ETSI SmartM2M Technical Report 103715; Study for oneM2M; Discovery and Query solutions analysis & selection

Luigi Liquori, Enrico Scarrone, Marie-Agnès Peraldi-Frati, Seung Myeong Jeong, Andrea Cimmino, Raúl García Castro, Joachim Koss, Abdul Qadir Khan, Sunil Kumar, Sara El Khatab

ETSI TR 103 715 V1.1.1 (2020-11)

SmartM2M;
Study for oneM2M;
Discovery and Query solutions analysis & selection

Reference
DTR/SmartM2M-103715

Keywords
interoperability, IoT, oneM2M, SAREF, semantic

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 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° 7803/88

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is 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 IPR Policy, no investigation, 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.

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.

Executive summary

The oneM2M system has implemented basic native discovery capabilities. 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: requirements, study, simulation and standardization phase. The present document covers the second phase and provides the basis for the other documents. It identifies, defines and analyses relevant approaches with respect to the use cases and requirements developed in ETSI TR 103 714 [i.1]. The most appropriate one will be selected.

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 [i.1], [i.3] and [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 [i.6], the oneM2M semantic approach [i.7], the current oneM2M capabilities and SAREF [i.6], [i.7], [i.8], [i.9], [i.10], [i.11], [i.12], [i.13], [i.14] 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 [i.15]) to extract (and potentially adapt) the applicable components and to assure a smooth interworking with non-oneM2M solutions. This is documented in the present document [i.3].

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/emulator and the simulation results are documented in ETSI TR 103 716 [i.3]. An extract of the simulation results is included in the present document [i.2] and in 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 second of the four phases and is related to the other documents listed below (the present document is highlighted in italic script in the list):

- 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 (this is the present document [i.2]).
- ETSI TR 103 716: SmartM2M; oneM2M Discovery and Query solution(s) simulation and performance evaluation [i.3].
- 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 identifies, defines and analyses relevant approaches with respect to the use cases and requirements developed in ETSI TR 103 714 [i.1]. The most appropriate one will be selected.

The need to plug in the solution on the oneM2M standard will drive the solution analysis, to determine the best approach to be followed. The activity will also look to the query and discovery mechanisms already available, starting from the ones defined by ETSI (e.g. the one included in NGSI-LD [i.15]) to extract (and potentially adapt) the applicable components and to ensure a smooth interworking with non-oneM2M solutions.

The present document is structured as follows:

- Clauses 1 to 3 set the scene and provide references as well as definition of terms, symbols and abbreviations which are used in the present document.

- Clause 4 describes the method used for selecting options for analysis and selection for advanced discovery.
- Clause 5 presents a state of the art related to discovery, i.e. the resources involved in resource discovery, different ontologies and query languages and discovery routing mechanisms.
- Clause 6 extracts and describes requirements from use cases presented in ETSI TR 103 714 [i.1]. For each requirement, it presents the related resources that can be impacted with these new requirements and the potential extensions in oneM2M.
- Clause 7 describes the Advanced Semantic Discovery functionalities and its impact in the oneM2M specification. The clause gives a high-level view of Advanced Semantic Discovery functionalities and a detailed forecast of the impact on the oneM2M specification (e.g. new feature, new resources, new parameters, modifications to the existing ones and on the oneM2M API).

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] oneM2M TS-0001 (V4.7.0): "Functional Architecture".

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

[i.6] ETSI TS 118 101 (V3.9.0): "oneM2M; Functional Architecture (oneM2M TS-0001 version 3.9.0 Release 3)".

[i.7] oneM2M TS-0034 (V4.2.0): "Semantics Support".

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

[i.8] ETSI TS 103 264: "SmartM2M; Smart Applications; Reference Ontology and oneM2M Mapping".

[i.9] ETSI TS 103 410-1: "SmartM2M; Extension to SAREF; Part 1: Energy Domain".

[i.10] ETSI TS 103 410-2: "SmartM2M; Extension to SAREF; Part 2: Environment Domain".

[i.11] ETSI TS 103 410-3: "SmartM2M; Extension to SAREF; Part 3: Building Domain".

[i.12] ETSI TS 103 410-4: "SmartM2M Extension to SAREF Part 4: Smart Cities Domain".

[i.13] ETSI TS 103 410-5: "SmartM2M; Extension to SAREF Part 5: Industry and Manufacturing Domains".

[i.14] ETSI TS 103 410-6: "SmartM2M; Extension to SAREF; Part 6: Smart Agriculture and Food Chain Domain".

[i.15] ETSI GS CIM 009: "Context Information Management (CIM); NGSI-LD API".

[i.16] Bizer, C., Heath, T., & Berners-Lee, T. (2011): "Linked data: The story so far. In Semantic services, interoperability and web applications: emerging concepts" (pp. 205-227). IGI Global.

[i.17] Heath, T., & Bizer, C. (2011): "Linked data: Evolving the web into a global data space. Synthesis lectures on the semantic web: theory and technology" 1(1), 1-136.

[i.18] Bizer, C., Heath, T., & Berners-Lee, T. (2008, April): "Linked data: Principles and state of the art. In World wide web conference" (Vol. 1, p. 40).

[i.19] World Wide Web Consortium (2014): "RDF 1.1 concepts and abstract syntax".

[i.20] W3C Recommendation 27 October 2009: "OWL 2 Web Ontology Language Document Overview" OWL Working Group.

[i.21] IETF RFC 3987 (January 2005): "Internationalized Resource Identifiers (IRIs)". M. Dürst; M. Suignard.

NOTE: Available at http://www.ietf.org/rfc/rfc3987.txt.

[i.22] 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.23] ETSI TS 103 264 (V2.1.1): "SmartM2M; Smart Appliances; Reference Ontology and oneM2M Mapping".

NOTE: After 2017 SAREF evolved and a newer version V3.1.1 (2020-02) of ETSI TS 103 264 was published.

[i.24] Berrueta, D., Phipps, J., Miles, A., Baker, T., & Swick, R. (2008): "Best practice recipes for publishing RDF vocabularies" Working draft, W3C, 7.

[i.25] W3C Recommendation (2013): "SPARQL 1.1 Query Language", 21(10), 778 Harris, S., Seaborne, A. & Prud'hommeaux, E.

[i.26] W3C Recommendation (2013): "SPARQL 1.1 query results JSON format" Seaborne, A., Clark, K. G., Feigenbaum, L. & Torres, E.

[i.27] W3C Recommendation (2013): "SPARQL 1.1 Query results CSV and TSV formats", 21 Seaborne, A.

[i.28] W3C Recommendation (2013): "SPARQL query results XML format" Hawke, S., Beckett, D. & Broekstra, J.

[i.29] W3C Recommendation (2013): "SPARQL 1.1 federated query", 21, 113 Prud'hommeaux, E. & Buil-Aranda, C.

[i.30] W3C Recommendation (2013): "SPARQL 1.1 Protocol" Feigenbaum, L., Williams, G. T. Clark, K. G. & Torres, E.

[i.31] W3C Recommendation 9 April 2020: "Web of Things (WoT) Architecture" Kovatsch, M., Matsukura, R., Lagally, M., Kawaguchi, T., Toumura, K. & Kajimoto, K.

[i.32] W3C Recommendation 9 April 2020: "Web of Things (WoT) Thing Description" Kaebisch, S., Kamiya, T., McCool, M., Charpenay, V. & Kovatsch, M.

[i.33] Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P. A. & Lindström, N. (2019): "JSON-LD 1.1-a JSON-based serialization for Linked Data" (Doctoral dissertation, W3C).

[i.34] S. K. Datta and C. Bonnet, "Advances in Web of Things for IoT Interoperability", 2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW), Taichung, 2018, pp. 1-2, doi: 10.1109/ICCE-China.2018.8448890.

[i.35] Bröring, A., Datta, S. K. & Bonnet, C. (2016, November): "A categorization of discovery technologies for the internet of things". In Proceedings of the 6th International Conference on the Internet of Things (pp. 131-139).

[i.36] Cimmino, A., Poveda-Villalón, M., & García-Castro, R. (2020): "eWoT: A Semantic Interoperability Approach for Heterogeneous IoT Ecosystems Based on the Web of Things". Sensors, 20(3), 822.

[i.37] Charpenay, V., Lefrançois, M., Poveda-Villalón, M., & Käbisch, S. (2020): "Thing Description (TD) Ontology" W3C Editor's Draft.

[i.38] W3C Recommendation 9 April 2020: "Web of Things (WoT) Thing Description" Käbisch, S., Kamiya, T., McCool, M., Charpenay, V. & Kovatsch, M.

[i.39] W3C Recommendation 21 March 2013: "SPARQL 1.1 Update. World Wide Web Consortium" Gearon, P., Passant, A. & Polleres, A.

NOTE: Available at http://www.w3.org/TR/sparql11-update.

[i.40] Zhou, Y., De, S., Wang, W. & Moessner, K. (2016): "Search techniques for the web of things: A taxonomy and survey" Sensors, 16(5), 600.

[i.41] Čebirić, Š., Goasdoué, F., Kondylakis, H., Kotzinos, D., Manolescu, I., Troullinou, G. & Zneika, M. (2019): "Summarizing semantic graphs: a survey". The VLDB Journal, 28(3), 295-327.

[i.42] Goldman, R., & Widom, J. (1997): "Dataguides: Enabling query formulation and optimization in semistructured databases". Stanford.

[i.43] Čebirić, Š., Goasdoué, F., & Manolescu, I. (2015, July): "Query-oriented summarization of RDF graphs". In British International Conference on Databases (pp. 87-91). Springer, Cham.

[i.44] Dudáš, M., Svátek, V. & Mynarz, J. (2015, May): "Dataset summary visualization with LODSight". In European Semantic Web Conference (pp. 36-40). Springer, Cham.

[i.45] Frisendal, Thomas. Visual Design of GraphQL Data: "A Practical Introduction with Legacy Data and Neo4j". Apress, 2018.

[i.46] Rath, Michael, et al.: "Are graph query languages applicable for requirements traceability analysis?" REFSQ Workshops. 2017.

[i.47] ArangoDB Documentation: "Comparing ArangoDB to Neo4j Cypher".

NOTE: Available at https://www.arangodb.com/comparing-arangodb-aql-neo4j-cypher.

[i.48] GraphQL programming language and server libraries support.

NOTE: Available at https://graphql.org/code/#server-libraries.

[i.49] oneM2M TS-0003 (V4.3.0): "Security Solutions".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=32192.

[i.50] Petar Maymounkov et David Mazières, Kademlia: "A Peer-to-Peer Information System Based on the XOR Metric", Springer Berlin Heidelberg, Lecture Notes in Computer Science, 2002.

[i.51] oneM2M TR-0045: "Developer Guide: Implementing Semantics".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=24354.

[i.52] oneM2M TS-0012 (V3.7.3): "oneM2M Base Ontology".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/?docId=20213.

[i.53] oneM2M TS-0030: "Ontology based Interworking".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/?docId=26806.

[i.54] oneM2M TS-0009 (V3.5.0): "HTTP Protocol Binding".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=31202.

[i.55] IETF RFC 1035: "Domain names - Implementation and specification".

[i.56] IETF RFC 3596: "DNS Extensions to Support IP Version 6".

[i.57] IETF RFC 6895: "Domain Name System (DNS) IANA Considerations".

[i.58] oneM2M REQ-2014-0005R01: "Semantics query for device discovery on Inter-M2M SP".

NOTE: Available at http://ftp.onem2m.org/Meetings/REQ/2014%20meetings/20140407_REQ10.0_Berlin/REQ-2014-0005R01-Semantics_query_for_device_discovery_on_Inter-M2M_SP.DOC.

[i.59] oneM2M TR-0001 (V3.1.1): "Use cases collection".

NOTE: Available at https://www.onem2m.org/images/files/deliverables/Release3/TR-0001-Use_Cases_Collection-V3_1_1.pdf.

[i.60] oneM2M TS-0002 (V4.6.0): "oneM2M Requirements".

NOTE: Available at http://member.onem2m.org/Application/documentapp/downloadLatestRevision/default.aspx?docID=29274.

[i.61] GSM Association Official Document IR.67: "IR.67 - DNS/ENUM Guidelines for Service Providers & GRX/IPX Providers".

NOTE: Available at https://www.gsma.com/iot/wp-content/uploads/2012/03/ir6741.pdf.

[i.62] IETF RFC 4271 (January 2006): "A Border Gateway Protocol 4 (BGP-4)". Y. Rekhter, Ed., T. Li, Ed., S. Hares.

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

[i.63] CAIDA.

NOTE: Available at https://www.caida.org/data/as-relationships/.

3 Definition of terms, symbols and abbreviations

3.1 Terms

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

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

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

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

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

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

NOTE: Defining a SDA is important in an Advanced Resource Discovery because the discovery routing can pass out of a single Trusted Domain, and therefore, an Advanced Search can "walk" through many different Trusted Domains with different Access Policies. SDA is set in advance and is respected by Trusted Domains to guarantee "fair play" discovery packet forwarding (see Figure 6.2.1-1: CAIDA topologies: (left) cash flow (right) valid and invalid paths). In oneM2M, 3 kinds of abstract relationships between CSE need to be settled: CUSTOMER-to-PROVIDER (C2P), PEER-to-PEER (P2P) and SIBLING-to-SIBLING (S2S): those relations are useful to set up Advanced Semantic Discovery routing paths intra and inter Trusted Domains and Service Providers.

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

i) reduces the Advanced Semantic Discovery Query (ASDQ) by means of the Semantic Query Resolution (SQR)

ii) solves and forwards in a distributed way the queries

iii) reconstructs the partial results, sending back to the originator of the Advanced Semantic Discovery Query (ASDQ)

NOTE: Generally, as described in [i.1] and [i.58] two kinds of routing are discriminated, namely:

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

2) "Non-exhaustive". As example, even in the case a semantic resource that exists somewhere in the CSEs network, the system will explore part of the distributed network until it will be stopped *because of different reasons (Timeout or exceeded TTL, etc.).*

Semantic Query Resolution (SQR): capability of each CSE to take as input an Advanced Semantic Discovery Query (ASDQ) and as output:

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

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

3.2 Symbols

Void.

3.3 Abbreviations

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

ACK	ACKnowledge message
ACP	Access Control Policy
AE	Application Entity
API	Application Program Interface
AQL	Arango Query Language
AS	Autonomous System
ASD	Advanced Semantic Discovery
ASDQ	Advanced Semantic Discovery Query
ASDQL	Advanced Semantic Discovery Query Language
ASN	Application Service Node
BGP	Border Gateway Protocol
BGP4	Border Gateway Protocol 4
C2P	Customer-to-Provider
CAIDA	Center for Applied Internet Data Analysis
CBOR	Concise Binary Object Representation
CC	Creative Commons
CMDH	Communication Management and Delivery Handling
CNF	Conjunctive Normal Form
CRUD	Create Retrieve Update Delete
CSE	Common Services Entity
CSF	Common Service Function
CSV	Certified Server Validation
DCL	Data Control Language
DDL	Data Definition Language
DIS	Discovery
DNF	Disjunctive Normal Form
DNS	Domain Name System
drt	Desired identifier Result
ETSI	European Telecommunications Standards Institute
FQDN	Fully Qualified Domain Name
GraphQL	Graph Query Language
HTTP	Hypertext Transfer Protocol
IN	Infrastructure Node
IN-CSE	Infrastructure Node - Common Services Entity
IoT	Internet of Things
IP	Internet Protocol
IRR	Internet level Routing Registries
ISP	Internet Service Provider
JSON	JavaScript Object Notation
KETI	Korean Electronics Technology Institute
lbl	label
M2M	Machine-to-Machine
Mca	Reference Point for M2M Communication with AE
Mcc	Reference Point for M2M Communication with CSE
MN	Middle Node
MN-CSE	Middle Node - Common Services Entity
NoSQL	Not only SQL
OA	Optional Announced
OLAP	Online Analytical Processing
OLTP	Online Transactional Processing
OSPF	Open Shortest Path First
OWL	Web Ontology Language
P2P	Peer-to-Peer

PDP	Policy Decision Point
PEP	Policy Enforcement Point
PHP	Hypertext Preprocessor (originally: "Personal Home Page Tools")
PIP	Policy Information Point
PPM	Privacy Policy Manager
PRP	Policy Retrieval Point
QoS	Quality of Service
RBAC	Role Based Access Control
RC	Remote Control
RDBMS	Relational Database Management System
RDF	Resource Description Framework
REST	Representational State Transfer
RS	Routing Recommendation System
S2S	Sibling-to-Sibling
SAREF	Smart Applications REFerence ontology
SDA	Semantic Discovery Agreement
SDPR	Semantic Discovery Routing Protocol
SDQ	Semantic Discovery Query
SDR	Semantic Discovery Routing
SDRM	Semantic Discovery Routing Mechanism
SLA	Service Level Agreement
SP	Service Provider
SPARQL	Simple Protocol and RDF Query Language
SQL	Structured Query Language
SQR	Semantic Query Resolution
SQRM	Semantic Resolution Query Mechanism
SQRS	Semantic Query Resolution System
SR	Semantic Recommendation system
SRT	Semantic Routing Table
SSN	Semantic Sensor Network
TD	Thing Description
TR	Technical Report
TTL	Time to life
UPM	Universidad Politecnica de Madrid
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WoT	Web of Things
XML	eXtensible Markup Language

4 Method for Discovery and Query options analysis and selection

The oneM2M template for the contribution served as the basis for structuring the present document which describes the analysis and the clause for an Advanced Semantic Discovery in oneM2M.

ETSI TR 103 714 [i.1] and the associated uses cases lead to potential requirements for this ASD. Based on this new set of requirements, the current work of the present document aims at identifying possible approaches (existing and new ones) to an extended semantic discovery 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.

5 State of the art related to discovery

5.1 Introduction

5.1.0 Foreword

The objective of clause 5.1 is to identify and present the different components and services involved in the current oneM2M resource discovery process. A discovery starts with a request expressed in a dedicated language, with the objective of retrieving the requested "thing" that matches the required characteristics as indicated in the query, according to the used query language.

In oneM2M this maps into the finding of oneM2M resources matching the query, expressed by means of filter criteria and/or ontology-based characteristics. The request initiated by a so-called originator is routed into the tree-like structure of oneM2M that contains the resources down to the destination.

In oneM2M the queries are always generated by an AE and addressed to one or many CSE, that are known *a priori* by that AE. The queries are expressed in SPARQL language extended to adapt to the oneM2M resources and topology structure, and currently requires to explicitly indicate the query targets (resources/CSEs) to address the query applicability domain.

The communication is matched on *Mca* and *Mcc* interfaces, and the routing is made according to the oneM2M tree-like structure of the oneM2M SP, that at the root (IN-CSE) is potentially connected with the other SP infrastructure in a mesh-like topology. This means that only the SPARQL query language is used, the rest of the SPARQL protocol suite is replaced by the oneM2M communication system.

The oneM2M Access Control Policies are used to regulate the right to access the information. In particular it is described who can discover and who can access the resources and under which conditions, so privacy and access could be properly respected.

To sustain the query, in oneM2M the resources could be associated with a semantic descriptor point to the associated semantic description. Moreover, to supersede the limitations due to the need of indicating the specific nodes in the topology, the implicit assumption that in case of resource that are needed to be generally found without pointing to their hosting CSE, these resources are announced to the CSE that may be target of the queries, typically the IN-CSE.

With Advanced Semantic Discovery (ASD), described in detail in clauses 6 and 7, it is expected that several oneM2M functionalities may need to be enhanced, in particular:

- It is expected that an Overlay Network-based routing mechanism of the queries would need to be introduced to ensure that the query would be properly propagated in the oneM2M system (*intra* and *inter* SP) to match the conditions and the logical and topological indications expressed in the query.
- It is expected that the Query language would need to be extended (in the language or in the associated parameters) to cope with the specific IoT oneM2M architecture and functionality; in particular it would be required to give the flexibility to address a query only by knowing the immediate CSE where an AE, raising the query, is registered-in, without requiring any other knowledge of the whole oneM2M system (e.g. knowledge of other CSEs to which address the query), or even to add non-topological indications such as the maximum response time.
- It is expected that the Access Control Policies (ACP) may need to be improved to cope with the enhanced overlay network discovery capability; in particular the case of inter SP providers may needs to be addressed, specifying ACP extensions to support proper Semantic Discovery Agreements.
- It is expected that the registration, announcement, notification and ontologies of oneM2M resources would need to be extended to cope with the specific overlay network-based features.
- It is expected on *Mcc'* to be able to interact with external systems adopting the standard SPARQL protocols.

In this clause, short references to the mentioned oneM2M resources functionalities and mechanism are provided, in particular to SPARQL usage, topology, intra and inter SP overlay network query routing, announced resources, ACP ontology support, and base ontology usage for syntactic and semantic interoperability.

This state of the art also integrated information about mechanisms specified by organizations that may become relevant to define the enhancements needed by oneM2M standard to support the Advanced Semantic Discovery, described in details in clauses 6 and 7.

5.1.1 Requirements involving Discovery

NOTE: Clause 5.1.1 includes requirements to the attention of oneM2M Change Requests concerning Discovery.

"Discovery" is a concept that is used pervasively, often in different settings, sometimes with a little overload, in oneM2M: as example, resource and object discovery, applications and gateway discovery, ontology discovery, LWM2M object discovery, area network discovery, infrastructure nodes discovery, fog/edge network discovery, just to mention a few. Here all the Requirements containing the keyword "discovery" in [i.60] are listed. The reader can check that most of those requirements are not yet implemented, even in the current oneM2M Release 3.

OSR-056	*The oneM2M System shall enable discovery of usable M2M Applications on an M2M Gateway or at an M2M Device.*	*Implemented in Rel-1*
OSR-057	*The oneM2M System shall enable discovery of M2M Gateways and M2M Devices available to an M2M Application for data exchange.*	*Implemented in Rel-1*
OSR-086 See REQ-2015-0611R02	*The oneM2M System shall enable M2M Gateways to discover M2M Infrastructure Nodes and M2M Devices available for data exchange.*	*Implemented in Rel-1*
OSR-087 See REQ-2015-0611R02	*The oneM2M System shall enable M2M Infrastructure Nodes and M2M Device to discover M2M Gateways available for data exchange.*	*Implemented in Rel-1*
OSR-098 See REQ-2016-0055R02	*The oneM2M system shall be able to support machine socialization functionalities (such as existence discovery, correlated task discovery, message interface discovery and process optimization for multiple machines with same tasks).*	*Not implemented*
OSR-101 See REQ-2017-0008R02	*The oneM2M System shall enable discovery of M2M Application Servers, M2M Management Servers and M2M Devices available to an M2M Gateway for data exchange.*	
OSR -102 See REQ-2017-0008R02	*The oneM2M System shall enable discovery of M2M Gateways available to a M2M Management Server and an M2M Device for data exchange.*	
OSR-0147 See REQ-2018-0011R03	*The oneM2M System shall enable data continuity services to be provided between Edge/Fog Nodes by enabling the discovery, retrieval, and combination of data sets dispersed across the Edge/Fog network.*	*Rel-4/future releases?*
OSR-0162 See REQ-2018-0018R01	*The oneM2M System shall enable the sharing and discovery of service capability information across fog/edge networks.*	*Rel-4*
MGR-002	*The oneM2M System shall provide the capability to discover the M2M Area Networks including information about devices on those networks and the parameters (e.g. topology, protocol) of those networks.*	*Implemented in Rel-1*
ONT-016 See REQ-2015-0521R01	*The oneM2M System shall support storage, management and discovery of ontologies.*	*Not implemented*
ONT-018 See REQ-2018-0057R01	*The oneM2M system shall support semantic query and discovery across heterogeneous ontologies including support of automatic ontology mapping and semantic reasoning.*	*Rel-4*
QRY-001 See REQ-2015-0521R01	*The oneM2M System shall provide capabilities to discover M2M Resources based on semantic descriptions.*	*Implemented in Rel-2*
SER-011	*The oneM2M System shall protect the use of the identity of an M2M Stakeholder within the oneM2M System against discovery and misuse by other stakeholders.*	*Implemented in Rel-1*
LWM2M-004 See REQ-2015-0517R04	*The oneM2M System shall provide the capability for M2M Applications to discover LWM2M Clients using the LWM2M Client's Endpoint Name.*	*Implemented in Rel-2*
LWM2M-005 See REQ-2015-0517R04	*When transparently transporting LWM2M Objects, the oneM2M System shall provide the capability for M2M Applications to discover the defining of LWM2M Objects transported by the oneM2M System.*	*Not implemented*
LWM2M-006 See REQ-2015-0517R04	*When interworking with LWM2M Objects, the oneM2M System shall provide the capability for M2M Applications to discover a LWM2M Object using the LWM2M Object's identifier.*	*Implemented in Rel-2*

Although not listed here, considering closely related to discovery also the following keywords: "Announcement", "Registration", "Notify" and "Subscribe"; the interested reader can try a search in [i.60] also on the above keywords.

5.2 Resource in oneM2M

5.2.1 Resource involved in Semantic Resource Descriptor

5.2.1.1 Introduction

As oneM2M follows RESTful architecture style, its APIs are basically defined as combinations of oneM2M REST resources and CRUD operations. Since oneM2M platform is not just data sharing platform but provides middleware functions (e.g. group fanout), oneM2M resources can be categorized into two:

1) data sharing resources; and

2) functional resources.

In the present document, the focus is to provide semantic discovery and query therefore this introduction illustrates oneM2M resources on the data sharing aspect.

Data sharing resources include <container>, <contentInstance>, <timeSeries>, <timeSeriesInstance> and <flexContainer> resources [i.5]. In release 4 standard, <flexContainerInstance> resource has been added. The notion of data sharing in oneM2M is sharing application data among different applications (i.e. AEs) via platforms (i.e. CSEs). Aforementioned different resource types provide different way of storing and managing data but still enable data sharing.

The target of semantic discovery and query in oneM2M is semantic annotation or triple data (RDF) which is stored with the semanticDescriptor resource type. A <semanticDescriptor> resource can have annotation of other data as well as oneM2M entities (i.e. AE, CSE). When oneM2M entities registers to another, there happens corresponding resource creations (i.e. <AE> and <remoteCSE> resource) and additionally semantic descriptions can be added for other semantics-enabled applications to find an oneM2M entity. Likewise, data sharing resources can be discovered via their semantic description resources.

Table 5.2.1.1-1 illustrates resource type specific attributes of the semanticDescriptor. As the essential attributes of the semanticDescriptor, the descriptor stores triples and the descriptorRepresentation of the serialization format of the descriptor. Addition to them, the ontologyRef refers the ontology definition that is applied to the descriptor.

Table 5.2.1.1-1: Attributes of semanticDescriptor resource type [i.5]

descriptorRepresentation	*Indicates the type used for the serialization of the descriptor attribute, e.g. RDF/XML, OWL/XML.*
descriptor	*Stores a semantic description pertaining to a resource and potentially sub-resources. Such a description shall be according to subject-predicate-object triples as defined in the RDF graph-based data model.*
semanticOpExec	*This attribute cannot be retrieved. Contains a SPARQL query request for execution of semantic operations on the descriptor attribute.*
ontologyRef	*A reference (URI) of the ontology used to represent the information that is stored in the descriptor attribute. If this attribute is not present, the ontologyRef from the parent resource is used if present.*
relatedSemantics	*List of resource identifiers containing related semantic information to be used in processing semantic queries. The resource identifiers may reference either a <group> resource or <semanticDescriptor> resources and <contentInstance> resources with semantic information in their content attributes as indicated by their contentInfo attribute. In the latter case, the resource identifier may reference a <latest> resource representing the most recent <contentInstance> in a container.*
semanticValidated	*A Boolean value representing the validation result of the triples in the descriptor attribute. The validation is against the referenced ontology as pointed by the ontologyRef attribute as well as other associated <semanticDescriptor> resources (and their referenced ontologies) linked by relatedSemantics attribute and triples in the descriptor attribute.*
validationEnable	*A Boolean value indicating whether the triples in the descriptor attribute needs to be validated by the hosting CSE.*

5.2.1.2 Announced resource

An announced resource contains a set of attributes of the original resource. An announced resource is updated automatically by the Hosting CSE of the original resource whenever the original resource changes. The announced resource contains a link to the original resource.

