Internet of Things Architecture

IoT-A

Project Deliverable D4.3

Concepts and Solutions for Entity-based Discovery of IoT Resources and Managing their Dynamic Associations

Project acronym: IoT-A
Project full title: The Internet of Things Architecture
Grant agreement no.: 257521

Doc. Ref.: D4.3
Responsible Beneficiary: UniS

Editor(s): Suparna De (UniS)

List of contributors:
Suparna De (UniS), Gilbert Cassar (UniS), Benoit Christophe (ALBLF), Sameh Ben Fredj (ALBLF), Martin Bauer (NEC), Nuno Santos (NEC), Tobias Jacobs (NEC), Ricardo de las Heras (TID), Gregorio Martin (TID), Gerd Völksen (SAG), Andreas Ziller (SAG)

Reviewers: Ebru Zeybek (CATTID)

Contractual Delivery Date: February 2012
Actual Delivery Date: March 2012
Status: Final

Version and date Changes Reviewers / Editors
V1.02, March 16, 2012 - / - - / -

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

<table>
<thead>
<tr>
<th>Dissemination Level</th>
<th>PU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public</td>
<td></td>
</tr>
<tr>
<td>Restricted to other programme participants (including the Commission Services)</td>
<td></td>
</tr>
<tr>
<td>Restricted to a group specified by the Consortium (including the Commission Services)</td>
<td></td>
</tr>
<tr>
<td>Confidential, only for members of the Consortium (including the Commission Services)</td>
<td></td>
</tr>
</tbody>
</table>
Executive Summary

Finding Virtual Entities and IoT Services is an essential aspect for a comprehensive Internet of Things. A suitable infrastructure needs to be in place that allows resolution, look-up and discovery of IoT Services and the proper abstraction level needs to be provided to application programmers. Therefore, we provide look-up and discovery on two abstraction levels, the level of the IoT Services themselves, e.g., a service providing a noise level or a service detecting the motion of people, and the entity level, where virtual entities representing physical entities can be discovered, e.g., the noise level in a room or the motion of a certain user. Due to the mobility of physical entities and devices and other dynamic changes, the relations between IoT Services and Virtual Entities may also change over time. Therefore, we investigate how new relations can be automatically found and how the validity of existing ones can be monitored.

The deliverable consists of two main parts. In the first part, we present more details on the reference architecture view. In the second part, we detail different approaches for look-up, discovery and finding new Associations between Virtual Entities and IoT Services.

The first part includes the high-level data models, the functional components from the functional view that are relevant to discovery, look-up and name/identifier resolution and the interfaces between them. The relevant data models are the Resource Model, modelling all the aspects that are directly related to the resource and the hosting device, the Service Model that provides information regarding the interface and how to access the service, the Entity Model that describes relevant aspects of a Virtual Entity, and finally the structure of Associations between Virtual Entities and IoT Services. Next, the functional components IoT Service Resolution, VE Resolution and VE & IoT Service Monitoring that have been identified in the Functional View of the IoT-A Architectural Reference Model are detailed regarding their functionality and their interfaces. The IoT Service Resolution is responsible for discovery and look-up of IoT Service descriptions, as well as name/identifier to locator resolution for IoT Services. The VE Resolution supports discovery and look-up of Associations. Finally, the functionality of the VE & IoT Service Monitoring component is to find new Associations to be added to the VE Resolution, thereby making them discoverable for Users of the IoT infrastructure. As Associations may become invalid, e.g., as a result of the mobility of Devices and Physical Entities, their validity has to be monitored by the VE & IoT Service Monitoring.

In the second part different approaches for realizing the look-up and discovery functionalities of the functional components are presented, as well as approaches for finding new Associations and monitoring the validity of existing Associations. Beyond the basic architectural aspects, this deliverable details mechanisms, protocols and algorithms required for the implementation of the functionality in different settings and under different assumptions. The first approach looks at how geographic location can be used for discovery and how this can be efficiently implemented using geographic index structures. Furthermore, geographic proximity plays an important role for finding new Associations between IoT Services and Virtual Entities. The second approach looks at using Semantic Web technology for the discovery and look-up in the IoT Service Resolution and the VE Resolution. Furthermore, it is shown how Semantic Web rules can be used for finding new Associations. The third approach presents a federation-based approach using semantic clustering for discovery and look-up. The fourth approach uses a Peer-to-Peer infrastructure for the look-up of Associations and Service Descriptions, whereas discovery can only be supported in a limited way based on keywords. Finally, a domain-based approach supporting discovery and look-up of IoT Service Descriptions and Associations is presented.

