE Source.
2. Select randomly a feature type - `FEATURE_TYPE`.
3. AE Source will query parent CSE for that feature type `FEATURE_TYPE`.
4. The Advanced Semantic Routing starts.
5. AE Source waits for the results (one or multiple responses).
6. Some AEs randomly going out of order, propagating `CANCELLATION`.
7. Go to step 1.

The number of simulation scenario repetitions have to be relatively big, approximately one orders of magnitude smaller than the number of AEs in the network. Given randomly selected AE to query the network, simulation scenarios will provide potentially unbiased results.

5.2 Choosing Key Performance Indexes (KPIs) in OMNeT++

Overall, proposed KPIs give good understanding of proposed Advanced Semantic Routing. Each metric is affected by one or more configuration parameters.

Success rate number of successful query responses;

Flood number of messages generated by CSEs;

Throughput total number of messages in the network at a given time;

Latency number of message retransmissions before receiving response.

When dealing with the number of messages, simulation should be tweaked in a way, that all the registration messages are processed beforehand, and impact of registration phase is minimized, e.g. an introduction of a so-called **warmup period** is required. Dealing with mean response time can be tricky and simulation should be run several times to eliminate random bias. Cost of routing is one of the trickiest KPIs, since cost metric should be attached to each node (depending on a node type, for example - IN-CSE and MN-CSE) and each link regarding SDA relationship between CSEs - namely Peer, Sibling, Customer, Provider. Success rate is an absolute number of successful responses to the sent query. A number of parameters can impact this metric. It can be affected by:

- `notification_depth`, depending on it, query can be rerouted using the SRT (Semantic Routing Table) records. If such records are absent, query will be multicasted in the network to different neighbours.

- **multicast parameters** (namely - `alpha, beta, gamma, delta`). Each individual multicast parameter impact is greatly affected by used network topology and actual SDA relationship between CSEs - namely Peer, Sibling, Customer, Provider.

The success rate of query heavily depends on the distribution of AE feature types, and to eliminate the randomness impact, scenario has to be run multiple times on the same stable network (with the same semantic routing table records). To say more, for results to be representative, a big number of simulations with different parameters should be run. Parameter impact should be thoroughly analysed, as their impact could differ with different KPIs.

As discussed in clause 4, multicast parameters and notification depth heavily impact the performance for almost all KPIs. Thus, simulations have to be configured with network topology in mind, since running simulations with different parameters which have no impact on metrics (KPIs) would give virtually no useful results.

Analysing latency, impact of multicast parameters can be demonstrated, along with `notification_depth`, which directly affects SRT (Semantic Routing Table). The less query being multicasted in the network, the more direct and shorter path is taken.

Analysing flood, impact of `queryBufferTTL` can be seen. The less time query is memorized by the nodes, the more retransmissions occur. Flood can be also impacted by the `maxHops` parameter, which determines the number of retransmissions of the query. Flood can be also expected to be impacted by `notification_depth`.

5.3 OMNeT++ implementation of KPIs

5.3.0 Foreword

Figure 5.3.0-1 presents the OMNeT ++ implementation of KPIs.

```
parameters:

        @signal[flood](type="long");
        @statistic[flood](title="number_of_messages"; source="flood";
                          record= vector,last);

        @signal[packet_size](type="long");
        @statistic[throughput](title="total_number_of_packets";
                               source="packet_size"; record=vector,last);

        @signal[hop_count](type="long");
        @statistic[latency](title="latency"; source="hop_count";
                            record= vector,last);

        @signal[number_replies](type="long");
        @statistic[successRate](title="total_number_of_replies";
                                source="number_replies"; record=vector,last);
```

Figure 5.3.0-1: OMNeT++ implementation of KPIs

To gather statistics, default statistics collection mechanisms offered by OMNeT++ are used. This approach combines the signal mechanism and NED properties in order to de-couple the generation of results from their recording, thereby providing more flexibility in what to record and in which form.

Statistics are declared in the NED files with the `@statistic` property, and modules emit values using the signal mechanism. The simulation framework records data by adding special result file writer listeners to the signals. By being able to choose what listeners to add, the user can control what to record in the result files and what computations to apply before recording.

The signals approach allows for calculation of aggregate statistics (such as the total number of packet drops in the network) and for implementing a warm-up period (as discussed in clause 5.2) without support from module code. It also allows to write dedicated statistics collection modules for the simulation, also without touching existing modules [i.30].

5.3.1 Success rate

The OMNeT++ collection of result relies on the `emit` directive, as shown in Figure 5.3.1-1 below.

```
void AE::handleMessage(cMessage *msg) {
    static int NumOfReplies = 0;
    //AE will receive the response
    //AEMessage *responseMsg = check_and_cast<AEMessage *>(msg);
    discoveryMessage *responseMsg = check_and_cast<discoveryMessage*>(msg);
    EV << "AE receives a response" << "\n";
    int number_of_response=0;
    number_of_response++;

    if (responseMsg->getReturnCode() == ResultCode::SUCCESS) {
        number_of_successfulResponse= 0;
        number_of_successfulResponse++;

        EV << "Resource of type " << responseMsg->getFeature_type()
                << " found in " << responseMsg->getURI_init() << "\n";
    }

    if (responseMsg->getReturnCode() == ResultCode::NOT_FOUND) {

        EV << "Resource of type " << responseMsg->getFeature_type()
                << " not found in" << responseMsg->getURI_init() << "\n";
    }

    number_of_replies++;
    emit(successRate, number_of_replies);
    delete responseMsg;
}
```

Figure 5.3.1-1: Success rate statistics collection

The success rate statistics is collected in an AE module, which triggered the query, which successful results are to be emitted into results file.

5.3.2 Flood & Throughput

Figure 5.3.2-1 below shows how flood and throughput statistics are collected in OMNeT++.

```
void CSE::handleMessage(cMessage *msg) {

    number_of_packets++;
    // assigning the values to the signal
    emit(totalpacketsSignal, number_of_packets);
    EV << "URI " << msg->getSenderModuleId() << "\n";

// if the message comes from the AE
    if (prefix("AE", msg->getSenderModule()->getName())) {
        handleAEMessage(msg);
    } else {
        handleDiscoveryMessage(msg);
        emit(flood, number_of_messages);
    }

} // end of handle message
```

Figure 5.3.2-1: Flood and throughput statistics collection

Total packets and throughput statistics are collected in the CSE module. Depending on type of message, different signals can be emitted.

5.3.3 Latency

Figure 5.3.3-1 shows how latency statistics are collected in OMNeT++.

```
/*
* processQuery is used to route query if local DB lookup failed.
* It tries to perform semantic routing, and if no records satisfying query were
found,
* if uses a so-called fallback routing to multicast query to the best match
neighbours.
*/
void CSE::processQuery(discoveryMessage *msg) {
    EV << "The Message is a query \n";
    EV << "DB Lookup not Successful" << "\n";
    if (msg->getHopCount() <= 0) {
        bubble("TTL: expired");
        //Respond to the URI_init that the discovery ends
        // TODO: DBLookup part to be added here
        msg->setOp_code(RESPONSE);
        // TODO: set the message op_codes according to result from DBLookup
        //You extract from the top of the list the gate that has to be used
        EV << "Hop count is 0 so we generate a self response message \n";
        number_of_messages++;
        generateResponseMessage(msg, ResultCode::NOT_FOUND);
        return;
    }

    // decrease the hop count
    EV << "we are in the else : hop count is currently " << msg->getHopCount()
            << "\n";
    msg->setHopCount(msg->getHopCount() - 1);
    number_of_hops++;
    emit(latency, number_of_hops);

    EV << "New hopCount=" << msg->getHopCount() << "\n";

    auto res = routeQuery(msg);

    if (res.size() > 0) {
        for (auto it : res) {
            auto gateit = this->Gates[it];
            int gateindex = gateit.second;
            std::string gateName = gateit.first + "$o";
            bubble("Semantic record found");
            number_of_messages++;
            sendDelayed(msg->dup(), delay, gateName.c_str(), gateindex);
        }

        return;
    }
    fallbackRouteQuery(msg);
}
```

Figure 5.3.3-1: Latency statistics collection

Latency statistics is being collected in the CSE module, using `TTL` (`hopCount`) message field. It is emitted on message retransmission.

5.4 Simulation results

5.4.0 Foreword

Multiple simulation runs have been processed to get simulation results. This clause will provide the details result of these experiments. The graphical results of the above mentioned KPIs will be shown and explained. More (large scale) simulation results, obtained with bigger topologies and more powerful hardware will be uploaded and made accessible in the future (see Annex A).

The simulation set up is as follows:

- Network hierarchy is as follows:
  - 40 IN-CSE. For each IN-CSE node: 100 AE-IN and 40 AE-ADN;
  - 40 CSE-ASN. For each node: 50 AE-ASN;
  - 30 MN-CSE. For each node: 20 MN-AE and 20 ADN;
- 20 % of AEs and ADNs send query;
- 5 % AEs randomly go out of order after warmup period.

NOTE: Due to a very specific network topology only `alpha` multicast parameter has meaningful effect on KPIs. Since CSE have big number of customer CSEs thus it impacts the KPIs. The rest of the parameters are set to default.

5.4.1 Total Number of Messages

As mentioned above, the number of messages sent between the entities in the network will be collected for one of the KPIs. The result of all the experiments for the KPI will be compared.

Figure 5.4.1-1 shows the first simulation result for the total number of times. Simulation setup varies with the values of the parameter $0= `notification_depth`, $1= `alpha`, $2= `TTL`. The parameter is set as $0=1, $1=1, $2=8 for the first simulation. The X-axis represents the simulation time (expressed in seconds). The time starts from 600 s onward because the warm up period is set to 600 s in the simulation. The Y-axis represents the value. Figure 5.4.1-1 shows that the number of messages increases with time and reaches to the maximum of 35 000 messages. The change in the number of messages in the graphs results from the fact that resources are distributed homogenously. Also, the query buffer (as discussed in clause 4.8.3) help to drop the duplicate messages.

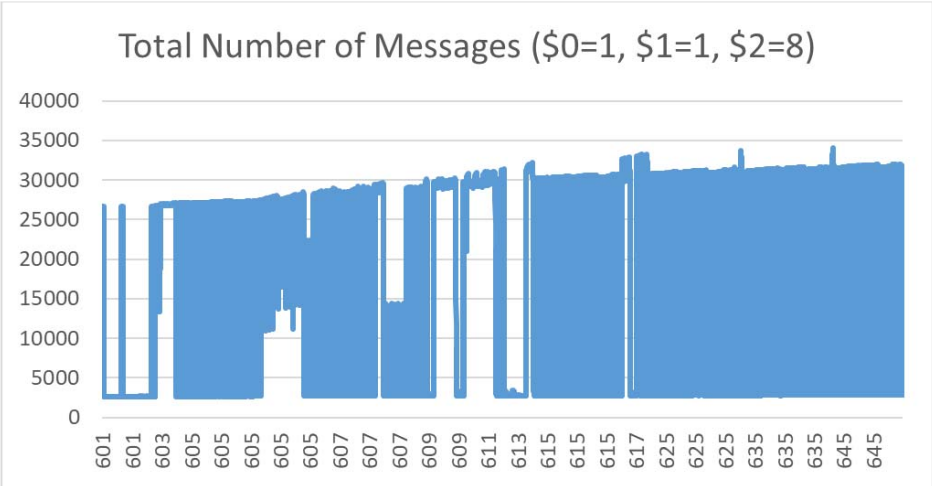

Figure 5.4.1-1: Total Number of Messages ($0=1, $1=1, $2=8)

For the second run the parameters have been changed to $0=1, $1=1, $2=10 and Figure 5.4.1-2 shows the result. The two Figures 5.4.1-1 and 5.4.1-2 demonstrate how changing the `TTL` from 8 to 10 will impact the number of total messages.

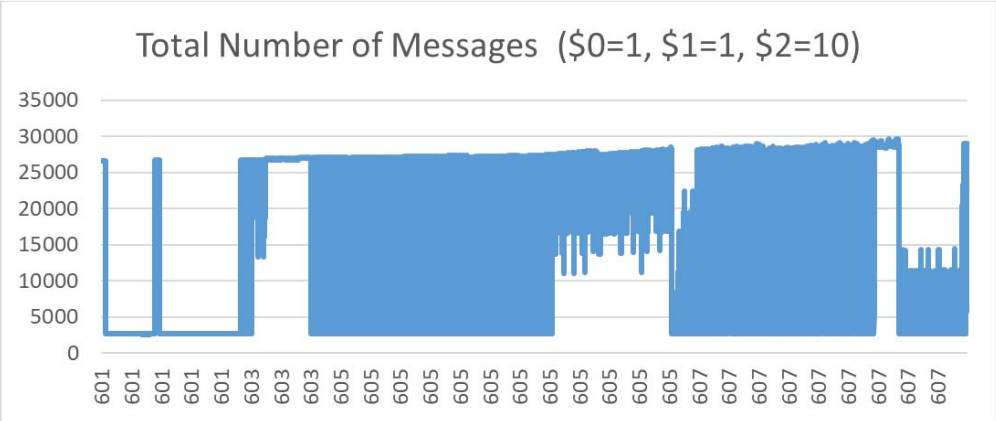

Figure 5.4.1-2: Total Number of Messages ($0=1, $1=1, $2=10)

The `alpha` parameter is changed for the third run. For this run, the parameter is set to $0=1, $1=3, $2=8, as shown in Figure 5.4.1-3. As compared in Figure 5.4.1-1, a minimal change in the number of messages with changing the value of `alpha` can be seen.

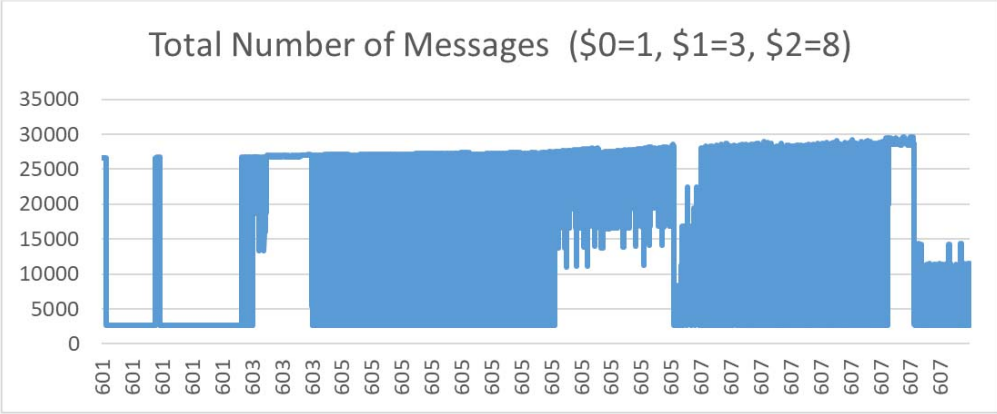

Figure 5.4.1-3 :Total Number of Messages ($0=1, $1=3, $2=8)

The value of the `notification_depth` has been changed to 2 for next simulation. Figure 5.4.1-4 shows that the number of messages and the time to finish the simulation both will increase, because the notification messages which are sent to just first neighbours by default. Changing its value to 2 means, that it will send the notify messages to two-hop neighbours now.