Resource announcement can facilitate resource discovery. The announced resource at a remote CSE can also be used for creating child resources at the remote CSE that are not present as children of the original resource or are not announced children of the original resource.

The following are the resource specification guidelines for resource announcement:

In order to support announcement of resources, the resource template specifies the attributes to be announced for inclusion in the associated announced resource type:

- For each announced <resourceType>, the addition of suffix "Annc" to the original <resourceType> is used to indicate its associated announced resource type. For example, resource <containerAnnc> indicates the announced resource type for <container> resource; <groupAnnc> indicates announced resource type for <group> resource, etc.

Table 5.2.1.2-1: Common Attributes linked to announcement

announceTo	This attribute may be included in a CREATE or UPDATE Request in which case it contains a list of addresses/CSE-IDs where the resource is to be announced. For the case that CSE-IDs are provided, the announced-to CSE decides the location of the announced resources. For the original resource, this attribute is only present if it has been successfully announced to other CSEs. This attribute maintains the list of the resource addresses to the successfully announced resources. Updates on this attribute will trigger new resource announcement or de-announcement. If announceTo attribute includes resource address(s), the present document does not provide any means for validating these address(s) for announcement purposes. It is the responsibility of the Hosting-CSE referenced by the resource address(s) to validate the access privileges of the originator of the Request that triggers the announcement.
announcedAttribute	This attribute is only present at the original resource if some Optional Announced (OA) type attributes have been announced to other CSEs. This attribute maintains the list of the announced Optional Attributes (OA type attributes) in the original resource. Updates to this attribute will trigger new attribute announcement if a new attribute is added or de-announcement if the existing attribute is removed.

5.2.2 Resource distribution in oneM2M

oneM2M architecture allows a distributed system deployment with more than one platform. The oneM2M resources summarized in clause 5.2.1.1 are hosted on the platform (i.e. CSE). When there is an oneM2M system deployed with several CSEs, from an application point of view, resources are distributed on different CSEs. Figure 5.2.2-1 depicts the multiple CSEs deployment and a oneM2M resource distribution situation that the present document is dealing with.

For registration, a Registree already knows its Registrar a priori such as its CSE-ID. With the identifier, the Registree can discover resource(s) on the Registrar simply because it knows where, target resource identifier (e.g. <CSEBase> resource name that was already provisioned for registration), to ask. However, for the other CSE hosted resources, the entity would have very limited information to send such discovery requests. Note that in oneM2M resource addressing scheme an Originator is requested to indicate the target CSE-ID in the target resource identifier unless the CSE is not its Registrar. Also note that there is no request flooding concept, but always a request targets a single resource wherever hosted.

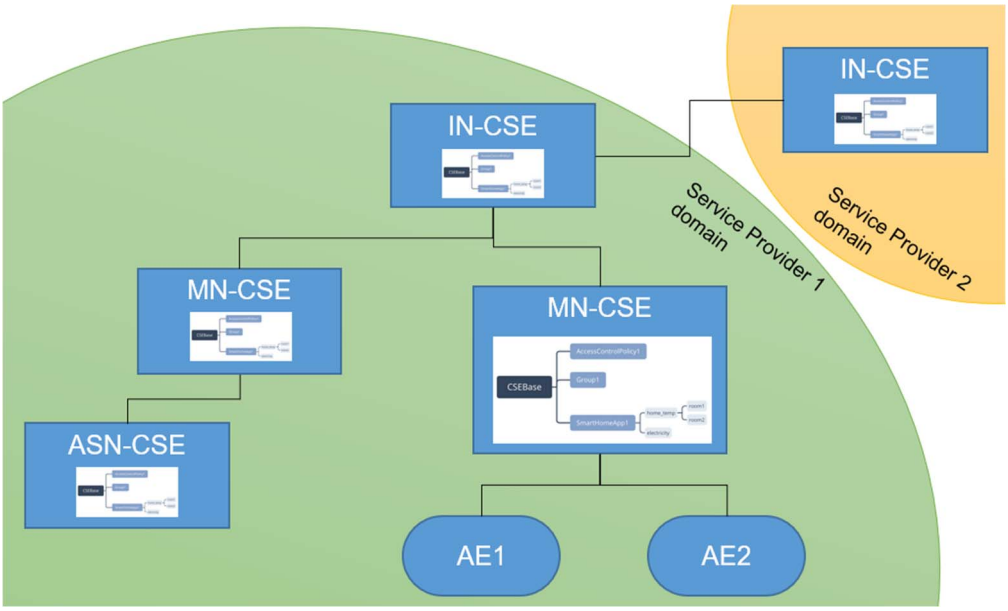


Figure 5.2.2-1: Platform and resource distribution in oneM2M

5.2.3 Resources in W3C Linked Data

Linked data refers to data expressed using the W3C standards that link resources distributed across the Web, providing a unique data set to consume [i.16]. Data has to follow the linked data principles in order to be considered as linked data [i.17]. These principles were drafted in 2005 [i.18]:

1) use URIs to identify things [i.21];

2) use HTTP URIs so that people can look up those names;

3) when someone looks up a URI, provide useful information using the standards (RDF [i.19] or SPARQL [i.25]); and

4) include links to other URIs, so that they can discover more things.

In order for a service to be linked data compliant, it publishes a piece of data that refers to a resource. By request, the data are expressed using the W3C standard Resource Description Framework (RDF) [i.19] and modelled according to an ontology (implemented with the OWL W3C standard) [i.20]. These ontologies could be any of the standard or well-known ones [i.22], or custom ones as long as they are correctly published [i.24]. RDF data relies on URIs to identify resources; in the linked data paradigm these URIs are differentiable and publicly accessible through the HTTP (or HTTPS) protocol [i.21]. Since resources are identified with URIs, the relation of one resource to another will be thus a link that should exist in the Web and should publish linked data as well.

Discovery in linked data relies on the previous assumptions [i.17]:

i) a resource is uniquely identified by and URI;

ii) URIs are differentiable; and

iii) URIs are available on the Web through HTTP (or HTTPS) and provide RDF data that contains relationships to the URIs of other resources.

Therefore, discovery in linked data consists of following the different links until a discovery criterion is fulfilled. In other words, discovery consists of performing a sort of web crawling.

For instance as an assumption, three resources from which two refer to sensors and one to an IoT platform. The RDF data of these three concepts is expressed using the SAREF ontology [i.23]. The first piece of data refers to a sensor named "Temperature Sensor" that has no data observations, such RDF data will be published at the URI http://snsrs.org/1 using the HTTP protocol; when accessed, the URI will return the following sample RDF expressed in the Turtle serialisation.

EXAMPLE 1:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix saref: < https://w3id.org/saref#>.
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
 <http://snsrs.org/1> rdf:type saref:TemperatureSensor;
 saref:hasManufacturer "MLF Corp."^^xsd:string;
 saref:hasModel "MLF-2020"^^xsd:string;
 saref:measuresProperty [
 rdf:type saref:Temperature;
 saref:isMeasuredByDevice <http://snsrs.org/1>, <http://snsrs.org/2>;
 saref:hasMeasurement [
     saref:isMeasuredIn om:degree_Celsius;
 ];
 ];
 saref:hasState [
 rdf:type saref:OffState;
 ].
```

From the piece of data referring to the first sensor, it can be observed that it follows the SAREF ontology in order to specify the type of the resource (saref:TemperatureSensor), and its attributes (saref:hasManufacterer, saref:hasModel). Some relationships to other URIs that are terms from the ontology convey the property measured (saref:Temperature), these state of the sensor (saref:OffState), or the units of the temperature measured (saref:isMeasuredIn om_degree_Celsius). Also, it should be noted how the sensor has a relation to another URI that identifies another sensor identified by the URI http://snsrs.org/2, which is measuring the same temperature property. If the URI http://snsrs.org/2 is accessed, it is requested to publish the RDF data of such sensor. For instance, the following excerpt of Turtle could be published.

EXAMPLE 2:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix saref: < https://w3id.org/saref#>.
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
 <http://snsrs.org/2> rdf:type saref:TemperatureSensor;
 rdfs:label "Temperature sensor 2"^^xsd:string;
 saref:measuresProperty [
 rdf:type saref:Temperature;
 saref:isMeasuredByDevice <http://snsrs.org/1>, <http://snsrs.org/2>;
 saref:hasMeasurement [
 saref:hasValue "32.4"^^xsd:float;
 saref:isMeasuredIn om:degree_Celsius;
 ];
 ];
 saref:hasState [
 rdf:type saref:OnState;
 ];
 saref:isLocatedIn <http://snsrs.org/building>.
```

The RDF excerpt of the relies as well on the SAREF ontology for specifying the type (saref:TemparatureSensor), and the relationships with other resources like http://ssnrs.org/1. The sensor http://snsrs.org/2 is similar to http://snsrs.org/1, nevertheless it should be noted that its sate is saref:OnState and also the sensor is reporting a temperature value (saref:hasValue "32.4"). Additionally, this sensor counts with a relationship that the former sensor lacked of, which is saref:isLocated. The URI http://snsrs.org/building identifies a third resource that is a location. If such URI is accessed the following example excerpt is output.

EXAMPLE 3:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix saref: < https://w3id.org/saref#>.
@prefix geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
<http://snsrs.org/building> rdf:type saref:BuildingSpace;
 saref:hasSpaceType "Laboratory"^^xsd:string;
 geo:lat "52.0705"^^xsd:string;
 geo:long "4.30070"^^xsd:string;
 saref:contains [
 rdf:type saref:Window;
 rdfs:label "Left window"^^xsd:string;
].
```

The previous RDF excerpt denotes a location that is a window (saref:Window) located in a room within a building (saref:BuildingSpace). In the piece of data, the latitude and longitude are reported (geo:lat and geo:long). Additionally, there is no relationship to the previous sensors.

Assuming the previous scenario, further assuming a discovery task that aims at finding sensors that are located in a window of a building, which measure temperature and that are switched on. As a result, there is a need for finding a URI that identifies a saref:TemperatureSensor that has a relationship by means of saref:locatedIn with an URI that identifies a window located in a building, and which also are switched on (saref:OnState) and measure a temperature property (saref:Temperature).

If the discovery task starts from http://snsrs.org/1 it is easy to check that the platform does not fulfil the discovery criteria and, thus, the discovery task is repeated for its known URIs; in this case, http://snsrs.org/2 partially fulfils the discovery criteria. Next, the discovery process will visit the known URIs, that are http://snsrs.org/1 and http://snsrs.org/building, after accessing the former URI the discovery realizes that a loop exists and finish the task. After accessing the latter URI all the discovery criteria are fulfilled and thus, the sensor 2 should be output as result.

If the discovery process starts at the http://snsrs.org/building URI, since this URI has no relationships to others the process would finish without discovering anything. This remarks the relevance of having links to URIs, since it is the basic pillar to discover new data.

5.3 Discovery query languages

5.3.1 oneM2M syntactic discovery query language

oneM2M, which follows RESTful architecture style, defines resources and operations (e.g. CREATE, RETRIEVE) for its interface design. Discovery operation in oneM2M is not specified as a separate one (i.e. DISCOVERY) but uses RETRIEVE with the specific parameter Filter Criteria to indicate discovery request. The Filter Criteria request parameter contains different filter conditions and among them when the filterUsage condition is set as 'discovery' the request becomes discovery request.

Like the other operations, as RESTful API, discovery request is formulated like the others. Target of the discovery is represented as the **To** parameter and discovery queries are carried by the **Filter Criteria** parameter.

Descendant resources of the request target are considered as the discovery targets. This means the resource identified by the **To** parameter is not included for discovery results. To understand this better, oneM2M resource tree structure concept needs to be understood first. Resources are organized in hierarchy, so when a new resource gets created it becomes a child of an existing resource. The target of the discovery request indicates a starting point of a discovery procedure.

Figure 5.3.1-1 provides the resource tree and discovery example. In the diagram, the discovery arrow illustrates that the discovery request targets "Humidity" resource and it contains the **Filter Criteria** (short name "fc"). Then the resources that apply the discovery will be the two as child and grandchild of the "Humidity" resource.

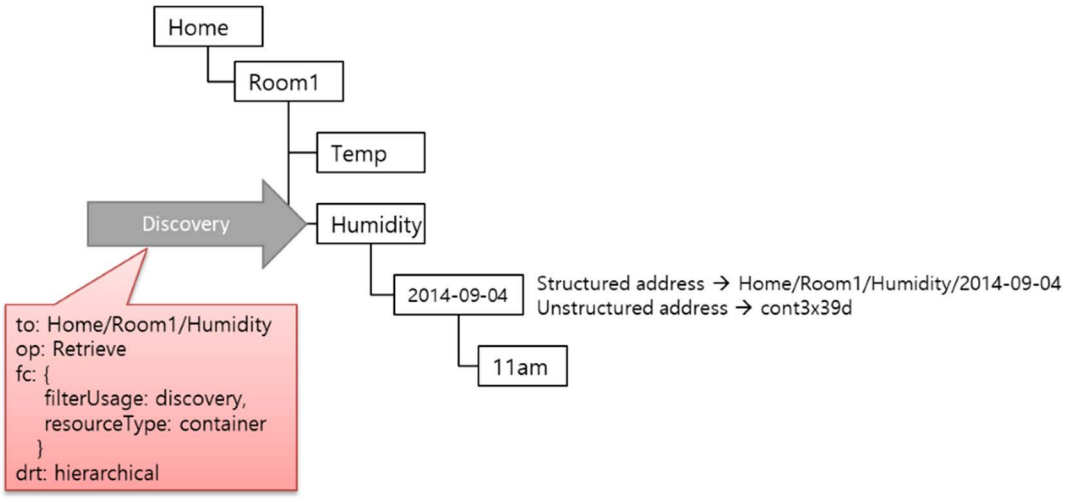


Figure 5.3.1-1: oneM2M resource tree and discovery

Discovery in oneM2M returns a list of resource identifiers that match with conditions defined in the **Filter Criteria**. This is the distinction in oneM2M as discovery vs. query. Normally query can specify what to be returned in which format, but oneM2M discovery by its interface definition returns the resource identifiers.

As well as the matching conditions, the **Filter Criteria** also includes filter handling conditions. Whatever the condition and its behaviour is, as the query language point of view, a discovery condition is delivered as key value pair in the request. For example, to discover the resources having "Sam" in the labels attribute, "lbl = Sam" is serialized depending on the preferred type in the request (lbl is the short name for labels).

Table 5.3.1-1: oneM2M filter conditions

Matching conditions	Filter handling conditions
createdBefore, createdAfter, modifiedSince, unmodifiedSince, stateTagSmaller, stateTagBigger, expireBefore, expireAfter, labels, labelsQuery, childLabels, parentLabels, resourceType, childResourceType, parentResourceType, sizeAbove, sizeBelow, contentType, attribute, childAttribute, parentAttribute, semanticsFilter, filterOperation, contentFilterSyntax, contentFilterQuery, geoQuery, operations	*filterUsage, limit, level, offset, applyRelativePath*

The **Desired identifier Result Type** (drt) parameter defines resource identifier format. The allowed formats that are specified in the oneM2M standards are structured (or hierarchical) and unstructured (or non-hierarchical) identifier. In the example above, structured identifier is made by concatenating resource names in the tree with slashes, while unstructured one is assigned by platform as one string token without slash.

Protocol in oneM2M consists of two levels, one is the core protocol and the other is protocol bindings. The core protocol supports request and response primitive serializations in XML, JSON and CBOR. The **Filter Criteria** parameter settings are serialized depending on oneM2M applications. The below is the JSON serialization request and response example in HTTP binding [i.54].

```
HTTP Request:
GET /~/mn-cse/home_gateway?fu=1&ty=3&drt=2 HTTP/1.1
Host: in.provider.com:8080
X-M2M-Origin: /in-cse/Csmartphone_ae
X-M2M-RI: mncse-99882
Accept: application/json

HTTP Response:
200 OK
X-M2M-RSC: 2000
X-M2M-RI: mncse-99882
X-M2M-CNST: 2
Content-Type: application/json
{
   "m2m:uril":
    [
       "cnt-582759912",
       "cnt-582769893"
    ]
}
```

oneM2M discovery results, array of identifiers, are returned as value of "m2m:uril" XML tag or JSON key (uril stands for URI list). Note that "fu=1", "ty=3" and "drt=2" represent "filterUsage = discovery", "resourceType = container" and "desired resource identifier type = unstructured", respectively.

5.3.2 Cypher-Gremlin, AQL, GraphQL

5.3.2.1 Cypher

Cypher is a graph query language supported for Neo4j graph database, to perform graph traversals as well as Create Retrieve Update Delete (CRUD) operations. Neo4j has been evolving since 2000. However, the first initial development was carried out by Andrés Taylor in early 2011. It is declarative in nature, therefore, the sequence and procedure execution cannot be handled through the query. Cypher is stated to be an expressive with respect to visual and logical representation of graph query patterns and relationship dependencies. Its syntax closely related to that of SQL in order to minimize the learning curve and ease the adaptation for the users. It is open source, and its current implementation.

5.3.2.2 Gremlin

Gremlin is a graph query language supported for Apache TinkerPop, which is a framework for different graph databases. Gremlin has been available since 2009. It is based on Groovy, an object oriented programming language for Java Platform. Gremlin is stated to be a functional and dataflow based query language where the query is structured it terms of functions which directs the graph traversals based on given patterns and constraints. Therefore it supports both imperative and declarative queries. Furthermore, it supports both Online Transactional Processing (OLTP) and Online Analytical Processing (OLAP) database queries. In addition, there are different variants currently available to support wide range of applications. Some of its variants include: Gremlin-Groovy, Gremlin Python, Gremlin-Java, Gremlin-Scala, etc. Due to this wide range of support features, it is supported by different graph vendors Amazon Neptune, Hadoop Giraph, Orient DB, Neo4j, etc.

5.3.2.3 Arango Query Language (AQL)

Arango Query Language (AQL) is supported for ArangoDB, a multimodal database management system developed by triAGENS GmbH. It is used to perform operations regarding data modifications in ArangoDB as well as for graph traversals, so it is not stated as a pure Data Definition Language (DDL) or Data Control Language (DCL). It is a declarative language in nature, and is developed in Java and C language. Its initial version was available in 2012. AQL is used to access all types of datasets stored in ArangoDB, therefore it is not limited to queries related to graph traversals.

In its purpose, AQL is similar to the Structured Query Language (SQL). AQL supports reading and modifying collection data. But unlike other graph data queries, AQL does not support creating and dropping data entities. AQL supports the complex query patterns and the different data model (documents, graphs, and key-values). In addition, AQL also supports different traversal functionalities such as searching path using inbound and outbound relationships, result filtrations, shortest path traversals, etc.

There are two fundamental types of AQL queries, data access queries and data modification queries: Data access queries retrieving data from the database with RETURN operation. Data modification queries support data-modification operations, like INSERT (insert new data), UPDATE (partially update existing data), REPLACE (completely replace existing data), REMOVE (remove existing data), UPSERT (conditionally insert or update data).

5.3.2.4 Graph Query Language (GraphQL)

GraphQL is a query language for supporting APIs and client-side applications, with simpler and flexible data description with syntax similar to JSON. GraphQL does not facilitate complex computations like OLAP features, however, its schema definitions and typed object with hierarchical structures provide the possibility to annotate a graph data having nodes and relationship. Unlike AQL and Cypher, GraphQL is not a language support for specific language or system, rather, it facilitates the application servers the mapping capabilities by translating data into common structure, to support interworking among different applications. GraphQL design consideration supports introspection, where schema is query able and can be examined at runtime. GraphQL enabled servers impose restriction of fixed schema definitions, based on which queries are formulated and executed, due to which most of the graph traversal capabilities cannot be supported or are too complex to formulate.

5.3.2.5 Query Language Summary and Comparison

Table 5.3.2.5-1 describes the feature-wise general comparison of Cypher, Gremlin, AQL and GraphQL, considered for graph database applications.

Table 5.3.2.5-1: Feature-wise General Comparison of Graph Query Languages

Graph Database	Cypher	Gremlin	AQL	GraphQL
Query Language Types	Declarative	Declarative/Imperative	Declarative/data manipulation	Declarative
Shortest Path Search	Yes	Yes	Yes	No
Effort (Query formation)	Low	Medium	Low	High
Readability	Medium	low	Medium	High
Expressiveness	High	medium	High	Low
Language support	Java	Languages with function compositions and function nesting features	Java, Javascript, PHP	Multiple
Access	Embedded, WebSocket, REST	Embedded, Websockets, HTTP	REST	HTTP
OLTP	Yes	Yes	Yes	Yes
OLAP	Yes	Yes	No	No

It can be seen that the GraphQL does not support graph search related functionalities such as shortest path support. The comparison regarding Effort (Query formation), Readability and Expressiveness are adopted from [i.46], where four different query languages are compared including Cypher and Gremlin. In this case the Effort is defined as the number of parameters required in the query formation. Readability has been defined as the difficulty in understanding the query and expressiveness is defined as the ability to formulate the query and its computations in intuitive ways. Here Gremlin has the lowest readability, due its structure based on nested function calls. Whereas GraphQL has high readability due to its structure closely resembled to JSON. Although, being a query language with structure identical to JSON, GraphQL often requires high effort in query formation due to its strict definition of schema [i.45]. Due to the same reason expressiveness for GraphQL is comparatively low, as having fixed defined schema also limits possibilities of making intuitive queries. In order to preserve simplicity, the schema is required in the query in order to get specific response, which omits many other varieties that other graph query languages provides, such as traversing a path of arbitrary length. For comparing these three criteria with AQL, a comparison between AQL and Cypher can be considered, which is provided by ArangoDB documentation [i.47]. There, different operations such as traversals, pattern matching, shortest path search and aggregation have been compared, which puts it close to Cypher. In addition, optimization capability of GraphQL is comparatively low due to some issues like n+1 query problems, in which some queries having nested resource hierarchy, will cause multiple round trip to data access resulting in higher response delays. Wide range of language support is available for GraphQL, where extensive list of support can be found at GraphQL website [i.48].

5.3.3 W3C SPARQL 1.1 Query Language

The SPARQL Query Language is a standard promoted by the W3C that aims at allowing users to write complex queries that will be solved over data described in RDF according to an ontology [i.25]. The SPARQL Query Language contains capabilities for querying required and optional graph patterns along with their conjunctions and disjunctions. Additionally, the SPARQL Query Language counts with a set of standard functions, filtering criteria, solution sequences, and modifiers. Although there are different types of SPARQL queries, all require a graph pattern matching that specifies a set of terms (URIs that can be either properties, types, or resources identifiers), variables, or literals. The graph pattern matching can be as complex as desired, however, since it has to match the data modelled by an ontology usually they can be as complex as the ontology used by the data allows.

The SPARQL Query Language allows specifying different types of queries depending on the operations that have to be performed; there are four types of queries for reading data (SELECT, ASK, DESCRIBE, and CONSTRUCT) and three types for writing data (INSERT, DELETE, and UPDATE). Depending on the type of query, the results will be output in different formats [i.26], [i.27], [i.28]; which are CSV, JSON or XML depending on the content negotiation for SELECT, ASK, INSERT, DELETE, and UPDATE, or directly RDF for DESCRIBE or CONSTRUCT queries.

In addition, the SPARQL Query Language has been designed to meet the distributed nature of RDF data, in which resources are identified with dereferenceable URIs that are publicly available on the Web through the HTTP protocol. For this reason the SPARQL Query Language offers two statements for query federation [i.29], i.e. FROM and SERVICE. The former allows solving a SPARQL query over a set of online URIs by specifying these URIs within the FROM statement. The latter allows solving a SPARQL query over a set of SPARQL endpoints that implement the SPARQL Query Protocol [i.30]. Both FROM and SERVICE statements can be combined in the same query, and used all together, to solve SPARQL queries over distributed RDF data that follows the linked data principles.

For the sake of discovery, the SPARQL query language can be used in different ways depending on how the statements FROM and SERVICE are used. A given SPARQL query can be solved over a set of RDF documents online using the FROM clause, allowing discovering new URIs, which can be iteratively included in the query. As a result, performing a sort of web crawling but relying on a SPARQL query to discover the new URIs and to check whether an RDF document fulfils the discovery criteria. Alternatively, the SERVICE statement can be used to include RDF documents published by a SPARQL endpoint. Bear in mind that a discovery task using these statements can combine both. Nevertheless, notice that discovery in this case is not straightforward and automatic since it requires changing the queries anytime new URIs are discovered and, also, it requires a first URI to issue the query and start the discovery process.

Considering the sample scenario presented in clause 5.3.3, a query to find the URI of a sensor that is located in the window of a building, and which state is on and reporting temperature in relying on the SAREF ontology is the following:

EXAMPLE 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
PREFIX saref: < https://w3id.org/saref#>.
PREFIX geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
PREFIX om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
SELECT DISTINCT ?uri {
     ?uri saref:measuresProperty ?property.
     ?property rdf:type saref:Temperature.
     ?property saref:hasMeasurement ?measurement.
     ?measurement saref:isMeasuredIn om:degree_Celsius.
     ?uri saref:hasState ?sate.
     ?state rdf:type saref:OnState.
     ?uri saref:locatedIn ?building.
     ?building rdf:type saref:BuildingSpace.
     ?building saref:contains ?space.
     ?space rdf:type saref:Window.
}
```

The previous query needs a set of URIs over which it will be issued; since the query is to be solved over the previous scenario, a set of FROM statements should be included in the query.