This deliverable still addresses the reference architecture level, discussing different approaches and different options for the approaches. As a next step, we will work on integrating the different approaches in a specific architecture instance. This will serve as a basis for a concrete implementation to be used within the IoT-A project. Naturally this will mean selecting options
and making specific assumptions, but also gives the opportunity for further evaluations albeit in a much more specific IoT setting.

**Progress of the Work**

This deliverable builds on the D4.1 (R. Heras & Santos, 2012) deliverable, which, among other things, discussed different architectural approaches for realizing the discovery, look-up and name/identifier resolution. This deliverable provides an update on the modelling, the functional components related to WP4 and their respective interfaces. Going beyond the architectural approaches, this deliverable discusses more detailed aspects like mechanisms, protocols and algorithms and shows how the functional components and their interfaces can be implemented based on the respective approach.

**Results Beyond State-of-the-Art**

In this deliverable we provide details on different approaches for realizing the discovery, look-up and resolution functionalities of an IoT infrastructure. We also show approaches for automatically finding new Associations between Virtual Entities and IoT Services, which we have not seen in any existing IoT system. The provided functionalities are based on the IoT-A information model and specifically the Service Model and the Entity Model. A previous version of the latter models has been published in a paper on “Service modelling for the Internet of Things” (De, Barnaghi, Bauer, & Meissner, 2011).

**Results Reported vis-a-vis Overall Architecture**

The results presented in this deliverable concern the detailed aspects of the architectural options for implementing the IoT Service Resolution, the VE Resolution and the VE & IoT Service Monitoring functional components as identified in IoT-A Deliverable D1.2 (Walewski, 2011). The work will serve as a basis for implementing an instance of the resolution infrastructure (D4.4), which will then be used in the use cases of WP7. Also the results provide feedback to WP1 for the next iteration detailing the Architectural Reference Model (D1.3), especially for the functional decomposition and the interactions and interfaces required in an overall IoT system.

**Role and Positioning of Deliverable in Overall Project**

The deliverable provides a further milestone towards the objective of developing “a novel resolution infrastructure for the IoT, allowing scalable look-up and discovery of IoT Resources, Entities and their Associations”. It discusses different approaches and aspects on how to build such an infrastructure in different settings. Concerning the more detailed WP4 objectives, the deliverable addresses the objectives O4.2 (finding and look-up of IoT Services), O4.3 (finding IoT Services based on Virtual Entities), O4.4 (mechanisms for discovery based on physical world aspects) and O4.5 (Finding and monitoring Associations).
# Table of Contents