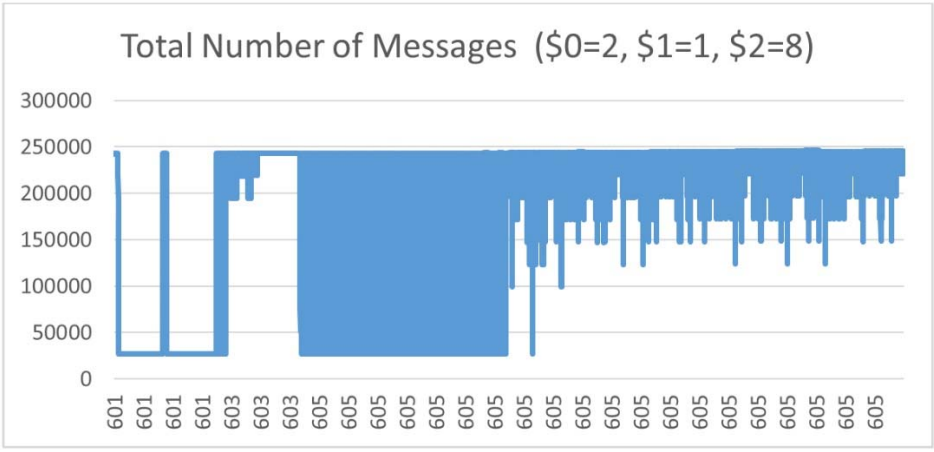

Figure 5.4.1-4: Total Number of Messages ($0=2, $1=1, $2=8)

From the results of the multiple simulation setups, it is clear that the number of messages in the network increases with change in the parameters `alpha` and `notification_depth`. The parameter `notification_depth` has a huge impact on the number of messages in the network. Figure 5.4.1-5 shows the detailed comparison of multiple runs of the simulation. The X-axis represents the total count of the messages and the Y-axis represents different simulation runs with different parameters.

The main reason for a huge difference between the runs, are the nodes randomly going out of order during the simulation. Retransmissions of notifications to neighbours greatly increase the overall number of messages. The impact of greater `notification_depth` is obvious and should be taken into consideration.

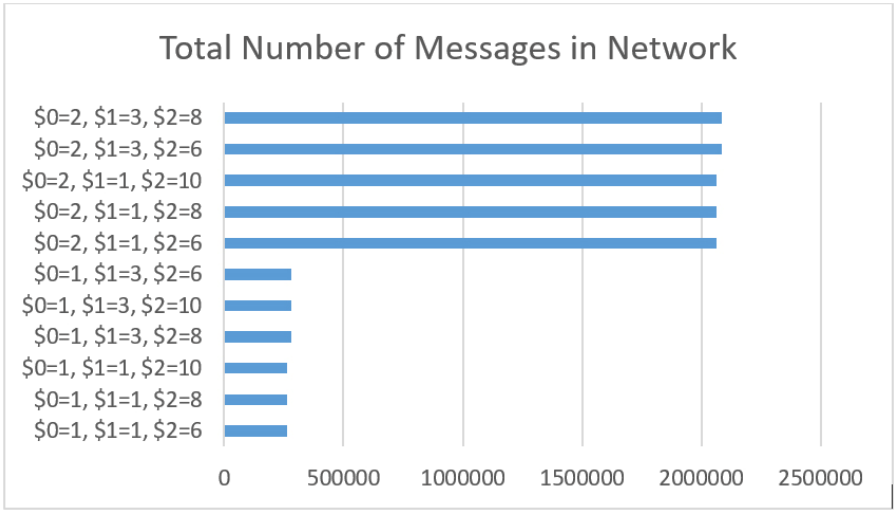

Figure 5.4.1-5: Comparison of Total Number of Messages

5.4.2 Latency

Latency is another KPI being intended to calculate in the simulation. Figure 5.4.2-1 shows the first simulation result for latency. For the first simulation, parameter is set as $0=1, $1=1, $2=8. The X-axis represents the simulation time in seconds and the Y-axis the value. Figure 5.4.2-1 demonstrates that the value of latency in this simulation reaches to maximum of approximately 1 300.

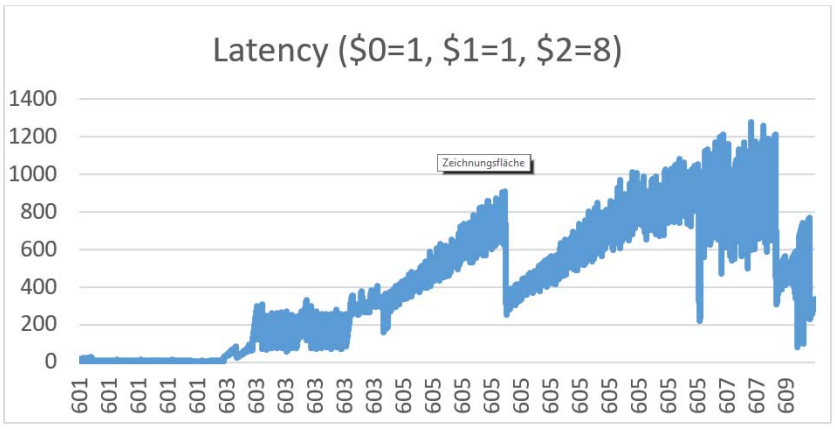

Figure 5.4.2-1: Latency ($0=1, $1=1, $2=8)

For second run, the parameter `TTL` is set to $0=1, $1=1, $2=10 and Figure 5.4.2-2 presents the result. It shows that the `TTL` has minimal impact on the latency.

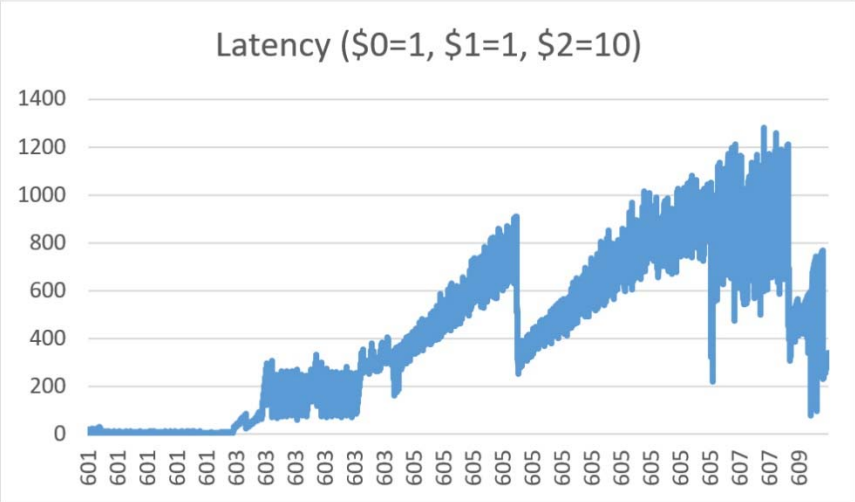

Figure 5.4.2-2: Latency ($0=1, $1=1, $2=10)

For the third run, the `alpha` parameter is changed to $0=1, $1=3, $2=8. Figure 5.4.2-3 shows the result of this run. It documents that changing the `alpha` parameter results in a slight increase of the value of latency.

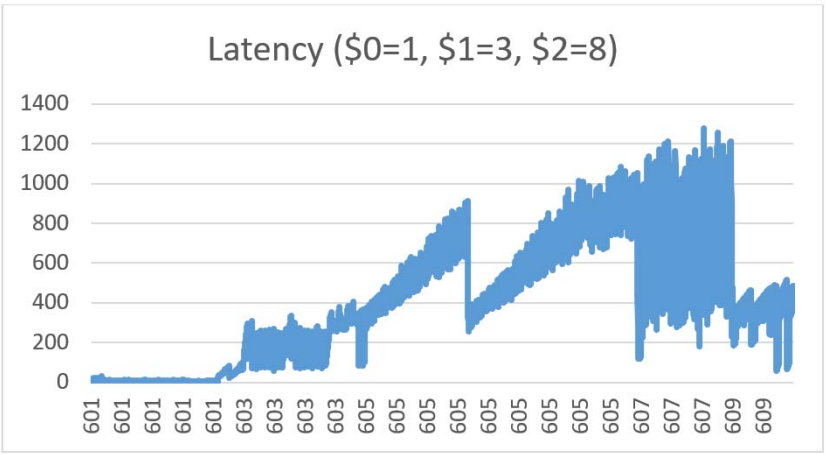

Figure 5.4.2-3: Latency ($0=1, $1=3, $2=8)

For the fourth run, the value of `notification_depth` is changed to parameter $0=2, $1=1, $2=8. Figure 5.4.2-4 below shows that the value of latency increases as the `notification_depth` increase.

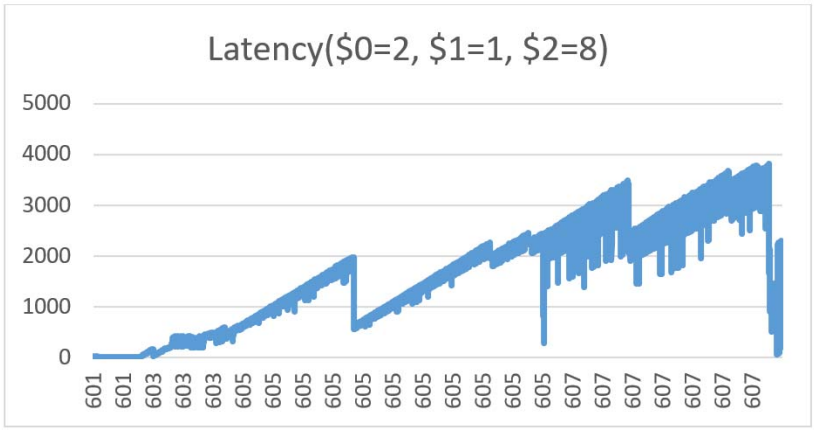

Figure 5.4.2-4: Latency ($0=2, $1=1, $2=8)

Figure 5.4.2-5 presents the comparison of the latency with different set of parameters values. The X-axis represents the value of latency and the Y-axis the different simulation runs with different parameters.

The impact of `notification_depth` can be seen in Figure 5.4.2-5.

There are a number of conclusions that can be drawn from Figure 5.4.2-5.

1) Since seen queries are being omitted at arrival (loop free routing, mentioned in clause 4.8.3), at some point, a big number of queries are being retransmitted to the same node, and dropped after. It demonstrates the importance and effectiveness of aforementioned loop free routing (namely seen queries buffer, mentioned in clause 4.8.3).

2) The `notification_depth` greatly impacts the number of retransmissions, since SRT records are exchanged with more distant neighbours.

3) The `alpha` multicast parameter has certain impact on this KPI, but fades in relation to `notification_depth`.

Keeping those conclusions in mind, it is important to note that this KPI metric has to be thoroughly investigated and probably redefined due to the unexpected effect of `notification_depth`.

It is important to note that well defined peaks and drops on charts above can be attributed to loop free routing approach, and events of dropping seen queries can be noted.

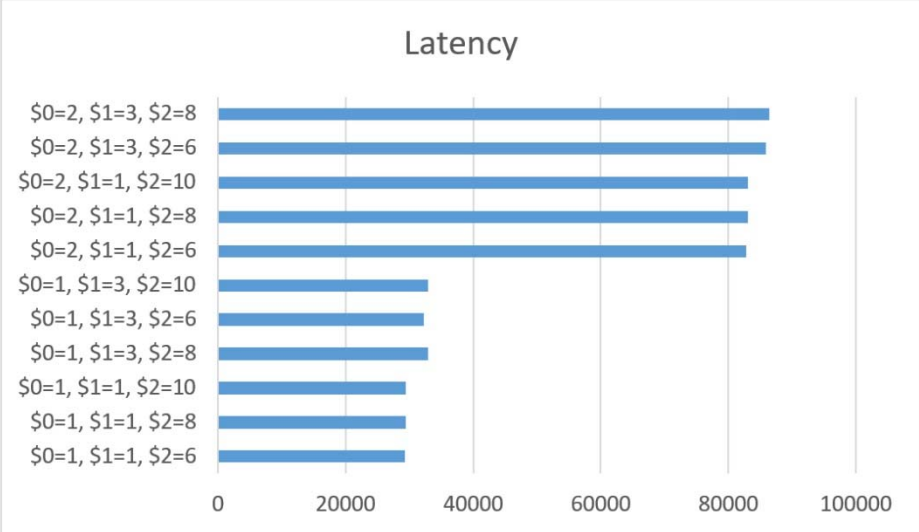

Figure 5.4.2-5: Comparison of Latency

5.4.3 Success Rate

The success rate is the number of responses received for a query. In this clause the impact of the parameters on the success rate is checked.

For the first run, the parameter is set as $0=1, $1=1, $2=8. The X-axis represents the simulation time(s) and the Y-axis the values. Figure 5.4.3-1 shows the result.

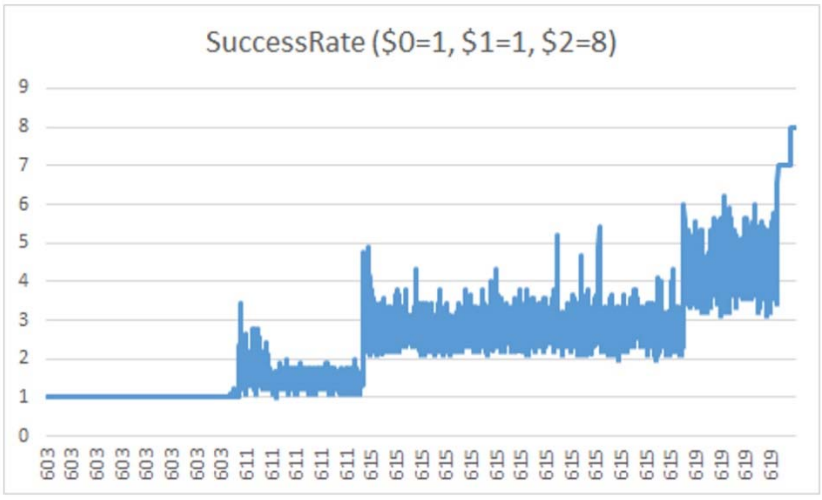

Figure 5.4.3-1: Success Rate ($0=1, $1=1, $2=8)

For the second run, the `TTL` parameter is changed. Figure 5.4.3-2 shows the result of the simulation.

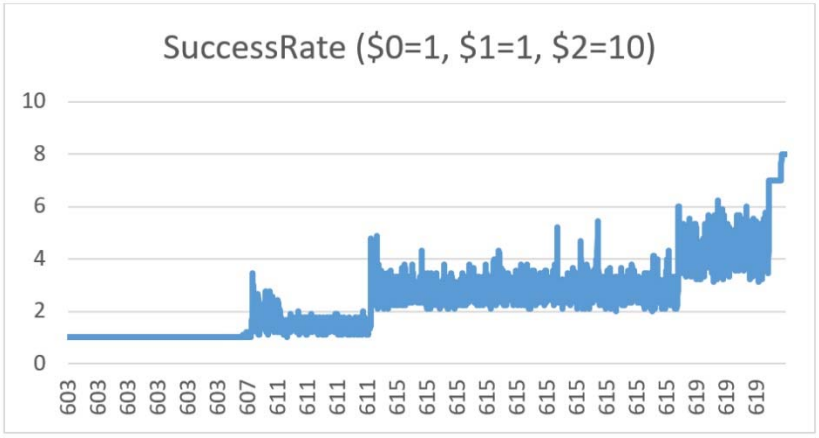

Figure 5.4.3-2: Success Rate ($0=1, $1=1, $2=10)

For the third run, the parameter `alpha` is changed. Figure 5.4.3-3 below shows the result. Figure 5.4.3-3 documents that the success rate increases when `alpha` is increased. It is matter of the fact that the message will be multicasted to `alpha` customers.

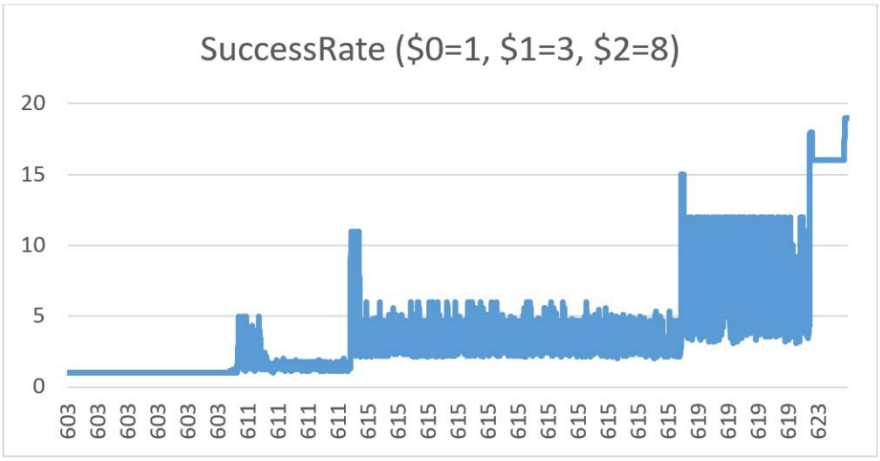

Figure 5.4.3-3: Success Rate ($0=1, $1=3, $2=8)

Figure 5.4.3-4 shows the result of the simulation run where the `notification_depth` parameter is changed.