EXAMPLE 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
PREFIX sosa: <http://www.w3.org/ns/sosa/>.
FROM <http://snsrs.org/building>
FROM <http://snsrs.org/1>
FROM <http://snsrs.org/2>
SELECT DISTINCT ?uri {
     ?uri saref:measuresProperty ?property.
     ?property rdf:type saref:Temperature.
     ?property saref:hasMeasurement ?measurement.
     ?measurement saref:isMeasuredIn om:degree_Celsius.
     ?uri saref:hasState ?sate.
     ?state rdf:type saref:OnState.
     ?uri saref:locatedIn ?building.
     ?building rdf:type saref:BuildingSpace.
     ?building saref:contains ?space.
     ?space rdf:type saref:Window.
}
```

Nevertheless, it should be noted that the previous SPARQL query has to be issued over a SPARQL endpoint, following the SPARQL protocol 1.1 [i.30], which is later explained.

5.3.4 Discovery in the Web of Things (WoT)

In the context of the Internet of Things the W3C has promoted an initiative named Web of Things (WoT). This approach considers any IoT device or service as a WoT Thing [i.31], which provides a set of services through the Web. The services of the Things can either publish data, perform actions, or produce/handle events. Additionally, these services can rely on heterogeneous payloads, security schemas, or protocols.

The WoT provides an identity to the Things relying on the W3C principles, by identifying a Thing with a dereferenced URI, as the Linked Data principles state, that provides an RDF document known as the Thing Description (TD) [i.32]. These Thing Descriptions specify the endpoints related to the Thing and their type of interaction (Property, Action, Event). Additionally, Thing Descriptions provide information about the schema used by the payloads published by those endpoints, their security schemas, protocols, and any relevant information to access the endpoints of a Thing.

The WoT relies on the Thing Descriptions to provide a standardized machine-understandable representation of a Thing with which a potential third-party system will be able to discover, interpret, and interact with a Thing. The Thing Descriptions are an RDF document that follows the ontology specification for Thing Descriptions [i.32]. These documents are expressed in JSON-LD format [i.33], a specific RDF serialization. Besides, TDs can reference other domain-specific ontologies to model pieces of data, like SAREF or SSN.

Currently, the W3C is working on a new standard document for discovery. However, the new discovery is based on the previous one that revolves around the concept of Directory Service, which allows services to perform two operations: registering and discovering Thing Descriptions [i.34]. The action of registering a Thing Description into a Directory Service consists of a Thing sending its Thing Description to such Directory Service [i.35], which stores the document as a central repository. The WoT discovery consists of finding Thing Descriptions registered that fulfil a search criterion. Notice that this discovery does not support finding Things based on the data values that they report [i.36]. The discovery criteria can be either a JSON Path, XPath or a SELECT SPARQL query, and the output can be either a fragment of the Thing Description or the whole document.

In order for a Thing to be discoverable using the WoT approach, first it has to provide a Thing Description. Assuming a thermometer IoT device, a sample Thing Description is the following (adapter from the TD published at https://www.w3.org/TR/wot-thing-description/#simple-thing-description-sample).

EXAMPLE 1:

```
{
 "@context": "https://www.w3.org/2019/wot/td/v1#",
 "id": "http://example-deferenciable-url.com/WoTThermometer-1324",
 "title": "Thermometer",
 "securityDefinitions": {
 "basic_sc": {"scheme": "basic", "in":"header"}
 },
 "security": ["basic_sc"],
 "properties": {
 "status": {
 "type": "string",
 "forms": [{"href": "https://thermometer.example.com/1324/temperature"}]
 }},
 "actions": { },
 "events":{
 "overheating":{
 "data": {"type": "string"},
 "forms": [{
 "href": "https://thermometer.example.com/1324/over_heating",
 "subprotocol": "longpoll"
 }]
 }}}
```

The previous Thing Description uniquely identifies the thermometer with the URI http://example-deferenciable-url.com/WoTThermometer-1324. Additionally, it defines two interaction endpoints for the thermometer that are: the property temperature value at https://thermometer.example.com/1324/temperature, and the event over heating at https://thermometer.example.com/1324/over_heating. Finally, the Thing Description specifies the security used by the thermometer, which is basic username/password authentication in the header.

Having the previous Thing Description, the Thing can be registered in a Thing Directory in order to become discoverable by third-party systems. To this end, the previous Thing Description has to be registered in a Thing Directory, as following described.

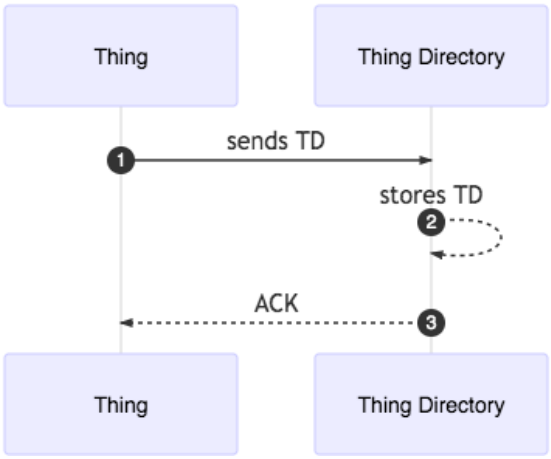


Figure 5.3.4-1: Thing registration in WoT

The Thing is requested to send the Thing Description following the specific Thing Directory API, which usually requires a POST request using either HTTP or HTTPS and with the headers Content-Type application/td+json. Then, the Thing Directory stores internally the Thing Description. Finally an ACK is sent back, which can be a 201 HTTP code or an acknowledgement of any sort.

After the Thing Description has been registered, the Thing will become discoverable to third-party systems through the Thing Directory, as depicted in Figure 5.3.4-2.

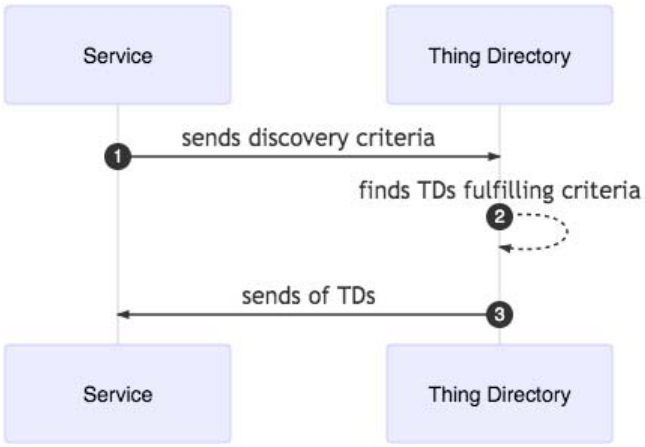


Figure 5.3.4-2: Thing discovery in WoT

When a Service aims at discovering Things it sends a discovery criterion to a Thing Directory, the criterion can be either a SPARQL query entailing that the Thing Directory implements the SPARQL 1.1. protocol, or JSON path entailing that the Thing Directory has a specific API supporting this criteria. Then, the Thing Directory finds those Thing Descriptions stored that fulfil the discovery criteria. Finally, the suitable Thing Descriptions are sent back. Bear in mind that the Thing Descriptions found can be the whole document or a fragment.

Assuming that the Thing Directory only contains the Thing Descriptions previously shown, a query to discover its related Thing, i.e. a thermometer, is the following.

EXAMPLE 2:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
PREFIX wot: <https://www.w3.org/2019/wot/td/v1#>.
SELECT DISTINCT ?uri ?propertyEndpoint {
     ?uri wot:title ?wotTitle.
     ?uri wot:properties ?property.
     ?property wot:forms ?propertyEndpoint.
     FILTER ( CONTAINS(?wotTitle, "Thermometer" ) )
}
```

The previous query will return the value http://example-deferenciable-url.com/WoTThermometer-1324 for the variable ?uri, and https://thermometer.example.com/1324/temperature for the variable ?propertyEndpoint. The query is expressed to find all the Things that have properties, and that contain the term "Thermometer" in its title. The query does not contain any semantic term referencing the type of the IoT device, i.e. a thermometer; this is a shortcoming due to the expressivity of the Thing Description ontology that does not cover this kind of concepts. As a final remark, the WoT discovery could include data values in the queries as filtering criteria as far as a distributed or federated approach is implemented [i.36].

5.4 Ontologies for discovery

5.4.1 oneM2M ontology

Ontologies and their OWL representations are used in oneM2M to provide syntactic and semantic interoperability of the oneM2M System with external systems. These systems are expected to be associated with ontologies that provide a description of the external system.

- Syntactic interoperability allows for interworking with non-oneM2M devices. In this case an ontology - represented as an OWL file - that contains the specific types of communication parameters (names of operations, input/output parameters, their types and structures, etc.) of the non-oneM2M device objects is used to configure an Interworking Proxy Entity that performs the effective mapping to the oneM2M solution.

- Semantic interoperability can be achieved by mapping ontologies from external organizations ultimately onto the oneM2M Base Ontology.

Interworking Proxy for ontology based interworking is described in oneM2M TS-0030 [i.53] (while the oneM2M Base Ontology, is described in oneM2M TS-0012 [i.52]). The oneM2M Base Ontology is the minimal ontology (i.e. mandating the least number of conventions) that is required such that other ontologies can be mapped into oneM2M.

Using Generic Interworking as described in [i.53] with a complete modelling of a functionality according to the oneM2M Base Ontology (or an external functionality represented by an ontology that has been mapped to the oneM2M Base Ontology), would allow the automatic creation of a oneM2M resource structure for the case and enable tool-support for a partially automated creation of the required interworking proxy.

External organizations and companies are expected to contribute their own ontologies that can be mapped (e.g. by sub-classing, equivalence, etc.) to the oneM2M Base Ontology. In such a way neM2M data can be supplemented with information on the meaning/purpose of the data/object.

The oneM2M Base Ontology is provided at the oneM2M Gitlab page, the same page links to an example built on the use of the SAREF ontology (https://git.onem2m.org/MAS/BaseOntology).

Figure 5.4.1-1 shows the mapping of the specific concepts of the simple ontology for the semantic annotation and discovery use case to the more general concepts of the oneM2M Base Ontology [i.52]. Thus, the semantic integration with the concepts identified for oneM2M itself is achieved. Figure 5.4.1-1 is extracted from oneM2M TR-0045 [i.51].

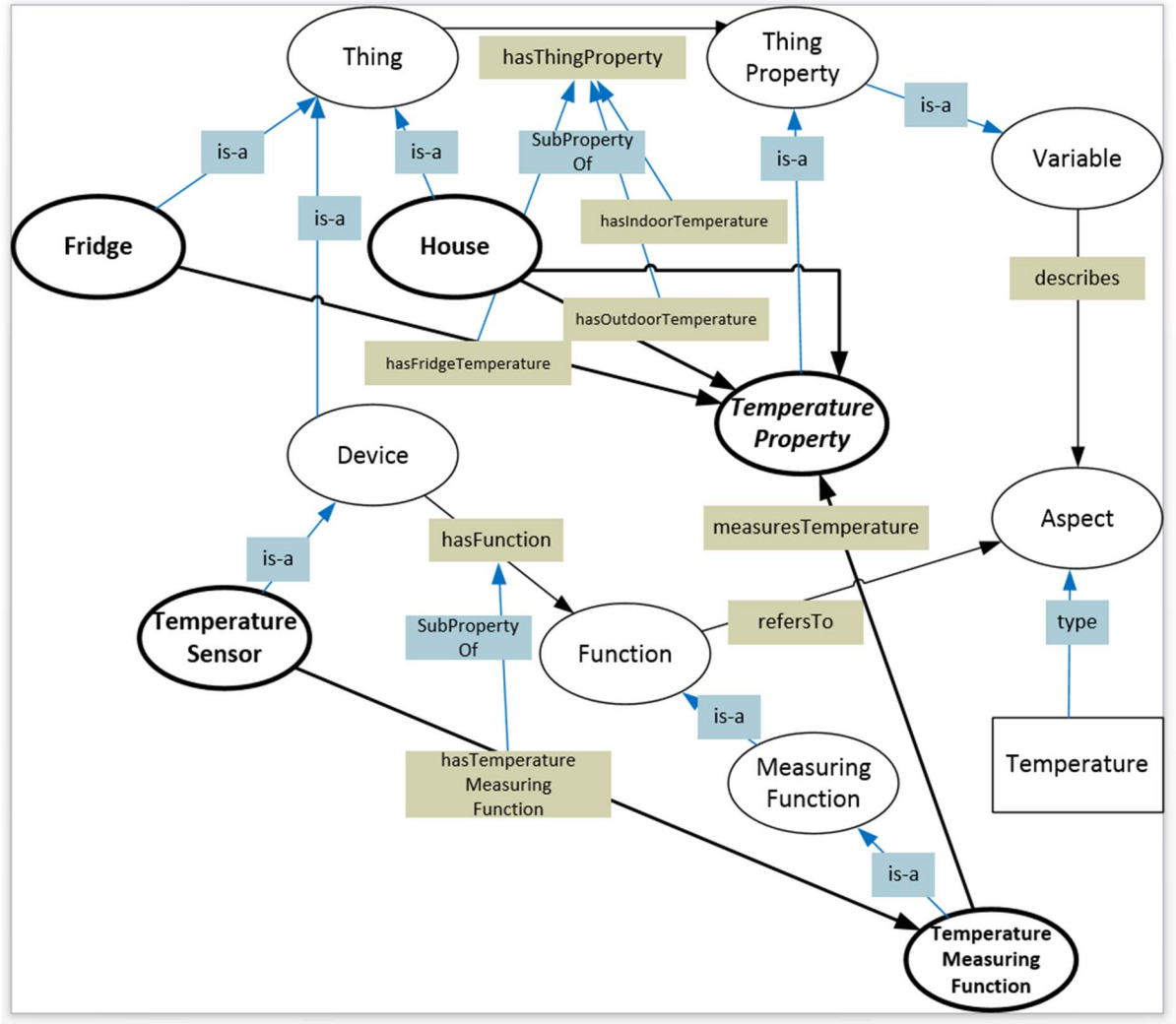


Figure 5.4.1-1: Example of mapping on the oneM2M Base Ontology

5.4.2 W3C Web of Things (WoT)

The W3C Web of Things initiative has promoted the Thing Description ontology [i.37], which is public available at https://www.w3.org/2019/wot/td. This ontology is an idiomatic RDF axiomatisation of the TD information model [i.38], which can be used to describe WoT Things and their interaction patterns (Affordance). This ontology meets the need of expressing an IoT infrastructure (wot:Thing), its endpoints to interact with (wot:InteractionAffordance), and its security schemas. Figure 5.4.1-1 presents an overview of the Thing Description information model covered by the ontology.

Assuming wot: as the prefix for https://www.w3.org/2019/wot/td# and wotsec: for https://www.w3.org/2019/wot/security#, the main terms in the WoT ontology for describing types are: wot:Thing, wotsec:SecurityScheme, wot:InteractionAffordance (and its subtypes wot:PropertyAffordance, wot:ActionAffordance, and wot:EventAffordance), wot:Forms, wot:Links, and wot:MultiLanguage.

As it can be observed, the WoT Thing Description ontology is especially suitable for describing endpoints but falls short describing specific domain data (like types of sensors, or measurements units), and also lacks of a standard ontology module to describe how to translate on the fly data coming from heterogeneous data sources described as WoT Things.

5.4.3 ETSI SAREF

The ETSI Smart Applications REFerence ontology (SAREF) is a standard ontology designed for describing domain specific IoT devices for smart applications. SAREF has been designed relying on the following principles: Reuse concepts and relationships that are defined in existing ontologies, Modularity allowing separation and recombination of fragments coming from different ontologies depending on the specific problem needs, Extensibility allowing easy mechanisms to extend the current ontology, and Maintainability enabling an easy process for identifying and correcting defects, accommodate new requirements, and cope with modifications.

Assuming saref: as prefix for https://saref.etsi.org/core/, the SAREF ontology counts with the following main terms for types: saref:Device, saref:Property, saref:FeatureOfInterest, saref:Measurement, saref:UnitOfMeasure, saref:Service, saref:Function, saref:Command, saref:Sate, saref:Profile, saref:Door_switch, saref:Energy_meter, saref:Energy_related, saref:Energy_unit, saref:Building_related, saref:Close_command, saref:Close_state, saref:Coal, saref:Gas, saref:Get_meter_history_command, saref:Get_sensing_data_command, saref:Humidity, saref:HVAC, saref:Illuminance_unit, saref:Level_control_function, saref:Light, saref:Light_switch, saref:Lighting_device, saref:Load, saref:Measurement, saref:Meter, saref:Metering_function, saref:Occupancy, saref:On_off_function, saref:On_off_state, saref:On_state, saref:Open_state, aref:Power, saref:Power_unit, saref:Pressure, saref:Pressure_unit, saref:Price, saref:Profile, saref:Property, saref:Sensing_function, saref:Sensor, saref:Service, saref:Set_absolute_level_command, saref:Set_level_command, saref:Set_relative_level_command, saref:Smoke, saref:Smoke_sensor, saref:Start_command, saref:Start_state, saref:Start_stop_function, saref:Start_stop_state, saref:State, saref:Switch, saref:Switch_on_service, saref:Task, saref:Temperature, saref:Temperature_sensor, saref:Temperature_unit, saref:Time, saref:Washing_machine, saref:Water.

As it can be observed, the SAREF ontology is especially suitable for describing specific domain data (like sensors, their measurements, or their properties). However, it falls short describing other information relevant for discovery tasks, such as how to access endpoints or how to translate on the fly their data into RDF.

5.5 Discovery query resolution

5.5.1 Introduction

The Discovery (DIS) CSF searches information about applications and services as contained in attributes and resources. The result of a discovery request from an Originator depends upon the filter criteria and is subject to access control policy allowed by M2M Service Subscription. An Originator could be an AE or another CSE. The scope of the search could be within one CSE, or in more than one CSE. The discovery results are returned back to the Originator.

5.5.2 Discovery query rewriting in oneM2M

Requests over the Mca and Mcc reference pointers, from an Originator to a Receiver, may contain optional parameters. Certain parameters may be mandatory or optional depending upon the Requested operation.

In the case of semantic discovery, a decomposition of a complex query into simplest ones have to be considered that will be propagated over different CSEs with a collect of the responses for sending the final response to the originator of the query. To that aim parameters for a request should be considered that can be used during the request propagation. Out of the list of parameters the following is considered:

Response Type: optional response message type: Indicates what type of response is to be sent to the issued request and when the response will be sent to the Originator:

- **nonBlockingRequestSynch:** In case the request is accepted by the Receiver CSE, the Receiver CSE responds, after acceptance, with an Acknowledgement confirming that the Receiver CSE will further process the request. In the response to an accepted request, the Receiver CSE includes a reference that can be used to access the status of the request and the result of the requested operation at a later time. Processing of Non-Blocking Requests is defined in clause 8.2.2 of [i.5] and in particular for the synchronous case in clause 8.2.2.2 of [i.5].

- **nonBlockingRequestAsynch {optional list of notification targets}:** In case the request is accepted by the Receiver CSE, the Receiver CSE responds, after acceptance, with an Acknowledgement confirming that the Receiver CSE will further process the request. The result of the requested operation needs to be sent as notification(s) to the notification target(s) provided optionally within this parameter as a list of entities or to the Originator when no notification target list is provided. When an empty notification target list is provided by the Originator, no notification with the result of the requested operation is sent at all. Processing of Non-Blocking Requests is defined in clause 8.2.2 of [i.5] in particular for the asynchronous case in clause 8.2.2.3 of [i.5].

- **blockingRequest:** In case the request is accepted by the Receiver CSE, the Receiver CSE responds with the result of the requested operation after completion of the requested operation. Processing of Blocking Requests is defined in clause 8.2.1 of [i.5]. This is the default behaviour when the Response Type parameter is not given the request.

- **flexBlocking {optional list of notification targets}:** When **Response Type** in the request received by the Receiver CSE is set to flexBlocking, it means that the Originator of the request has the capability to accept the following types of responses: nonBlockingRequestSynch, nonBlockingRequestAsynch and blockingRequest.

- The Receiver CSE makes the decision to respond using blocking or non-blocking based on its own local context (memory, processing capability, etc.) if not defined in the resource handling procedure. If the Receiver CSE choose to respond using non-blocking mode or blocking mode, based on the presence of notification targets in the request:

  - If the notification targets are provided in the request and the Receiver CSE is responding, the Receiver CSE choose and respond with nonBlockingRequestAsynch, nonBlockingRequestSynch or blockingRequest mode.

  - If notification targets are not provided, the Receiver CSE chooses and respond with nonBlockingRequestSynch or blockingRequest mode.

- Example usage of the response type set to nonBlockingRequestSynch: An Originator that is optimized to minimize communication time and energy consumption wants to express a Request to the receiver CSE and gets an acknowledgement on whether the Request got accepted. After that the Originator may switch into a less power consuming mode and retrieve a Result of the requested Operation at a later time.

Result Persistence: optional result persistence: indicates the time for which the response may persist to. The parameter is used in case of non-blocking request where the result attribute of the <request> resource should be kept at the CSE, for example, with the purpose of sharing, tracking and analytics.

- In the case the response of a request is required to be kept in the CSE, for example the procedures of <request> resource, <delivery> resource and <group> resource, the **Result Persistence** indicates the time duration for which the CSE keeps the response available after receiving it.

- Example usage of result persistence includes requesting sufficient persistence for analytics to process the response content aggregated asynchronously over time. If a result expiration time is specified, then the result persistence lasts beyond the result expiration time.

Delivery Aggregation: optional delivery aggregation on/off: Use CRUD operations of <delivery> resources to express forwarding of one or more original requests to the same target CSE(s). When this parameter is not given in the request, the default behaviour is determined per the provisioned CMDH policy if available. If there is no such CMDH policy, then the default value is "aggregation off".

NOTE: Since **Delivery Aggregation** is optional, there could be a default value to be used when not present in the Request. This parameter could not be exposed to AEs via Mca.

Example usage of delivery aggregation set on: the CSE processing a request uses aggregation of requests to the same target CSE by requesting CREATE of a <delivery> resource on the next CSE on the path to the target CSE.

5.5.3 oneM2M discovery (semantic and non-semantic)

5.5.3.1 Non-semantic/syntactic discovery

The present clause describes how oneM2M non-semantic/syntactic discovery gets performed on oneM2M resource trees. As briefly illustrated in clause 5.3.1, oneM2M discovery target defines discovery starting point and the scope of the discovery is the set of descendant resources.

In the example of Figure 5.5.3.1-1, all resources under "Humidity" are the discovery target. Logically (meaning implantation can be different), when a descendant resource is selected, the oneM2M platform checks if the resource (e.g. "2020-06-04") matches with the conditions in the **Filter Criteria** parameter. When more than one conditions are contained in the discovery request, logical AND operation is applied. When a condition includes more than one value, logical OR operation is applied for matching. When a resource is determined as matching resource its identifier is included in the result.

The scope can be limited with specific conditions limit and level. limit limits the number of identifiers in the results and level indicates the maximum level or depth in the tree. For instance, in the following example diagram, when the level = 1, "2020-06-04", "2020-06-03" and "sub1" become discovery targets. The discovery starting point can be shifted from the **To** parameter with offset condition.

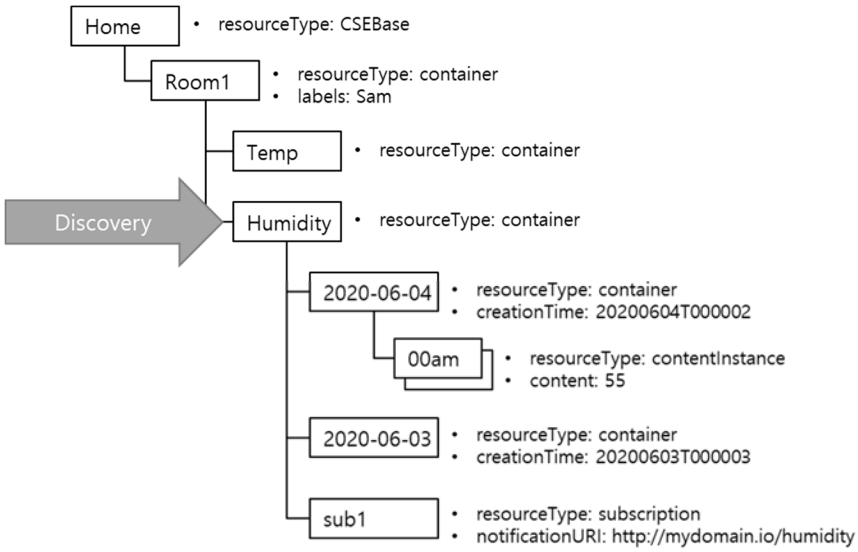


Figure 5.5.3.1-1: oneM2M resource tree example

Addition to the above discovery procedure, access control is applied for discovery. oneM2M access control defines privileges per operation type. Even discovery is not a separate operation, privileges include DISCOVERY. This means that applications or users can set whether a resource can be discovered by others or not. Only if the Originator has privileges to discover, corresponding resource can be included in the result.

5.5.3.2 Semantic discovery

The fundamental difference between oneM2M semantic and non-semantic discovery is that semantic discovery is performed on triple database with SPARQL while non-semantic discovery applies to oneM2M resource tree, note that databases e.g. RDBMS, NoSQL, for oneM2M resource tree is implementation specific. In the request it is decided whether the **Filter Criteria** parameter includes semanticFilter condition. The semanticFilter condition delivers SPARQL query.

Semantic query or discovery in oneM2M is performing SPARQL against the semantic description in oneM2M platform. In oneM2M, semantic description is stored in <semanticDescriptor> resources. When a <semanticDescriptor> resource gets created, updated or deleted, it is applied to the triple store by implementation. This means SPARQL query contained in the request primitive, is performed over semantic graphs on triple store.

The scope for semantic query and discovery is determined and performed on the triple store. When a semantic query or discovery is requested targeting a specific resource on the tree, descendant <semanticDescriptor> resources will form a discovery target set and only those resources and corresponding graphs on the triple store will be the targets.

Semantic discovery returns identifiers of <semanticDescriptor> resources that match with SPARQL. By the **Semantic Query Indicator** parameter setting, semantic query can be requested. In that case, SPARQL query result is send back to applications.

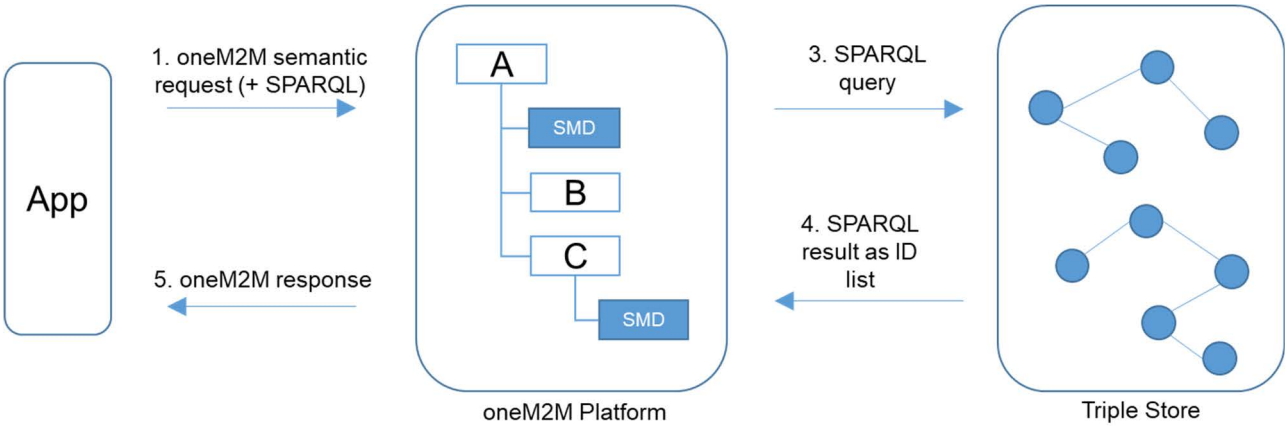


Figure 5.5.3.2-1: oneM2M semantic discovery procedure

5.5.4 Cypher, Gremlin, AQL, GraphQL

5.5.4.1 Introduction

The discovery process and query in a graph database can be mostly dependent on its underlying graph model as well as the query features supported by the considered query language. In this regard following clauses describe the features for each query language.

5.5.4.2 Conventional Graph Models

Most graph databases follow conventional graph modeling concepts. The model comprises of a node or a vertex, and edge or relationship and property or attributes. The node or a vertex describes a data entity, which can potentially represent a class or concept in the considered scenario. In graph models, they are described with nouns. The edge or a relationship is used to represent a link of specific type, connecting two nodes or vertices. They are often described as verbs in the graph models. Properties or attributes, represent the data value specifications or constraints for their respective Nodes, Vertices, edges or relationships. Same data modeling convention is considered and followed in the query languages considered.

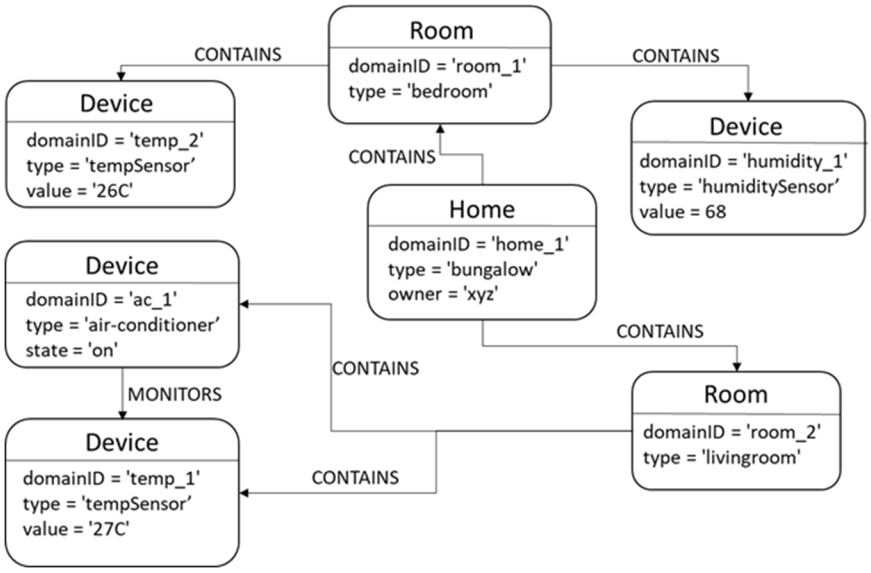


Figure 5.5.4.2-1: Example of graph data visualization

Figure 5.5.4.2-1 describes an example of a graph data visualization of a considered smart home environment. The rectangular box represents a vertex, where its upper part contains its label, the lower part contains the property description. The arrow represents the directed edge between two vertices. The vertex, labelled as 'Home', has two outbound edges to the vertices labelled as 'Room', representing a real-world relationship that the home contains two rooms. Similarly, each room contains two devices, represented by vertices with label 'Device'. The edge labelled as 'MONITORS', connecting one device (vertex with property type = 'air-conditioner'), with the other (vertex with property 'tempSensor'), represents that the Air-conditioner monitors the temperature sensor. In this case, all the labels and the properties are considered as user defined. Whereas in some graph databases, there are default properties such as id, type, etc. which may also be declared and initialized by graph database management system. For simplicity, edge properties have not been considered. These concept and relationship representations can be hypothetical and are usually based on considered scopes and use cases.