Executive Summary ........................................................................................................... - 1 -
List of Figures ..................................................................................................................... - 5 -
1.   Introduction .................................................................................................................. - 7 -
2.   Data Models and Functional Components ................................................................... - 8 -
   2.1 Data models ................................................................................................................ - 8 -
       2.1.1 Resource Model ................................................................................................. - 8 -
       2.1.2 Service Model .................................................................................................... - 10 -
       2.1.3 Entity Model ....................................................................................................... - 11 -
       2.1.4 Associations ....................................................................................................... - 13 -
   2.2 Functional Components and Interfaces..................................................................... - 14 -
       2.2.1 IoT Service Resolution (I: TID, NEC) ................................................................. - 15 -
       2.2.2 VE Resolution .................................................................................................... - 19 -
       2.2.3 VE & IoT Service Monitoring ............................................................................ - 23 -
3.   Discovery and Association Methodologies .................................................................. - 26 -
   3.1 Geographic Locations Approach ............................................................................. - 26 -
       3.1.1 Discovery Mechanisms ....................................................................................... - 26 -
       3.1.2 Association Methods .......................................................................................... - 32 -
       3.1.3 Conclusions ......................................................................................................... - 38 -
   3.2 Semantic Web Approach ........................................................................................... - 38 -
       3.2.1 Discovery Mechanisms ....................................................................................... - 39 -
       3.2.2 Association Mechanisms .................................................................................... - 49 -
       3.2.3 Conclusions ......................................................................................................... - 59 -
   3.3 Federation-based Approach ....................................................................................... - 60 -
       3.3.1 Federation based architecture ............................................................................. - 60 -
       3.3.2 Discovery Mechanisms ....................................................................................... - 63 -
       3.3.3 Association Mechanisms ..................................................................................... - 70 -
       3.3.4 Concluding remarks ............................................................................................ - 75 -
   3.4 P2P Infrastructure – DHT Approach ......................................................................... - 76 -
       3.4.1 Hash Tables ......................................................................................................... - 76 -
       3.4.2 Hash Functions .................................................................................................... - 77 -
       3.4.3 Distributed Hash Tables ...................................................................................... - 78 -
       3.4.4 Peer-to-Peer Routing .......................................................................................... - 79 -
       3.4.5 Further Peer-to-Peer Features ............................................................................. - 80 -
       3.4.6 Assessment of DHT-based P2P Infrastructures .................................................. - 81 -
       3.4.7 Implementation Aspects ...................................................................................... - 81 -
       3.4.8 IoT Service Resolution ....................................................................................... - 82 -
       3.4.9 IoT Service Discovery ......................................................................................... - 83 -
       3.4.10 Virtual Entity Resolution ................................................................................... - 84 -
       3.4.11 Managing Subscriptions ................................................................................... - 84 -
       3.4.12 Conclusions ........................................................................................................ - 85 -
   3.5 Domain-based Approach ............................................................................................ - 85 -
       3.5.1 Introduction .......................................................................................................... - 85 -
       3.5.2 Directory Servers ............................................................................................... - 86 -
       3.5.3 Master Index catalogue ....................................................................................... - 88 -
       3.5.4 Discovery mechanisms ....................................................................................... - 89 -
       3.5.5 Lookup mechanisms ......................................................................................... - 92 -
       3.5.6 Management functions ....................................................................................... - 94 -
4.   Conclusions .................................................................................................................. - 96 -
5.   References ..................................................................................................................... - 98 -
6.   Terminology and Glossary ............................................................................................ - 100 -
List of Figures

Figure 1 Service Related part of Domain Model ................................................................. - 8 -
Figure 2 Resource Model ...................................................................................................... - 9 -
Figure 3 Service Model ......................................................................................................... - 10 -
Figure 4 Entity Model ........................................................................................................... - 12 -
Figure 5 Structure of Associations ................................................................................... - 14 -
Figure 6 Functional Component IoT Service Resolution in the functional view of IoT reference architecture D1.2 ........................................................................................................ - 15 -
Figure 7 Conceptual internal structure of IoT Service Resolution ................................ - 18 -
Figure 8 Functional Component VE Resolution in the functional view of IoT reference architecture ......................................................................................................................... - 19 -
Figure 9 Conceptual internal structure of the VE Resolution ........................................... - 22 -
Figure 10 Functional Component VE & IoT Service Monitoring in the functional view of IoT reference architecture ......................................................................................................... - 23 -
Figure 11 Conceptual internal structure of the VE & IoT Service Monitoring .................. - 25 -
Figure 12 Figure 7 Mercator map projection of the world (http://en.wikipedia.org/wiki/File:Mercator_projection_SW.jpg) .................................................................................. - 27 -
Figure 13 Example of a VE-Resolution Spatial Query ................................................................ - 28 -
Figure 14 R-Tree representation of a VE-Resolution Spatial Query ................................ - 29 -
Figure 15 Example of an IoT-Services-Resolution Spatial Query ...................................... - 31 -
Figure 16 Dynamics of Spatial Relations between Physical Entities and Devices ................. - 35 -
Figure 17 Discovering new Associations ........................................................................... - 37 -
Figure 18 IoT-A Service instance example .......................................................................... - 39 -
Figure 19 Abstraction of Service description in terms of Latent Factors .......................... - 41 -
Figure 20 IoT Service Clustering ......................................................................................... - 42 -
Figure 21 Ontology Example and parameters calculation .................................................. - 45 -
Figure 22 Example of Sensor Ontology .............................................................................. - 46 -
Figure 23 Overview of the Discovery Process ....................................................................... - 48 -
Figure 24 Indoor location model (edges denotes relations between terms) ....................... - 50 -
Figure 25 Relations between “Place”, “Region” and “IndoorPoint” concepts ...................... - 51 -
Figure 26 Defining an indoor point by translating a GPS coordinate .................................. - 52 -
Figure 27 Computing coordinates of a resource (the same would work for an entity) ........ - 53 -
Figure 28 Types of Region one can create ......................................................................... - 54 -
Figure 29 Associations along thematic, spatial, temporal axes ......................................... - 55 -
Figure 30 Spatial Matching ................................................................................................. - 57 -
Figure 31 Federation based architecture as a “polytree” with all non source nodes having an in-degree equals to 1 .................................................................................................................. - 61 -
Figure 32 Clustering in each Vertex of the Federated Architecture .................................. - 64 -
Figure 33 Example of IoT-A services and their related concepts ...................................... - 64 -
Figure 34 SimMatrix for 8 services ....................................................................................... - 65 -
Figure 35 Hierarchical clustering of services ...................................................................... - 65 -
Figure 36 Routing table construction in each vertex ............................................................. - 66 -
Figure 37 Request forwarding in the federated architecture ............................................. - 68 -
Figure 38 Executing rules when a resource/entity joins/leaves a place then updating triple store of closed nodes .................................................................................................................. - 70 -
Figure 39 A path between two nodes, using Federated approach ..................................... - 73 -
Figure 40 Sharing Associations between nearby nodes: Overall process ............................. - 74 -
Figure 41 Illustration of a Hash Table Storage Mechanism ................................................ - 78 -
Figure 42 Distributed Hash Table ...................................................................................... - 79 -
Figure 43 The Distributed Hash Table Architecture of the Chord Protocol ....................... - 80 -
Figure 44 Basic Structure of Entry Types .......................................................................... - 82 -
Figure 45 Inserting a service description ............................................................................. - 83 -
Figure 46 Updating a service description .......................................................................... - 83 -
Figure 47 Deleting a service ............................................................................................... - 83 -
Figure 48 Service description lookup .................................................................................. - 83 -
Figure 49 Resolving a service URL ..................................................................................... - 83 -
1. Introduction