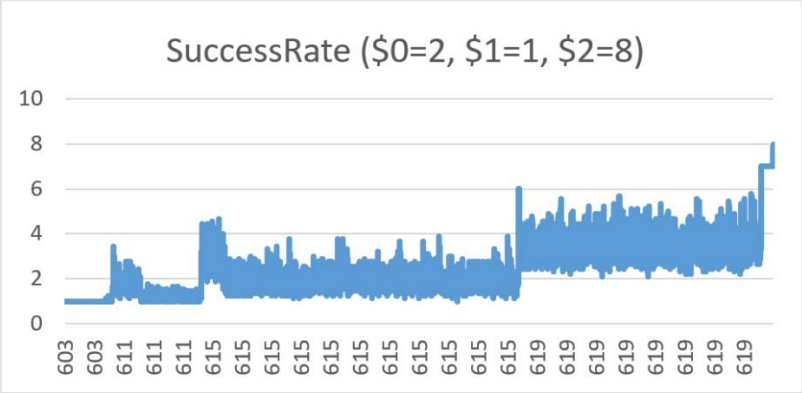

Figure 5.4.3-4: Success Rate ($0=2, $1=1, $2=8)

Figure 5.4.3-5 shows the comparison of the success rate. The X-axis represents the value of the success rate and the Y-axis the different simulation runs with different parameters. The value of the success rate is higher when `notification_depth` is high and when `alpha` is high.

It shows, that the impact of parameters can be ordered as follows (from greater to lesser):

1) `alpha`: it appears, that multicast turned out to be one of the most effective ways to find resources in the network.

2) `notification_depth`: it affects the success rate ever so slightly.

Main lesson to be learned from it is simple: multicast allows to find farthest resources, if SRT lacks entries, and the semantic routing allows to efficiently find the closest resources, in case of existence of SRT entries.

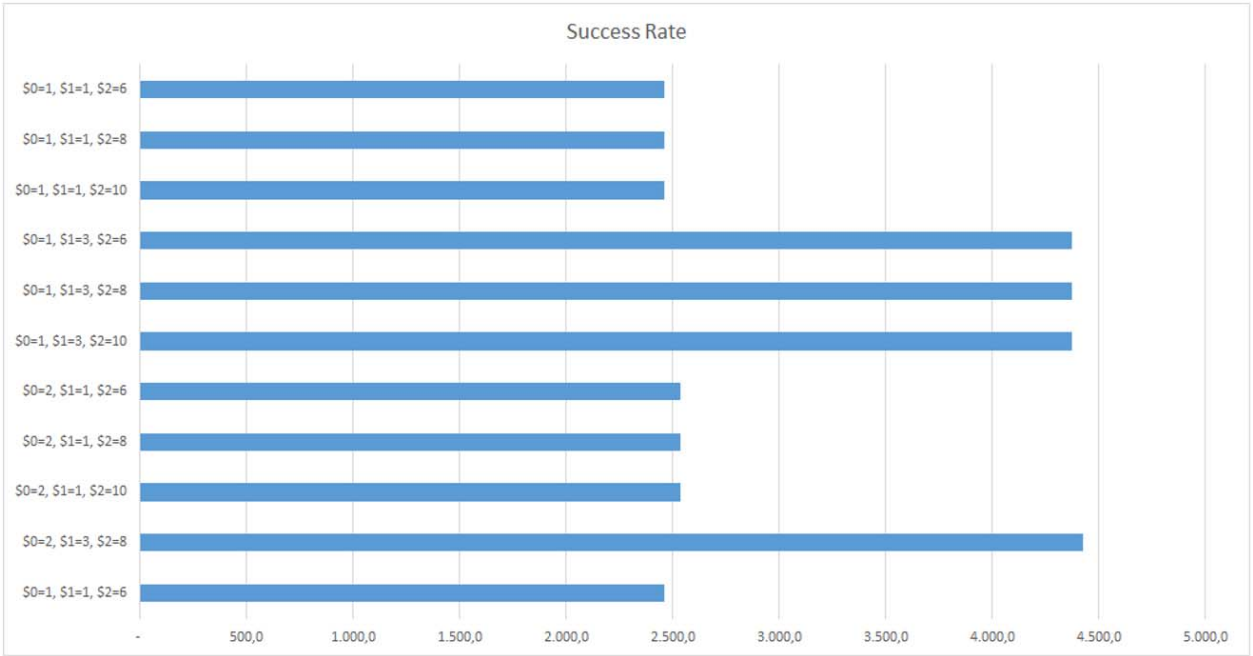

Figure 5.4.3-5: Comparison of Success Rate

5.4.4 Number of Messages Exchanged by CSEs (Flood)

Flood is the number of messages exchanged by CSEs. This clause provides the detail comparison about the number of messages exchanged by CSEs depending on the parameters.

For the first run, the parameter is set as $0=1, $1=1, $2=8. The X-axis represents the simulation time(sec) and the Y-axis the values. Figure 5.4.4-1 shows the result. It documents that the number of the messages can reach up to 17 000.

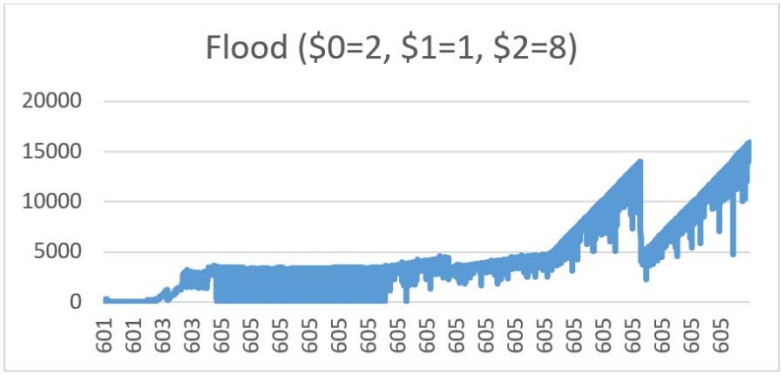

Figure 5.4.4-1: Flood ($0=1, $1=1, $2=8)

In the next simulation the TTL is changed from 8 to 10. Figure 5.4.4-2 shows the result of that simulation. It documents that the value of flood increases.

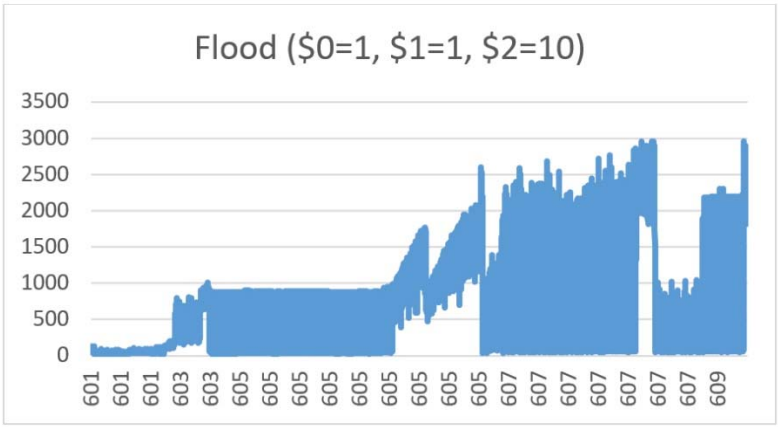

Figure 5.4.4-2: Flood ($0=1, $1=1, $2=10)

In the next simulation, the value of `alpha` parameter is changed, and Figure 5.4.4-3 shows the results. It documents that the number of the messages generated will increase because the CSE will forward the query to `alpha` customers.

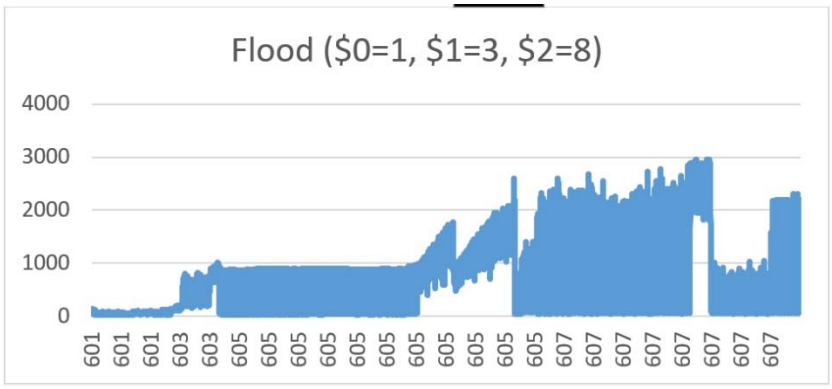

Figure 5.4.4-3: Flood ($0=1, $1=3, $2=8)

For the fourth run, the value of `notification_depth` parameter is changed. Figure 5.4.4-4 shows the result. The following graphs point out, that the value of flood increases when the `notification_depth` parameter is changed from 1 to 2. The reason is, when the `notification_depth` is set to 2, the CSEs will send Notify messages to the two hop neighbours.

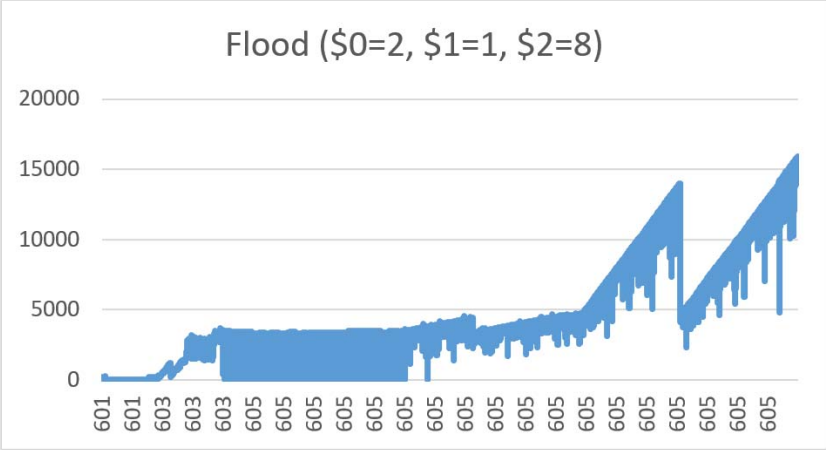

Figure 5.4.4-4: Flood ($0=2, $1=1, $2=8)

Figure 5.4.4-5 provides the comparison of the total number of messages generated by queries. It documents that the number of messages is higher when the value for `notification_depth` is higher. The result below explains, that the number of messages exchanged between CSEs will increase with `notification_depth` because the CSEs sends the SRT entries using Notify message to the 2-hops neighbours.

Flood is an important metric, as it shows the correlation between the `notification_depth` and number of messages exchanged by CSEs. It once again demonstrates the great importance of properly tweaking network configuration parameters.

An important conclusion is the following: current approach to propagation of changes in the network (namely AEs registering in the network and randomly going out of order) can easily flood the system with service messages.

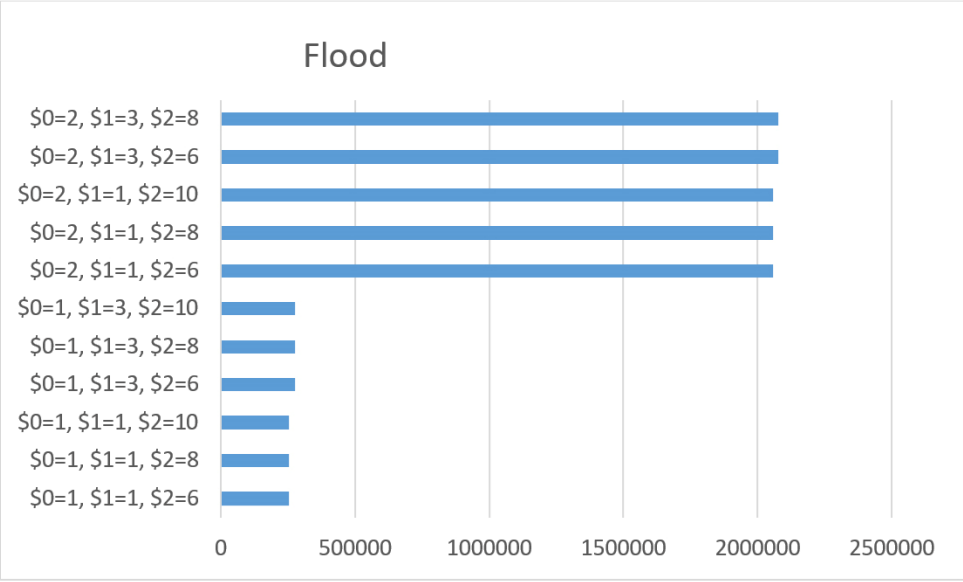

Figure 5.4.4-5: Comparison of Flood

5.4.5 Overall conclusions

Analysing all the results obtained, the conclusion shows:

1) Notifications about resources registration and deregistration (cancellation) should be investigated and possibly optimized.

2) Multicast parameters should be investigated more closely, minimizing the impact of `notification_depth`.

3) Query TTL (`maxHops`) should be analysed more precisely, minimizing impact of both `notification_depth` and multicast parameters.

4) **Recommendation system**, that will be developed in the near future, could possibly decrease the overall network load, and probably most of the KPIs.

5) Possible buffering and packing of service messages (namely - notifications) should be investigated due to the great and disproportionate impact to all the KPIs presented here.

5.5 Some simulation improvements in further versions of OMNeT++ implementation

This clause enumerates some possible improvements that could be implemented in the simulator. The following improvements are strongly inspired and adapted to [i.32]:

- **Implementing CACHES in the CSE to reduce routing hops**. A CSE can put the result of a successful queries lookup in a cache giving positive results not in the current CSE. The positive effect of caches applied to all the CSE databases can leverage the number of message exchanges between CSEs.

- **Implementing a LIVENESS politics in the Semantic Recommendation System**. Each publication can have a LIFESPAM: after the end of the lifespan, either the publisher republishes the content in the CSE, or the record is simply dropped out from the CSE.

- **Reduce TRAFFIC and FLOODING attacks**. Each CSE can limit the number of packets arriving from a CSE-customer, CSE-provider and CSE-peer and CSE-sibling; their number can be fixed on a CSE-to-CSE basis.

- **Adding INCENTIVES to republication**. To improve participation, incentives to locally republish contents retrieved abroad can be introduced: republication can be a simple pointer to another CSE. A tit-for-tat strategy could be installed between clients (looking for contents) and purveyors (distributing the contents) were the CSE should play a special role being in the middle of the above two actors.

- **Improve LOAD DISTRIBUTION**. To improve load distribution, CSE can perform load distribution among replicated copies of a single content. If CSE tables map a semantic resource name into a list of IPs, then the CSE can respond with the entire list of purveyors, or it can rotate the ordering of the addresses within each reply. As such, IP rotation performed by CSE can distribute among multiple purveyors.

- **Improve SUCCESS RATE and FOCUS the discovery search by modifying the multicast parameters**. To improve the discovery success rate and focus the discovery search, each CSE can dynamically refine its α,β,γ and δ multicast parameters by combining with the success probability of a given tag in the previous queries.