Considering graph data from Figure 5.5.4.2-1, there can be many different ways to discover a resource based on the given information as well the features provided by the query language. Among those, three possible ways are described in the present clause which are as follows:

1) device information, with given 'domainID': Requesting the device information of given the 'domainID' (known by the client);

2) list all the devices in each room of the given home: Requesting the 'deviceID', along with their respective room's 'domainID' in which they are contained;

3) list all the humidity sensors, having value greater than 60.

5.5.4.3 Cypher

The query modeling in Cypher is based on the Neo4j property graph model, where a single data entity is represented by a Node, having properties defining the related data attributes. Thus, different Nodes are interconnect via relationships, forming a graph of the given data.

Cypher provides different functionalities for graph data such as CRUD operations, pattern searching based on filter and sub-queries. It also provides other support functions such as aggregation and ordering functions, handing data duplications as well as user defined functions. Cypher queries can also be embedded in Java code which is an additional language support. Following some examples of search in Cypher are given:

EXAMPLE 1: Requests device info, with the given 'domainID':

Table 5.5.4.3-1 shows the Cypher query, and its respective response. The first line in the query contains the '**MATCH**' clause to identify a node of label '**Device**', having a '**domainID**' as '**ac_1**'. This Node has to be stored in a query variable named '**device**', and will be returned as specified in the next line. The response contains the required node information in JSON format, which represents the device of type '**air-conditioner**' as described in the data model.

Table 5.5.4.3-1: Cypher query and result: device info by 'domainID'

Cypher Query	Result

`MATCH (device:Device {domainID:'ac_1'})`
`RETURN device`
	`[{`
	`"device": {`
	`"identity": 7,`
	`"labels": [ "Device" ],`
	`"properties": {`
	`"state": "on",`
	`"type": "air-conditioner",`
	`"domainID": "ac_1"`
	`}`
	`}`
	`}]`

EXAMPLE 2: Lists all the devices in each room of the given home:

In this case, the only the **'domainID'** of house is known, and the list of all devices are requested along with their respective room information. Table 5.5.4.3-2 shows the query and results for this case. The first line of the query selects the home of given type and id. The next line searches for all the edges with label 'CONTAINS' having other connected node of type 'Room'. In order to filter out all the other rooms without any devices, the next line specifies that the Room node is requested to have edge 'CONTAINS' connecting to a node of type 'Device'. Finally, the traversed room node, edge and the device node are stored in variables: rooms, c, and devices respectively and their respective node id and edge type are returned. The results show the list of variables' values in a JSON format.

Table 5.5.4.3-2: Cypher query and result: List of devices in each room of the home

Cypher Query	Result
MATCH (:Home {domainID:'home_1'}) -[:CONTAINS*]-> (rooms:Room) -[c:CONTAINS]->(devices:Device) RETURN rooms.domainID, type(c), devices.domainID	[{ "rooms.domainID": "room_2", "type(c)": "CONTAINS", "devices.domainID": "ac_1" }, { "rooms.domainID": "room_2", "type(c)": "CONTAINS", "devices.domainID": "temp_1" }, { "rooms.domainID": "room_1", "type(c)": "CONTAINS", "devices.domainID": "humidity_1" }, { "rooms.domainID": "room_1", "type(c)": "CONTAINS", "devices.domainID": "temp_2" }]

EXAMPLE 3: Lists all the humidity sensors, whose value is greater than 60:

Here two parameters are considered given. One is device type, which in this case is 'humiditySensor' and other is the value threshold, which is 60. The respective query and results can be seen in Table 5.5.4.3-3 where the first line in the query searches for the devices of type 'humiditySensor'. The next line filters the devices, whose value is greater than 60. The filtered devices are stored in variable 'device' and its 'domainID' and 'value' are returned. The results include the node information of domainID 'humidity_1' which is the only matched device in the data.

Table 5.5.4.3-3: Cypher query and result: Device search based on threshold value

Cypher Query	Result

MATCH (device:Device {type:'humiditySensor'})
WHERE device.value > 60
RETURN device.domainID, device.value
	[{
	"keys": [
	"device.domainID",
	"device.value"
	],
	"length": 2,
	"_fields": [
	"humidity_1",
	{ "low": 68,
	"high": 0 }
	],
	"_fieldLookup": {
	"device.domainID": 0,
	"device.value": 1
	}}]

5.5.4.4 Gremlin

The Gremlin query language provides many different functionalities for managing and traversing graph data. These functionalities include centrality measures, patterns searching, tree traversal and analysis, etc. Gremlin is stated to be Turing complete, based on its feature that through query execution, the states of the graph data flow can be manipulated. Gremlin provides support for embedding in different other languages, which can allow function composition and nesting, such as Java, JavaScript, Groovy, Python and Scala.

The terminology to represent graph data in gremlin has small variation. The graph node is referred to as vertex and the relationship between vertices is represented as an edge. The CRUD operations such as Insert Delete and Update are referred to as add, drop and set respectively.

EXAMPLE 1: Requests device info, with the given 'domainID':

Table 5.5.4.4-1 shows the Gremlin query, and its respective (console) output. The query in gremlin contains consecutive functions indicating the respective task to perform. Here the first line of the query requests the vertex having label as '**Device**' and the property '**domainID**' = '**ac_1**'. The next line, which is continuation of the query, requests all properties of the vertex along with its label and id, in the form of Java HashMap, i.e. key-value pairs. Notice that in the first line of the result, returned id is **22,** which is different from the '**domainID**' property. This is the default id, defined and initialized by Gremlin database. Each vertex, edge and even property is assigned with such unique id. If known before hand, graph data can also be queried using this id. Otherwise, separate property can be explicitly defined in the data model, which can be used in the query process.

Table 5.5.4.4-1: Gremlin query and result: Device info by 'domainID'

Gremlin Query (Groovy)	Result (Groovy Console)
gremlin> g.V().has('Device' , 'domainID' , 'ac_1'). gremlin> valueMap()with(WithOptions.tokens).unfold()	=>id=22 =>label=Device =>state=[on] =>type=[air-conditioner] =>domainID=[ac_1]

EXAMPLE 2: Lists all the devices in each room of the given home:

Table 5.5.4.4-2 shows the query and results for this case. This first line in the query follows the same step to get home vertex. Second line traverses the outbound edges in a loop, with the stopping condition that the traversed vertex has the label '**Device**'. Here the loop has been used considering the possible case where the path from home vertex to the device vertex may have varying number of hops for each device. In addition **as('room')** function stores the traversed vertices in temporary '**room**' variable. The following line traverses the next outbound edge and its connected vertex and stores it in variable '**has**' and '**device**' respectively. Finally, the last line selects those variables sequentially, by specifying the '**domainID**' of the room and device to be included as the variables' values in the result.

In the result, each line displays three selected variables (comma-separated), where each includes the variable name, followed by colon, followed by variable's value. First is the '**room**' variable followed by domainID. Second is the '**has**' variable, followed by the edge definition. The edge definition contains the edge id, followed by source vertex id, edge label and the destination vertex id. Finally, '**device**' variable is defined with '**domainID**' as its value.

Table 5.5.4.4-2: Gremlin query and result: List of devices in each room of the house

Gremlin Query (Groovy)	Result (Groovy Console)
gremlin> g.V().has('Home','domainID','home_1'). gremlin> repeat(out().as('room')).until(hasNot('Device')). gremlin> outE().as('has').inV().as('device'). gremlin> select('room','has','device').by('domainID').by()	=>[room:room_1, has:e[28][4-CONTAINS->14],device:temp_2] =>[room:room_1, has:e[29][4-CONTAINS- >18],device:humidity_1] =>[room:room_2, has:e[30][7-CONTAINS->10],device:temp_1] =>[room:room_2, has:e[31][7-CONTAINS->22],device:ac_1]

EXAMPLE 3: Lists all the humidity sensors, whose value is greater than 60:

Table 5.5.4.4-3 shows the respective query and result. In the query, the first line requests the devices of type 'humiditySensor'. Second line filters those devices, having value greater than 60. The third line requests the two properties, 'domainID' and 'value', arranged as key-value pairs.

Table 5.5.4.4-3: Gremlin query and result: Device search based on threshold value

Gremlin Query (Groovy)	Result (Groovy Console)
gremlin> g.V().has('type','humiditySensor'). *gremlin> has('value',gte(60)).* *gremlin> valueMap('domainID','value').unfold()*	*=>value=[68]* *=>domainID=[humidity_1]*

5.5.4.5 Arango Query Language (AQL)

Data in ArangoDB is modelled using two main concepts, which are collections and documents. With respect to modelling in relational databases, collections can correspond to tables and documents as records or rows. However, their schema is not predetermined. Hence, the graph data is also organized and stored using same model. Unlike other graph data queries, AQL does not support creating and dropping data entities. It supports different traversal functionalities such as searching path using inbound and outbound relationships, result filtrations, shortest path traversals, etc.

In ArangoDB, the stored graph data, based on Figure 5.5.4.2-1, has some specific considerations. The nodes are arranged into collections based on node types. In case of edges, all are stored in single edge collection. This storage consideration can be varied depending on different use cases models and their respective graph complexities. For example, in case of an environment having large number of devices, can be stored in collections representing different device type, such as sensors, actuators, etc.

EXAMPLE 1: Requests device info, with the given 'domainID':

This task can be formulated in following query, shown in Table 5.5.4.5-1. In the query, the device is searched in '**Device**' collection, having '**domainID**' as '**ac_1**'. The selected device node data is stored in the '**device**' variable and is returned as the query response. The result shows the returned device node data, which is an air-conditioner. The field '_rev' is a unique key assigned to all the nodes and edges in ArangoDB.

Table 5.5.4.5-1: Arango query and result: Device info by 'domainID'

Arango Query	Result

`FOR device IN Device`
`FILTER device.domainID == 'ac_1'`
`RETURN device`
	`[{`
	`"_key": "ac_1",`
	`"_id": "Device/ac_1",`
	`"_rev": "_axd_SL6---",`
	`"domainID": "ac_1",`
	`"type": "air-conditioner",`
	`"value": "on"`
	`}]`

EXAMPLE 2: Lists all the devices in each room of the given home:

In order to search rooms containing devices, 'Edge' collection needs to be traversed, where the starting point of edge is a node of 'Collection' (type) 'Room' and the ending point is of 'Collection' (type) 'Device'. This can be shown in Table 5.5.4.5-2, where 'FILTER LIKE' defines these constrains. The resultant edges are stored in 'edge' variable and the starting node, ending node and the edge type are returned with labels 'room', 'device' and 'type' respectively. The result clause shows the list of edges, where each includes fields based on above mentioned labels.

Table 5.5.4.5-2: Arango Query and result: List of devices in each room of the house

Arango Query	Result
FOR edge IN Edge *FILTER LIKE(edge._from, '%Room%')* *FILTER LIKE(edge._to, '%Device%')* *RETURN {room: edge._from, device: edge._to,* *type: edge.type}*	*[{ "room": "Room/room_2",* *"device": "Device/ac_1",* *"type": "CONTAINS" },* *{ "room": "Room/room_2",* *"device": "Device/temp_1",* *"type": "CONTAINS" },* *{ "room": "Room/room_1",* *"device": "Device/temp_2",* *"type": "CONTAINS" },* *{ "room": "Room/room_1",* *"device": "Device/humidity_1",* *"type": "CONTAINS" }]*

EXAMPLE 3: Lists all the humidity sensors, whose value is greater than 60:

The query in Table 5.5.4.5-3 searches devices of type 'humiditySensor' and value greater than 60. The returned device information , shown in the table, involves only one device, having the '**domainID**' as '**humidity_1',** as described in Figure 5.5.4.2-1.

Table 5.5.4.5-3: Arango query and result: Device search based on threshold value

Arango Query	Result
FOR device IN Device FILTER device.type == 'humiditySensor' && device.value > 60 RETURN device	[{ "_key": "humidity_1", "_id": "Device/humidity_1", "_rev": "_axd-7gy---", "domainID": "humidity_1", "type": "humiditySensor", "value": 68 }]

5.5.4.6 Graph Query Language (GraphQL)

GraphQL queries are formulated based on its schema definition, which includes the object and field definitions as well as definitions for queries and mutations (for updating the data). Based on the use case defined in Figure 5.5.4.2-1, following schema definition can be considered. Here the first object of type **"Query"**, defines the possible queries that can be requested from the server, along with their respective input parameters. The next three objects define the corresponding nodes in the use case, as well as the fields contained, represent their respective properties. The relationships are also defined as fields inside the objects, having the list of target objects as value.

Schema:

```
type Query {
 getHome(domainID: String): Home
 getRoom(domainID: String): Room
 getDevice(domainID: String): Device
 getDeviceWithHighValue(type: String, value: String): Device
}
type Home {
 domainID: String!
 type: String!
 owner: String
 CONTAINS: [Room]
}
type Room {
 domainID: String!
 type: String!
 CONTAINS: [Device]
}
type Device {
 domainID: String!
 type: String!
 value: String!
 MONITORS: [Device]
}
```

1) Request device info, with the given 'domainID':

Table 5.5.4.6-1 shows the GraphQL query and result. Here, the device is requested with input parameter as **"$domainID"**, having value of **"ac_1"**, as query variable's value. It also includes the specific fields of the particular object type. The requested fields does not include the relationship **"CONTAINS"**, therefore it is omitted by the server in the response. This highlights the feature of this language that the system returns exactly what is queried.

Table 5.5.4.6-1: GraphQL query and result: Device info by 'domainID'

GraphQL Query	Result

query getDevice($domainID: String!) {
 getDevice(domainID: $domainID) {
  domainID
  type
  value}}
	{
	 "data": {
	  "getDevice": {
	  "domainID": "ac_1",
	  "type": "air-conditioner",
	  "value": "on"
	  }}}

Query Variables	
{"domainID": "ac_1"}	

2) List all the devices in each room of the given home:

Table 5.5.4.6-2 shows that, traversal can be performed by explicitly mentioning the underlying fields and relationships of the queried object in an appropriate manner as described in schema. The input parameter is **"domainID"** of the home, and the requested fields contains the **"domainID"**, **"type"** and **"CONTAINS"** as relationship. So, the first clause with CONTAINS relationship will return the Room objects, and the second one returns the Device objects in these rooms. It can be observed that there is no anonymity of the types of objects to be traversed in the path. This means that the intermediate objects (nodes) have to be explicitly defined in the query, in order to find the target object, which is one limitation, as compared to the existing graph queries explained above. The result contains the objects and fields in the same structure as described in the query, as well as complied with the schema.

Table 5.5.4.6-2: GraphQL Query and Result: List of devices in each room of the house

GraphQL Query	Result

query getHome($domainID: String!){
 getHome(domainID: $domainID){
  domainID
  CONTAINS{
  domainID
  type
  CONTAINS{
  domainID
  type
  }
  }
 }
}
	{"data": {
	  "getHome": {
	  "domainID": "home_1",
	  "CONTAINS": [
	  {
	  "domainID": "room_1",
	  "type": "bedroom",
	  "CONTAINS": [
	  {
	  "domainID": "temp_2",
	  "type": "tempSensor"
	  },
	  {
	  "domainID": "humidity_1",
	  "type": "humiditySensor"
	  }
	  ]
	  },
	  {
	  "domainID": "room_2",
	  "type": "livingroom",
	  "CONTAINS": [
	  {
	  "domainID": "temp_1",
	  "type": "tempSensor"
	  },
	  {
	  "domainID": "ac_1",
	  "type": "air-conditioner"
	  }
	  ]
	  }
	  ]
	}}}

Query Variables	

{
 "domainID": "home_1"
}	

3) List all the humidity sensors, whose value is greater than 60.

The query in Table 5.5.4.6-3 has been provided in the schema, defined by the server, which gets the Device object having value higher than the threshold value as query variable. The other query variable is the type field, which here is specified as "humiditySensor". The result contains the respective Device object with its fields. The underlying execution of procedure and instructions is dependent on the server and the supporting database. Hence its performance and sequence of execution varies based on its backend implementation and functionalities.

Table 5.5.4.6-3: GraphQL query and result: Device search based on threshold value

<table>
<tr><th>GraphQL Query</th><th>Result</th></tr>
<tr><td>

```
query getDeviceWithHighValue($type: String, $value:
String){
 getDeviceWithHighValue(type: $type, value: $value){
  domainID
  type
  value
 }
}
```

</td><td rowspan="2">

```
{
 "data": {
  "getDeviceWithHighValue": {
   "domainID": "humidity_1",
   "type": "humiditySensor",
   "value": "68"
  }
 }
}
```

</td></tr>
<tr><th>Query Variables</th></tr>
<tr><td>

```
{
 "type": "humiditySensor",
 "value": "60"
}
```

</td></tr>
</table>

5.5.5 SPARQL1.1 protocol

The W3C has promoted the standard SPARQL 1.1 protocol that describes how to query data expressed in RDF that is modelled by means of a given ontology [i.30]. Specifically, this standard specifies how SPARQL processors should receive and process SPARQL queries expressed with the SPARQL 1.1 Query Language [i.25] (operational query), or the SPARQL 1.1. Update Language for RDF [i.39] (update query).

The SPARQL protocol is built on top of the HTTP protocol, specifying a set of conveyed operations and parameters that can be sent along the queries. The SPARQL protocol distinguishes different operational frameworks depending on the type of query. It is required, that Clients issuing an operational query use HTTP GET or HTTP POST, additionally, that the request includes only one SPARQL query (using the parameter query) and optionally zero or more default graph URIs (using the parameter default-graph-uri). In addition, requests may include the URIs of named graphs (relying on the parameter named-graph-uri). Finally, depending on the query form (SELECT, CONSTRUCT, DESCRIBE or ASK) and the content negotiation, the response of the query can be expressed in either SPARQL XML Results Format [i.28], SPARQL JSON Results Format [i.26], SPARQL CSV/TSV Results Format [i.27], or an RDF serialization [i.21]. Table 5.5.5-1 extracted from the SPARQL protocol documentation summarizes the specification for the operational queries.

Table 5.5.5-1: Operational queries in SPARQL

HTTP Method	URL Parameters	Request Content Type	Request Message Body
GET	query (exactly 1) default-graph-uri (0 or more) named-graph-uri (0 or more)	None	None
POST	None	application/x-www-form-urlencoded	URL-encoded, ampersand-separated query parameters. query (exactly 1) default-graph-uri (0 or more) named-graph-uri (0 or more)
POST	default-graph-uri (0 or more) named-graph-uri (0 or more)	application/sparql-query	Unencoded SPARQL query string

The SPARQL queries are endowed to be executed against an RDF dataset [i.27], such dataset may be specified relying on the parameters default-graph-uri or named-graph-uri via the SPARQL protocol or using the FROM and FROM NAMED keywords in the SPARQL query itself; in the case that no RDF dataset is specified the implementation may execute the query over a defined default RDF dataset.

Clients issuing an update query are requested to use an HTTP POST request that include only one SPARQL query (using the parameter update) and may include zero or more default graph URIs (using the parameter using-graph-uri). Additionally, the request can specify a named graph URIs (using parameter using-named-graph-uri). The response to an update query is indicated by means of an HTTP response status code. Table 5.5.5-2 extracted from the SPARQL protocol documentation summarizes the specification for the update queries.

Table 5.5.5-2: Update queries in SPARQL

HTTP Method	URL Parameters	Request Content Type	Request Message Body
POST	None	application/x-www-form-urlencoded	URL-encoded, ampersand-separated query parameters. query (exactly 1) default-graph-uri (0 or more) named-graph-uri (0 or more)
POST	default-graph-uri (0 or more) named-graph-uri (0 or more)	application/sparql-update	Unencoded SPARQL update request string

Assuming a SPARQL endpoint at http://example-sparql.com, that implements the SPARQL 1.1 protocol and the following SPARQL query.

EXAMPLE 1:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
PREFIX saref: < https://w3id.org/saref#>.
PREFIX geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
PREFIX om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
SELECT DISTINCT ?uri {
     ?uri saref:measuresProperty ?property.
     ?property rdf:type saref:Temperature.
     ?property saref:hasMeasurement ?measurement.
     ?measurement saref:isMeasuredIn om:degree_Celsius.
     ?uri saref:hasState ?sate.
     ?state rdf:type saref:OnState.
     ?uri saref:locatedIn ?building.
     ?building rdf:type saref:BuildingSpace.
     ?building saref:contains ?space.
     ?space rdf:type saref:Window.
}
```

A client could issue the previous query sending the following request:

EXAMPLE 2:

```
GET /sparql/?query= PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-
     ns%23%3E%20.%0APREFIX%20saref%3A%20%3C%20https%3A%2F%2Fw3id.org%2Fsaref%23%3E.%0APREFIX%20geo
     %3A%20%3C%20http%3A%2F%2Fwww.w3.org%2F2003%2F01%2Fgeo%2Fwgs84_pos%23%3E%20.%0APREFIX%20om%3A%
     20%3Chttp%3A%2F%2Fwww.wurvoc.org%2Fvocabularies%2Fom-
     1.6%2F%3E.%0ASELECT%20DISTINCT%20%3Furi%20%7B%0A%09%3Furi%20saref%3AmeasuresProperty%20%3Fpro
     perty%20.%0A%09%3Fproperty%20rdf%3Atype%20saref%3ATemperature%20.%0A%09%3Fproperty%20saref%3A
     hasMeasurement%20%3Fmeasurement%20.%0A%09%3Fmeasurement%20saref%3AisMeasuredIn%20om%3Adegree_
     Celsius%20.%0A%09%3Furi%20saref%3AhasState%20%3Fsate%20.%0A%09%3Fstate%20rdf%3Atype%20saref%3
     AOnState%20.%0A%09%3Furi%20saref%3AlocatedIn%20%3Fbuilding%20.%0A%09%3Fbuilding%20rdf%3Atype%
     20saref%3ABuildingSpace%20.%0A%09%3Fbuilding%20saref%3Acontains%20%3Fspace%20.%0A%09%3Fspace%
     20rdf%3Atype%20saref%3AWindow%20.%0A%7D%0A HTTP/1.1

Host: http://g-sparql.com

User-agent: sparql-client/1.0

Accept: application/json
```

The request counts with the ?query parameter and the SPARQL query encoded as an URL. Additionally, for receiving the answer of the query in JSON format the request counts with the header Accept: application/json. Finally, the request does not specify any RDF dataset and, therefore, depends on the implementation of the SPARQL endpoint to solve the query sent over a set of RDF documents. Assuming that the default RDF dataset is the following:

EXAMPLE 3:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix saref: < https://w3id.org/saref#>.
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
@prefix geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
 <http://snsrs.org/2> rdf:type saref:TemperatureSensor;
 rdfs:label "Temperature sensor 2"^^xsd:string;
 saref:measuresProperty [
 rdf:type saref:Temperature;
 saref:isMeasuredByDevice <http://snsrs.org/1>, <http://snsrs.org/2>;
 saref:hasMeasurement [
 saref:hasValue "32.4"^^xsd:float;
 saref:isMeasuredIn om:degree_Celsius;
 ];
 ];
 saref:hasState [
 rdf:type saref:OnState;
 ];
 saref:isLocatedIn <http://snsrs.org/building>.

<http://snsrs.org/building> rdf:type saref:BuildingSpace;
 saref:hasSpaceType "Laboratory"^^xsd:string;
 geo:lat "52.0705"^^xsd:string;
 geo:long "4.30070"^^xsd:string;
 saref:contains [
 rdf:type saref:Window;
 rdfs:label "Left window"^^xsd:string;
 ].
```

The query issued would have the following answer in JSON, due to the content negotiation.

EXAMPLE 4:

```
 {
"head": { "vars": [ "uri" ] },
 "results": {
 "bindings": [
 { "uri": { "type": "uri", "value": "http://snsrs.org/2" } }
 ]
 }
}
```

The resource http://snsrs.org/2 is the only one that fulfils the restrictions of the query, i.e. "Find the URI of a resource which type is saref:TemperatureSensor and that has a state on (saref:OnSate), and that measures temperature in Celsius (saref:isMeasuredIn om:degree_Celsius, saref:Temperature), and that is located in a windown (saref:Window)".

Finally, the SPARQL protocol has not been designed specifically for discovery of resources on the Web but to provide a mechanism to solve queries over an RDF dataset (discovery and access) that could consist of a set of distributed RDF documents. As a result, the SPARQL protocol is suitable for discovering resources within a given set of RDF documents, although these resources could potentially lead to other RDF documents available online. Nevertheless, including them in the query would require additional effort. In the following clause, it is explained how the whole SPARQL protocol is usually used for discovery, and to deal with heterogeneous data sources.

5.5.6 SPARQL-based architectures for discovery

In the last decade, SPARQL-based architectures have been proposed to enable the discovery, and access, of a set of data sources [i.40]. These data sources are not always compliant with the W3C Linked Data principles, or are not even RDF-based. These architectures enable a SPARQL endpoint that is able to answer queries considering a set of RDF meta-data that profiles the data sources and, depending on the architecture, also considering the data being published by those data sources [i.36]. It is important to remark that a query answering mechanism has two steps: discovery and query solving (that may entail data access).

The most extended SPARQL-based architecture is the centralized one that consists of a repository storing an RDF dataset, such dataset accommodates a set of RDF documents commonly known as Descriptions that contain meta-data of the data sources. When a SPARQL query is issued, it is solved over the RDF dataset and if the data sources follow the Linked Data principles their data values can be included in the query as a discovery criteria. Alternatively, if data sources rely on heterogeneous data (non RDF-based) they may asynchronously inject data in the repository by previous translation to RDF. Therefore, their data values can still be included as a query discovery criterion. Although if the values of the data sources could not be considered as discovery criteria, solving a SPARQL over their Descriptions could discover their endpoints.

The advantages of this architecture are that query solving time is low since all the data that should be considered by a discovery criterion is already present in the repository. The disadvantages are that considering data values of endpoints requires translators of heterogeneous data to RDF and that data sources have to continuously send their data to the repository in order to provide the latest values. If discovery does not need data values, and those are not injected, this approach is very suitable for discovery. Well-known discovery mechanisms that are based on this architecture are the ones of oneM2M, and WoT.

The Distributed architecture is an alternative to the Centralized one. This architecture relies also on a repository that stores an RDF dataset containing the Descriptions of the data sources. However, in this architecture the Descriptions contain meta-data of the data sources and also translation rules, i.e. mappings, which allow accessing and translating on the fly the data sources data into RDF expressed with a specific ontology. Therefore, when a SPARQL query is received this architecture first analyses the query to decide if it requires accessing the data sources and translating their data. If the query can be solved only with the meta-data, an answer is computed; otherwise, relying on fragments of the query a set of suitable Descriptions is chosen. Then, their related data sources are accessed and their data translated thanks to the mappings. Finally, the query is solved using the Descriptions and the translated data.

The advantage of this architecture is that data sources do not need to enable mechanisms to translate their data, instead, they only need to provide Descriptions containing mappings if they wish their data values to be considered as a discovery criterion. Additionally, another advantage is that not all the data sources are accessed, only the relevant ones for answering the query, avoiding an overhead of the network. The disadvantage of this architecture is that the query answering time is slightly higher than in the Centralized architecture due to the distributed access.