One of the key elements of an Internet of Things reference architecture is a framework that facilitates finding the IoT Services that expose resource functionalities and provide access to them. Furthermore, finding resources that are relevant for interactions with any particular physical world object is a key precursor to achieving the IoT vision. In the context of the IoT-A project, this translates to finding the IoT services that can be associated to Virtual Entities (VE). This link between a VE and an IoT service that allows interaction with it, along some VE attribute, is termed as ‘Association’ within the scope of IoT-A. A VE is the digital representation of a real world Physical Entity (PE). The IoT can mediate the interaction between a User and a PE based on a VE and associated IoT services that expose resource functionality hosted on devices providing some form of physical access to the PE.

Three key functionality components identified in the IoT-A architectural reference model are IoT Service Resolution, VE Resolution and VE and IoT Service Monitoring. Resolution frameworks are used for resolving queries to IoT services that expose the capabilities of a particular resource (or resource type) or provide information on associations between VEs and IoT Services. Resolution in IoT-A has been identified as involving three main functionalities: discovery, lookup and name/ID resolution. Thus, the term ‘resolution’ has been used in both a narrow and a wide sense. In a narrow sense, it is used for mapping names or identifiers to locators or addresses through which the IoT services can be accessed. In the wider sense, as used in the terms IoT Service ‘Resolution’, it encompasses the three functionalities as previously stated. For VE Resolution, the discovery and lookup functionalities are supported. IoT Service Resolution is concerned with providing the service descriptions for selecting and accessing the services. Associations that have been asserted between VEs and IoT Services that provide information/actuation capabilities for some attribute of the VE can be known through the VE Resolution component. VE and IoT Service Monitoring looks at finding new associations and monitoring those already in place.

The previous deliverable from Work Package 4 (WP4) of the IoT-A project, D4.1 (R. Heras & Santos, 2012), has looked at investigating the issues of identification and addressing in the heterogeneous world of IoT and presented the architectural design options for the resolution and discovery infrastructure. Following on from there, this current deliverable delves into details of those architectural design alternatives. It is aimed at detailing the mechanisms, protocols and algorithms to realize the functionality components of IoT Service Resolution, VE Resolution and VE and IoT Service Monitoring. Additionally, the interfaces and structures of these functionality components will also be presented.