- **Improves content aggregation in CSE**. The data quality can be compromised by many factors, including data entry errors ("OpenOffice" instead of "OpenOf- fice"), missing integrity constraints ("eat before December 12018"), multiple formats ("1st, rue Prés. Wilson, Antibes", versus "1, rue du Président Wilson, Antibes"), optional arguments ("+33(0)678123456" versus "0033678123456"), see [i.16] for a survey of data deduplication techniques.

- **CSE Mobility**. Since traffic from wireless and mobile devices has exceeded traffic from wired devices, most contents are requested and delivered by both wireless and mobile devices.

- **Nomadism**. When a mobile CSE publishes a content.

- **Security**. Design attacks that could corrupt the CSE routing tables; the Discovery Service is not vaccinated by either DDoS bandwidth-flooding attack, or man in the middle attack, or poisoning attack, or spoofing an IP of a node below a CSE.

6 Distributed Semantic Resource Directory Simulator (DSRDS)

6.1 Introduction

The different discovery approaches that exist in the context of IoT infrastructures and sensors have been classified in the following ETSI TS 103 264 [i.5]: nearby discovery (like RFID or Bluetooth®), network discovery (like mDNS IETF RFC 6762 [i.8]), and resource directory discovery (like oneM2M). From the aforementioned list, the Distributed Semantic Resource Directory Simulator (DSRDS) focuses on providing a virtual environment to simulate and test a specific sub-type of resource directory discovery known as Semantic Resource Directory Discovery. The resource directory discovery and the semantic resource directory discovery approaches rely on the following concepts:

- **Resource descriptions.** Resource descriptions are documents that describe services (endpoints, security, or meta-data among other elements) of IoT infrastructures and/or IoT devices. These descriptions are the objects that a resource directory discovery task aims at finding. These documents may take different forms and formats: from an URI with a set of attributes IETF RFC 6690 [i.9], to RDF documents expressed according to an ontology [i.10]. In the latter case, the resource descriptions are usually known as semantic resource descriptors. As a result, when the resource directory discovery relies on semantic resource description it is known as semantic resource directory discovery.

- **Resource directory.** A resource directory is a service that publishes an API for either users or machines. Although resource directories differ from one implementation to another, they have to implement at least two API endpoints: resource description endpoint that allows to create, update, or remove resource descriptions or semantic resource descriptions; and the resource discovery endpoint, or Semantic Resource Discovery endpoint, that allows to find relevant resource description, or semantic resource descriptions, for a given discovery criterion.

oneM2M relies on the resource directory discovery approach for implementing its syntactic discovery oneM2M TR-0057 [i.11]. Besides, it relies on the semantic resource directory discovery approach for implementing its semantic discovery oneM2M TR-0045 [i.12]. The semantic discovery of oneM2M assumes that different AEs send a set of semantic descriptors (e.g. the resource semantic descriptors) to one CSE; which in this case is an implementation of a resource directory. The CSE stores the different semantic descriptors and, then, implements a semantic resource directory discovery by answering SPARQL queries. The CSE provides a list of identifiers belonging to those semantic descriptions that fulfil the restrictions specified in the SPARQL query.

For the sake of clarity, from now on, the deliverable will align with the terms used by oneM2M to refer to the generic terms of the resource directory discovery, e.g. CSE referring to a resource directory, semantic descriptors referring to semantic resource description, and semantic discovery for semantic resource directory discovery. The DSRD simulator aims at providing a virtual environment that allows simulating virtual CSEs, assign to them semantic descriptors, and for a specific CSE performing the semantic discovery over the semantic descriptors associated with such CSE.

Finally, when there is a specific type of semantic discovery known as distributed semantic discovery. This kind of semantic discovery involves several resource directories, e.g. CSEs, The bottom line is that a client, e.g. AE, issues a semantic discovery criterion to a CSE, then the CSE forwards the query to other beforehand known CSEs (which can perform this operation recursively). Finally, the former CSE aggregates the results from the other CSEs with its own and return a unified query answer.

6.2 DSRDs implementation

6.2.0 Foreword

The DSRD simulator is built upon two functional layers. The former provides configuration capabilities that allow CRUD operations of virtual CSEs, CRUD operations for semantic descriptions and, also, allow associating semantic descriptors to existing CSEs. The latter layer provides the simulation capabilities that allow selecting an existing CSE and solving a SPARQL query, e.g. perform the semantic discovery specified by oneM2M. As an addition, the DSRD is able to allow CSEs to answer full SPARQL queries according to the standard [i.13], instead of returning only a list of identifiers as specified in oneM2M TR-0057 [i.11].

The DSRD threats the virtual CSEs as RDF entities that are related to semantic descriptors. Therefore, the DSRD relies on a triple store to perform any CRUD operation for the CSEs or the semantic descriptors. When a user demands the creation of a virtual CSE, the DSRD creates a named graph in the triple store that will contain all the semantic descriptors related to such CSE. Following this approach, the DSRD simulates the distributed nature of the CSEs as different named graphs that will contain the associated semantic descriptors. The whole interaction with the DSRD simulator is performed through a SPARQL endpoint, that is used to perform any of the supported operations.

The semantic descriptors supported by the DSRD, the virtual CSEs, and how the semantic descriptors are assigned to those CSE follows the ontology depicted by Figure 6.2.0-1.

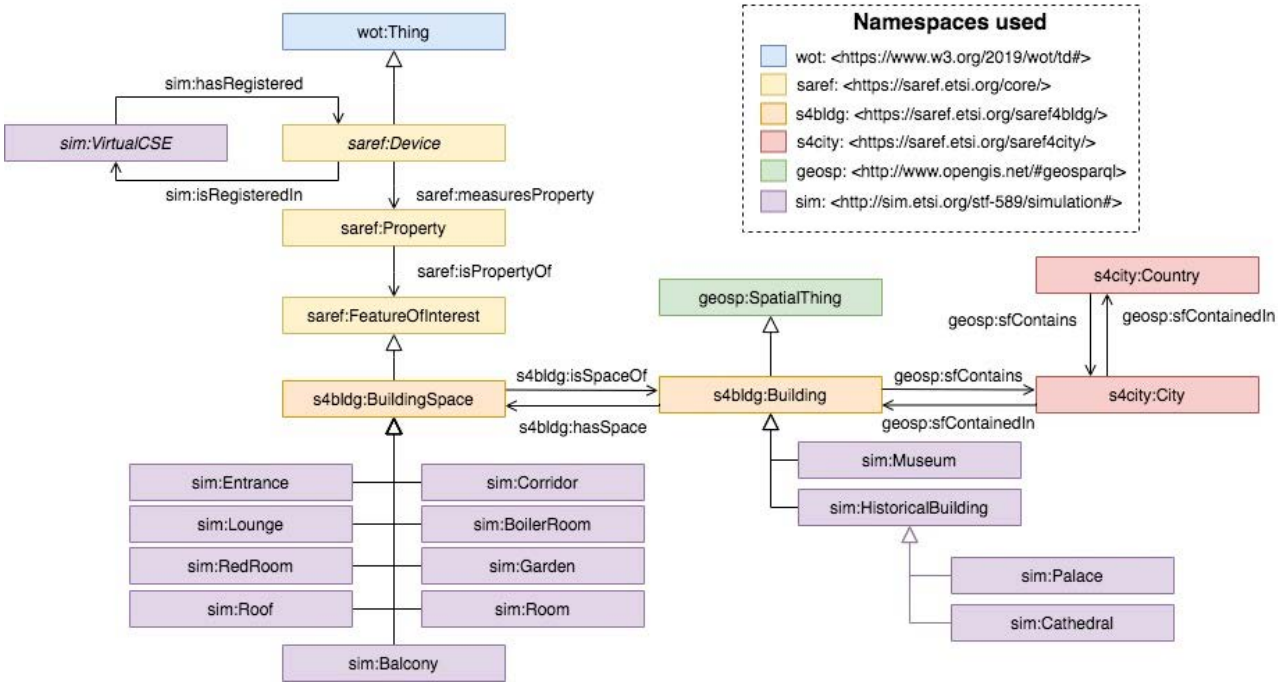

Figure 6.2.0-1: DSRD Ontology

It can be observed that the ontology re-uses well known and standard ontologies, namely: WoT, GeoSPARQL, SAREF, the SAREF extensions for cities (SAREF4CITY) and buildings (SAREF4BLDG). Additionally, the ontology has specific terms developed for the sake of the simulation, which belong to the `sim:namespace`. The previous ontologies have been re-used to express the following informational requirements of the semantic descriptors:

- API information: in order to specify the endpoints, security, protocols and data schema of the IoT devices published data, the ontology relies on the well-known Web of Things ontology.
- Specific sensor information: in order to specify specific sensor types, properties measured and, also, features of interest measured by those sensors the ontology relies on the well-known and standard SAREF and its extension for buildings SAREF4BLDG.
- Building location information: in order to express information about buildings the ontology re-uses the standard SAREF extension for buildings SAREF4BLDG.
- Geolocation information: this kind of information is expressed using the well-known GeoSPARQL that allows to express latitude and longitude coordinates.

As it can be observed, the ontology from Figure 6.2.0-1 allows to create a virtual CSE with an RDF resource of type `sim:CSE`. Then, the ontology allows to assign a set of semantic descriptors to that resource by means of the property `sim:hasRegistered`. The semantic descriptors are RDF sources typed as `saref:Device` or `wot:Thing`, that may have more meta-information related; for instance: information about what is measured (saref:Property), location information (`s4bldg:BuildingSpace, s4bldg:Building, s4city:City, s4city:Country`), or geospatial information (`geosp:SpatialThing`).

The following clauses explain how to perform the different operations supported by the DSRD simulator. For the sake of the following examples, they are expected to be run in a deployed instance of the DSRD; however, the examples of following clauses should work with any SPARQL endpoint that allows UPDATE queries and the creation of named graphs.

6.2.1 DSRD API for CSE

The DSRD API for CSE counts with two basic operations: create a virtual CSE or delete a virtual CSE. All these operations are performed using SPARQL queries that will create or delete an RDF resource typed as `sim:CSE`. In addition, the query specifies a named graph with the same URI as the one of the resource typed as `sim:CSE`. For this end, the following templates have been created in order to ease the use of the DSRD; bear in mind that all the identifiers have to be URIs.

- Query template for creating a CSE in its own named graph is depicted by Figure 6.2.1-1. Consider that `#CSE_id` has to be replaced with a valid URI. Figure 6.2.1-2 depicts a sample instantiation of the template.

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


INSERT DATA {
 GRAPH <#CSE_id> {
 <#CSE_id> a sim:VirtualCSE .
```

Figure 6.2.1-1: Query template for creating a virtual CSE

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


INSERT DATA {
 #CSE_id was replaced for http://etsi.stf-584/samples/CSE/1
 GRAPH <http://etsi.stf-584/samples/CSE/1> {
 <http://etsi.stf-584/samples/CSE/1> a sim:VirtualCSE .
```

Figure 6.2.1-2: Sample instantiation of the template for creating a virtual CSE

- Query template for deleting a CSE is depicted by Figure 6.2.1-3. `#CSE_id` has to be replaced with a valid URI.
- Deleting the CSE will also delete any semantic descriptor related. Figure 6.2.1-4 depicts a sample instantiation of the template.

```
CLEAR GRAPH <#CSE_id>
```

Figure 6.2.1-3: Query template for deleting a virtual CSE

```
#CSE_id was replaced for http://etsi.stf-584/samples/CSE/1
CLEAR GRAPH <http://etsi.stf-584/samples/CSE/1>
```

Figure 6.2.1-4: Sample instantiation of the template for deleting a virtual CSE

In order to run one of the previous queries in the DSRD simulator and create or delete a virtual CSE, a user will send an instantiated query (e.g. Figure 6.2.1-2 or Figure 6.2.1-4) to the DSRD following the SPARQL protocol [i.13]. For this end, the query will be codified as a URL and sent as value of the argument 'update' in a GET request. Alternatively, the query should be sent as part of the body in a POST request. In the specific case of the DSRD, it supports the GET protocol for running these queries.

Relaying on the Figure 6.2.1-2 example, the excerpt depicted by Figure 6.2.1-5 shows how the query will be codified and sent through a GET request to the DSRD. Instead, Figure 6.2.1-6 depicts how the request should be for the query depicted by Figure 6.2.1-4.

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf/statements?update=PREFIX%20sim%3A%20
%3Chttp%3A%2F%2Fetsi.stf-
583%2Fdef%2Fsimulation%23%3E%0A%0AINSERT%20DATA%20%7B%0A%20%20%20%20GRA
PH%20%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20%7B%0A%20%20%20%20%20%20%20%20%3Chttp%3A%2
F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20a%20sim%3AVirtualCSE%20.%0A%20%20%20%20%7D
%0A%7D%0A%0A
```

Figure 6.2.1-5: Sending query from Figure 6.2.1-2 to the DSRD using the SPARQL protocol

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf/statements?update=
CLEAR%20GRAPH%20%3Chttp%3A%2F%2Fetsi.stf-584%2Fsamples%2FCSE%2F1%3E%20
```

Figure 6.2.1-6: Sending query from Figure 6.2.1-4 to the DSRD using the SPARQL protocol

6.2.2 DSRD API for Semantic Descriptors

The DSRD API for Semantic Descriptors counts with two basic operations: register one semantic descriptor into an existing virtual CSE, or delete an existing descriptor from such virtual CSE. As explained, these operations are performed using SPARQL queries. For this end, the following templates have been created in order to ease the use of the DSRD. All identifiers have to be URIs.

- The query template for creating a semantic descriptor into an existing CSE is depicted by Figure 6.2.2-1. The `#CSE_id` has to be replaced with a valid URI of an existing virtual CSE, `#SM_id` with a valid URI that belongs to the semantic descriptor resource types as wot:Thing or any of its subclasses according to the DSRD ontology, and finally, the `<#SM>` has to be replaced with the semantic descriptor RDF. Figure 6.2.2-2 depicts a sample instantiation of the template that inserts the description of a sensor that measures the temperature of one room, which building is located in Madrid.

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


INSERT DATA {
 GRAPH <#CSE_id> {
 <#CSE_id> sim:hasRegistered <#SM_id> .
 <#SM>
 }
```

Figure 6.2.2-1: Query template for registering a semantic descriptor into a virtual CSE

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>
PREFIX saref: <https://saref.etsi.org/core/>
PREFIX geosp: <http://www.opengis.net/ont/geosparql#>
PREFIX s4city: <https://w3id.org/def/saref4city#>
PREFIX s4bdlg: <https://saref.etsi.org/saref4bldg/>


INSERT DATA {
 #CSE_id was replaced for http://etsi.stf-584/samples/CSE/1
 #SM_id was replaced for http://etsi.stf-584/samples/SM/1
 GRAPH <http://etsi.stf-584/samples/CSE/1> {
 sim:hasRegistered <http://etsi.stf-584/samples/SM/1> .
 #SM was replaced for the following RDF:
 <http://etsi.stf-584/samples/SM/1> a saref:Device ;
 sim:isRegistedIn <http://etsi.stf-584/samples/CSE/1>;
 saref:masuresProperty [
 a saref:Temperature ;
 saref:isPropertyOf [
 a sim:Roof ;
 s4bdlg:isSpaceOf <http://etsi.stf-584/samples/Building/1> ;
 ]
 ].
```

Figure 6.2.2-2: Sample instantiation of the template for creating registering a semantic descriptor

- Query template for deleting an existing semantic descriptor from a virtual CSE is depicted by Figure 6.2.2-3. Consider that #CSE_id has to be replaced with a valid URI, #SM_id has to be replaced with the identifier of the semantic descriptor that will be removed, and SM_full_triples has to be replaced with all the RDF triples from the semantic descriptor containing non-blank nodes that will be removed. Figure 6.2.2-4 and Figure 6.2.2-5 depict two sample instantiations of the template.