Another widely adopted architecture is the Federated one [i.29], which consists of an SPARQL endpoint that when receiving a query, it broadcasts it to a set of known SPARQL endpoints that enclose different RDF datasets. In this architecture the final query answer is computed gathering the answers of the other SPARQL endpoints and providing a unified query answer.

The main advantage of this approach is that query federation is a mechanism implemented in the SPARQL protocol 1.1 [i.30] and, thus, is reliable. Additionally, it does not require Descriptions to be registered; instead, it only needs the SPARQL endpoint URI to be registered. Nevertheless, the main disadvantages are that data sources have to implement a SPARQL endpoint, which hinders integrating third-party data sources that are already developed based on different standards or technologies. In addition, the federation broadcasts the query to all the known endpoints flooding the network with queries.

5.6 Discovery routing mechanisms

5.6.1 Introduction

To enable M2M entities (e.g. CSE, AE) within an M2M domain or in different M2M Service Provider (SP) domains to communicate, configuration within the M2M domain determines if such communications are allowed. Two kinds of communications namely intra-domain or inter-SP domains use routing mechanisms. If inter-SP domain communications are allowed, the M2M System supports routing of the traffic across the originating M2M SP domain and within the target M2M SP domain. Inside a given SP domain, a CSE routes M2M requests targeting another CSE by forwarding the request to the next hop towards the target CSE by first checking each of its <remoteCSE> resources to determine whether the CSE-ID specified in the **To** parameter of the request matches either the CSE-ID or descendantCSEs attributes of a <remoteCSE> resource.

Out of the potential requests that can be routed, the discovery request intensively used routing mechanisms.

5.6.2 Inter M2M Service Provider Communication in oneM2M

5.6.2.1 Overview

To enable M2M entities (e.g. CSE, AE) in different M2M Service Provider (SP) domains to communicate, configuration within the M2M domain determines if such a communication is allowed. If allowed, the M2M System supports routing of the traffic across the originating M2M SP domain and within the target M2M SP domain.

Communication between different M2M SPs which occurs over the reference point Mcc', is subject to business agreements. The offered functionality is typically a subset of the functionality offered over the Mcc reference point.

Any interM2M SP communication in support of a request originating from one M2M SP domain is to be processed and forwarded through the Infrastructure Node of the originating M2M domain towards the Infrastructure Node of the target M2M SP domain and finally forwarded to its target CSE, if different from the Infrastructure Node. Hence the Infrastructure Node in both M2M domains are the exit and entry points, respectively, for all inter M2M SP communication traffic.

In this configuration approach, public DNS [i.57] is used to support traffic routing for inter M2M SP communication in accordance with [i.61]. This relies on public domain names being allocated to communicating CSE entities within the oneM2M architecture, and to whom access across domains is permitted through policies. To that effect, an M2M SP supporting inter- M2M SP communication ensures that the public domain names for the CSEs whose functionality is available across domains are held in its public DNS and always points to the IP address associated with the Infrastructure Node for the domain (being the entry point) for accessibility purposes.

The M2M SP could optionally also have additional policies (e.g. black list or white list) that governs accessibility from other domains to CSE functionality located within its own domain. These policies are however out of scope of the present document.

The public domain names of CSEs to whom access from other domains is allowed by policies, is created in the DNS of the M2M SP by the Infrastructure Node at registration time of these CSEs, and is removed at de-registration. DNS entries for CSEs can also be created/removed for registered CSEs at any time by the M2M SP through administrative means to handle dynamic policies.

5.6.2.2 Public Domain Names of SP and CSEs

To enable the usage of public DNSs as described above, there is a need for a naming convention for public names for CSEs. This naming convention facilitates the creation of the necessary entries of the public domain names of CSEs in the DNS by the infrastructure node.

CSEs public domain names is a sub-domain of the Infrastructure Node's public domain name. This naming convention allows the Infrastructure Node to include the needed DNS entry corresponding to the CSE to whom access from other domains is allowed. This would typically occur when the CSE registers with the Infrastructure Node, subject to policies, or administratively.

Accordingly, the structure of the public domain of the CSEs in IN/MN/ASN follows the following naming convention, which relies on the CSE identifier (CSE-ID) as part of the naming convention to facilitate the DNS entry creation:

- Infrastructure Node CSE public domain name: <Infrastructure Node CSE Identifier>.<M2M Service Provider domain name>.
- Middle Node CSE public domain name: <Middle Node CSE Identifier>.<Infrastructure Node public domain name>.
- Application Service Node CSE public domain name: <Application Service Node CSE Identifier>.<Infrastructure Node public domain name>.

Both the MN-CSE and the ASN-CSE public domain names are sub-domains of the Infrastructure Node public domain name.

The A/AAAA records in the DNS, as per [i.55], [i.56] and [i.58] consist of the public domain name of the CSE and the IP address of the M2M Infrastructure Node, since the M2M Infrastructure Node is the entry point of the M2M Service Provider domain name where it belongs to.

Entries in the public domain names of the three nodes depicted above do not imply that the actual CSE-Identifier allocated for that node has to be used in the DNS entry. Rather any name, including indeed the CSE Identifier for the node, can be used there as long as the entry resolves to the intended Node.

EXAMPLE: These 3 host entries are valid entries in the DNS:

MN-CSEID.IN-CSEID.m2m.myoperator.org

node1.node2.m2m.myoperator.org

MN-CSEID.node22.m2m.myoperator.org

5.6.3 Routing mechanisms in oneM2M

5.6.3.1 Intra-domain Routing Policies

The routing policies in oneM2M are described in [i.5]. A CSE routes M2M requests targeting another CSE in the same SP domain by forwarding the request to the next hop towards the target CSE by first checking each of its <remoteCSE> resources to determine whether the CSE-ID specified in the **To** parameter of the request matches either the CSE-ID or descendantCSEs attributes of a <remoteCSE> resource.

If a match is found, the CSE retargets the request to the pointOfAccess of the matching <remoteCSE> resource.

If a match is not found, and the CSE received the request from an AE or a descendant CSE, and the CSE is not the IN-CSE, then it retargets the request to its Registrar CSE. If a match is not found and the CSE is the IN-CSE, then the CSE does not forward the request and it responds with an error. If a match is not found and the CSE is not the IN-CSE and the CSE receives the request from its registrar CSE, then the CSE does not forward the request and it responds with an error. Anytime a CSE re-targets a request to another CSE, it keeps track of the requestID and the corresponding Originator's ID. This information is used to route re-targeted responses back to the Originator.

5.6.3.2 Inter-Service Providers Domain Routing Policies

If a CSE in the originating SP domain is not the IN-CSE and it receives a request targeting another CSE in a different SP domain, it retargets the request to its Registrar CSE. This is done by retargeting the requests to the pointOfAccess of the <remoteCSE> of its Registrar CSE. If a CSE in the originating SP domain is the IN-CSE and it receives a request targeting another CSE in a different SP domain, the IN-CSE routes the request to the IN-CSE in the targeted SP domain using either the DNS-based procedures or the inter-M2M SP registration procedures defined in clause 5.6.2.

For the inter-M2M SP registration based procedure, the IN-CSE in the originating SP domain forwards the request to the IN-CSE in the target SP domain by checking each of its <remoteCSE> resources to determine whether the SP-ID specified in the **To** parameter of the request matches the SP-ID specified in the CSE-ID attribute containing a SP-relative CSE-ID. If the IN-CSE finds a match, it retargets the request to the pointOfAccess of the matching <remoteCSE> resource. If the IN-CSE does not find a match, then it does not forward the request and it responds with an error. An IN-CSE receiving a request from an IN-CSE in another SP domain, routes the request to the targeted CSE residing in its own domain using the Intra SP Domain routing as described in clause 8.2.1.1 of [i.5] to route the request to the CSE in the targeted SP domain. Anytime a CSE re-targets a request to another CSE in its own SP domain or another SP domain, it keeps track of the requestID and the corresponding Originator's ID. This information is used to route re-targeted responses back to the Originator.

5.6.3.3 Announcement mechanisms

A resource can be announced to one or more remote CSEs to inform the remote CSEs of the existence of the original resource. An announced resource can have a limited set of attributes and a limited set of child resources from the original resource. The announced resource includes a link to the original resource hosted by the original resource-Hosting CSE.

In case that the original resource is deleted, all announced resources for the original resource is deleted, except for <AEAnnc> resources that were created during the registration of an AE with AE-ID-Stem starting with "S", which is not deleted. If the announced resource is not deleted promptly (e.g. the announced resource is not reachable), the announced resource can be deleted later either by the original resource Hosting CSE or by the expiration of the announced resource itself. The original resource stores the list of links for the announced resources for those purposes. Synchronization between the attributes announced by the original resource and the announced resource is the responsibility of the original resource Hosting CSE.

5.6.4 Routing recommendation system

The objective of a Routing Recommendation System (RS) in our advanced discovery is to:

i) analyse available data, essentially the query;

ii) analyse the routing table information of the CSE receiving an ASD routing packet and the CSE neighbours (i.e. those directly connected with the current CSE) and make suggestions for routing the query to the next CSE, according to some criteria that are not fixed a priori but that can reasonably can be the one of the following:

   1) number of remaining hops;

   2) successful response rate of the neighbours CSE;

   3) query popularity;

   4) past routing history, etc.

The mathematical tools, often employed in designing RS, are Statistic and Machine Learning. Caching Techniques are often good Computer Science tools to effectively implement RS. RS are widely used in a variety of areas, mostly commercial applications: notable examples are audio/video playlist generators, content recommenders for social media platforms, product recommenders, online dating, real-time routes, banking, financial and insurance services, tourism, etc.

Often RC are subject to patents, and, because of this, literature on the topic is not "exterminate".

Applying SR applied to resource discovery routing protocols seems to the little author knowledge, a "new application field", partially used in some peer-to-peer overlay networks, like, e.g. Kademlia [i.50] that could be fruitfully be employed in the domain of IoT semantic resource discovery, i.e. a domain characterized by the following elements:

i) heterogeneity that captures the diversity of objects;

ii) churn that characterizes the high intermittence of the object lifespan; and

iii) small-is-beautiful that characterizes the fact that the routing actors have tiny memories (read routing table) and they never could memorize all the previous successful routing experiences.

5.6.5 Routing table specification

5.6.5.1 RDF Summarization

In the last decade, the number of data expressed in RDF using ontologies has become pervasive on the Web. Due to the large amount of data, and the necessity of consuming such data in and for different tasks, researchers have focused during the past years on developing algorithms to produce summaries of large RDF documents. RDF summarization aims at extracting concise and meaningful RDF documents that are smaller than the original ones and provide an overview of the original RDF providing a close view of the original data [i.41]. The use of RDF summaries has become the pillar element for several well-known tasks, which are [i.41]: indexing [i.42], SPARQL query processing (estimating the size of query results, making BGPs more specific, source selection) [i.43], graph visualization [i.44], vocabulary usage analysis, and ontology discovery [i.45].

Numerous approaches exist for preforming RDF summarization; they can be classified into [i.41]:

1) Structural, this approach consists of producing an RDF summary focusing on the structure of the RDF data been summarized.

2) Pattern-mining, this approach consists of discovering data-patterns and creating the summaries on top of such patterns.

3) Statistical, this approach consists of producing summaries relying on statistical evidences and indicators, they produce quantitative and frequency-based summaries of data.

4) Hybrid, this approach consists of combining strategies from the structural, statistical and pattern-mining approaches in order to exploit their benefits and compensating their potential drawbacks with the benefits brought by combining these approaches.

Depending on the task for which the RDF summarization is required some of the previous approaches can be more or less relevant. Since discovery is the matter at hand in the present document approaches like Pattern-mining and Statistical are likely to be unsuitable whereas Structural and Hybrid approaches may be suitable. Notice that source selection, which is one of the specific tasks addressed with RDF summarization, is the one specifically performed in a routing table.

Discovering resources distributed across the Web for a given SPARQL query is a hard task; some proposals like oneM2M or WoT assume that resources are described with RDF documents that specify information about the resources. These descriptions are stored in discovery services that are able to find relevant descriptions for a given SPARQL query. Nevertheless, RDF documents with large content, or a large number of simple RDF documents, can hinder this discovery and will also consume more storage resources. In this kind of scenarios, relying on RDF summarization techniques leverages the two main drawbacks by firstly saving storage space and secondly by reducing the search space that will be be analysed in order to fulfil the discovery criterion.

Assuming an IoT device that measures the air quality of a location, it has the following RDF document that describes its infrastructure as follows.

EXAMPLE 1:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix saref: < https://w3id.org/saref#>.
@prefix om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
@prefix geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
 <http://snsrs.org/2> rdf:type saref:TemperatureSensor;
 rdfs:label "Temperature sensor 2"^^xsd:string;
 saref:measuresProperty [
 rdf:type saref:Temperature;
 saref:isMeasuredByDevice <http://snsrs.org/1>, <http://snsrs.org/2>;
 saref:hasMeasurement [
 saref:hasValue "32.4"^^xsd:float;
 saref:isMeasuredIn om:degree_Celsius;
 ];
 ];
 saref:hasState [
 rdf:type saref:OnState;
 ];
 saref:isLocatedIn <http://snsrs.org/building>.
```

A naive summary that could be done is just extracting the RDF types present in the previous description, and the relationships with URIs that are not resources in the document.

EXAMPLE 2:

```
<http://snsrs.org/2> rdf:type saref:TemperatureSensor;
 saref:measuresProperty [
     rdf:type saref:Temperature;
     saref:hasMeasurement [
         saref:isMeasuredIn om:degree_Celsius;
     ];
     ];
     saref:hasState [
     rdf:type saref:OnState;
     ];
     saref:isLocatedIn [
     rdf:type saref:BuildingSpace;
     ].
```

The first RDF document requires a large number of lines to be stored, whereas the second one only a few. Assuming that one of the previous RDF documents should be stored in a routing table, the latter seems more suitable than the former one due to its size. Assuming that the following SPARQL query is issued, which looks for the URI of a sensor that measures temperature.

EXAMPLE 3:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
PREFIX saref: < https://w3id.org/saref#>.
PREFIX geo: < http://www.w3.org/2003/01/geo/wgs84_pos#>.
PREFIX om: <http://www.wurvoc.org/vocabularies/om-1.6/>.
SELECT DISTINCT ?uri {
 ?uri saref:measuresProperty ?property.
 ?property rdf:type saref:Temperature.
 ?property saref:hasMeasurement ?measurement.
 ?measurement saref:isMeasuredIn om:degree_Celsius.
 ?uri saref:hasState ?sate.
 ?state rdf:type saref:OnState.
 ?uri saref:locatedIn ?building.
 ?building rdf:type saref:BuildingSpace.
 ?building saref:contains ?space.
 ?space rdf:type saref:Window.
}
```

It can be observed that the summary has enough information to know that the SPARQL query can be answered with the description of the resource http://snsrs.org/2 and, thus, the routing table could potentially forward the query to the endpoint where the full RDF document is stored, or point the client who issued the query to such resource.

5.6.6 Routing Table Population

5.6.6.1 W3C WoT registration

The Web of Things (WoT) discovery mechanism relies on the assumption that IoT services have been described with an RDF document that follows the WoT Thing Description ontology, and that such document has been sent to a third-party service known as Thing Directory. On the one hand, the Thing Directory stores the Thing Descriptions received. On the other hand, the Thing Directory is the service in charge of discovering relevant Thing Description for a given SPARQL query. Therefore, in the Web of Things context the action of sending a Thing Description to a Thing Directory is the paramount step in order a Thing to become discoverable; this task is known as registration.

The registration of Thing Descriptions is not detailed in the WoT standard [i.31]; however, some assumptions are made: the Thing Description document is required to be expressed with JSON-LD 1.1 that is and RDF serialisation [i.33], additionally the Thing Description is required to be correctly expressed according to the WoT Thing Description ontology [i.37], and finally, The Thing Description been registered is required to be sent relying on the mime type application/td+json [i.38].

5.7 Discovery Agreement Mechanism

5.7.1 Access Control Policy

In oneM2M, the Access Control Policies (ACPs) is a specific resource, attached to CSE and used by them to control access to the resources and their attributes as specified in [i.5]. The <accessControlPolicy> resource is comprised of privileges and selfPrivileges attributes which represent a set of access control rules defining which entities (defined as accessControlOriginators) have the privilege to perform certain operations (defined as accessContolOperations) within specified contexts (defined as accessControlContexts) and are used by the CSEs in making Access Decision to all or specific parts (i.e. child resources or attributes) of the targeted resource (defined as accessControlObjectDetails and accessControlAttributes).

In a privilege, each access control rule defines which AE/CSE is allowed for which operation. So for sets of access control rules an operation is permitted if it is permitted by one or more access control rules in the set.

For a resource that is not of <accessControlPolicy> resource type, the common attribute accessControlPolicyIDs for such resources (defined in Table 9.6.1.3.2-1 of [i.5]) contains a list of identifiers which link that resource to <accessControlPolicy> resources. The CSE Access Decision for such a resource follows the evaluation of the set of access control rules expressed by the privileges attributes defined in the <accessControlPolicy> resources.

The selfPrivileges attribute represents the set of access control rules for the <accessControlPolicy> resource itself.

The CSE Access Decision for <accessControlPolicy> resource follows the evaluation of the set of access control rules expressed by the selfPrivileges attributes defined in the <accessControlPolicy> resource itself.

Logically an authorization system may comprise four sub-functions: enforcing access control decision, making access control decision, providing access control policies and providing access control information (e.g. roles). As specified in oneM2M TS-0003 [i.49], these sub-functions are modelled as policy enforcement point (PEP), Policy Decision Point (PDP), Policy Retrieval Point (PRP) and Policy Information Point (PIP) respectively. In an oneM2M System these authorization sub-functions may coexist in one CSE or may be distributed in different CSEs in different combinations.

In the <accessControlPolicy> resource three operational attributes are defined for holding the information about where to find the distributed authorization sub-functions. These attributes are: authorizationDecisionResourceIDs, authorizationPolicyResourceIDs and authorizationInformationResourceIDs.

The authorizationDecisionResourceIDs attribute contains a list of addresses of <authorizationDecision> resources. Each <authorizationDecision> resource represents a PDP to which an access control decision request is sent in order to obtain an access control decision. See clause 9.6.41 in [i.49] for further details of <authorizationDecision> resource type.

The authorizationPolicyResourceIDs attribute contains a list of addresses of <authorizationPolicy> resources. Each <authorizationPolicy> resource represents a PRP to which an access control policy request is sent in order to obtain access control policies. See clause 9.6.42 in [i.49] for further details of <authorizationPolicy> resource type.

The authorizationInformationResourceIDs attribute contains a list of addresses of <authorizationInformation> resources. Each <authorizationInformation> resource represents a PIP to which an access control information request is sent in order to obtain requested access control information (e.g. role and/or token) for making an access control decision. See clause 9.6.43 in [i.49] for further details of <authorizationInformation> resource type.

The access decision is the authorization reached when an entity's Privileges, as well as other Access Control Attributes, are evaluated.

The applicability of the authorizationDecisionResourceIDs, authorizationPolicyResourceIDs and authorizationInformationResourceIDs attributes for the distributed authorization depends on the deployment form of authorization sub-functions:

- In the case the privileges attribute is not NULL, the access control rules in the privileges attribute is used for access control, and the authorizationDecisionResourceIDs, authorizationPolicyResourceIDs and authorizationInformationResourceIDs attributes is not to be present.

- In the case the privileges attribute is NULL, how to process further depends on which authorization method is adopted. In the case distributed authorization method is supported, authorizationDecisionResourceIDs or authorizationPolicyResourceIDs attribute is considered for obtaining access control decision or access control policies from another CSE. However, authorizationDecisionResourceIDs and authorizationPolicyResourceIDs attributes are not to be present at the same time.

- In case the authorizationInformationResourceIDs attribute is present, the access control information request (e.g. for role information) related to the access control policy specified in the privileges attribute is sent to one of the addresses listed in this attribute.

The details of distributed authorization procedures are described in [i.49].

The Security Solution Specification [i.49] proposes different functionalities that enriches the ACP resource:

Authorization

- Role Based Access Control (RBAC) allows the Hosting CSE to authorize accesses on resources according to the roles assigned to the Originators.

- Token Based Access Control allows the Hosting CSE to authorize accesses on resources according to the authorization information in tokens provided by the Originators.

- Dynamic Authorization provides an interoperable framework for an Originator to be dynamically issued with temporary permissions providing the Originator with access to one or more resources on one or more CSEs at runtime.
- The entire authorization function can be split into four sub-functions: Policy Enforcement Point (PEP), Policy Decision Point (PDP), Policy Retrieval Point (PRP) and Policy Information Point (PIP). Distributed Authorization provides an interoperable framework which allows PEP, PDP, PRP and PIP to be distributed in different CSEs.
- Privacy Policy Manager (PPM) provides a standardized list of privacy attribute value pairs, automatic comparison of a user's privacy preferences with applications privacy policies and management of related access control policies.

5.7.2 Agreement in Autonomous System in Internet

In the internet a Service Level Agreement (SLA) is a contractual obligation between the service provider and the service consumer by specifying mutually agreed understandings and expectations of the provision of a service. The core of a SLA is a specification of the service guarantees, which defines both functional and non-functional guarantees of a service provision. On one hand, the interactions that need to be carried out are defined in order to specify the functionality (service) to be offered. On the other hand, a set of Quality of Service (QoS) constraints, for example, price and response time constraints, are agreed in order to specify how well the service should be offered. In addition, a SLA often contains general information such as the parties involved in service provision and their respective roles, and the validity period of the SLA.

The Internet connects thousands of Autonomous Systems operated by different administrative domains also known as Internet Service Providers (ISPs). Each Autonomous Systems has the responsibility to carry the traffic from providers to customers. They use OSPF as intra domain routing protocol. When the traffic needs to be propagated to peers Autonomous Systems it is an interdomain routing protocol such as Border Gateway Protocol (BGP). One characteristic of this protocol is the ability to determine routes according to contractual relations and agreements between neighbouring Autonomous Systems.

The commercial agreements between pairs of administrative domains can be classified into customer–provider, peering, mutual-transit, and mutual-backup agreements to select routes to propagate information to others.

At the global Internet level Routing Registries (IRR) was created as a repository of routing policies. However, some ISPs are not willing to reveal their policies, and even if they were, these routing policies might not specify Autonomous Systems relationships.

6 Extracting Potential Requirements from Use Cases

6.1 Introduction

In order to enhance the semantic capabilities of the oneM2M Discovery by providing solid contributions to the oneM2M standards, a Technical Report ETSI TR 103 714 [i.1] has been produced that proposes new operational conditions and associated uses cases for an Advance Semantic Resource Discovery. These use cases lead to potential requirements, which extend the oneM2M existing requirements. They are documented in oneM2M TR-0001 [i.59], clause 12.9 with a focus on the discovery and query capabilities. These new requirements introduce a direct relation with the semantic aspects and enable more sophisticated semantic queries as e.g. a capability in the CSE that takes routing decisions for forwarding a received Advanced Semantic Discovery Query.

In the present clause 6, specifically in clauses 6.x.1 and 6.x.3 (x runs from 2 to 18), quoted requirements (*in italic*) of ETSI TR 103 714 [i.1] have been refined and some of them are listed to the attention of oneM2M Change Requests that are related to the Advanced Semantic Discovery.

For each of them a requirement description that gives details in plain text and/or with figures is provided firstly. This first sub-clause is addressed to oneM2M semi experts. A second sub-clause references the oneM2M specifications that can be a basis for the solution or can be affected by this solution. The last part is addressed to oneM2M expert and elaborates the requirement in the oneM2M "style" by using the existing oneM2M features and pointing out the missing one.

6.2 Semantic Discovery Agreement (SDA)

6.2.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.3, 5.6, 5.7, 5.10 and 6.4.

Semantic Discovery Agreement (or mechanism) aims at adding a semantic registering information for the cooperation between CSEs.

Defining a SDA is important in an Advanced Resource Discovery because the discovery routing can pass out of a single Trusted Domain, and therefore, an Advanced Search can "walk" through many different Trusted Domains with different Access Policies. SDA should be set in advance and should be respected by Trusted Domains to guarantee "fair play" discovery packet forwarding. The SDA chosen in the present document is inspired (hence is also compatible) with the one employed by the Border Gateway Protocol 4 (BGP4 [i.62]) in Internet, that fixes cooperation among Autonomous Systems (AS), see Figure 6.2.1-1 below: left part, shows how the CAIDA AS-relation and right part show some valid and invalid advanced semantic discovery routing paths.

NOTE 1: See BGP4 [i.62].

The following text is taken from CAIDA:

*The justification for this classification is that an AS shall buy transit services for any traffic destined to parts of the Internet that this AS neither owns nor can reach through its customers. In figure 6.2.1-1 (left), arrow directions reflect flows of money -- ASes at lower levels are customers who pay ISPs (providers) at higher levels in exchange for access to the rest of the Internet, also known as transit. We refer to links between a customer and a provider as **c2p** (**p2c**) links. In Figure 6.2.1-1 (left), D->B, E->B, F->C, B->A, and C->A are c2p links.*

*A **p2p** link connects two ISPs who have agreed to exchange traffic on a quid pro quo basis. Peers should exchange traffic only between each other and each other's customers. Peering allows growing ISPs to save money on transit costs they would otherwise have to pay to deliver traffic to/from their customers. In Figure 6.2.1-1 (left), B-C is a p2p link, unidirectional since neither B nor C pays the other for the traffic they exchange.*

*An **s2s** link connects two ASes with a common administrative boundary. Such links usually appear as a result of mergers and acquisitions, or under certain network management scenarios.*

*We use the notion of money transfers between ASes to define **valid and invalid AS paths**. A valid path between source and destination ASes is one in which for every ISP providing transit (a transit provider), there exists a customer immediately adjacent to the ISP in the AS path. An invalid path has at least one transit provider not paid by a neighbour in the path.*

Figure 6.2.1-1 (left) show the Types of AS relationships. The ASes at the bottom of the graph, D, E, and F, are customers of those above. ISPs in the middle, B and C, are both providers of ASes below and customers of ISPs above. ISPs B and C are also peers of each other. ISP A at the top is a provider to B and C and a customer of no one.

Figure 6.2.1-1 (right) the top two examples are valid paths, while the bottom two are invalid. In Example 1 the transit providers are A, B, and C. ISPs B and C pay to A, D pays to B, and F pays to C. In Example 2 the transit providers are B and C, and they are paid by D and F, respectively. In contrast, in Example 3 the transit provider is B, but not only does no one pay B, but B itself pays both A and Z. Example 4 also illustrates a situation where nobody pays transit provider B.

In other words, a valid path shall have the following valid path pattern: zero or more c2p links, followed by zero or one p2p link, followed by zero or more p2c links. In addition, s2s links can appear in any number anywhere in the path.

See CAIDA Internet homepage for more on that topic.

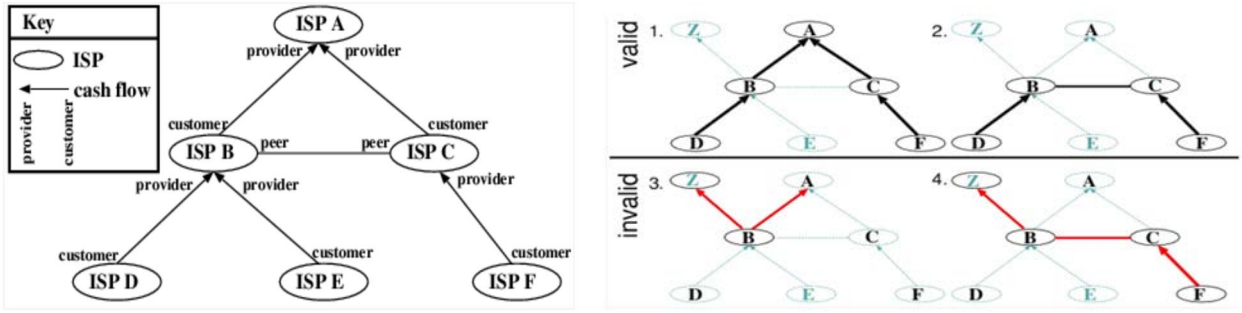


Figure 6.2.1-1: CAIDA topologies (left) cash flow; (right) valid and invalid paths

NOTE 2: Source [i.63].