This document is structured into 3 main sections. Section 2 presents the data models that form the basis of the discovery, lookup and association mechanisms. It also details the IoT-A functional components introduced in the functional view that implement the Resolution and Discovery infrastructure, namely the IoT Service Resolution, Virtual Entity Resolution and VE and IoT Service Monitoring, together with their respective interfaces and structures. The core part of the deliverable consists of the description of the different options with their respective mechanisms and concepts that have been identified for the implementation of look-up and discovery, which is presented in Section 3. Section 4 provides some conclusions. Section 6, following the references in section 5, provides an overview of the important terminology used in the IoT-A project and which is relevant for the understanding of this deliverable.
2. Data Models and Functional Components

The Discovery Interface is a key element of the Resolution and Discovery Infrastructure being developed in IoT-A. In this section we look at the modelling of the elements to be discovered, how the architecture of the infrastructure looks like and finally how to design and describe the discovery interface.

2.1 Data models

The Domain Model (Walewski, 2011) presents a UML representation of the key actors in the IoT domain. With an entity forming the main focus of interactions by humans and/or software agents, its digital representation, i.e. the Virtual Entity, is a key component in the Domain Model as shown in Figure 1 below. The service aspects are captured by the Service model presented in Section 2.1.2, elaborating on the relationships with Virtual Entities and Resources. As shown in Figure 1, services are associated to Virtual Entities but they expose resource functionalities and provide a standardised access mechanism to the resource capabilities.

It is worth noting that the purpose of these models is to present a structure for the respective information and relationships. Though the visualisation leans towards a RDF/OWL representation, especially while showing the links to other modelled concepts, this is not a technology decision of the IoT-A project.

2.1.1 Resource Model

A resource is the software component of a device that provides information on the entity or enables controlling of the device. The device in turn either attaches to or is part of the environment of an entity so it can monitor it. The device allows the entity to be part of the digital world by mediating the interactions. Figure 2 details the Resource Description Model.
The resource concept has datatype properties that specify its name (hasName), an ID (hasResourceId) and time zone URI (hasTimeZone). A Resource Description Framework (RDF) (W3C, 2004b) realisation of the model can specify the name property through rdfs:label and the hasResourceId property through the rdf:ID property. The resource provider can specify certain keywords describing the resource through the hasTag property. This is an optional property to allow the resource provider to provide a free text search for the resource instance. A resource also has a location property (hasResourceLocation) that links to the Location that links to a modelled WGS-84 Location concept. This location could be the location of the device the resource runs on. The cardinality restriction on the ‘hasResourceLocation’ property is either 0 or 1. When the resource is of type storage and hosted somewhere in the network cloud, specifying location through geographical information does not apply, hence the ‘0’ cardinality. For the other resource types, the ‘exactly 1’ restriction denotes that a resource can only have a link to one location instance. The definition of the location concept is similar to that in the entity model, as specified in the following section. The resource type is denoted in terms of the type property (hasType) to the ResourceType concept. Resources can be instances of either of the following types: sensor, actuator, RFID tag, storage or processing resource. The different resource types are not disjoint. A semantic

---

1 http://earth-info.nga.mil/GandG/wgs84/index.html
realisation can map the ‘hasType’ property to ‘rdf:type’. Then, a semantic engine can, for instance, infer that a resource asserted to be of type sensor is also of type actuator. When the type is a sensor, the hasType property serves as a link to an instance of a sensor that conforms to an available sensor model (e.g., SSN sensor ontology). This allows linking the resource concept to external models which already define in detail related concepts, without the need of repeating them in the Resource Model.

As the access to a resource is provided by an IoT Service, this link to the service is denoted by the ‘isExposedThroughService’ object property that links the resource model to an IoT Service instance of the IoT-A service model. The resource model also captures the link to the hardware ‘device’ on which the resource is hosted by defining a property ‘isHostedOn’, binding the Resource Model to a Device Model (whose implementation could be a Device ontology).

2.1.2 Service Model

The Service Model proposed here reflects the service related aspects of IoT-A’s domain model. The Service Model contains information needed for discovering and looking up the service as well as information on how to invoke the service. The service model is shown in Figure 3.

![Figure 3 Service Model](image)

The actual technology used to invoke the service is modelled through the hasServiceType parameter, which could take a value such as ‘REST’ for a RESTful Web Service. The link to the resource to which the service provides access is specified through the exposes property that links back to an instance of the Resource Model.