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


DELETE DATA {
 GRAPH <#CSE_id> {
 <#CSE_id> sim:hasRegistered <#SM_id> .
 <#SM_full_triples>
 }
```

Figure 6.2.2-3: Query template for removing existing sematic descriptor from virtual CSE

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>
PREFIX saref: <https://saref.etsi.org/core/>
PREFIX geosp: <http://www.opengis.net/ont/geosparql#>
PREFIX s4city: <https://w3id.org/def/saref4city#>
PREFIX s4bdlg: <https://saref.etsi.org/saref4bldg/>


DELETE DATA {
 #CSE_id was replaced for http://etsi.stf-584/samples/CSE/1
 #SM_id was replaced for http://etsi.stf-584/samples/SM/1
 GRAPH <http://etsi.stf-584/samples/CSE/1> {
 <http://etsi.stf-584/samples/CSE/1> sim:hasRegistered
<http://etsi.stf-584/samples/SM/1> .
 # Removing information of the device, not from the building
```

Figure 6.2.2-4: Sample instantiation for removing partially a semantic descriptor from Figure 6.2.2-2

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>

PREFIX saref: <https://saref.etsi.org/core/>

PREFIX geosp: <http://www.opengis.net/ont/geosparql#>

PREFIX s4city: <https://w3id.org/def/saref4city#>

PREFIX s4bdlg: <https://saref.etsi.org/saref4bldg/>


DELETE DATA {

 #CSE_id was replaced for http://etsi.stf-584/samples/CSE/1

 #SM_id was replaced for http://etsi.stf-584/samples/SM/1

 GRAPH <http://etsi.stf-584/samples/CSE/1> {

 <http://etsi.stf-584/samples/CSE/1> sim:hasRegistered
<http://etsi.stf-584/samples/SM/1> .

 #Removing information of the device
```

Figure 6.2.2-5: Sample instantiation for removing the semantic descriptor from Figure 6.2.2-2

As it can be observed, both instantiations depicted in Figure 6.2.2-4 and Figure 6.2.2-5 remove all the triples of the semantic descriptor. However, certain information can be shared by one or more semantic descriptors, like the building on these examples. Different semantic descriptors may describe sensors located in the same building, or even city. For this reason it is suitable for the DSRD to allow users to partially remove a semantic descriptor and preserve the RDF resources that may be shared among other semantic descriptors. Nevertheless, the DSRD allows as well to remove this shared information.

In order to run one of the previous queries in the DSRD simulator, a user should use an instantiated query template (e.g. Figure 6.2.2-4 or Figure 6.2.2-5) and send them using the SPARQL protocol [i.13]. The excerpt shown in Figure 6.2.2-6 shows how the request for sending the query depicted by Figure 6.2.2-4 should be.

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf/statements?update=
PREFIX%20sim%3A%20%3Chttp%3A%2F%2Fetsi.stf-
583%2Fdef%2Fsimulation%23%3E%0APREFIX%20saref%3A%20%3Chttps%3A%2F%2Fw3i
d.org%2Fsaref%23%3E%0APREFIX%20geosp%3A%20%3Chttp%3A%2F%2Fwww.opengis.n
et%2Font%2Fgeosparql%23%3E%0APREFIX%20s4city%3A%20%3Chttps%3A%2F%2Fw3id
.org%2Fdef%2Fsaref4city%23%3E%0APREFIX%20s4bdlg%3A%20%3Chttps%3A%2F%2Fs
aref.etsi.org%2Fsaref4bldg%2F%3E%0A%0ADELETE%20DATA%20%7B%0A%0A%20%20%2
0%20GRAPH%20%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20%7B%0A%20%20%20%20%20%20%20%20%20%20%20%20
%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20sim%3AhasRegistered%20%3Chttp%3A%2F%2Fetsi
.stf-
584%2Fsamples%2FSM%2F1%3E%20.%0A%20%20%20%20%20%20%0A%20%20%20%20%20%20
%20%20%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FSM%2F1%3E%20a%20saref%3ADevice%20.%0A%20%20%20%20%7D%0A
%7D%0A%0A
```

Figure 6.2.2-6: Sending query from Figure 6.2.2-4 to the DSRD using the SPARQL protocol

6.2.3 DSRD API for Semantic Discovery

As previously mentioned, in the oneM2M semantic discovery a SPARQL query is answered with a list of identifiers that belong to those semantic descriptors registered that fulfil the query restrictions. However, this behaviour is not the one established by the SPARQL protocol [i.13]. The DSRD allows to retrieve both types of query answers for a given SPARQL query. The following query templates show how to emulate both behaviours.

- Query template for answering a SPARQL query with a list of semantic descriptor identifiers is depicted by Figure 6.2.3-1. This behaviour is achieved forcing the query to return always the same variable, that is the semantic descriptor resource URI, e.g. the semantic descriptor identifier. Figure 6.2.3-2 depicts an instantiation of the template in which the identifier of any semantic descriptor describing sensors that measures temperature are discovered.

```
SELECT DISTINCT ?#SM_id {
 GRAPH ?cse {
 <#SPARQL_triples>
 }
 values ?cse {<#CSE_id >}
}
```

Figure 6.2.3-1: Query template for solving a SPARQL in a CSE as specified in oneM2M

```
PREFIX saref: <https://saref.etsi.org/core/>

SELECT DISTINCT ?smId {
 GRAPH ?cse {
 #The <#SPARQL_triples> triples are:
 ?smId a saref:Device .
 ?smId saref:masuresProperty ?property .
 ?property a saref:Temperature .
 }
```

Figure 6.2.3-2: Sample instantiation for querying a virtual CSE following oneM2M

- Query template for answering a SPARQL in a virtual CSE following the standard is depicted in Figure 6.2.3-3. Notice that this behaviour is achieved allowing users to specify any variable in the query results.
  Figure 6.2.3-4 depicts an instantiation of the template in which any sensor that measures temperature is discovered. The output of this instance will be more complex than the one of the query from Figure 6.2.3-2.

```
SELECT DISTINCT * {
 GRAPH ?cse {
 <#SPARQL_triples>
 }
 values ?cse {<#CSE_id >}
                                    }
```

Figure 6.2.3-3: Query template for solving a SPARQL in a CSE as specified by the W3C standard

```
PREFIX saref: <https://saref.etsi.org/core/>


SELECT DISTINCT ?smId ?propertyMeasured {
 GRAPH ?cse {
 #The <#SPARQL_triples> triples are:
   ?smId a saref:Device .
   ?smId saref:measuresProperty ?property .
 ?property a ?propertyMeasured .
 }
```

Figure 6.2.3-4: Sample instantiation for querying a virtual CSE following the SPARQL W3C standard

Finally, as mentioned in the previous clauses, in order to run these queries in the DSRD a user will send a request to the SPARQL endpoint of the DSRD. Nevertheless, the type of query used for the semantic discovery is different, and thus, the requests are slightly different. For this end, Figure 6.2.3-5 and Figure 6.2.3-6 show an excerpt of the request that should be performed in order to send the queries depicted in Figure 6.2.3-2 and Figure 6.2.3-4 respectively.

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf?query=PREFIX%20saref%3A%20%3Chttps%3
A%2F%2Fsaref.etsi.org%2Fcore%2F%3E%0A%0ASELECT%20DISTINCT%20%3FsmId%20%
7B%0A%20%20%20%20GRAPH%20%3Fcse%20%7B%0A%20%20%20%20%20%20%20%20%23The%
20%3C%23SPARQL_triples%3E%20triples%20are%3A%0A%09%20%20%20%3FsmId%20a%
20saref%3ADevice%20.%0A%20%20%20%20%20%20%20%20%3FsmId%20saref%3Ameasur
esProperty%20%3Fproperty%20.%0A%20%20%20%20%20%20%20%20%3Fproperty%20a%
20saref%3ATemperature%20.%0A%20%20%20%20%7D%0A%20%20%20%23CSE_id%20was%
20replaced%20for%20http%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%0A%20%20%20%20values%20%3Fcse%20%7B%3Chttp%3A%2
F%2Fetsi.stf-584%2Fsamples%2FCSE%2F1%3E%7D%0A%7D%0A
```

Figure 6.2.3-5: Sending query from Figure 6.2.3-2 to the DSRD using the SPARQL protocol

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf?query=PREFIX%20saref%3A%20%3Chttps%3
A%2F%2Fsaref.etsi.org%2Fcore%2F%3E%0A%0ASELECT%20DISTINCT%20%3FsmId%20%
3FpropertyMeasured%20%7B%0A%20%20%20%20GRAPH%20%3Fcse%20%7B%0A%20%20%20
%20%20%20%20%20%23The%20%3C%23SPARQL_triples%3E%20triples%20are%3A%0A%0
9%20%20%20%3FsmId%20a%20saref%3ADevice%20.%0A%20%20%20%20%20%20%20%20%3
FsmId%20saref%3AmasuresProperty%20%3Fproperty%20.%0A%20%20%20%20%20%20%
20%20%3Fproperty%20a%20%3FpropertyMeasured%20.%0A%20%20%20%20%7D%0A%20%
20%20%23CSE_id%20was%20replaced%20for%20http%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%0A%20%20%20%20values%20%3Fcse%20%7B%3Chttp%3A%2
F%2Fetsi.stf-584%2Fsamples%2FCSE%2F1%3E%7D%0A%7D%0A
```

Figure 6.2.3-6: Sending query from Figure 6.2.3-4 to the DSRD using the SPARQL protocol

6.2.4 Advanced API for CSE interconnectivity

6.2.4.0 Foreword

Although previous clauses provided a basic guide of the operations that the DSRD is able to perform, some advanced ones are available. The relevance of these advanced operations will be discussed later in clause 6.4.

6.2.4.1 Querying several CSEs with one query

Since the DSRD has the implicit knowledge of all the virtual CSEs that a user created, and also, all the semantic descriptors registered in each virtual CSE, the DSRD is able to answer a SPARQL query taking into consideration these CSEs. This functionality not only provides the results that these CSEs would output if queried one by one, but also, provides the traceability of the results output. In other words, the DSRD answers a query over a series of virtual CSEs and is able to return which answer is provided by each CSE. This behaviour can be used with both the oneM2M semantic discovery behaviour, and the SPARQL standard behaviour.

Figure 6.2.4.1-1 shows the template for querying a set of virtual CSEs and specify in the results which answer was provided by which virtual CSE. Figure 6.2.4.1-2 shows an instantiation of the template in which any sensor that measures temperature is discovered following the oneM2M approach, and Figure 6.2.4.1-3 the same instantiation following the SPARQL W3C standard behaviour [i.13]. These instantiations allow to retrieve the provenance of the results.

```
SELECT DISTINCT ?cse {
 GRAPH ?cse {
 <#SPARQL_triples>
 }
 #Remove the line below to target all the created virtual CSE
 values ?cse { <#CSE_id_1> <#CSE_id_2> … >#CSE_id_N>}
}
```

Figure 6.2.4.1-1: Query template for querying a set of virtual CSEs

```
PREFIX saref: <https://saref.etsi.org/core/>


SELECT DISTINCT ?cse ?smId {
 GRAPH ?cse {
 #The <#SPARQL_triples> triples are:
 ?smId a saref:Device .
 ?smId saref:masuresProperty ?property .
 ?property a saref:Temperature .
 }
```

Figure 6.2.4.1-2: Sample instantiation for querying a set of virtual CSE following oneM2M

```
PREFIX saref: <https://saref.etsi.org/core/>


SELECT DISTINCT ?cse ?smId ?propertyMeasured {
 GRAPH ?cse {
 #The <#SPARQL_triples> triples are:
 ?smId a saref:Device .
 ?smId saref:measuresProperty ?property .
 ?property a ?propertyMeasured .
 }
```

Figure 6.2.4.1-3: Sample instantiation for querying a set of virtual CSE following the SPARQL W3C standard

In order to run the previous queries, they will be sent to the DSRD SPARQL endpoint following the W3C standard [i.13]. Figure 6.2.4.1-4 and Figure 6.2.4.1-5 show how the requests should be performed to send the queries from Figure 6.2.4.1-2 and Figure 6.2.4.1-3 respectively.

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf?query=PREFIX%20saref%3A%20%3Chttps%3
A%2F%2Fsaref.etsi.org%2Fcore%2F%3E%0A%0ASELECT%20DISTINCT%20%3Fcse%20%3
FsmId%20%7B%0A%20%20%20%20GRAPH%20%3Fcse%20%7B%0A%20%20%20%20%20%20%20%
20%23The%20%3C%23SPARQL_triples%3E%20triples%20are%3A%0A%09%20%20%20%3F
smId%20a%20saref%3ADevice%20.%0A%20%20%20%20%20%20%20%20%3FsmId%20saref
%3AmasuresProperty%20%3Fproperty%20.%0A%20%20%20%20%20%20%20%20%3Fprope
rty%20a%20saref%3ATemperature%20.%0A%20%20%20%20%7D%0A%20%20%20%20value
s%20%3Fcse%20%7B%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%2C%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F2%3E%2C%20%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F3%3E%7D%0A%7D%0A%0A
```

Figure 6.2.4.1-4: Sending query from Figure 6.2.4.1-2 to the DSRD using oneM2M

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf?query=PREFIX%20saref%3A%20%3Chttps%3
A%2F%2Fsaref.etsi.org%2Fcore%2F%3E%0A%0ASELECT%20DISTINCT%20%3Fcse%20%3
FsmId%20%20%3FpropertyMeasured%20%7B%0A%20%20%20%20GRAPH%20%3Fcse%20%7B
%0A%20%20%20%20%20%20%20%20%23The%20%3C%23SPARQL_triples%3E%20triples%2
0are%3A%0A%09%20%20%20%3FsmId%20a%20saref%3ADevice%20.%0A%20%20%20%20%2
0%20%20%20%3FsmId%20saref%3AmasuresProperty%20%3Fproperty%20.%0A%20%20%
20%20%20%20%20%20%3Fproperty%20a%20%3FpropertyMeasured%20.%0A%20%20%20%
20%7D%0A%20%20%20%20values%20%3Fcse%20%7B%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%2C%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F2%3E%2C%20%3Chttp%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F3%3E%7D%0A%7D%0A%0A
```

Figure 6.2.4.1-5: Sending query from Figure 6.2.4.1-3 to the DSRD using the SPARQL protocol

6.2.4.2 Register a topology of virtual CSEs for distributed discovery

The DSRD aims at simulating a variable number of CSEs from the oneM2M. Nevertheless, it does not aim at providing distributed Semantic Resource Discovery, e.g. a SPARQL query is answered by several CSEs. However, extending the basic ontology of the DSRD users can model relationships between CSEs. This functionality can be used to benchmark proposals that perform distributed semantic discovery as discussed in clause 6.4. Figure 6.2.4.2-1 depicts the extension of the DSRD ontology for providing the distributed discovery functionalities.

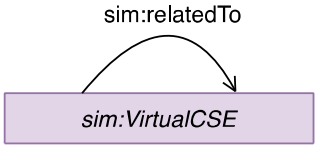

Figure 6.2.4.2-1: DSRD ontology extension for distributed semantic discovery

As it can be observed, the only addition to the original ontology is the relationship `sim:relatedTo`. This relationship represents that a virtual CSE is aware of the existence of others. Although this will now allow to perform distributed semantic discovery behaviours, it can provide support to benchmark proposals that follow such approach. Also, the property `i` entails a direction, and thus, it can be represented by a virtual CSE that is aware of another but this other is not aware of the former virtual CSE.

- Query template for creating a relationship between two virtual CSE is depicted by Figure 6.2.4.2-2, `#CSE_id_1` and `#CSE_id_2` have to be replaced with two valid URIs from existing virtual CSEs. Figure 6.2.4.2-3 depicts a sample instantiation of the template. Figure 6.2.4.2-4 depicts how the instantiated query is sent to the DSRD in order to persist the relationship between virtual CSEs.

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


INSERT DATA {

 <#CSE_id_1> a sim:relatedTo <#CSE_id_2> .