In oneM2M, 3 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 Advanced Semantic Discovery routing paths intra and inter Trusted Domains and Service Providers.

Concluding, setting correct SDA intra and inter oneM2M Trusted Domains is useful to control the following oneM2M features:

- access control policies between IN-CSEs belonging to different Trusted Domains/Service Providers;
- access control policies between IN-CSE and MN-CSE and ASN-CSE inside a single Trusted Domain/Service Provider;
- accounting issues (some spaces can or cannot be searched in) intra and inter Trusted Domains/Service Providers;
- discovery scoping intra and inter Trusted Domains/Service Providers;
- discovery pricing inter Trusted Domains/Service Providers;
- discovery efficiency intra and inter Trusted Domains/Service Providers;
- discovery load balancing intra and inter Trusted Domains/Service Providers;
- driving discovery results intra and inter Trusted Domains/Service Providers;
- IPR and licensing issues CC BY* related to TRANSIT of information between different Trusted Domains.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) 3 kinds of relationship between CSE are settled: CUSTOMER-to-PROVIDER (C2P), PEER-to-PEER (P2P) and SIBLING-to-SIBLING (S2S);

2) SDA is useful to define VALID and INVALID Advanced Semantic Discovery routing paths;

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;

4) each CSE contains a SDA-table containing the URI of all CSEs which is connected classified following the C2P, P2P and S2S relationship.

6.2.2 References to oneM2M Specifications

Relevant information of relationship of CSEs pertaining to the registration is represented in <remoteCSE> resources. In oneM2M TS-0001 [i.5], clause 9.6.4, the remoteCSE resource type defines the cseType and descendantCSEs attributes. The CSE Type represents the CSE type of a registrar or registree CSE (e.g. IN-CSE, MN-CSE). The descendantCSEs attribute contains the CSEs hierarchy as the list of descendant CSE IDs.

With the request target (***To*** parameter) information, CSEs registration information is used for request forwarding when a CSE receives a request but it is not the CSE to process that request. The procedure is illustrated in clause 8.2 of oneM2M TS-0001 [i.5].

6.2.3 Elaborated oneM2M Requirements

A semantic discovery agreement is made between CSEs, who performs request forwarding/routing. Each CSE share their information during registration and if the agreement is decided and recorded it needs to be done during the CSE registration. Note that M2M Node is the requirement phase term which replaces CSE, that is used in solution illustrations:

1) *The oneM2M System shall support relationship type information during M2M Node registration which can be used for selecting target node for semantic discovery/query.*

6.3 Advanced Semantic Discovery (ASD)

6.3.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.10 and 6.1.

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

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 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.

The Advanced Semantic Discovery aims to discover AEs (also called Resources) that are registered/announced to some CSEs. The Advanced Semantic Discovery could start from any AE, even these ones not belonging to the same Trusted Domain. The Advanced Resource Discovery differs from the usual one present in oneM2M in the sense that one (or many) AE could be searched for even without knowing its identifier but just knowing its TYPE or ONTOLOGY membership, as shown in Figure 6.3.1-1.

Figure 6.3.1-1: ASD in one image

The result of a successful Advanced Semantic Discovery is the HANDLER (URI in oneM2M) that can be then used to perform ordinary oneM2M queries.

As a simple example, an advanced semantic query should search for a set of discovery criteria specified in the Advanced Semantic Discovery Query language, like, e.g. NUMBER=1 or NUMBER=N or NUMBER=ALL and resources of TYPE=THERMOMETER.

Some extra parameters (aka, SEMANTIC FILTER CRITERIA) can be used to refine the Advanced Semantic Discovery, like e.g. all THERMOMETERS in TRUSTED_DOMAIN=MY_TD or in TIME[60] or SPACE[1000], etc.

One Advanced Semantic Discovery can be aggregated by AND and OR and NOT logical operators to build complex queries and that may vary on the way discovery is performed, like e.g.:

[NUMBER=1,TYPE=THERMOMETER] AND [NUMBER=1,TYPE=HUMIDITYMETER].

Intuitively a CSE receiving a complex query:

1) reduce a complex query into a number of unary queries;

2) route the unary queries one-by-one; and

3) aggregate and return the results (see Figure 6.3.1-2).

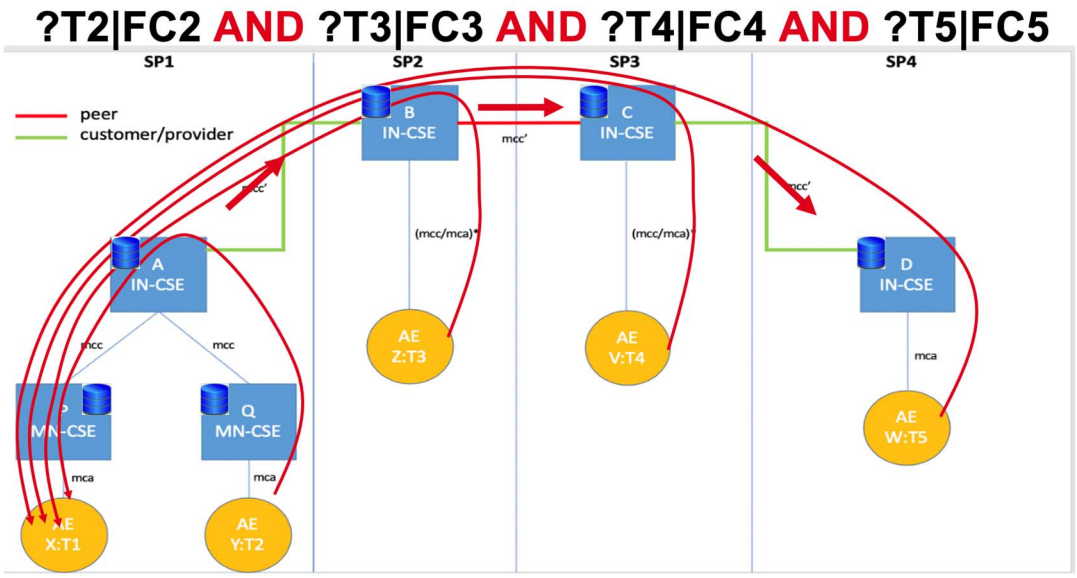


Figure 6.3.1-2: ASD routing in one slide

Summarizing the following potential requirements for the oneM2M system are advocated:

1) the oneM2M system provides an Advanced Semantic Discovery (ASD) across a distributed network of IoT nodes within a single oneM2M Service Provider;

2) the ASD is performed from any AE, even these ones not belonging to the same Trusted Domain and across different IoT Service Providers;

3) the ASD is time and space aware;

4) the ASD is guided by a Semantic Recommendation (SR) system, implemented in each CSE, in order to improve the performance of the service;

5) the ASD supports semantic reasoning between the oneM2M base ontology (search on TYPES);

6) the oneM2M Access Control Policy is compatible with SDA relationship and includes discovery permissions to support ASD;

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.

6.3.2 References to oneM2M Specifications

Discovery in oneM2M is the procedure which returns matching resource identifiers: it is illustrated in clause 10.2.6 of oneM2M TS-0001 [i.5]. Discovery actually features a very limited form of "CSE-to-CSE routing" (sometimes called "CSE-to-CSE traversal"), as illustrated in clause 8.2 of oneM2M TS-0001 [i.5]. A single request having one resource can be reached in a remote CSE and then request routing happens as specified. This means in oneM2M notion, when a CSE receives a request but it does not handle or process, but forward it to another CSE, this is routing.

Communications between different SPs is illustrated in oneM2M TS-0001 [i.5], clause 6.5. The clause defines that an M2M Service Provider has its identifier which is in the FQDN format. An IN-CSE forward a received request to a proper IN-CSE by checking the SP-ID that is the part of the request target resource ID.

Figure 6.3.2-1 describes oneM2M as-is semantic discovery involving multiple CSEs. As a simple scenario, AE1 can semantically discover AE2 on CSE1, which is the AE1's Registrar, when AE2 or another entity provides semantic description of the AE2 on CSE1. In this case, the strong assumption made is *AE2 already knows AE1* would do the semantic discovery on its Registar, CSE1. A bit more feasible use case would be AE2 provides its semantic annotation on the IN-CSE of the Service Provider domain that the AE2 belongs to. With the assumption made in oneM2M TS-0001 [i.5], an entity in a SP gets provision information including its IN-CSE identifier. Therefore in this second setup, AE2 provides semantic annotation with the announcement to its IN-CSE and AE1, when it joins the system later, do the discovery on its IN-CSE, which is CSE2. These two examples are depicted to show that currently oneM2M release *does not* support the ASD discovery, as described in clause 6.3.1. A oneM2M discovery targets one CSE with one resource as a discovery starting point, and it does not get propagated to other CSEs.

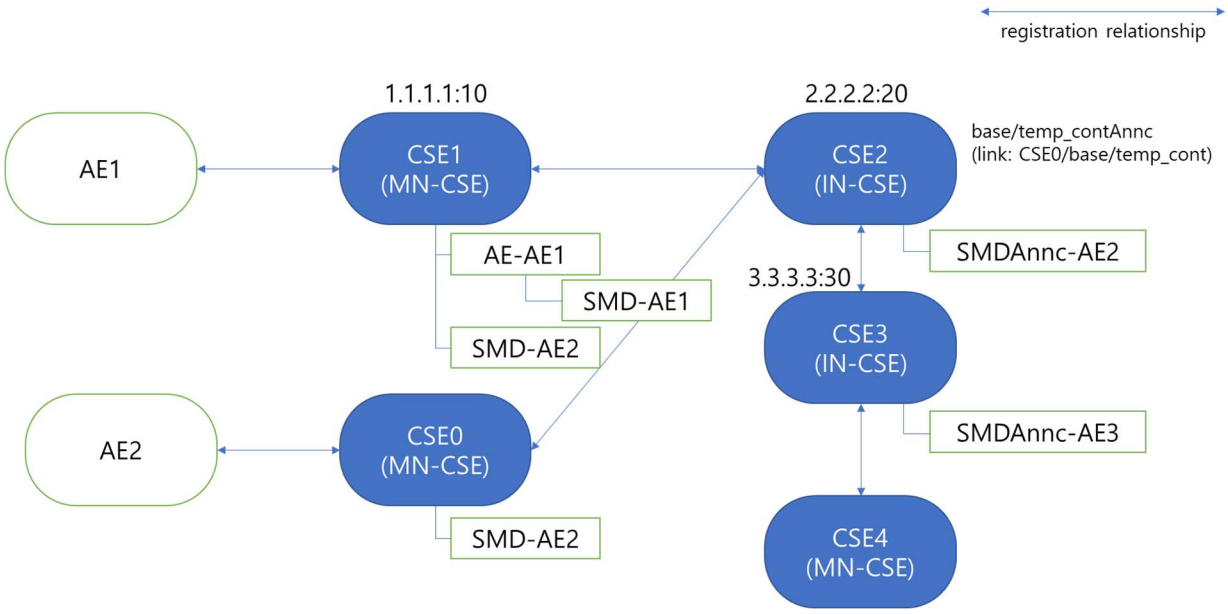


Figure 6.3.2-1: Actual oneM2M limited discovery routing in one slide

The use case in clause 6.3.1 illustrates the "other" (from oneM2M specification point of view) type of routing and it is rather similar to group-fanout mechanism since:

i) intermediate CSE initiates other requests than forwarding the original request; and

ii) responses are aggregated by the intermediary.

This group fan-out feature is illustrated in clause 10.2.7 of oneM2M TS-0001 [i.5].

6.3.3 Elaborated oneM2M Requirements

The needs (or requirement) for semantic reasoning is captured in oneM2M TS-0034 [i.7] already in oneM2M but no solution is provided yet. To illustrate the semantic recommendation in oneM2M TS-0034 [i.7], detailed mechanism needs to be provided above. Instead if this is implementation scope, there is no specification on this:

1) *The oneM2M System shall support semantic queries which supports temporal and spatial aspects.*

2) *The oneM2M System shall support semantic query results aggregation for a semantic query that was fanned-out targeting multiple M2M Nodes with different or fragmented semantic query criteria.*

3) *The oneM2M System shall support a context-aware access control mechanism including entity relationships of M2M Nodes.*

4) *The oneM2M System shall support access granting mechanism for discovery-related resource(s) between M2M Nodes when they have specific relationship(s).*

6.4 Advanced Semantic Discovery Query (ASDQ)

6.4.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 5, 6 and 7.

oneM2M currently relies on the any protocol that can be bind to a oneM2M primitive for performing semantic discovery (oneM2M TR-0045 [i.51]), nevertheless it does not implement the SPARQL 1.1 protocol. Additionally, the semantic discovery requests sent with primitives lack of routing parameters, which are paramour for the Advanced Semantic Discovery Query (ASDQ). Therefore, in order to be retro compatible the new ASDQ does not implement SPARQL 1.1 protocol, instead it will extend the parameters currently supported by oneM2M in order to include routing capabilities. The SPARQL 1.1 protocol is well explained in clause 5.5.6 of the present document, there a table with the supported HTTP methods and parameters is presented. Nevertheless, the SPARQL protocol implemented in oneM2M differs with the standard one.

The same query from the example presented in clause 5.5.6, but written according to this SPARQL 1.1 protocol extended is the following:

```
GET /sparql?max_hops=5&current_hops=3&query=
PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-
ns%23%3E%20.%0APREFIX%20saref%3A%20%3C%20https%3A%2F%2Fw3id.org%2Fsaref%23%3E.%0
APREFIX%20geo%3A%20%3C%20http%3A%2F%2Fwww.w3.org%2F2003%2F01%2Fgeo%2Fwgs84_pos%2
3%3E%20.%0APREFIX%20om%3A%20%3Chttp%3A%2F%2Fwww.wurvoc.org%2Fvocabularies%2Fom-
1.6%2F%3E.%0ASELECT%20DISTINCT%20%3Furi%20%7B%0A%09%3Furi%20saref%3AmeasuresProp
erty%20%3Fproperty%20.%0A%09%3Fproperty%20rdf%3Atype%20saref%3ATemperature%20.%0
A%09%3Fproperty%20saref%3AhasMeasurement%20%3Fmeasurement%20.%0A%09%3Fmeasuremen
t%20saref%3AisMeasuredIn%20om%3Adegree_Celsius%20.%0A%09%3Furi%20saref%3AhasStat
e%20%3Fsate%20.%0A%09%3Fstate%20rdf%3Atype%20saref%3AOnState%20.%0A%09%3Furi%20s
aref%3AlocatedIn%20%3Fbuilding%20.%0A%09%3Fbuilding%20rdf%3Atype%20saref%3ABuild
ingSpace%20.%0A%09%3Fbuilding%20saref%
```

The query looks for a device that measures temperature and that is located in a building. Notice in the request the parameter `max_hops` that specifies that the query should be routed 5 number of hops, and currently it already travelled 3 hops according to the parameter `current_hops`.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *Advanced Semantic Discovery Query (ASDQ) is sent using the oneM2M primitives;*

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);*

3) *Advanced Semantic Discovery Query (ASDQ) includes a set of parameters configuring routing options.*

6.4.2 References to oneM2M Specifications

The semanticFilter condition of the **Filter Criteria** request parameter in oneM2M (clause 8.2.1 of oneM2M TS-0001 [i.5]) carries SPARQL query in base64 encoding. The HTTP binding specification [i.54] defines that the semantic filter gets delivered in HTTP query string (see clause 6.2.2.2).

6.4.3 Elaborated oneM2M Requirements

Any oneM2M requests, including a discovery or query, can be sent over HTTP protocol. The operation mapping (e.g. RETRIEVE to HTTP GET) is determined in the binding specification. Requirements 3 and 4 is captured as below:

1) The oneM2M System shall support different query conditions for semantic query which is used for semantic query resolution and fan-out.

6.5 Advanced Semantic Discovery Query Language (ASDQL)

6.5.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 6.1, 6.6, 6.7, 6.10, 5.1, 5.6, 5.7 and 5.10.

According to the current oneM2M TR-0045 [i.51] CSEs in oneM2M implement the semantic discovery by means of the SPARQL 1.1 Query Language, although, CSEs do not implement the official SPARQL 1.1 protocol. This query language is used for querying RDF data, specifically; the CSEs solve these queries over the resource descriptions sent by the registered AEs. Notice that RDF and the SPARQL 1.1 Query Language are W3C standards, and allow expressing data as a graph, or query such data. RDF data have to rely on a specific and well-defined ontology, the SPARQL queries are recommended to use the terms specified in the ontology in order to find suitable RDF documents.

From the aforementioned features of oneM2M, the SPARQL query language could perfectly fit the requirements derived for the ASDQL. Examples of SPARQL queries using terms from the oneM2M ontology, and terms from the SAREF ontology combined with the oneM2M can be found in clauses 5.3.3, 5.5.6 and 5.6.5.1 of the present document.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) ASDQL is able to query RDF data, since discovery in oneM2M consists in finding relevant semantic descriptors which are expressed in RDF;

2) ASDQL is implemented using a well-known and established standard;

3) ASDQL allows expressing queries over graph data;

4) ASDQL have the capabilities to use the terms of the oneM2M ontology and any other ontology used by the data.

6.5.2 References to oneM2M Specifications

1) oneM2M supports RDF data query while embedding SPARQL query that will be performed for RDF data of the targeted <semanticDescriptor> resources. This is illustrated in clauses 7.4 and 7.5 of [i.7];

2) oneM2M supports any ontology as well as oneM2M defined Base Ontology. The ontology can be indicated by the ontologyRef attribute of a <semanticDescriptor> resource. See clause 9.6.30 of [i.5].

6.5.3 Elaborated oneM2M Requirements

oneM2M supports W3C defined RDF triples to store semantic data with SPARQL query execution feature. When stores semantic annotations the reference ontology can be oneM2M base ontology or could be any ontologies.

No new requirement is required since oneM2M already supports them.

6.6 Semantic Discovery Routing Mechanism (SDRM)

6.6.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

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

1) reduces the Advanced Semantic Discovery Query (**ASDQ**) by means of the Semantic Query Resolution Mechanism (**SQRM**);

2) solves and forwards in a distributed way the queries;

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

Generally, as described in ETSI TR 103 714 [i.1] two kinds of routing are discriminated, namely:

- **Exhaustive**. As example, in the case that a semantic resource exists somewhere in the CSEs network, then the system will explore the entire distributed network until it will find it.

- **Non-exhaustive**. As example, even in the case a semantic resource that exists somewhere in the CSEs network, the system will explore part of the distributed network until it will be stopped because of different reasons (Timeout or exceeded TTL, etc.).

Advanced Semantic Routing is the protocol interpreting an Advanced Semantic Query: in order to work it allow to define USER DEFINED TYPES of object entities in the oneM2M Base Ontology.

As example, let the type THERMOMETER, or WATER_VALVE denote all the AE of type thermometers or water valves. Types are needed to perform query asking for all entities of that kinds without knowing the CSE where the entity is registered (or announced). Suitable SEMANTIC FILTER CRITERIA can be provided to refine the Advanced Semantic Discovery but their presence is not enforced in the query syntax of the query language.

Each CSE have a LOCAL RESOURCE DATABASE (aka, SEMANTIC ROUTING TABLE), see clause 6.9 listing, for each type:

- the URI of the objects entities of that type registered in that CSE;

- for all CSE connected with the current CSE with a C2P, P2P and S2S relationship, a quantitative information of how many AE (with respective URI) are (recursively) registered with that type.

Consider the following routing example built by considering:

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

- 2 Middle Node Common Service Entities (MN-CSE) P, and Q. A MN-CSE has a local database containing information on their registered AE. The local database includes location information (where each device is presently located), the device type, etc. Assuming P and Q have some Semantic Discovery Agreement (SDA) with A. Semantic Discovery Agreement (SDA) can be relaxed inside a single Service Provider, see note 2 of SDA definition in clause 3.1;

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

Figure 6.6.1-1 considers the following CSE topology.

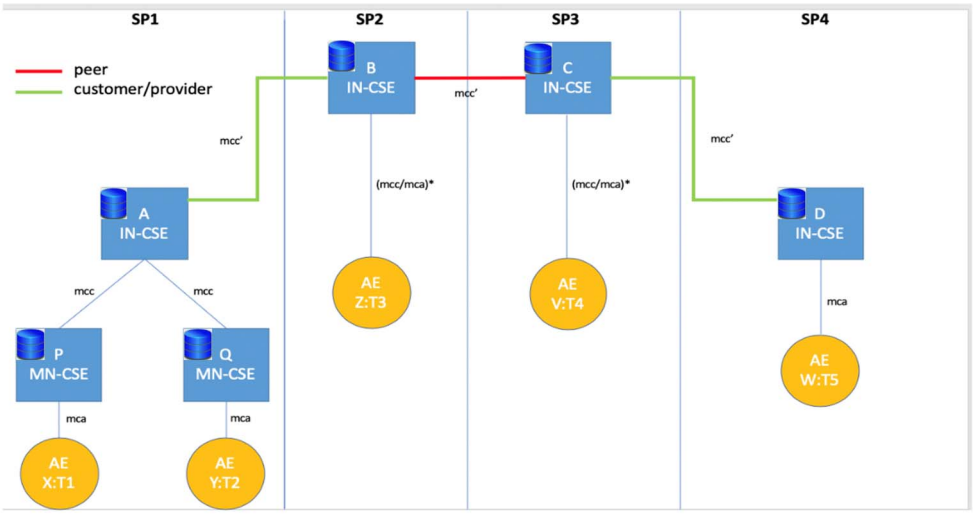


Figure 6.6.1-1: Simple topology listing SDA

Assuming AND and OR and NOT, be non-terminals used in an Advanced Semantic Query and T means a type T to be resolved.

Following some examples in an abstract syntax that to be translated in the oneM2M Query Language are listed.

EXAMPLE 1: X:T1 send to MN-CSE P the following query (written in an abstract matter).

ASDQ1 = T2|FC2 AND ?T3|FC3 AND ?T4|FC4 AND ?T5|FC5

The query can be intuitively read as follows: X is looking for:

- some AE of type T2 registered in any CSE satisfying the filter criteria FC2; AND
- some AE of type T3 registered in any CSE satisfying the filter criteria FC3; AND
- some AE of type T4 registered in any CSE satisfying the filter criteria FC4; AND
- some AE of type T5 registered in any CSE satisfying the filter criteria FC5.

The next examples list other potential Advanced Semantic queries that could be routed in the oneM2M semantic overlay network.

EXAMPLE 2: X:T1 send to MN-CSE P

ASDQ = ?T2|FC2 OR ?T3|FC3 OR ?T4|FC4 OR ?T5|FC5

The query can be intuitively read as follows: X is looking for:

- some AE of type T2 registered in any CSE satisfying the filter criteria FC2; OR
- some AE of type T3 registered in any CSE satisfying the filter criteria FC3; OR
- some AE of type T4 registered in any CSE satisfying the filter criteria FC4; OR
- some AE of type T5 registered in any CSE satisfying the filter criteria FC5

EXAMPLE 3: X:T1 send to MN-CSE P

ASDQ = (?T2|FC2 OR ?T3|FC3) AND (?T4|FC4 OR ?T5|FC5)

EXAMPLE 4: X:T1 send to MN-CSE P

ASDQ = (?T2|FC2 AND ?T3|FC3) OR (?T4|FC4 AND ?T5|FC5)

EXAMPLE 5: X:T1 send to MN-CSE P

ASDQ = (?T2|FC2 AND ?T3|FC3) OR (?T4|FC4 AND (NOT ?T5|FC5))

Simple queries (in SPARQL syntax) composed by AND, OR and NOT and ROUTING SEMANTIC FILTER CRITERIA to describe directives to walk the Search Routing Space. Examples of some filter criteria are: (this list not exhaustive and is here just to foster the reader intuition):

ANY = search all resources matching the criteria;

CURRENT = search in own registered CSE;

PROVIDER = search in own Trusted Domain;

PROVIDER[SP1...SPn] = search in SP1... SPn Service Providers;

TIME[SEC] = search should give the results found in SEC;

SPACE[METERS] = search should give the results within METERS from a given point.

Figure 6.6.1-2 below shows an alternative topology where CUSTOMER-PROVIDER SDA are reversed.

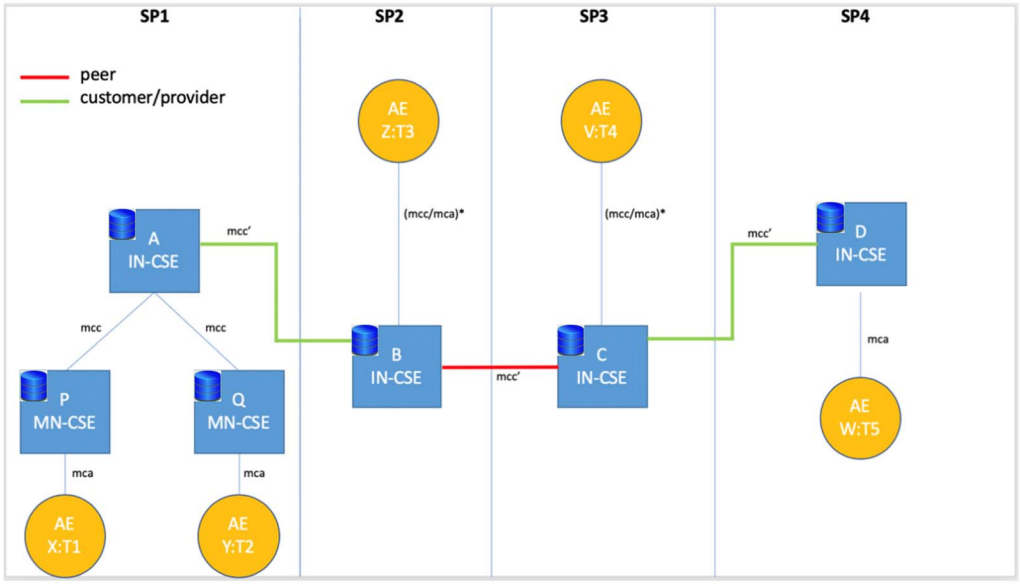


Figure 6.6.1-2: Alternative topology listing SDA

In that topology, the AE X could not get fully access to AE of type T3, T4 and T5 because the semantic routing will break the potential SDA requirements for the oneM2M system listed in clause 6.2.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *It is possible to introduce in the ASDQL new SEMANTIC FILTER CRITERIA for defining suitable routing policies.*

2) *The SDRM manages user defined types of objects.*

3) *The SDRM reduces the ASDQ.*

4) *The SDRM solves the ASDQ.*

5) *The SDRM forwards the simplified ASDQ.*

6) *The SDRM reconstructs the partial results sending back to the originator.*

6.6.2 References to oneM2M Specifications

See clause 6.3.2 which includes the semantic discovery routing aspects.

6.6.3 Elaborated oneM2M Requirements

During a oneM2M request forwarding, including discovery and query, an original request is not modified. In the SDRM, an original query can be fragmented and each fragment can be sent, as a new request, to a proper target. To illustrate that additional semantic filter criteria can be included.

1) The oneM2M system shall support semantic data conditions for semantic query routing.

2) The oneM2M system shall support semantic query fragmentation and fan-out of fragments to different oneM2M Nodes.

See also clause 6.3.3 which includes the semantic query fragmentation and results aggregation.

6.7 Semantic Query Resolution Mechanism (SQRM)

6.7.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

As described in the Advanced Semantic Discovery Routing clauses 6.3 and 6.6, upon reception of a Complex Advanced Semantic Query the CSE parse the query and reduce it to either a CONJUNCTIVE NORMAL FORM (CNF) or a DISJUNCTIVE NORMAL FORM (DNF).