One of the important aspects of a Service is to allow for Associations with Virtual Entities in the IoT domain. In a semantic realisation, the IoT-A proposed service model could utilise the existing standard OWL-S (W3C, 2004a) model as its upper ontology. The ‘ServiceModel’ part of the OWL-S ontology is used to specify the input, output, preconditions and effects (IOPE) related parameters of the Service Model. Since the Service Model exposes the underlying resource’s functionalities, the resource attribute that is exposed through an IoT Service either as output data type (hasOutput) or as an input parameter (hasInput) is captured in the service specification. The feature can then be matched with the attribute type of the Virtual Entity with which it can be associated. For instance, a Virtual Entity can have an attribute that represents its ‘indoorTemperature’. The generic type of this particular attribute is ‘temperature’. Then, if there is a service exposed by a resource that measures temperature, specified as the service’s hasOutput parameter, the corresponding service can be a candidate for a possible Association to the relevant Virtual Entity. The input and output parameters can be specified in terms of the
generic instance quantities from the QU ontologies (SySML), such as ‘temperature’ or ‘luminosity’.

For actuating services, the state of the entity attribute being controlled is also important. This post-condition state is modelled through the hasEffect parameter in the Service Model. Similarly, any pre-conditions that need to be met before the service execution can be specified through the hasPrecondition parameter. An example scenario illustrating the use of these two properties could be the case of a composite service that requires an atomic service that is able to satisfy the precondition defined in its service specification. An atomic service that has exactly the same condition specified in its hasEffect property would be a candidate for the composite service. For instance, a ‘price display service’ displays the price of an item in a store on an Electronic Shelf Label. The label is an actuator resource and can be configured to display different numbers. The service is triggered only when the price of an item placed on the shelf has been updated, thus requiring the precondition ‘priceUpdated = true’ to be satisfied. A ‘price update service’ would have the required condition (priceUpdated) specified as effect in its description and its execution would set the condition to true. Thus, the effect of the ‘price update service’ matches the precondition of the ‘price display service’.

The state object properties link to instances of the ‘Condition’ class of the SSN ontology (SSN, 2011), so that conditions that affect the resource’s measurement or actuating capabilities can be specified. This is also an example where the SSN ontology concepts could be extended to include actuating conditions.

With location being an important criterion for service search and resolution, the area affected by the service is specified through the hasServiceArea property. For sensing services, this would be the observed area, while actuating services would specify the area of operation. The observation area of sensors can be different to their actual location. An example for that are camera resources observing areas at some distance to their position. The service area could be mapped to a location defined in the Location Model. The possibility of specifying time constraints on service availability is captured through the hasServiceSchedule property. A semantic instantiation of the ServiceSchedule concept could, for instance, be a DurationDescription borrowed from W3C’s Time Ontology\(^2\).

2.1.3 Entity Model
The IoT-A Information Model (Walewski, 2011) describes a Virtual Entity in terms of an identifier, a type and attributes, with the attributes further including metadata information for attribute type and value. In addition to these properties, an entity can have certain other aspects that need to be taken into account. For example, we may need to know about the location of an entity and the features that can be observed by a sensing mechanism to provide data about the observed feature. A diagram of the main attributes of the IoT-A Entity Model is shown in Figure 4.

\(^2\) http://www.w3.org/TR/owl-time/#duration
An entity has certain features, which include domain attributes, temporal features and location. A particular entity instance can have either or all of these features. Moreover, an entity instance can have multiple values for the domain, temporal or location feature. Domain attributes tie the entity instance to a particular domain. A semantic realisation of the model would for instance, link the entity instance to a domain ontology. The domain attribute is specified in terms of the attribute name (hasAttributeName), attribute type (hasAttributeType) and value. These attribute properties together describe an observable feature of the entity. Following the Information Model, having the attribute name and type as distinct properties allows for two levels of data specification. The DomainAttribute instance's name property refers to the domain specific attribute of the Virtual Entity, e.g. Ambient Temperature. What a resource (e.g. sensor) will be able to measure will be the attribute type, i.e. Temperature, in this case. Thus, for two distinct domain attributes of the same Virtual Entity, e.g. Ambient Temperature and Body Temperature, what a resource would be concerned with, would be the attribute type, i.e. Temperature, which is the same for both domain attributes. Only the domain attribute property, which is intrinsic to the entity, puts what the resource senses, into context. This is evident from the Entity Model, where the domain attribute name is specified as a string,
whereas the attribute type could link to other models, for instance, in the case of a semantic realisation, a vocabulary of physical phenomena, such as temperature or acceleration, which can be measured by sensors or influenced by actuators. The value itself has a literal ‘value’ and associated metadata information (ValueMetadata). The metadata could include information on, for instance, the units of measurement. It is specified in terms of the metadata value and metadata type. Where the metadata specifies the units of measurement, the metadata type could, for instance, refer to a vocabulary describing different possible measurement units.