}
```

Figure 6.2.4.2-2: Query template for relating two virtual CSEs

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


INSERT DATA {

 #CSE_id_1 was replaced for http://etsi.stf-584/samples/CSE/1

 #CSE_id_2 was replaced for http://etsi.stf-584/samples/CSE/2

 <http://etsi.stf-584/samples/CSE/1> a sim:relatedTo <http://etsi.stf-
584/samples/CSE/2> .

}
```

Figure 6.2.4.2-3: Sample instantiation of the template for creating a relationship between virtual CSEs

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf/statements?update=
PREFIX%20sim%3A%20%3Chttp%3A%2F%2Fetsi.stf-
583%2Fdef%2Fsimulation%23%3E%0A%0AINSERT%20DATA%20%7B%0A%20%20%20%3C%20
http%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20a%20sim%3ArelatedTo%20%3Chttp%3A%2F%2Fetsi
.stf-584%2Fsamples%2FCSE%2F2%3E%20.%0A%7D%0A%0A
```

Figure 6.2.4.2-4: Sending query from Figure 6.2.4.2-3

- Query template for deleting a relationship between two virtual CSE is depicted by Figure 6.2.4.2-5, `#CSE_id_1` and `#CSE_id_2` have to be replaced with two valid URIs from existing virtual CSEs. Figure 6.2.4.2-6 depicts a sample instantiation of the template. Figure 6.2.4.2-7 depicts how the instantiated query will be sent to the DSRD in order to persist the relationship between virtual CSEs.

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


DELETE DATA {

 <#CSE_id_1> a sim:relatedTo <#CSE_id_2> .

}
```

Figure 6.2.4.2-5: Query template for removing an existing relationship between two virtual CSEs

```
PREFIX sim: <http://sim.etsi.org/stf-589/simulation#>


DELETE DATA {

 #CSE_id_1 was replaced for http://etsi.stf-584/samples/CSE/1

 #CSE_id_2 was replaced for http://etsi.stf-584/samples/CSE/2

 < http://etsi.stf-584/samples/CSE/1> a sim:relatedTo <http://etsi.stf-
584/samples/CSE/2> .

}
```

Figure 6.2.4.2-6: Sample instantiation of the template for deleting a relationship between virtual CSEs

```
HTTP 1.1 GET http://stf-589-
sim.linkeddata.es/repositories/stf/statements?update=
PREFIX%20sim%3A%20%3Chttp%3A%2F%2Fetsi.stf-
583%2Fdef%2Fsimulation%23%3E%0A%0ADELETE%20DATA%20%7B%0A%20%20%20%3C%20
http%3A%2F%2Fetsi.stf-
584%2Fsamples%2FCSE%2F1%3E%20a%20sim%3ArelatedTo%20%3Chttp%3A%2F%2Fetsi
.stf-584%2Fsamples%2FCSE%2F2%3E%20.%0A%7D%0A%0A
```

Figure 6.2.4.2-7: Sending query from Figure 6.2.4.2-6

6.3 oneM2M considerations related to the DSRD simulator

The DSRD simulator aims at providing an environment that simulates the behaviour of oneM2M CSEs. Nevertheless, in oneM2M the semantic discovery is not aligned with the SPARQL W3C standard [i.13]. oneM2M semantic discovery consists of a CSE answering a SPARQL query, and as result, the query answer can only contain the list of identifiers from those registered semantic descriptors that fulfil the query restrictions. On the other hand, the SPARQL W3C standard specifies that the query answer may contain a more complex set of results, rather than only a list of identifiers. The query answers that oneM2M specifies belong to a sub-set of all the potential query answers that the SPARQL standard supports.

As a result, the DSRD extends the semantic discovery that the virtual CSEs implement by offering the possibility of answering a SPARQL query following the oneM2M specification or the SPARQL standard specification. In oneM2M the semantic discovery flow [i.12] and [i.13] is the following: an AE issues a SPARQL query to a CSE, then, for each identifier retrieved the AE fetches from the CSE the semantic descriptors of those identifiers, following the AE aggregates the descriptors, and finally, solves more complex queries that require as result information different from only identifiers. For instance, the query depicted by Figure 6.2.4.2-6 is a kind of query that cannot be solved from a oneM2M CSE. The AE has to perform several requests to the CSE, which degrades the efficiency due to network latencies.

In addition, although the AE retrieves the semantic descriptors some information may not be retrieved preventing complex queries to be solved due to lack of information, and thus, hindering the effectiveness. For instance, if several semantic descriptors point to a RDF resource that represents a building, even if those descriptors are fetched the information about the building will not be retrieved. As a result, a query that looks for sensors located in buildings will not be answered.

Finally, oneM2M may enable distributed semantic discovery implementations, however, the DSRD is not able to simulate such behaviour. Nevertheless, in terms of distributed discovery the DSRD can provide a gold standard to know which CSEs can answer a SPARQL query, which is the answer of each CSE involved, and also, starting from one CSE and following the directional relationship sim:relatedTo to which CSE a distributed semantic discovery proposal could navigate.

6.4 DSRD for benchmarking semantic discovery approaches

In terms of benchmarking the DSRD can be used to measure different KPIs for both semantic discovery and distributed semantic discovery. For semantic discovery the following KPIs can be measured:

- **Semantic Discovery efficiency**: this KPI can be measured by comparing the time that the DSRD takes to answer a SPARQL query with the time that a CSE takes. For this case the queries issued to the DSRD should follow the template for oneM2M.
- **Query answer completeness**: this KPI can be measured by solving the same SPARQL query in a virtual CSE from the DSRD and in a oneM2M CSE. Then, both results can be compared and should be the same.
- **Query answer complexity**: this KPI can be measured by issuing SPARQL queries to the DSRD using a template that follow the W3C standard, then, following the same query to a oneM2M CSE.

For the distributed semantic discovery, the following KPIs can be measured:

- **Distributed Semantic Discovery efficiency**: the KPI can be measured by comparing the time that the DSRD takes to answer a SPARQL query with the time of a distributed semantic discovery implementation. The DSRD have all the information in local, and thus, is not affected by network latencies. As a result, the query answering time of the DSRD can be used as baseline.

- **Query answer completeness**: this KPI can be measured by solving a SPARQL query considering all the virtual CSE from the DSRD and solving the same query with a distributed semantic discovery implementation. The results from the DSRD will contain the identifiers of those CSEs that are suitable to answer such query, and also their results. Therefore, by comparing the results of both proposals it can be known if the distributed semantic discovery implementation query results are complete.

- **Routing completeness**: starting from one virtual CSE the DSRD is able to compute to which other CSEs a distributed semantic discovery implementation should navigate. Therefore, this KPI can be used to know if the implementation hit all the available CSEs.

Combining the KPI from Query answer completeness and Routing completeness, a user may know for a given query which CSEs have to be visited to answer a query, and which would be their results for that query.

6.5 Routing Gold Standard using the DSRD simulator

6.5.0 Foreword

As mentioned, the DSRD simulator does not implement any distributed semantic discovery behaviour. Nevertheless, it allows establishing the relationships that the different service nodes, e.g. CSEs, have among them. Using that information, and since using the simulator it is possible to known in which CSE the different semantic descriptors are such simulator can be used to verify the correctness of a simulation result.

As a result, the DSRD provides a benchmark for the simulation and also a gold standard. For this end, it allows answering two paramount questions related to the routing. On the one hand, for a given CSEs to which other CSEs a query can be routed in order to be correctly be answered, e.g. all the paths that the routing of the distributed discovery can take. On the other hand, for a given query the DSRD returns which CSEs have to be visited during the routing of the distributed discovery, regardless if the topology allows them to be visited. Using these two elements, a gold standard can be provided that for a given query establishes the CSEs that have relevant information for answering such query, and also, which of those are likely to be involved in the discovery due to the topology restrictions. Additionally, the DSRD can answer a given ASD query considering all the semantic descriptors distributed in the simulation, and therefore, providing a gold standard for the answer.

As a result, the data simulator and its benchmark provide a solid set of theoretical results that can be compared with the experimental results. In fact, effectiveness measurements can be established such as Precision, Recall and F1 score when comparing the experimental results and the theoretical ones; additionally, efficiency measurements can be also established. Considering that the simulator solves one query (considering all the CSEs in the simulator) in a centralized service, the query answering time can be considered as the baseline of the experimentation. Therefore, comparing the Advanced Semantic Discovery query answering time with this baseline will show the overhead that the hop-to-hop routing ("fan-out" in oneM2M jargon) between CSEs adds to solving the query in a regular single-service topology.

For this end, the DSRD has a synthetic data generator called Semantic Descriptor Generator (SDG). This generator is able to automatically setup an experimental environment with a variable number of CSEs with different semantic descriptors that are registered with a certain probability. The generated data follows the ontology depicted by Figure 6.2-1. In order to provide an experimental environment, a data scenario has been generated and uploaded into the DSRD repository (https://labs.etsi.org/rep/iot/smartm2m-semantic-and-query/distributed-semantic-resource-directory); this scenario can be downloaded and loaded into any SPARQL endpoint that allows UPDATE queries and named graphs. Notice that the data scenario does not include relationships among CSEs, which will be created by a user. Clause 6.5.1 profiles the data scenario.

6.5.1 Sample experimental scenario for benchmarking

The SDG is able to automatically build a set of semantic descriptors with different features. The produced semantic descriptors are expressed in RDF, specifically Turtle, and modelled according to the ontology depicted by Figure 6.2-1. Additionally, these semantic descriptors are automatically assigned to a set of CSEs. The SDG relies on a template for the devices that consists of a set triples that relate a sensor type, a property that such sensor measures and also a building space. Table 6.5.1-1 shows an excerpt of these combinations, considering that there are 101 combinations available. The SDG will pick randomly one of these triples and instantiate a semantic descriptor, then, with a probability will assign this semantic descriptor to a CSE, or will discard it.

Table 6.5.1-1: Excerpt of triples used by the SDG to instantiate semantic descriptors

Sensor Type	Measured property	Building space
s4bldg:CoolingTower	saref:Temperature	sim:Room
s4bldg:SpaceHeater	saref:Temperature	sim:Room
sim:FogSensor	saref:Load	sim:BoilerRoom
s4bldg:Humidifier	saref:Humidity	sim:Lounge
s4bldg:DistributionDevice	saref:Occupancy	sim:Corridor
s4bldg:DistributionDevice	saref:Occupancy	sim:RedRoom
…	…	…

In addition, the SDG will assign the devices described in the semantic descriptors generated to a certain building with a probability. For this end, the SDG relies on 99 building instances already created (depicted by Table 6.5.1-2). Anytime a semantic descriptor is generated by the SDG, its building spaces are assigned to one of these buildings with a probability.

Table 6.5.1-2: Excerpt of buildings available in the SDG

Building type	Number of instances
sim:Cathedral	24
sim:HistoricalBuilding	17
sim:Museum	5
sim:Palace	20
s4bldg:Building	33

All the buildings belonging to the namespace *i* are also s4bldg:Bulding; however, the 33 instances of s4bldg:Building in Table 6.5.1-2 are not overlapped with all the other instances shown in the table. An important issue to remark is that two different sensors can be located in the same building. Also, the building spaces were designed to be compatible with the buildings selected.

Relaying on the SDG, a scenario with 1 000 CSE has been created for a sample dataset to be imported in any DSRD deployment. Each of those CSEs have an average of 19,8 semantic descriptors registered. The semantic descriptors created are 19 808, Table 6.5.1-3 shows their types. Any sensor type is always a saref:Device and a wot:Thing according to the ontology depicted in Figure 6.2.0-1.

Table 6.5.1-3: Sensors created in the experimental environment

Sensor types	Count
sim:FogSensor	157
sim:PV	197
sim:PanicButton	616
sim:SoundSensor	1 153
sim:WeatherStation	1 579
saref:Device	19 808
saref:DoorSensor	1 698
saref:LightSwitch	1 360
saref:Lightbulb	1 417
saref:SmokeSensor	1 395

Sensor types	Count
saref:TemperatureSensor	1 579
s4bldg:Alarm	406
s4bldg:Boiler	201
s4bldg:CoolingTower	1 168
s4bldg:DistributionDevice	1 613
s4bldg:ElectricGenerator	204
s4bldg:ElectricMotor	197
s4bldg:Fan	590
s4bldg:FlowMovingDevice	1 692
s4bldg:Humidifier	1 227
s4bldg:SolarDevice	195
s4bldg:SpaceHeater	1 164
wot:Thing	19 808

From those sensors, Table 6.5.1-4 shows the saref:Property that they measure. Similarly, Table 6.5.1-5 shows in which s4bldg:BuildingSpaces those sensors were allocated. Finally, in order to perform the testing the SDG also has a set of 20 SPARQL queries that will be used during the simulation in order to obtain results and compute the KPIs shown in clause 6.4. These queries should be sent to different CSEs in order to study how the routing behaves, and also, keep record of the query answering time and content in order to measure if the routing algorithm provides correct and complete results in an efficient time.

Table 6.5.1-4: Excerpt of properties measured by the generated sensors

Properties measured	Sensors measuring the property
saref:Load	157
sim:BinaryStatus	590
saref:Energy	793
saref:BinaryStatus	1 022
sim:sound	1 153
saref:Humidity	1 227
saref:Smoke	1 395
saref:Motion	1 692
saref:Occupancy	3 311
saref:Light	3 565
saref:Temperature	4 903
saref:Property	19 808

Table 6.5.1-5: Excerpt of building spaces where the generated sensors are allocated

Building Spaces where the sensors are allocated	Count
sim:Balcony	1 197
sim:BoilerRoom	3 165
sim:Corridor	2 296
sim:Entrance	2 600
sim:Garden	1 572
sim:Lounge	2 271
sim:RedRoom	2 808
sim:Roof	1 544
sim:Room	2 355
s4bldg:BuildingSpace	19 808

6.6 ASD requirements simulated by the DSRD simulator

The goal of the DSRD is to assist to simulate the Advanced Semantic Discovery (ASD), which relies on a set of requirements elicited in the document ETSI TR 103 715 [i.2]. For this end, this clause aims to present the requirements elicited and which of those can be simulated using the DSRD. Table 6.6-1 lists these requirements and its coverage, in summary, there are 61 requirements from which the DSRD covers 39, e.g. 64 % of them.