Examples of CNF are:

- $(A \vee \neg B \vee \neg C) \wedge (\neg D \vee E \vee F)$
- $(A \vee B) \wedge C$
- $A \vee B$

Examples of DNF are:

- $(A \wedge \neg B \wedge \neg C) \vee (\neg D \wedge E \wedge F)$
- $(A \wedge B) \vee C$
- $A \wedge B$

It is well-known in the literature that a CNF is a conjunction-of-disjunctions of literals and that a DNF is a disjunction-of-conjunctions of literals. In both CNF and DNF the negation (NOT) can appear only in an atomic query. Resolving a complex query into a CNF (respectively DNF) is useful to solve locally as much atomic query as possible and start the most appropriate number of Advanced Semantic Routing Queries. The algorithmic complexity of this pre-routing is important to optimize the number of network queries.

Summarizing the following potential requirements for the oneM2M system is advocated:

1) The SQRM is embedded in each CSE and is able to reduce a complex formula into a CNF (resp. DNF).

6.7.2 References to oneM2M Specifications

oneM2M request parameter **Filter Criteria** (except the semanticFilter that delivers SPARQL query) does not support complex condition composition as illustrated above.

6.7.3 Elaborated oneM2M Requirements

oneM2M System shall support complex queries with multiple logical operators and decomposition of the complex query.

6.8 Semantic Recommendation system (SR)

6.8.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

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

1) read the SDA-Table and the SRT;

2) take a decision on the next hop(s) for the Advanced Semantic Query; and

3) annotate all the CSE belonging to the SDA-Table that participate pro- or contra-actively to a successful routing.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) The SR can access the SDA-Table and the SRT.

2) The SRT contains a list of most active CSEs.

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.

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.

6.8.2 References to oneM2M Specifications

In oneM2M, when a CSE receives a request and forward the request to a transit CSE. The receiving CSE needs to decide which CSE needs to get the forwarding. It is either its Registree or Registrar. This means a request can be forwarded to an Entity that in registration relationship.

6.8.3 Elaborated oneM2M Requirements

When CSE access its resources, any specific requirements or features on it (see the above requirement 1) are not needed. A semantic routing table can have an indicator or index of a CSE which can be used for this routing, but how to use the data would be implementation specific. The indicator or the index can be updated by the CSE after the successful routing:

1) The oneM2M System shall support semantic data aggregation and maintenance mechanisms for semantic query fan-out to multiple M2M nodes.

2) The oneM2M System shall support request routing information which can be updated applying successful routing.

6.9 Semantic Routing Table (SRT)

6.9.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

Informally speaking, a Semantic Routing Table is a data structure, embedded in each CSE, containing sufficient information to route an Advanced Semantic Query -- generated either by a registered AE or received by another CSE -- to the "next CSE", so allowing the Advanced Semantic Routing possible. The data structure should contain at least the following information (this list not exhaustive but it is given to foster the reader imagination):

1) a SDA-table listing the URI of all the CSE connected with C2P, P2P and S2S relationship;

2) a FEATURE-TYPE-table listing the URI of all the AE, having the FEATURE-TYPE and directly registered (or announced) in the current CSE;

3) for each FEATURE-TYPE, the SRT should indicate the "number" of AEs of that type grouped by the SDA relationships: grouping is advocated to allow the Semantic Recommendation system (SR) to choose the best route maximizing successful queries and minimizing routing hops.

The SRT should be kept updated as much as possible, especially in case of high intermittence of AE.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) The SRT contains a SDA-table.

2) The SRT contains a FEATURE-TYPE-table.

3) The SRT refers all the URI of AE registered in the adjacent CSEs, according to the C2P, P2P and S2S relationship.

4) The SRT is kept fresh and updated to avoid FAULTY ROUTES.

6.9.2 References to oneM2M Specifications

Regarding the request forwarding see clause 6.8.2. A <remoteCSE> resource contains such information (see clause 9.6.4 of oneM2M TS-0001 [i.5]). An IN-CSE or MN-CSE can also look-up the descendantCSEs attribute of the remoteCSE resource type for forwarding when the discovery or query target is in more than zero-hop. The hop definition in oneM2M is different from traditional hop concept from the underlying network protocols and it is illustrated in clause 8.2 of oneM2M TS-0001 [i.5].

If a new resource type for the SRT is needed, then it can be specified in clause 9.6 of oneM2M TS-0001 [i.5].

6.9.3 Elaborated oneM2M Requirements

SDA information can be contained in the routing table which can consist of annotations of any resources, not just to the <AE> resources. The following candidate requirements captures the clause 6.9.1:

1) The oneM2M System shall support semantic query routing tables that consist of registration relationship type for semantic queries.

2) The oneM2M System shall support mechanisms to maintain semantic query routing table (e.g. semantic data summarization) for semantic query fan-out.

6.10 Building a CSE Topology Linked with the oneM2M actual topologies

6.10.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1]. All use cases showed oneM2M topology composed by a finite number of trees where the roots are connected together, in a mesh-like fashion. In all examples, SDA are respected in order to enforce oneM2M overlay routing compliant with the underlying networks (Internet, 3GPPP/2, etc.). Figures 6.10.1-1 and 6.10.1-2 present some preliminaries topologies that are evaluated in ETSI TR 103 716 [i.3].

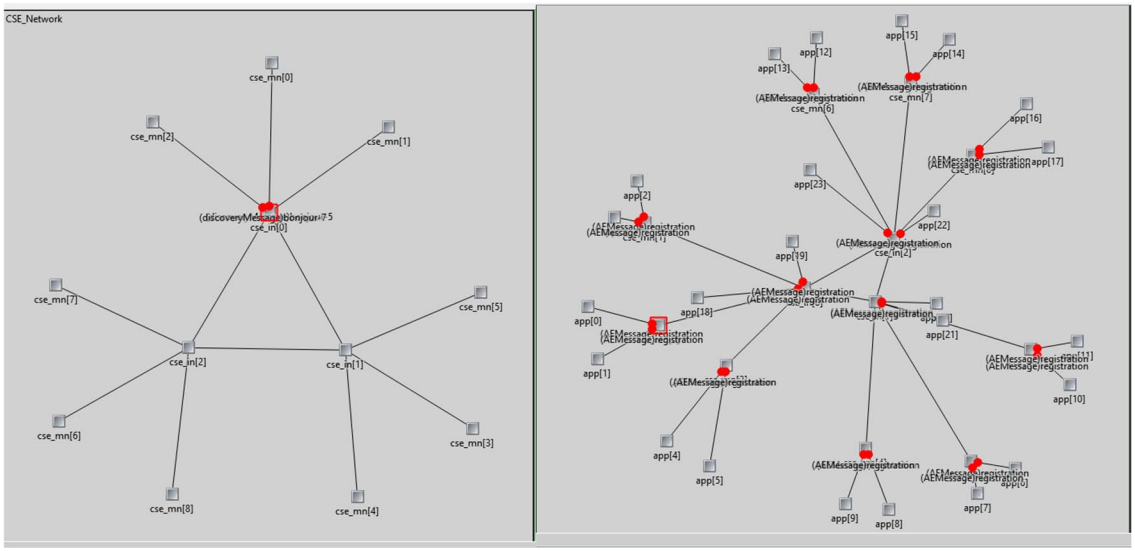


Figure 6.10.1-1: Simple topologies evaluated in ETSI TR 103 716 [i.3]

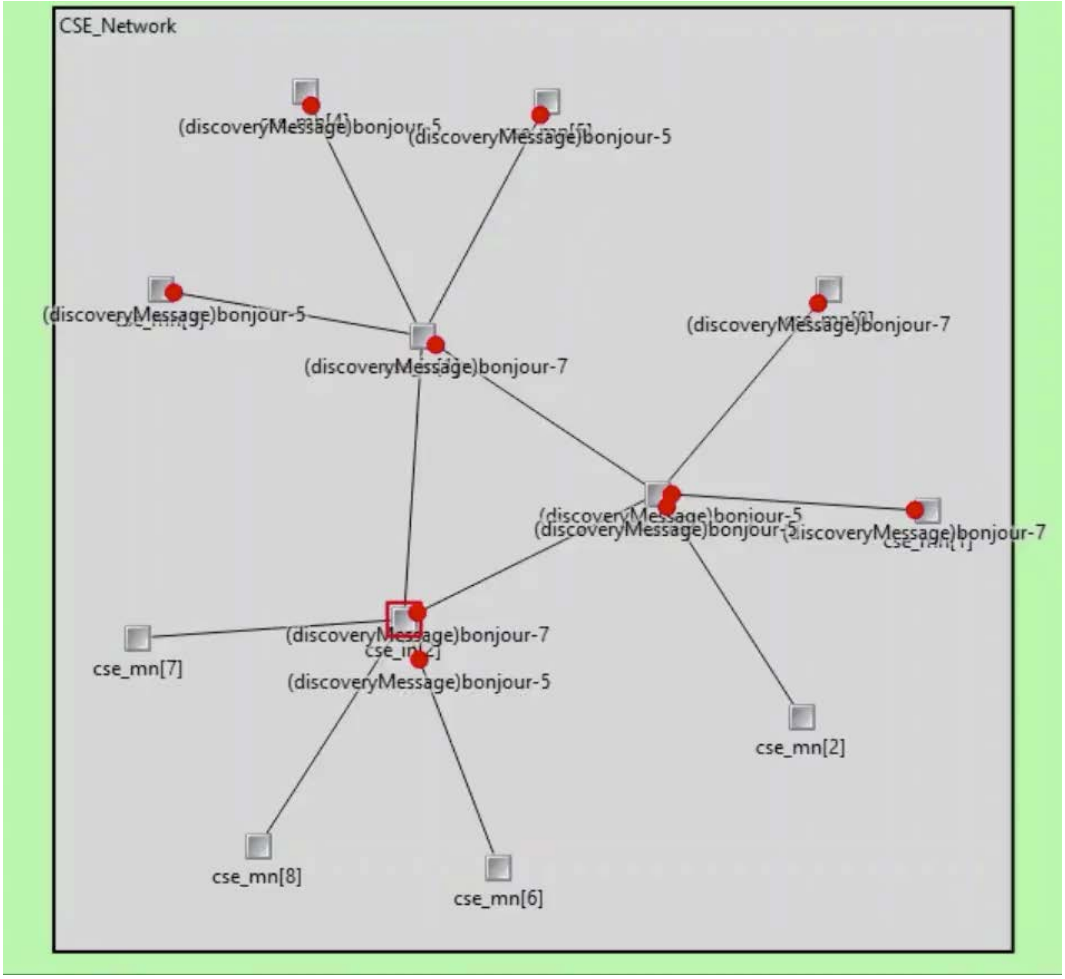


Figure 6.10.1-2: Advanced Semantic Routing query evaluated in ETSI TR 103 716 [i.3]

Summarizing the following potential requirements for the oneM2M system are advocated:

*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.22*].*

6.10.2 References to oneM2M Specifications

oneM2M request routing is handled among CSEs, so the topology consists only CSEs in general. When the semantic routing is involved, there could be the specification to perform the semantic query routing (or fan-out) based on the table, but it would be written separately in clause 7.2.

6.10.3 Elaborated oneM2M Requirements

A new topology for semantic query routing (fan-out) can be maintained by a CSE and this is not relevant to existing non-semantic request routing:

1) The oneM2M System shall maintain topologies for semantic queries forwarding.

6.11 Queries Integrating Baseline and Specific Domain Ontology (SAREF)

6.11.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 6.4 and 7.4.

Currently oneM2M provides an ontology (oneM2M TS-0012 [i.52]), which covers concepts to describe generic devices and sensors, as well as, their endpoints. Nevertheless, the current ontology lacks of domain-specific terms such as types of sensors or devices, units, properties reported (like humidity or temperature). This hinders the semantic discovery that needs to refer to generic terms instead of specific ones that allow fine-grained discovery queries. Although the oneM2M system counts with extension mechanisms, the ASD relies instead on at least one already-built extension with a domain-specific ontology, i.e. the Advanced Semantic Discovery Ontology. As a result, users will not need to produce the necessary alignments between the oneM2M ontology and the domain-specific one; instead, users will be able to directly use fine-grained queries referring to those specific terms. As preferable candidate, SAREF would suit such requirement: on the one hand, this new ontology counts with a wide number of extensions for expressing domain specific terms (extracted from SAREF), on the other hand, the Advanced Semantic Discovery Ontology includes terms referring to concepts from oneM2M (like AEs or CSEs among others) that will allow expressing queries that refer to such terms.

The present document already presented different scenarios with examples in which SAREF and the oneM2M ontologies were used, namely in clauses 5.2.3, 5.3.3, 5.6.6 and 5.6.5.1.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) ASDQ refers to terms in the Advanced Semantic Discovery Ontology.

Previous clauses presented different scenarios with examples in which SAREF and the oneM2M ontologies were used, namely: 5.2.3, 5.3.3, 5.6.6 and 5.6.5.1.

6.11.2 References to oneM2M Specifications

oneM2M supports any ontology as well as oneM2M defined Base Ontology in oneM2M TS-0012 [i.52]. The ontology can be indicated by the *ontologyRef* attribute of a <semanticDescriptor> resource. See clause 9.6.30 of [i.5]. oneM2M base ontology is somehow recommended to use. To guarantee semantic interoperability among different ontologies oneM2M also defined ontology mapping with SAREF in oneM2M TS-0012 [i.52].

6.11.3 Elaborated oneM2M Requirements

Regarding the Advanced Semantic Discovery Ontology, the same new requirement in clause 6.17.3 is applied here.

6.12 Queries Integrating Multiple Set of Targets and Multiplicity of Searches

6.12.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5, 6 and 7.

In clause 5.5.6 of the present document, the SPARQL 1.1 Query Language was presented. As previously shown, this query language allows referring online RDF documents over which the query will be solved. Additionally, the queries can refer to endpoints SPARQL, for this case, the query is federated to those endpoints and a unified view is provided. Example of these capabilities are presented in clause 5.3.3 and clause 5.5.6.

Despite the previous functionalities, oneM2M CSEs lack of those capabilities since they rely on the SPARQL Query Language but do not implement the SPARQL 1.1 protocol. Therefore, oneM2M CSEs should implement the functionalities required to solve SPARQL queries that refer to online RDF documents, as well as, those required to federate the queries.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *The query language allow expressing a set of URIs that belong to oneM2M systems that implement a query solving mechanism.*

2) *The query protocol is able to solve queries over a set of distributed oneM2M systems specified in the query.*

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.*

4) *The query protocol is able to solve queries over a set of distributed oneM2M URIs that publish RDF.*

An example of R1 and R3 is presented in clause 5.3.3 in which it was shown how the SPARQL queries could reference public available URIs that publish RDF documents. Furthermore, an example of R2 and R4 is presented in clause 5.5.6.

6.12.2 References to oneM2M Specifications

The scope, or in other words targeted semantic description resources, of semantic query or discovery can be decided based on resource tree concept, which is illustrated in clause 5.5.3.2. Alternatively, multiple resources no matter where those are located in a tree, or even spread in different CSEs, can be the semantic discovery and query target. This is supported by the semanticFanOutPoint resource type and its handling procedures (clauses 9.6.14 and 10.2.7.12, respectively of [i.6]), which leverage group fan-out mechanism in oneM2M.

6.12.3 Elaborated oneM2M Requirements

A complex query fragmentation, fan-out and result aggregation is captured in clauses 6.3 and 6.6.

Requirement 2 can be fulfilled with group mechanism having semantic fan-out feature. It is able to provide multiple targets for semantic queries. In case targets are not specified but needs to be resolved, the candidate requirement can be introduced as below:

1) *The oneM2M System shall support semantic queries that does not explicitly targets multiple M2M Nodes but processed to specifically targets multiple M2M Nodes.*

6.13 Advanced Queries with Priority

6.13.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

By Query with Priority Advanced Semantic Queries are meant where the time of the response matter. The positive result of a query also depends on temporal FILTER CRITERIA such as, TIME[SEC] or SPACE[METERS]. Queries with time constraints should be treated in priority, hence a simple mechanism of scheduling, deciding which query should be forwarded in priority, should be implemented in each CSE.

Summarizing the following potential requirements for the oneM2M system is advocated:

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].*

6.13.2 References to oneM2M Specifications

The **Result Expiration Timestamp** request parameter acts as the time constraint condition. By the parameter definition in clause 8.1.2 of [i.7], this timestamp indicates that the request receiver needs to return the result of the request before the time expires.

The **Event Category** request parameter defined in oneM2M TS-0001 [i.5], clause 8.1, plays as priority indicator. The predefined parameter value "immediate" refers the request is expected to be handled as top-priority. Other custom values can also be defined for an oneM2M system.

6.13.3 Elaborated oneM2M Requirements

Clause 6.4 captures temporal and spatial aspect for query conditions. ANY, CURRENT, PROVIDER terms are already supported by the resource identifier constructing rule in oneM2M TS-0001 [i.5], clause 7.2.

No new requirement needed.

6.14 Performance Requirements of the Advanced Discovery

6.14.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

By Performance Requirements the capability of self-measure the performance of the Advanced Semantic Discovery is meant. This capability can be advocated at any moment by any CSE that will, easily speaking, the "temperature" of the oneM2M Semantic Overlay Network. Those information will be useful to oneM2M Trusted Domain System Administrators to tune routing parameters intra and inter CSE.

Summarizing the following potential requirements for the oneM2M system is advocated:

1) Each CSE is able to register ASD network statistic and send it on demand to the respective IN-CSE.

6.14.2 References to oneM2M Specifications

Utilizing underlying network information in oneM2M overlay network is captured by existing requirements as below in [i.60], clause 6.6.

OPR-005	*The oneM2M System shall be able to exchange information with M2M Applications related to usage and traffic characteristics of M2M Devices or M2M Gateways by the M2M Application. This should include support for the 3GPP feature called: "Time controlled" (see note).*	*Implemented in Rel-2*
OPR-006	*Depending on availability of suitable interfaces provided by the Underlying Network the oneM2M System shall be able to provide information related to usage and traffic characteristics of M2M Devices or M2M Gateways to the Underlying Network.*	*Implemented in Rel-2*

6.14.3 Elaborated oneM2M Requirements

A rate limit is specified in oneM2M regarding subscription/notification feature. A certain rate can be set to limit the number of notifications in a certain time period. It is set by a resource subscriber. However, in this clause there may be dynamic rate control among CSEs.

1) The oneM2M System shall support semantic query rate control mechanisms referring past query history, for example.

6.15 Semantic Registration of Resources

6.15.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5 and 6.

The current oneM2M specification demands AEs to register in a CSE. Every AE will provide a description of their infrastructures (oneM2M TS-0034 [i.7]). This description is an RDF document expressed according to the oneM2M ontology (oneM2M TS-0012 [i.52]). The registration of an AE into a CSE is performed by sending the description to such CSE by means of a oneM2M primitive. Additionally, CSEs count with an endpoint that provides all the registered description.

Although the current CSE registration procedure fits the Advanced Semantic Discovery necessities, a new capability should be implemented in the CSE. In the Advanced Semantic Discovery the CSEs will be required to hold certain data of others CSEs in order to route a query, as well as, information about the AEs behind. For this purpose, CSEs should have the capability of summarizing the registered descriptions, so that summary could be the information that a CSE provides to another regarding its underneath AEs. Clause 5.6.5.1 of the present document presents several RDF summarization techniques, as well as, some examples.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *Semantic Registration of Resources is performed using the currentoneM2M primitives, as specified in oneM2M TR-0045 [i.51].*

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.51].*

3) *Semantic Registration of Resources implements a mechanism to allow accessing the registered semantic descriptors, as specified in oneM2M TR-0045 [i.51].*

4) *Semantic Registration of Resources summarizes semantic descriptors registered using any technique presented in clause 5.6.5.1.*

6.15.2 References to oneM2M Specifications

The descendantCSEs attribute of <remoteCSE> resource covers the requirement above partially. Descendant CSE information in an entity hierarchy, formed by entity registrations, is kept in the descendantCSEs attribute. However, the underneath AE information is not contained together.

Except the summary aspect, currently it can be realized with the <AE> resource announcement. An <AEAnnc> resource which is the partial copy of the <AE> resource basically can be hosted by IN-CSE for example which gathers all descendant CSEs information.

6.15.3 Elaborated oneM2M requirements

Protocol binding aspect of above requirements 1 and 2 in clause 6.15.1 is not requirement aspect and already supported by oneM2M. Also as they say, Requirements 3 and 4 are already covered. The remaining Requirement 5 is captured as below:

1) *The oneM2M System shall provide semantic data management capabilities in terms of aggregation, summarization, propagation, etc.*

6.16 Semantic Routing Table Upgrade

6.16.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3, 5.1, 5.6, 5.7, 5.10 and 6.4.

It is needless to say that faulty routing table implies faulty Advanced Semantic Routing. Therefore it is important to keep SRT as much up-to-date as possible, in order to feed correct data to the Semantic Recommendation system (SR). Nevertheless, by the intrinsic distributed nature of the Advanced Semantic Routing, each CSE have only a partial view of AE registered in other CSEs belonging to the oneM2M semantic overlay network. As an example consider that at given time, a number N of AE of type LIST_OF<FEATURE-TYPE> registers on a CSE: naturally the FEATURE-TYPE-Table will be updated with the new URI, one corresponding for each FEATURE-TYPE. This would suffice to keep fresh the SRT of the CSE but further NOTIFICATION messages should be send to all CSEs in the SDA-Table indicating that N new AE of type listing all the FEATUE-TYPEs that are now available in the current CSE. The outcome of those NOTIFICATIONS (once for each CSE in the SDA-table) will be a modification of the SRT of the connected CSE indicating the presence of N new AEs of type LIST_OF<FEATURE-TYPE>.

NOTE 1: In principle, those new arrivals to all the CSE using gossiping-like protocol techniques could be propagated, but this seems, as the moment too much expensive to be envisaged and would need to consider that each CSE would exactly contains the amount of information of all AE registered in the oneM2M semantic overlay networks.

NOTE 2: A summary of the AEs semantic descriptors in RDF can be synthetized from the SRT. The summary can be very narrow, like storing only the rdf:type statements, or very naive containing the whole semantic descriptors. In clause 5 RDF summarization techniques were provided, and thus, any of them could be potentially used. For the sake of this clause, from now on the assumption is, only the FEATURE-TYPE is stored in the SRT.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) Each CSE keeps its FEATURE-TYPE-table updated.

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.

6.16.2 References to oneM2M Specifications

oneM2M already supports subscription/notification mechanism. So when a resource contains semantic routing table(s), it can be updated by UPDATE operation and that update event can generate a notification for a <subscription> resource. See <subscription> resource and corresponding CRUD and notifications procedures in clauses 9.6.8 and 10.2.10 of oneM2M TS-0001 [i.5].

6.16.3 Elaborated oneM2M Requirements

As described in semantic routing table clause, routing information can be in RDF data format or not. The following requirements cover both aspects:

1) the oneM2M System shall provide overlay network capabilities to update semantic data (i.e. RDF triples) without a given SPARQL query but automatically composed one by a set of rules;

2) the oneM2M System shall support notification mechanism which can update data on a notification target M2M Node.

6.17 Advanced Semantic Resource Descriptors

6.17.1 Description of Potential Requirements for the oneM2M system

Potential requirements for the oneM2M system taken from ETSI TR 103 714 [i.1], clauses 3.1, 3.3 and 6.

The actual oneM2M specification demands AEs to submit an RDF document, named description, to a CSE in order to become semantic discoverable (oneM2M TR-0045 [i.51]). These descriptions should be expressed using the oneM2M ontology (oneM2M TS-0012 [i.52]), which covers generic terms for sensors or devices and allows specifying the AEs endpoints from which data can be fetched. Nevertheless, descriptors require a deeper specification of the AEs:

1) they should be able to refer to more specific terms, like those defined in SAREF;

2) descriptors shall contain information on how to interact with the data of the corresponding AE, since only accessing the data is not enough for automatic processing it;

3) descriptors shall contain information that allows third-party systems validate the fetched data. Additionally, since the descriptors are expressed in RDF, they shall be identified by differentiable URIs that are accessible by other oneM2M systems.

A sample of a semantic descriptor that fulfils the aforementioned requirements is the one presented in clause 5.3.4, that is the Web of Things Description presented as first example.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *oneM2M semantic descriptors are expressed in RDF, as specified in oneM2M TR-0045 [i.51];*

2) *oneM2M semantic descriptors are expressed following the Advanced Semantic Discovery Ontology;*

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;*

4) *Advanced Semantic Discovery Ontology contains domain specific terms (e.g. SAREF extensions);*

5) *Advanced Semantic Discovery Ontology contains information on how to interact with the data of the corresponding systems;*

6) *Advanced Semantic Discovery Ontology contains information on how to validate the data of the corresponding systems.*

A sample of a semantic descriptor that fulfils the aforementioned potential requirements for the oneM2M system from R1 to R7, with the exception of R5, is the Web of Things Description presented in clause 5.3.4 as first example.

6.17.2 References to oneM2M Specifications

Any resource identifiers in oneM2M is protocol agnostic. In HTTP binding, host server information will be concatenated with the resource identifier to generate a de-referenceable URI. Therefore <semanticDescriptor> resource identifier itself is not de-referenceable. See clause 6.2.2.1 of the oneM2M HTTP binding oneM2M TS-0009 [i.54].

In case of semantic validation, oneM2M supports semantic validation mechanism. The interfaces are illustrated in clause 6.9 of oneM2M TS-0034 [i.7].

6.17.3 Elaborated oneM2M Requirements

The oneM2M System shall support a specific ontology for semantic discovery or query.

6.18 Semantic Data Representation

6.18.1 Description of Potential Requirements for the oneM2M system

In oneM2M, two types of data are to be considered: the data that the AEs provide to the rest of oneM2M components within an IN-CSE, and the semantic descriptors currently used for discovery in oneM2M. The former type of data usually consists of data values captured by IoT infrastructures namely, whereas the latter type refers to meta-data about those IoT infrastructures. Currently, oneM2M forces the semantic descriptors to be expressed in RDF according to the oneM2M ontology; instead, there are a wider number of formats and models in which the data provided by the AEs can rely on.

Summarizing the following potential requirements for the oneM2M system are advocated:

1) *AEs can express their data using heterogeneous formats (JSON or XML), nevertheless, they also provide an RDF version of such data;*

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;*

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.51];*

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;*

5) *oneM2M data expressed in RDF have links that point to other RDF data URIs;*

6) *RDF data of oneM2M systems identifies every resource by a URI, as RDF specifies;*

7) *oneM2M systems publish RDF data using de-referenceable URIs (i.e. HTTP URIs);*

8) *RDF data of oneM2M systems shall be accessible through the oneM2M primitives, although inherent security mechanisms of oneM2M may prevent such data to be public available.*

Potential requirements for the oneM2M system R1 and R2 are addressed by discovery architectures presented in the previous clause 5.5.6, reader is referred to such clause in order to find examples. An example of how the oneM2M ontology and the SAREF ontology could be combined to represent data was presented in clause 5.2.3, reader is referred to that clause to find different examples. Examples of data fulfilling the potential requirements for the oneM2M R5 to R8 can be found in clause 5.2.3.

6.18.2 References to oneM2M Specifications

A <*semanticDescriptor*> resource can be the child of <AE>, <node> and other type resources. A list of possible parent resource types of the *semanticDescriptor* is defined in clause 9.6.1.1 of oneM2M TS-0001 [i.5]. When a semantic description as RDF format data is created, then it describes the parent resource referring a specific ontology indicated in the *ontologyRef* attribute. When the <AE> resource, or AE as the entity, has child resources for its devices or services, corresponding resources (e.g. a <container> resource) can also have its own semantic annotation as a child <*semanticDescriptor*> resource of that resource.

In the *semanticDescriptor* resource type definition in clause 9.6.30 of oneM2M TS-0001 [i.5], the *descriptorRepresentation* refers the serialization format of the RDF triples contained in the descriptor attribute. Among different <semanticDescriptor> resources they can refer each other with the *relatedSemantics* attribute.

6.18.3 Elaborated oneM2M Requirements

No new requirement needed since requirements in clause 6.18.1 are supported by oneM2M.

7 Advanced Semantic Discovery

7.1 Introduction

The present clause is based on the ETSI Technical Report ETSI TR 103 714 [i.1] (4 use cases and 17 potential requirements concerning an Advanced Semantic Discovery related Work Item on oneM2M).