Temporal features of a Virtual Entity are specified by the time zone, hasAvailabilityDate and hasAvailabilityTime properties. These capture the temporal properties of entities that may have temporal attributes, e.g., Meeting Rooms. The hasAvailabilityTime property is specified in terms of the TimeRange concept (which has start and end time properties) and the hasAvailabilityDate property links to the DateRange concept (in terms of start and end date). In a Semantic Web realisation of the proposed model, the values of these properties could be compared with other dates by using date and time comparison built-ins (such as those available in Semantic Web Rule Language (SWRL) (Horrocks et al., 2004)) to deduce facts about temporal aspects of the relevant entity.

The entity location is defined in terms of the geographical coordinates (hasLatitude, hasLongitude, hasAltitude). The location concept also has properties that could link to global (hasLocationURI) and local location (hasLocation) models. The local location model provides detailed location description, such as rooms and buildings on a campus. To specify the global location, an instantiation of the Entity Model could specify a URI from existing standards such as GeoNames (GeoNames, 2011), that models well-known location aspects such as cities, districts, countries and universities.

Additionally, an entity has datatype properties that specify the URI of an owner (hasOwner) where an example instantiation of the URI could point to a standardised vocabulary for modelling relations to persons, such as foaf ((http://www.foaf-project.org/docs) profile; a literal name (hasName) and a Boolean property to denote if the entity could be mobile (isMobile). An important attribute of an entity is the entity type (hasType). Typical examples for entity types could be person, place, room, car, table or pallet and they could also be structured in a type hierarchy, e.g., an office may be a specialization of a room with additional attributes. In a Semantic Web realisation such as that using RDF, the name can be specified through the rdfs:label property and the type by the ‘rdf:type’ property. Specifying the entity type through rdf:type property would allow a Semantic Web engine to infer the type of the entity from its asserted properties, especially in cases where the entity could have multiple types. The local identifier (hasLocalIdentifier) property is the ID of the virtual entity. It could as well point to a local naming schema and be specified through the rdf:ID in a RDF realisation of the entity model. The global identifier (hasGlobalIdentifier) property is a placeholder to associate the entity to other globally referenceable models. For instance, an Entity Model instance could link its hasGlobalIdentifier property to the open Linked Data (http://linkeddata.org) platform; for instance, to a Dbpedia (http://dbpedia.org/) entry.

### 2.1.4 Associations

An Association relates a Virtual Entity to an IoT Service. This means that the IoT Service provides certain functionality with respect to an aspect of the Physical Entity that is modelled as a Virtual Entity in the digital world. The aspect of the Physical Entity is modelled as a Domain Attribute of the Virtual Entity, following the Entity Model presented in Section 2.1.3. The IoT Service may provide information about the Physical Entity, which corresponds to reading the Domain Attribute of the Virtual Entity, or the IoT Service may enable actuation on the Physical Entity, which, in the simplest case, may correspond to setting the value of the Domain Attribute.

In more complex cases, the effect of the IoT Service on the Physical Entity modelled by the Virtual Entity has to be specified in more detail. This means that the Association also has to contain the Domain Attribute and what the IoT Service can do with respect to it.

Figure 5 shows the logical structure of an Association. It contains a description of the Virtual Entity, the IoT Service and the VEServiceDescription that describes what the IoT Service provides with respect to the Virtual Entity.
The VEServiceDescription contains the Domain Attribute and a serviceType attribute that specifies whether the service provides information or enables actuation. Optionally, the VEServiceDescription can contain a description of the serviceCapabilities, which may explain further what the service can do, e.g., increase the attribute value up to a certain maximum.

The Virtual Entity contains additional aspects of the Entity Model that may be relevant when it comes to the discovery of Associations like the location of the modelled Physical Entity and the entity type.