Table 6.6-1: Requirements from ETSI TR 103 715 [i.2] that DSRD can simulate

ETSI TR 103 715 [i.2]	Elaborated requirements	Requirements simulated by DSRD
6.2 Semantic Discovery Agreement (SDA)	1. 3 kinds of relationship between CSE are settled: CUSTOMER-to-PROVIDER (C2P), PEER-to-PEER (P2P) and SIBLING-to-SIBLING (S2S);	Yes
	2. SDA is useful to define VALID and INVALID Advanced Semantic Discovery routing paths;	Yes
	3. SDA promotes/enforces only VALID Advanced Semantic Discovery routing paths, so avoiding "unaccounted routing transit" graphically expressed as "valley", and defined schematically as PROVIDER->CUSTOMER->PROVIDER paths or CUSTOMER->PEER->PEER->PROVIDER paths;	Yes
	4. Each CSE contains a SDA table containing the URI of all CSEs which is connected classified following the C2P, P2P and S2S relationship.	Yes
6.3 Advanced Semantic Discovery (ASD)	1. The oneM2M system provides an Advanced Semantic Discovery (ASD) across a distributed network of IoT nodes within a single oneM2M Service Provider;	Yes
	2. The ASD is performed from any AE, even these ones not belonging to the same Trusted Domain and across different IoT Service Providers;	Yes
	3. The ASD is time and space aware;	Yes
	4. The ASD is guided by a Semantic Recommendation (SR) system, implemented in each CSE, in order to improve the performance of the service;	No
	5. The ASD supports semantic reasoning between the oneM2M base ontology (search on FEATURE_TYPES);	Yes
	6. The oneM2M Access Control Policy is compatible with SDA relationship and includes discovery permissions to support ASD;	No
	7. Each CSE involved in the ASD, have a FEATURE_TYPE-table (aka, as "the local Database") containing, for each FEATURE_TYPE, the list of all IDs of AEs of that type directly registered to the CSE: the FEATURE_TYPE table is kept updated as soon as new AE register or unregister.	Yes
6.4 Advanced Semantic Discovery Query (ASDQ)	1. Advanced Semantic Discovery Query (ASDQ) is sent using the oneM2M primitives;	Yes
	2. Advanced Semantic Discovery Query (ASDQ) includes a query expressed using the (ASDQL) and a set of parameters configuring query-solving characteristics (e.g. the output format, solving the query in a specific sub-graph);	Yes
	3. Advanced Semantic Discovery Query (ASDQ) includes a set of parameters configuring routing options.	No
6.5 Advanced Semantic Discovery Query Language (ASDQL)	1. ASDQL is able to query RDF data, since discovery in oneM2M consists in finding relevant semantic descriptors which are expressed in RDF;	Yes
	2. ASDQL is implemented using a well-known and established standard;	Yes
	3. ASDQL allows expressing queries over graph data;	Yes
	4. ASDQL have the capabilities to use the terms of the oneM2M ontology and any other ontology used by the data.	Yes

ETSI TR 103 715 [i.2]	Elaborated requirements	Requirements simulated by DSRD
6.6 Semantic Discovery Routing Mechanism (SDRM)	1. It is possible to introduce in the ASQL new SEMANTIC FILTER CRITERIA for defining suitable routing policies;	No
	2. The SDRM manages user defined types of objects;	No
	3. The SDRM reduces the ASDQ;	No
	4. The SDRM solves the ASDQ;	Yes
	5. The SDRM forwards the simplified ASDQ;	No
	6. The SDRM reconstructs the partial results sending back to the originator.	Yes
6.7 Semantic Resolution Query Mechanism (SRQM)	1. The SRQM is embedded in each CSE and is able to reduce a complex formula into a CNF (resp. DNF).	No
6.8 Semantic Recommendation System (SRS)	1. The SR can access the SDA-Table and the SRT;	No
	2. The SRT contains a list of most active CSEs;	No
	3. According to the kind of Advanced Semantic Query one CSE can receive (exhaustive vs. non-exhaustive) the SR embedded in that CSE can extract the best CSE-set to forward the query;	No
	4. According to the responses received by the forwarded queries, the SR can annotate each CSE in the SDA-Table leading to successful semantic routing results.	No
6.9 Semantic Routing Table (SRT)	1. The SRT contains a SDA-table;	No
	2. The SRT contains a FEATURE_TYPE table;	No
	3. The SRT refers all the URI of AE registered in the adjacent CSEs, according to the C2P, P2P and S2S relationship;	No
	4. The SRT is kept fresh and updated to avoid FAULTY ROUTES.	No
6.10 Building a CSE Topology Linked with the oneM2M actual topologies	1. The topology of CSEs should be setup in accordance with the SDA. the ASD overlay oneM2M routing will always respect the NO VALLEY property routing by Gao [i.30].	Yes
6.11 Queries Integrating Baseline and Specific Domain Ontology (SAREF)	1. ASDQ refers to terms in the Advanced Semantic Discovery Ontology.	Yes
6.12 Queries Integrating Multiple Set of Targets and Multiplicity of Searches	1. The query language allow expressing a set of URIs that belong to oneM2M systems that implement a query solving mechanism;	Yes
	2. The query protocol is able to solve queries over a set of distributed oneM2M systems specified in the query;	Yes
	3. The query language is able to express a set of URIs that belongs to oneM2M systems publishing RDF documents, which are identified by those URIs;	Yes
	4. The query protocol is able to solve queries over a set of distributed oneM2M URIs that publish RDF.	Yes
6.13 Advanced Queries with Priority	1. Each CSE is equipped of a simple mechanism of packets scheduling, selecting, for a set of query received, the one/s with the biggest priority according to the received FILTER CRITERIA such as, TIME[SEC] or SPACE[METERS].	No
6.14 Performance Requirements of the Advanced Discovery	1. Each CSE is able to register ASD network statistic and send it on demand to the respective IN-CSE.	No
6.15 Semantic Registration of Resources	1. Semantic Registration of Resources is performed using the current oneM2M primitives, as specified in oneM2M TR-0045 [i.12];	Yes
	2. Semantic Registration of Resources includes one or more semantic descriptors profiling meta-data of the oneM2M systems been registered, as specified in oneM2M TR-0045 [i.12];	Yes
	3. Semantic Registration of Resources implements a mechanism to allow accessing the registered semantic descriptors, as specified in oneM2M TR-0045 [i.12];	Yes
	4. Semantic Registration of Resources summarizes semantic descriptors registered using any technique presented in clause 5.6.5.1 of ETSI TR 103 715 [i.2].	No
6.16 Semantic Routing Table Upgrade	1. Each CSE keeps its FEATURE_TYPE table updated;	No
	2. Each CSE participates to keep the SRT of all the CSEs registered in its SDA-table updated using a lightweight NOTIFICATION protocol message suite.	No

ETSI TR 103 715 [i.2]	Elaborated requirements	Requirements simulated by DSRD
6.17 Advanced Semantic Resource Descriptors	1. oneM2M semantic descriptors are expressed in RDF, as specified in oneM2M TR-0045 [i.12];	Yes
	2. oneM2M semantic descriptors are expressed following the Advanced Semantic Discovery Ontology;	Yes
	3. oneM2M semantic descriptors are identified with de-referenceable URIs, entailing that the semantic descriptors are accessible through their URIs and uniquely identified. Nevertheless, security mechanisms may prevent those URIs to be public;	Yes
	4. Advanced Semantic Discovery Ontology contains domain specific terms (e.g. SAREF extensions);	Yes
	5. Advanced Semantic Discovery Ontology contains information on how to interact with the data of the corresponding systems;	Yes
	6. Advanced Semantic Discovery Ontology contains information on how to validate the data of the corresponding systems.	Yes
6.18 Semantic Data Representation	1. AEs can express their data using heterogeneous formats (JSON or XML), nevertheless, they also provide an RDF version of such data;	Yes
	2. CSEs, either MN or IN, provide an RDF version of the data provided by the AEs registered when they do not provide such data in an RDF version;	No
	3. RDF data of oneM2M systems (semantic descriptors and data values) are modelled according to the oneM2M ontology, as described in oneM2M TR-0045 [i.12];	Yes
	4. RDF data of oneM2M systems (semantic descriptors and data values) are also modelled according to other OWL ontologies (e.g. the Advanced Semantic Discovery Ontology) to include domain-specific information;	Yes
	5. oneM2M data expressed in RDF have links that point to other RDF data URIs;	Yes
	6. RDF data of oneM2M systems identifies every resource by a URI, as RDF specifies;	Yes
	7. oneM2M systems publish RDF data using de-referenceable URIs (e.g. HTTP URIs);	Yes
	8. RDF data of oneM2M systems is accessible through the oneM2M primitives, although inherent security mechanisms of oneM2M may prevent such data to be public available.	Yes

6.7 DSRD simulation, results, and conclusions

6.7.0 Foreword

Using the dataset previously profiled and generated as explained in clause 6.5.1 and relying on the KPIs specified in clauses 6.4 and 5.2, the current clause first presents a set of hypotheses regarding the existing oneM2M discovery approaches and some improvements to be taken into consideration. Then, this clause presents a set of experiments that advocate some relevant improvements and benefits that the current oneM2M semantic discovery approach could acquire. Before introducing the hypotheses it is important to contextualize that improvements hypothesized and the experiments are applied to the semantic discovery and not directly to the distributed semantic discovery. Specifically, the improvements apply to the Semantic Resource Discovery regardless of if it targets or not a oneM2M `<semanticFanOutPoint>`, e.g. although the improvements are tailored to the semantic discovery, the distributed semantic discovery could benefit from them since it usually consists in applying iteratively semantic discovery operations across a given CSE's topology.

oneM2M has two mechanisms that allow to solve SPARQL queries over previously registered semantic descriptors oneM2M TS-0034 [i.7], namely: Semantic Query and Semantic Resource Discovery. Although both mechanisms have as input a SPARQL query the former provides as answer a standard W3C payload for SPARQL queries, instead, the latter only provides a set of identifiers that are URIs idenfying existing resources oneM2M TS-0034 [i.7]. The following hypotheses aim at proving that the current discovery mechanism, that is the Semantic Resource Discovery, could be improved if it was implemented using the existing Semantic Query mechanism. There is a wide number of reasons to deprecate the current oneM2M Semantic Resource Discovery and upgrade the current Semantic Query as the new discovery mechanism. Following, the hypotheses to support such statement are presented and then, a set of experiments to prove these hypotheses.

- **Hypothesis A**: the proposed upgraded Semantic Query mechanism is more efficient than the current oneM2M Semantic Resource Discovery one. This hypothesis leans towards demonstrating the efficiency KPI of clause 6.4 and KPIs like mean number of messages to fulfil a query, mean response time, and success rate of query.

- **Hypothesis B:** the current oneM2M Semantic Resource Discovery mechanism may suffer of information loss. This hypothesis leans towards demonstrating the KPIs from clause 6.4 completeness and complexity.

6.7.1 Validating Hypothesis A

From a theoretical point of view, the current oneM2M Semantic Resource Discovery returns a list of URIs that are identifiers, which identify the semantic descriptors that can be latter retrieved one by one oneM2M TS-0034 [i.7]. Assuming this behaviour and assuming that any SPARQL query that asks for something more than just a list of identifiers requires a client, an AE, to send first the query and then retrieve one by one the semantic descriptors identified by those identifiers. Then it requires the client to aggregate them, and finally solve the query. The query depicted by Figure 6.7.1-1 illustrates this example.

```
PREFIX saref: <https://saref.etsi.org/core/>

SELECT ?SM_ID ?type {

  ?SM_ID a saref:Device .

  ?SM_ID a ?type .

}
```

Figure 6.7.1-1: Sample SPARQL query to find devices and their types

As it can be observed in Figure 6.7.1-1, the query is looking for anything that is a device and also the type of such device. In order to solve this upgraded Semantic Query, in our solution, the following flow depicted by Figure 6.7.1-2 has to be implemented by any client or AE.

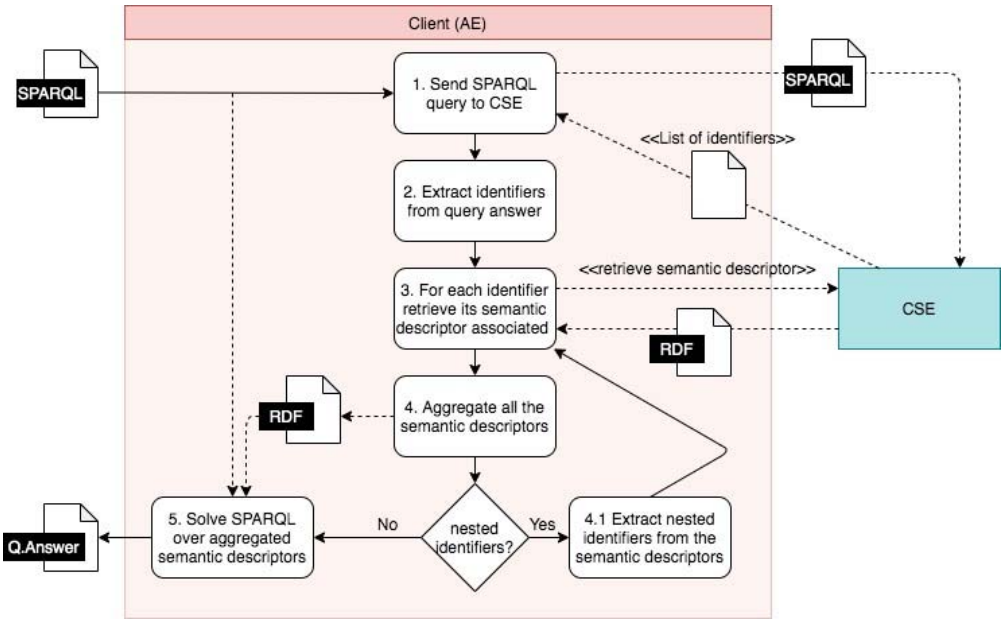

Figure 6.7.1-2: Workflow to be implemented to answer a SPARQL query by means of the current oneM2M Semantic Resource Discovery

Solving the query as depicted by Figure 6.7.1-2 will require a first request to retrieve the list of identifiers (steps 1 and 2), then one request per identifier to retrieve their semantic descriptors and aggregate them (steps 3 and 4). After, if those semantic descriptors reference another one, e.g. they have URLs within that are retrievable identifiers from the CSE, those semantic descriptors have to be retrieved recursively and aggregated (steps 3, 4, and 4.1). Finally the AE has to solve the original query using the aggregated semantic descriptors (step 5).

From a theoretical point of view, the current oneM2M Semantic Resource Discovery mechanism has to make a large number of requests. Instead, in the upgraded Semantic Query approach the query is directly solved in the CSE taking into consideration all the registered semantic descriptors, and thus it only requires one request with the SPARQL query.

As input data for the experiment a set of 13 SPARQL queries has been defined. These queries look for something else than just a set of identifiers and therefore are useful to reproduce the behaviour previously explained. Then, these queries were adapted to behave as the oneM2M Semantic Resource Discovery, e.g. they only return the list of identifiers. Finally, a client was developed to simulate the functionality that the AE should implement depicted by Figure 6.7.1-2.

Using the original queries and the DSRD simulator the query answering time was recorded for each query. Then, the adapted queries were solved also with our client using the DSRD. In order to analyse the impact of having to retrieve a variable number of semantic descriptors the queries were bounded with limit of results, e.g. the number of semantic descriptors involved. The same queries were sent first with a limit of 50, then 100, and so on incrementing the number of semantic descriptors involved by 50 until 500 semantic descriptors. Having that limit does not mean that the query have to return such number of semantic descriptors, but instead, it entails that in the worst case scenario that is the number of identifiers that will be retrieved.

Figure 6.7.1-3 shows the results for the queries solved by the client following the current oneM2M Semantic Resource Discovery approach; instead Figure 6.7.1-4 shows the results for the queries solved directly by the CSE following the upgraded Semantic Query approach. Both Figure 6.7.1-3 and Figure 6.7.1-4 measure the KPIs efficiency by measuring the mean response time.

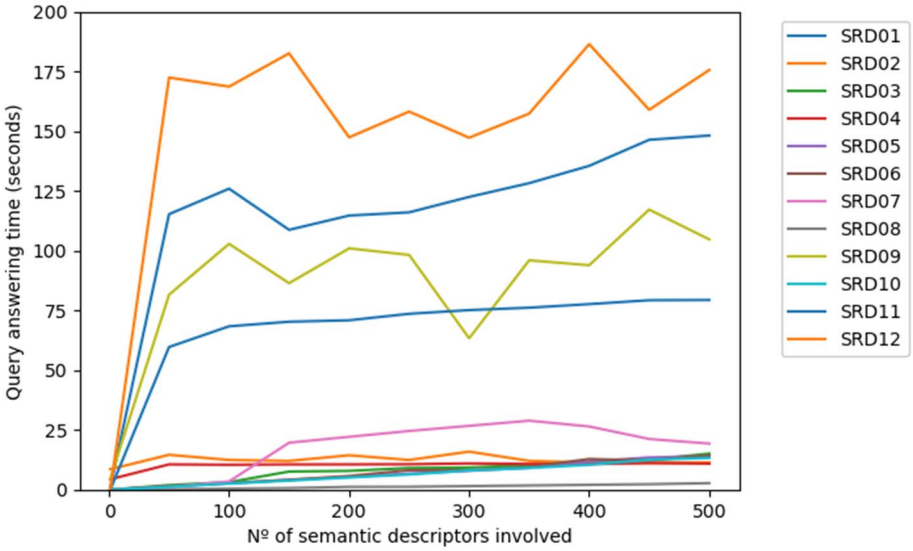

Figure 6.7.1-3: Queries solved by the client following the current oneM2M Semantic Resource Discovery approach

Figure 6.7.1-4 shows the average query answering time. It should be noticed, how the query answering times grows as more semantic descriptors identifiers are allowed to be retrieved. This behaviour is explained by the fact that solving this kind of queries, when 10 semantic descriptors have to be retrieved, will have a query answering time lower since less data has to be processed and then a scenario where 1 000 semantic descriptors have to be retrieved and processed. Finally, it is important to remark that some queries obtain low query answering times, because these queries provide the same number of semantic descriptor identifiers regardless the limit fixed. Additionally some of these queries retrieve semantic descriptors IDs, whose semantic descriptors do not have other nested semantic descriptors identifiers and thus less processing is needed.