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

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

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

It presents a high-level description of the mechanism designed to support Advanced Semantic Discovery and the main building blocks that will be part of the Advanced Semantic Discovery. It may contain figures and examples. The information contained are the basis to develop the detailed contribution to oneM2M in ETSI TR 103 717 [i.4] and such as could be written with precise references to the oneM2M elements and terminology.

It also presents a Low-Level (impact in the oneM2M specification) description of the main building blocks that will be part of the Advanced Semantic Discovery oneM2M extension and of the principal modifications to be done inside the oneM2M current release (e.g. new capabilities, new resources, new parameters, modifications to the existing one and on the oneM2M API). It may contain figures and examples.

7.2 Functional (high-level) description of oneM2M Advanced Semantic Discovery

7.2.1 Semantic Discovery Agreements

The rationale of SDA is rather simple: Advanced Semantic Discovery induces some traffic flow: each CSE is considered as an Autonomous System in BGP. As example, Figure 7.2.1-1, inspired by the oneM2M architecture (see oneM2M TS-0001 [i.5]), presents some CSEs with their respective abstract relationships.

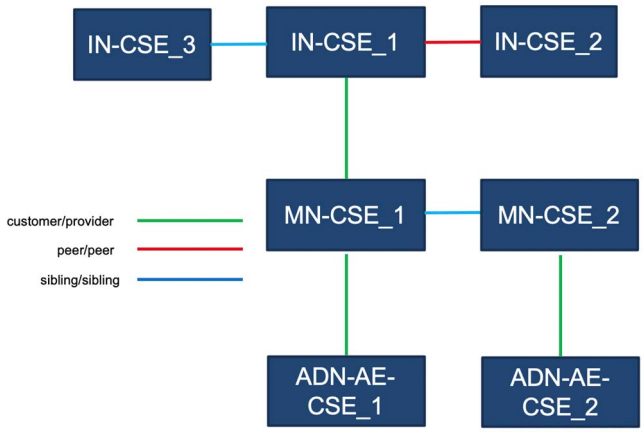


Figure 7.2.1-1: C2P and P2P and S2S CSE relationships

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

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

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

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

PROVIDER -> CUSTOMER -> PROVIDER

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

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

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

7.2.2 Semantic Non-Functional Issues

7.2.2.1 ASD Query with Priorities

As said in clause 6.13, AEs can generate Advanced Semantic Queries looking for some discovery TYPES and adding special FILTER CRITERIA, such as TIME[SEC] or SPACE[METERS]. When a CSE receives this query it should put it "on the top of the stack" and route before the others scheduled queries.

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

std::map<int,std::tuple<int,time_t,int>> schedulerMap; // <Id,<GateIndex,StartTime,MaxTime,Direction,>>

where:

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

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

7.2.2.2 Performance of ASD

As said in clause 6.14.1, every CSE should be able to self-measure the local performance of the Advanced Semantic Discovery. The CSE will produce a LOG file listing the following information (list not exhaustive):

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

- Peers CSEs.

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

7.2.2.3 Searching Multiple set of Targets

According to the requirement elicitation presented in clause 6.12, the Advanced Semantic Discovery Query (ASDQ) should be implemented according to the SPARQL 1.1 protocol, and the Advanced Semantic Discovery Query Language (ASDQL) should be implemented according to the SPARQL 1.1 Query Language. Following both SPARQL standards, the Advanced Semantic Discovery will be able of searcher over multiple set of targets; as explained in clause 6.12.

7.2.3 Semantic Discovery Query and Query Language

According to the requirement elicitation presented in clauses 6.4, 6.5 and 6.12, the Advanced Semantic Discovery Query (ASDQ) needs to be implemented using the well-known and established W3C SPARQL protocol. Additionally, as for the Advanced Semantic Discovery Query Language the W3C SPARQL query language can be adopted. Both standards meet the requirements listed in the aforementioned clauses; nevertheless the SPARQL protocol may be adjusted to meet the potential requirement number 4 for the oneM2M system of clause 6.4.

7.2.4 Semantic Discovery Routing and Resolution Mechanism

A "trace" of the Semantic Discovery Routing (SDR) generated by Example 1 of clause 6.6.1 is presented in Figure 7.2.4-1 (red lines represents request and reply Advanced Semantic routing paths from the AE that originates the complex query till the searched AE). The other examples in the same clause can be easily traced following the same logic. This trace is inspired to a semantic discovery routing as described in [i.24] and [i.25] and proceeds as follows:

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

ASDQ2 = ?T2|FC2

ASDQ3 = ?T3|FC3

ASDQ4 = ?T4|FC4

ASDQ5 = ?T5|FC5

1) A starts lookups in its local database, trying to solve {ASDQ 2,3,4,5} but fails;
2) A down-forwards ASDQ1 to Q via an mcc pointer;
3) Q solve the subquery ASDQ2 ?T2|FC2 in its local database returning Y to A;
4) A sends back Y to P and X;
5) A up-forwards ASDQ3 and ASDQ4 and ASDQ5 to B via an mcc' pointer;
6) B solves the ASDQ3 ?T3|FC3 in its local database returning Z to A (and back to P and X);
7) B side-forwards ASDQ4 & ASDQ5 to C;
8) C solves the ASDQ4 ?T4|FC2 in its local database returning V to B (and back to A, P and X);
9) C down-forwards ASDQ5 to D;
10) D solves the ASDQ5 ?T5|FC5 in its local database returning W to C (and back to B, A, P and X).

NOTE 1: When A up-forwards to B, it follows that A respect the CUSTOMER-to-PROVIDER SDA with B (e.g. A respects the SDA directives of B). When B side-forwards to C, it follows that B and C respect the PEER-to-PEER SDA directives. When C down-forwards to D, it follows that D respects the PROVIDER-to-CUSTOMER SDA with C (e.g. D respects the SDA directives of C).
The moral is: B and C should be "acknowledged" for their "routing job".

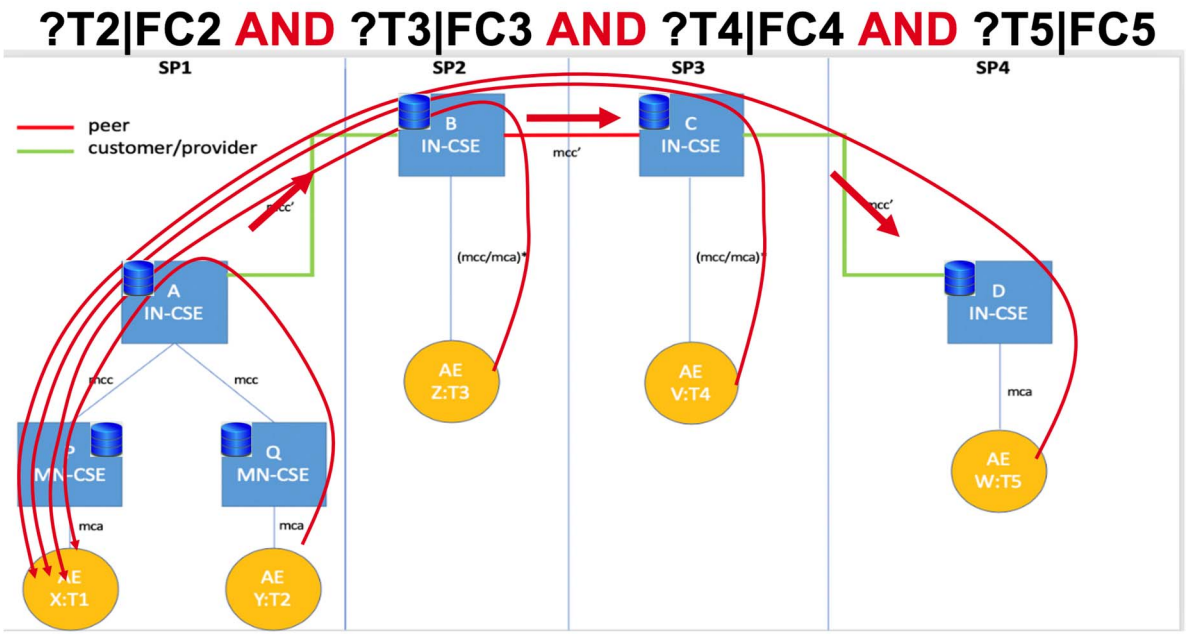


Figure 7.2.4-1: Advanced Semantic Discovery Routing Flow

In Figure 7.2.4-2, an alternative flow is represented. In this alternative topology the CUSTOMER-to-PROVIDER SDA are reversed.

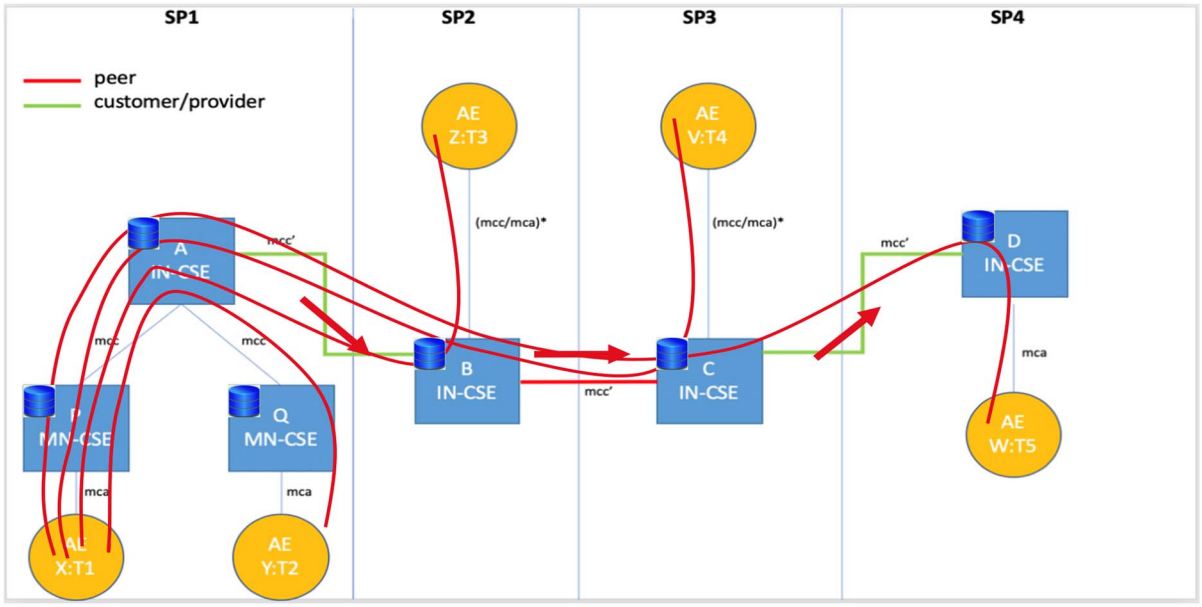


Figure 7.2.4-2: Alternative Advanced Semantic Discovery Routing Flow

A possible "trace" of the SDRM, again inspired to [i.22], [i.23], [i.19] and [i.24] proceeds as in clause 5.5 of ETSI TR 103 714 [i.1], excepting for the following caveat.

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

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

Alternative traces are possible but they should not be incentivized by the SDA in practice. Because of the distributed nature of the SDR, it is beneficial to try to incentivize routing respecting the SDA, and, as such, avoid routing not respecting the SDA. Those situations are not new in Internet and are referred as VALLEY ROUTING by Gao [i.22]. "Good Advanced Semantic Routing" in oneM2M should guarantee that routing is always "valley preserving" (or "no valley"). In the literature Valley routing property is also preserved in the network aware Resource Discovery Protocol of Liquori et al. [i.24] from which our protocol proposal is inspired.

Algorithmic description of the routing

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

1) It extracts the Advanced Semantic Discovery Query (ASDQ).

2) It reduces the Advanced Semantic Discovery Query (ASDQ) into a set of #m Semantic Discovery Queries (SDQ) by means of the Semantic Query Resolution System (SQRS).

3) It solve as much Semantic Discovery Queries (SDQ) as it can, using resources registered in the CURRENT CSE, say #n.

4) It forwards the remaining #m-n Semantic Discovery Queries (SDQ) to α-CSE Customer (in "downstream"), taken from the CURRENT Semantic Routing Table (SRT), resolving say #p.

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

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

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

NOTE 2: α, β, and γ are protocol specific parameters that can be locally modified in each CSE.

NOTE 3: if #r=#(m-n-p-q), then the query is exhaustive, and this is in contrast with non-exhaustive routing. Generally, two kinds of routing are discriminated, namely:

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

NOTE 4: A similar approach is described in [i.24].

7.2.5 Semantic Routing Tables

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

Table 7.2.5-1: Semantic Routing Table

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

Each CSE has a SDA Routing Table listing (aka, PHYSICAL Routing Table) for each CSE the number of CSE physically connected with their corresponding SDA, see Table 7.2.5-2 below.

Table 7.2.5-2: SDA-Table

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

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

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

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

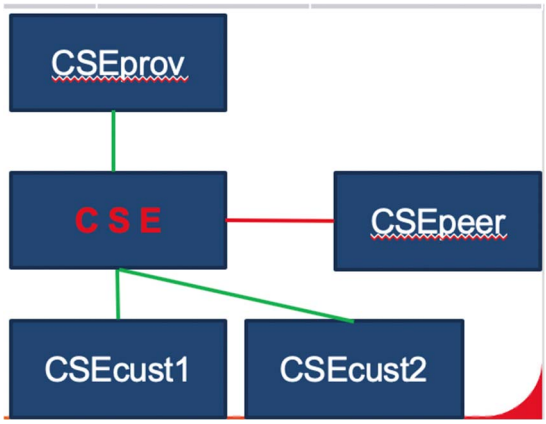
The previous Semantic Routing Table is implemented as an RDF document using an example ontology, i.e. ads, to describe its concepts. This routing table contains two entries, one for a peer CSE (http://cse1.com/002) and another one for a customer CSE (http://cse1.com/003). For those entries the table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

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

7.2.6 Semantic Routing Tables Upgrade and Propagation

Each time a resource register inside a CSE, this should send a SEMANTIC UPGRADE message to all neighbours CSEs. As example, when y news AE-THERMOMETER with URI_1 ... URI_y register on that CSE, the Semantic Routing Table will be upgraded (in red the upgrades) as in Table 7.2.6-1.

Table 7.2.6-1: Upgrading a Semantic Routing Table with y new AE-THERMOMETERS



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

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

NOTIFY CSEcust1 WITH #+y THERMOMETER

NOTIFY CSEcust2 WITH #+y THERMOMETER

NOTIFY CSEpeer WITH #+y THERMOMETER

NOTIFY CSEprov WITH #+y THERMOMETER

When an adjacent CSE will receive the NOTIFY message, its routing table will be modified shown in Table 7.2.6-2 (in yellow the upgrades).

Table 7.2.6-2: Upgrading the adjacent SRT CSEs with the new y AE-THERMOMETERS

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

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

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

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

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

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

will be sent to all adjacent CSEs.

7.2.7 Semantic Recommendation System

As descripted in clause 6.8.1 by Semantic Recommendation system (SR) a mechanism is meant, embedded in each CSE, that, upon a reception of an Advanced Semantic Query from a registered AE or from another CSE, should be able to:

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

The solution proposed to implement is inspired to the Kademlia P2P overlay network [i.50].

Table 7.2.7-1: SRT with Recommendation System

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

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

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

1) IF the CSE answers successfully to an Advances Semantic Query for that TYPE and already is in the BUCKET[TYPE], then move it to the tail of the list;

2) OTHERWISE:

   a) IF the BUCKET[TYPE] has fewer than 100 entries, then inserts the news CSE at the tail of the list;

3) OTHERWISE:

   a) ping the least-recently seen CSE;

   b) IF the least-recently seen CSE fails to respond, THEN it is evicted from BUCKETS[TYPE] and the new CSE is inserted at the tail;

4) OTHERWISE it is moved to the tail of the list, and the new CSE is discarded.

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

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

7.2.8 Semantic Ontologies (baseline and domain specific)

According to the requirement elicitation presented in clause 6.11 oneM2M systems rely on the oneM2M ontology (as specified in oneM2M TR-0045 [i.51] and oneM2M TS-0012 [i.52]), nevertheless this ontology falls short for performing Advanced Semantic Discovery.

For this end, the Advanced Semantic Discovery will count with a bespoke ontology, the Advanced Semantic Discovery Ontology.

7.2.9 Semantic Data Representation

According to the requirement elicitation presented in clause 6.18, all data related to the Advanced Semantic Discovery should be expressed in RDF and follow the Linked Data principles. Additionally, RDF translation techniques SHOULD be used in order to provide an RDF version of the data values belonging to oneM2M systems that rely on heterogeneous non-RDF formats.

The RDF data can be modelled according to the Advanced Semantic Discovery Ontology. Since this new ontology is designed as an extension of the current oneM2M ontology, using this new ontology will not require any extension task of the current oneM2M ontology, as required now in oneM2M TS-0012 [i.52].

7.2.10 Semantic Registration of Resources

According to the requirement elicitation presented in clause 6.15, resources perform a semantic registration process as described in the current oneM2M TR-0045 [i.51]. Additionally, all registered semantic descriptors are available as specified in oneM2M TR-0045 [i.51]. Finally, semantic descriptors registered should be summarized using any of the techniques presented in clause 5.6.4.1.

7.2.11 Semantic Resource Descriptors

According to the requirement elicitation presented in clause 6.17, semantic descriptors are expressed in RDF according to the Advanced Semantic Discovery Ontology. Furthermore, these semantic descriptors include information on how to interact with the oneM2M systems that they describe, and also, how to validate the data retrievable from the oneM2M systems. Although no validation mechanism is intent to occur during the discovery phase.

Finally, semantic descriptors can be accessible, as specified in oneM2M TR-0045 [i.51], but they are also identified by a de-referenceable URIs and follow the Linked Data principles.

7.3 Candidate solution (low-level) for a oneM2M Advanced Semantic Discovery

7.3.1 Semantic Discovery Agreements

Since semantic discovery/query agreements are made between CSEs, who performs discovery/query, during the CSE mutual registration relationship information needs to be agreed and recorded in <remoteCSE> resources. This would be used to determine the target of semantic fan-out discovery/query depending on the relationship type.

This agreement is also has impacts on access control. With the agreement, if a CSE receives an advanced semantic query, for example, the CSE performs the local semantic query for its semantic data. The data could be aggregated data or summarized data of resources on other CSEs. When the CSE access that semantic data there would be access grant due to a specific semantic discovery agreement.

As well as the authorization aspect, this agreement concept is utilized during semantic query fan-out which is different from the existing one briefly suggested in clauses 7.3.2.3 and 7.3.3. Depending on the relationship per agreement, a subset of the fragmented query can be fanned out to a CSE.

7.3.2 Semantic Non-Functional Issues

7.3.2.1 ASD Query with Priorities

There could be new *Event Category* parameter values added for semantic queries. The existing values for the parameter is like "immediate" or "bestEffort" which can be applied for any type of oneM2M requests. However, compared to the ordinary request handling, there might take longer time to handle SPARQL queries, it seems to be correct to have semantic query specific event category values, if needed. Then new values can be chosen by an intermediary CSE during semantic query fragmentation for fan-out.

Along with the new *Event Category* parameter values, the CSE who handles the semantic query request at the first time, can choose optimal *Request Expiration Timestamp* value. Since this timestamp is serves as an indicator for receiving CSEs as the lifetime of requests, when the event category is set high and the timestamp gets expired soon, the CSEs would handle the request in a high priority.

7.3.2.2 Control of ASD forwarding

Underlying network congestion information can be gathered over Mcn reference point and be used by a CSE for semantic query routing. How to leverage that information for performance would be specified as an implementation choice.

On the contrary, as the overlay network, oneM2M platforms can set limits for semantic query fan-out from/to other CSEs. The limits can be written in a resource per number of fan-out, specific SPARQL terms, specific targeted CSEs for example. This limit as configuration information can be shared during CSE registrations.

7.3.2.3 Searching Multiple Set of Targets

The SPARQL query federation can be realized based on group fan-out mechanism, especially with the *semanticFanoutPoint* virtual resource and its handling procedure. Depending on the semantic resource type, it could existing <semanticDescriptor> or a new to meet the requirements from by the requirements, the semantic fan-out point can leverage the new semantic resources. Behavioural difference between the SPARQL protocol and oneM2M is, in oneM2M case, aggregated RDF triples are queried at a single CSE while SPARQL queries are executed and aggregates the results. oneM2M could adopt the SPARQL way if needed. Then issue on access control for distributed triples on different CSEs can be relieved.

The limitation on group fan-out extended mechanism is that the same query is applied for multiple targets. The query fragmentation would derive different fragments, not the same one, for different targets. To cope with this, there should be a feature to fragment a semantic query per target by looking up its semantic routing table.

7.3.3 Semantic Discovery Query and Query Language

oneM2M already supports SPARQL execution over semantic triples pertaining to specific <semanticDescriptor> resources and this is done by <semanticFanoutPoint> resource which is the virtual child resource of a <group> resource. Since the existing mechanism can apply the same semantic query over multiple targeted semantic data, in this candidate solution, differently fragmented query targeting multiple targets can be executed and the results can be summed-up. To support this, the query language needs to support relevant terms, parameters which can be used during query fragmentation.

7.3.4 Semantic Discovery Routing and Resolution Mechanism

SPARQL query fragmentation can be introduced and it could be done by looking up the semantic routing tables. Unlikely the oneM2M request forwarding, the semantic discovery routing herein would be interpreted as a sort of fan-out mechanism or initiating subsequent requests.

When a CSE receives a semantic discovery/query request, the CSE as the request target resolves the query. If the query cannot be completely resolved by the CSE or requested to search for multiple CSEs by an indication, the CSE refers the semantic routing table and initiate necessary new semantic query requests to other CSEs. The new requests could include a different SPARQL query from the original one in case there is a need for query fragmentation. This would be also be decided by the information given in the routing table.

7.3.5 Semantic Routing Tables

A new resource type of semantic triple data could be suggested for semantic routing tables. Because a routing table needs to provide necessary information of routing targets, table information would consist of summary, for example, of distributed triples in different CSEs which is minimal information for semantic query routing.

Hierarchy information, from the oneM2M registration, that is used for oneM2M request forwarding can be augmented with summarized triples. This means there would be SPARQL query for semantic query routing (finding a next hop for forwarding) than just simple string match for ordinary routing implementation.

Even if the routing table consist of non-semantic data, a dedicated new resource type seems to be needed for CSE to handle semantic query fan-out by looking-up this table.

7.3.6 Semantic Routing Tables Upgrade and Propagation

Since the table suggested in the previous clause is based on RDF triples, table updates or propagation between CSEs can rely on SPARQL. As suggested alternatively, in case that non-semantic routing data forms the tables, update or propagation can rely on existing oneM2M subscription/notification mechanism between CSEs.

Irrelevant to the table data format, there would be a new resource type for this routing table maintenance. The resource type represents its functionality with routing information update condition, update target, update query, etc. Then whenever routing relevant information from the source CSE gets updates it can be propagated to another CSE which handles the semantic query routing.

7.3.7 Semantic Recommendation System

The semantic recommendation system in this present document consumes semantic data in the semantic routing table to choose a right target CSE during semantic routing. To have an optimal selection, there can be a semantic rule expression which can be composed by applications or administrators depending on system policies. This semantic rule can be embedded in the semantic query or can be pre-set in a CSE which can be re-used for different semantic queries.

When a routing table data is written as non-semantic (i.e. not RDF triples), a corresponding rule description can be set to choose the right routing target.

7.3.8 Semantic Ontologies (baseline and domain specific)

oneM2M already allows usage of any ontologies including oneM2M base ontology, ETSI SAREF and their extensions. For example, in oneM2M Release 4, there has been a study for smart city ontologies which does not necessarily based on oneM2M base ontology.

There is the need for a new and specific ontology for advanced semantic discovery/query in this present document. This ontology would provide better interoperability as the standard ontology. This fills the gap of existing standard ontologies, providing necessary aspects for discovery/query, so it can be used by semantic annotation in the semantic routing table.

7.3.9 Semantic Data Representation

Semantic data, which is RDF triples, is already supported by the *semanticDescriptor* resource type. Linkage for the <*semanticDescriptor*> resources is supported, too. Except the routing aspects, which are suggested as new resource types in clauses 7.3.5 and 7.3.6, no new solution is required for semantic data representation extension.

7.3.10 Semantic Registration of Resources

Overall this can be covered by clauses 7.3.5 and 7.3.6. Semantic descriptors can be aggregated or summarized in one CSE so it can be used for other purposes including the semantic query routing.

7.3.11 Semantic Resource Descriptors

As turned out in clause 6.17.3, no new extended features for the existing semantic descriptor resources.

8 Lessons learned and conclusions

The objective of the present document [i.2] is the study of an advanced semantic Discovery and associated Query capabilities in oneM2M and the development of the associated solution in the oneM2M standard (see the oneM2M specifications in clause 2.2).

To that purpose, 4 use cases and 18 potential requirements for the oneM2M system have been defined that emphasizes the needs for an extension of oneM2M for an 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. This work is documented in ETSI TR 103 714 [i.1].

From these requirements a solution of an Advance Semantic Discovery is proposed in the present document by considering the query and discovery mechanisms already available in oneM2M and by proposing some extensions when needed in order to implement and Advanced Semantic Discovery solution in oneM2M.

At this stage of the work, a solution is proposed that will be evaluated in terms of performances and that will provide a proof of concept of this solution (see ETSI TR 103 716 [i.3]).

Annex A:
Change History

Date	Version	Information about changes
March 2020	0.0.1	TR Skeleton derived from the ETSI TR template, with fill in descriptions deleted, structure of clauses 4 to 8 adapted to oneM2M template for input contributions on T2 task. Structure discussed during Task 2 meeting #1 of 23-03-2020
March 2020	0.0.2	Adaptation of TR Skeleton to the ETSI TR standard rules, discussed during the Task 2 point to point meeting with STF leader of 25-03-2020
March 2020	0.1.0	Early Draft version for review by TC SmartM2M
May 2020	0.1.1	Stable Draft structure
June 2020	0.1.2	Stable draft structure after T2 meeting#4 and partial contributions UPM &INRIA
June 2020	0.1.3	Early Draft version for review at TC SmartM2M #54
June 2020	0.1.4	Integration of contribution in clause 5 after T2 meeting #7
June 2020	0.1.5	Integration of contribution in clause 5 after T2 meeting #8
June 2020	0.1.4	Integration of contribution in clause 5 after T2 meeting #7
July 2020	0.1.7	Integration of clause 6 structure after meeting #9
July 2020	0.1.8	Integration of clause 7 and integrations of UPM Keti and INRIA contributions
July 2020	0.1.9	Integration of contributions from INRIA and alignment of the document format with ETSI TR writing rules
July 2020	0.3.0	Stable Draft version for review by TC SmartM2M
July 2020	0.3.1	Filled with text: clauses 5.3.2.4, 5.5.4.6, 6.4.1, 6.5.1, 6.11.1, 6.12.1, 6.15.1, 6.17.1 and 7.2.2.3
July 2020	0.3.2	Updated: clauses 2.1, 3.1 and 3.3, editorial modifications according to ETSI drafting rules
August 2020	0.3.3	Version updated after meeting #10
August 2020	0.3.4	Version after integration of Keti, INRIA and TIM contributions
August 2020	0.3.5	Version after integration of UPM contributions
August 2020	0.3.6	Version after integration of clauses 6 and 7 from Keti and INRIA comments for clause 6
August 2020	0.3.7-> 0.5.0	Suite of versions result of 5 ad hoc meetings asked by INRIA (T2 rapporteur) to KETI and UPM
August 2020	0.5.3	Final Draft version for approval by TC SmartM2M #55
September 2020	1.1.1	TB Approval ok. ETSI Technical Officer review before submission to ***editHelp!*** the remarks in rev-marks + comments to process during the resolution of other ***editHelp!*** remarks to come

History

Document history		
V1.1.1	November 2020	Publication