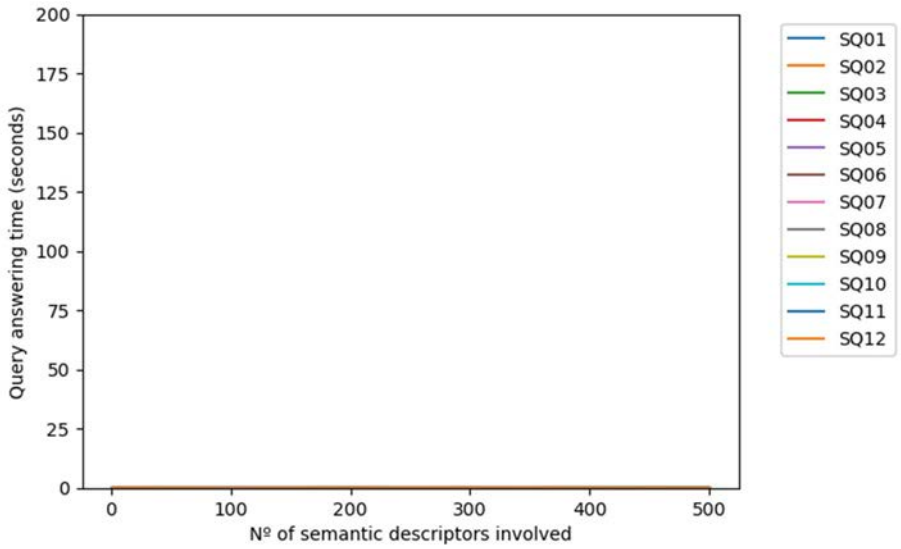

Figure 6.7.1-4: Queries solved by the CSE following the upgraded Semantic Query approach

The results depicted by Figure 6.7.1-3 and Figure 6.7.1-4 clearly show how solving these queries using the current oneM2M Semantic Resource Discovery takes a large amount of time that only increases as the number of semantic descriptors that are involved increases as well. Instead, when the CSE is the one solving the query the answering times are extremely low, less than 1 second observing Figure 6.7.1-4. Due to this reason, Figure 6.7.1-5 shows in detail the results of Figure 6.7.1-4. It can be observed that even if the query answering times increase with the number of semantic descriptors involved the ratio of growth is negligible in comparison with the growth when the query is solved by the AE following the current oneM2M Semantic Resource Discovery.

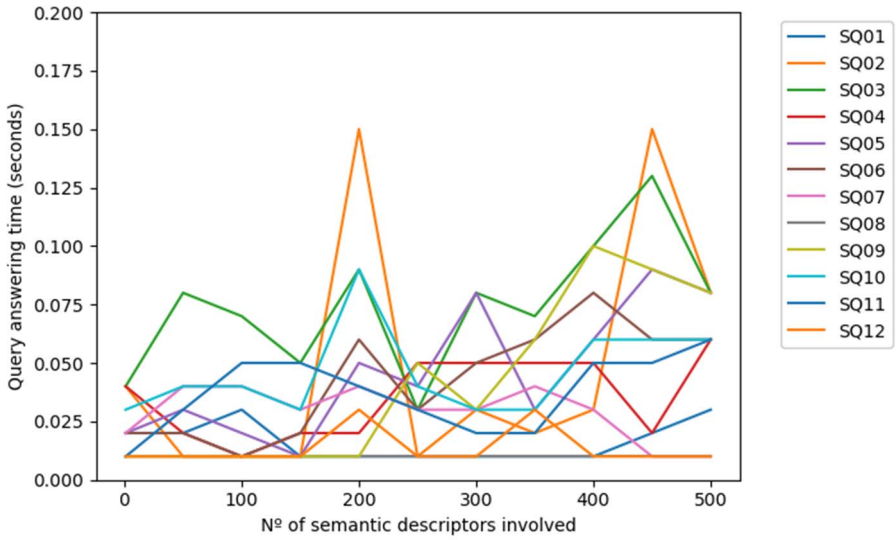

Figure 6.7.1-5: Detailed results for the queries solved by the CSE following the Semantic Query approach

During the previous experimentation the mean number of messages to fulfil a query was also recorded in order to measure this KPI. Figure 6.7.1-6 depicts the results for the current oneM2M Semantic Resource Discovery approach, whereas the results for the upgraded Semantic Query was one request always regardless the limit in the semantic descriptors involved.

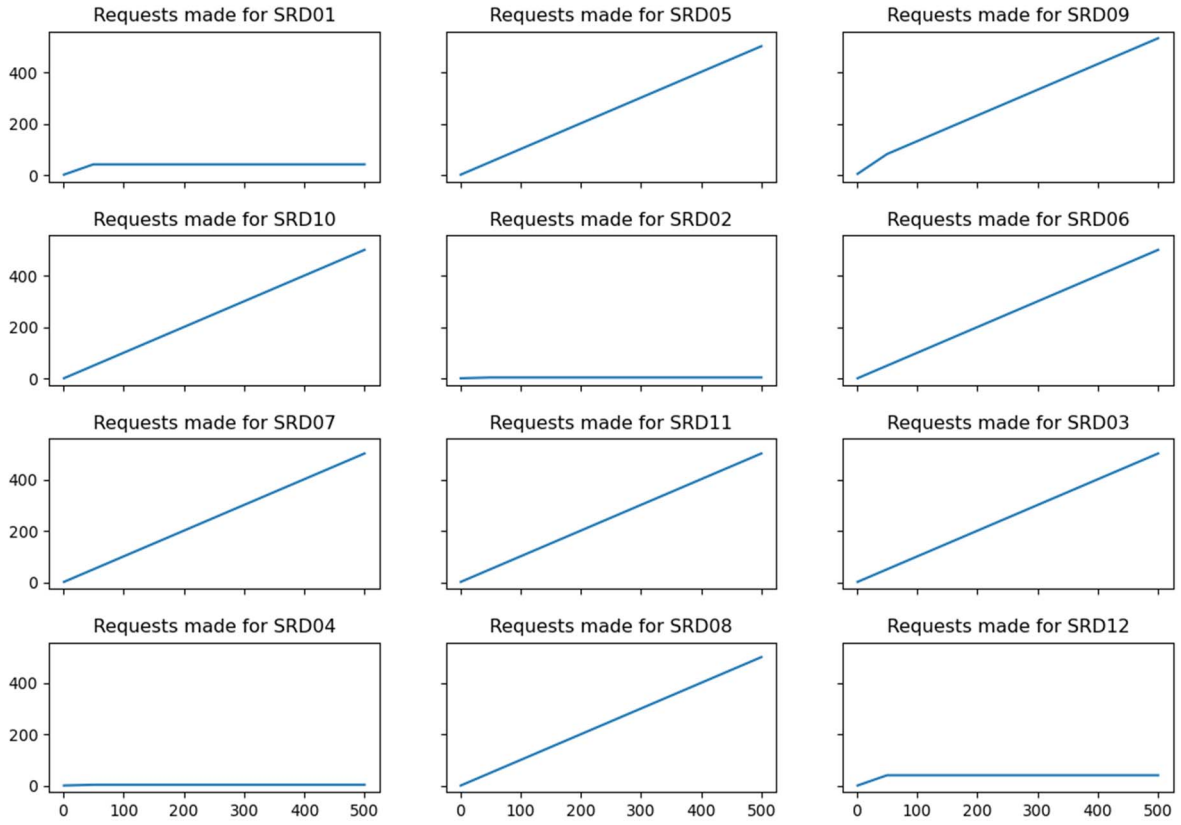

Figure 6.7.1-6: Requests required to solve the different queries using the Semantic Resource Discovery

Figure 6.7.1-6 shows how some queries solved following the semantic resource discovery required a low number of query requests. Notice that SRD01, SRD02, SRD04, and SRD12 always require the same number of requests regardless of the limit in the number of semantic descriptor identifiers that could be retrieved. The reason is because such queries do not provide more identifiers after some limit, i.e. they always return the same.

The results shown for the mean number of messages required to solve a query following the current oneM2M Semantic Resource Discovery have to be compared against the upgraded Semantic Query approach that always requires one request to fulfil any query. Instead, the current oneM2M Semantic Resource Discovery will need a large number of them. The behaviour of the current oneM2M Semantic Resource Discovery leads this approach to require always a wider number of approaches than the upgraded Semantic Query needs obtaining high query answering times. As a final remark towards the KPI of success rate of queries, during these experiments all the executions obtained a 100 % of success.

At the light of these results, it can be concluded that the upgraded Semantic Query approach outperforms the current oneM2M Semantic Resource Discovery for solving SPARQL queries in terms of efficiency and number of requests needed to fulfil one query. In terms of success rate for solving the query both approaches reach the 100 %. The upgraded Semantic Query approach will solve any query that only looks for identifiers as fast as the current oneM2M Semantic Resource Discovery, with the addition that it will also solve more complex queries in a very short time as shown in the experiments.

6.7.2 Validating Hypothesis B

In order to explain how the current oneM2M Semantic Resource Discovery approach may suffer of information loss for solving queries it is important to understand that an oneM2M CSE creates a tree of resources and associates to each resource a semantic descriptor. However, there is a bottom line problem with this mechanism that is the fact that the resources in the CSE tree may not be aligned with all the resources contained in a semantic descriptor. Furthermore, the semantic descriptors are expressed in RDF, and thus, are graphs which cannot be always fitted as a tree of resources as a CSE requires.

The information leak will occur anytime a CSE has one or more URIs that are an object in any semantic descriptor and that are also subjects without a resource associated in the tree of the CSE. For example, assuming the CSE resource tree and semantic descriptors depicted by Figure 6.7.2-1, which assumes the RDF prefixes defined by Figure 6.2-1 and takes the prefix `stf:` for the URI of the CSE would have.

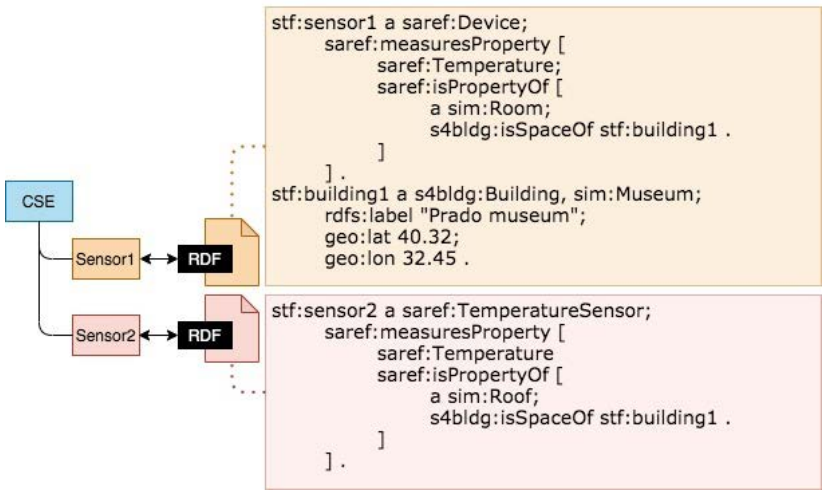

Figure 6.7.2-1: CSE with information leak during the current oneM2M Semantic Resource Discovery

As it can be observed the URI of the building `stf:building1` was stored inside the semantic descriptor of `stf:sensor1`. For this case any attempt to find the building will fail since it does not exist in the resource tree of the CSE. Furthermore, a query that looks for a sensor in a building should return the identifiers `stf:sensor1` and `stf:sensor2` since both are located in the same building. Nevertheless, since the semantic resource directory mechanism tries to solve the query isolate over one semantic descriptor, in this case only the `stf:sensor1` would be provided as an answer because the `stf:sensor2` does not have the information of the building, only the pointer. Additionally, if a client would request the semantic descriptor of the `stf:sensor1` only the triples where the `stf:sensor1` is a subject will be retrieved. As a result, the information of `stf:building1` will be only useful to filter the `stf:sensor1` (although other semantic descriptors may reference this URI) and will never be retrievable.

The only way to prevent this information leak is to create a resource in the CSE tree for each subject in the RDF identifying a real-world concept, which is also the way that RDF data has to be published according to the W3C [i.31]. However, this will create very verbose resource trees and will require users to split an RDF document into very tiny RDF sub-documents each sub-document containing the set of triples with a common subject. Performing this division is not a trivial task and will heavily depend on the information requirements that an AE may have.

On the opposite, the upgraded Semantic Query mechanism does not have information leak since the query is always solved by the CSE which is considering and taking into account all the semantic descriptors stored. Therefore, regardless the CSE resource tree, the queries will always find all the data required to provide a full and correct answer, because the query solving does not attempt to find suitable semantic resources one by one that may fulfil the restrictions of the query. Instead, for the upgraded Semantic Query the CSE solves the query over all the semantic descriptors all together at once, achieving complete and correct results.

6.7.3 The moral of the DSRD simulator

At the light of the results presented in the previous clauses, the upgraded Semantic Query mechanism is on the one hand more efficient in time and able to answer more complex queries than the current oneM2M Semantic Resource Discovery; on the other hand, it is able to always provide complete and correct answers. The upgraded Semantic Query is fully aligned with the W3C SPARQL standard [i.13], whereas the current oneM2M Semantic Resource Discovery is not. As a conclusion of this clause, the reccomendation is that the current oneM2M Semantic Query mechanism should be upgraded as a discovery mechanism replacing the current oneM2M Semantic Resource Discovery. These benefits are transferable when performing a distributed Semantic Discovery as well.

Annex A:
Source Code of the OMNeT++ ASD Network Simulator and more (large scale) ASD network simulations and of the DSRD simulator

See https://labs.etsi.org/rep/iot/smartm2m-semantic-and-query/omnet-asd-network-simulation.

See https://labs.etsi.org/rep/iot/smartm2m-semantic-and-query/distributed-semantic-resource-directory.

Annex B:
Change History

Date	Version	Information about changes
May 2020	0.0.1	TR Skeleton derived from the TR template, clauses 1-3 filled with initial text
June 2020	0.0.2	Early Draft proposal (Inria)
June 2020	0.1.0	Early Draft version for review at TC SmartM2M #54
December 2020	0.1.0	Clean-up done by ***editHelp!*** E-mail: mailto:edithelp@etsi.org
December 2020	0.2.0	Stable Draft version including modifications by UPM+INRIA+EDITHELP, adding TOC and contents by UPM and TOC by Inria
December 2020	0.2.2	Modifications after the 9/12 meeting
December 2020	0.2.4	Modifications after pass by MAP
December 2020	0.2.5	Merging Inria and UPM
December 2020	0.2.6	Stable Draft version for review by TC SmartM2M
December 2020	0.3.0	Adding material by Inria +JK + MAP + Annex A
January 2021	0.4.0	Adding material by UPM + AC+RGC
January 2021	0.4.1	General pass by INRIA+UPM
February 2021	0.4.2	General pass by INRIA
February 2021	0.4.3	Final pass by INRIA with adds by OK and AQK
February 2021	0.5.0	Final Draft for approval by TC SmartM2M
March 2021	1.1.1	Final check by ETSI Technical Officer for EditHelp publication pre-processing
April 2021	1.1.1	Final check by the TR rapporteur LL

History

Document history		
V1.1.1	April 2021	Publication