**Figure 5 Structure of Associations**

### 2.2 Functional Components and Interfaces

In this Section we discuss the three functional components as identified in D1.2 (Walewski, 2011) that are part of the Resolution Infrastructure developed in this work package. The IoT Service Resolution provides the name/identifier resolution, resource description look-up and discovery. The VE Resolution provides the look-up and discovery on the abstraction level of Virtual Entities. The latter allows finding the IoT Services that can provide certain information about a Virtual Entity or allow a certain actuation on the Physical Entity that corresponds to the Virtual Entity. The VE & IoT Service Monitoring functional component is used to dynamically discover Associations between Virtual Entities and IoT Services. These Associations can then be registered with the VE Resolution component. As the validity of the Association may depend on dynamically changing aspects like the location of a Physical Entity or a device, the VE & IoT Service Monitoring component will monitor these aspects and remove the respective Association from the VE Resolution once it is no longer valid.
2.2.1 IoT Service Resolution (I: TID, NEC)

The IoT Service Resolution (Figure 6) is a functional component which provides all the functionalities needed by the User in order to find and be able to contact IoT Services. On the other hand, the IoT Service Resolution also gives Services the capability to manage their service descriptions, so they can be looked up and discovered by the User. The User can be either a human user or a software component.

The functionalities needed in brief are:

- **Discovery functionality** finds the IoT Service without any prior knowledge about the ServiceID. The functionality is used by providing a service specification as part of a query.
- **Lookup** is a functionality which enables the User to access the service description having prior knowledge regarding the ServiceID.
- **Resolution function** resolves the ServiceIDs to locators through which the User can contact the service.

Other functionalities provided by the IoT Service Resolution are the management of the service descriptions. IoT Services can update, insert or simply delete the service descriptions from the IoT Service Resolution component. It is also possible that these functions are called by management components and not the IoT Services themselves.

### 2.2.1.1 IoT Service Resolution Interface

In the IoT-A, the services are identified with a unique identifier but these identifiers cannot be used for network routing. Thus they have to be mapped to an address or URL, and this task is performed by the `resolveService` function. This address or URL can be either on the same device which provides the Resource, or it can be hosted elsewhere.

Resolution function enables the user of the system to have the address information which enables him to contact the service of interest. The User has prior knowledge of the unique ServiceID and, upon the entry of this information, IoT Service Client calls the Resolution component which returns the URL of the service, respectively, which is associated to the given ID.

Upon the successful return of the URL, User can contact the Service.
resolveService(ServiceID):ServiceURL

Whereas the ServiceID should remain the same over the lifetime of the service – if it changes we may also interpret this as a new service – the point of network attachment and thus the address and the ServiceURL may change during the lifetime of the service, e.g., due to mobility of the device or the (sub-)network. Thus Users that require the service may subscribe to the resolution based on the ServiceID and receive notifications whenever the ServiceURL changes. The IoT Service Resolution returns a SubscriptionID, which can be used to uniquely identify the subscription and to map notification, which include the SubscriptionID to the respective subscription.

subscribeServiceResolution(ServiceID, notificationCallback):SubscriptionID

The User then has to implement the callback functionality, which is available through the provided notification callback:

notifyServiceResolution(SubscriptionID, ServiceURL)

For the management of the subscriptions the respective unsubscribe functions are needed:

unsubscribeServiceResolution(SubscriptionID)

There are scenarios when the User knows the ServiceID of a needed Service and would like to have the ServiceDescription and contact information of the Service which is defined as a part of the ServiceDescription. LookupService function is called by the IoT Service Client to answer the User’s query input, The returned result is the ServiceDescription. The assumption is that the ServiceID is a prior knowledge of the user. Upon the entry of the ServiceID into the interface, the returned result is the full description of the associated service.

lookupService(ServiceID):ServiceDescription

As important aspects of the service description may change over time, the User can also subscribe for the service description of relevant services based on the ServiceID. An important change may also be that the service is no longer available, which can be indicated by a special ServiceDescription containing the ServiceID and a “no longer available” status. Again, a unique SubscriptionID is returned the subscribing User that can be used to match notifications to the subscription.

subscribeServiceLookup(ServiceID, notificationCallback):SubscriptionID

The User then has to implement the callback functionality, which is available through the provided notification callback:

notifyServiceLookup(SubscriptionID, ServiceDescription)

For the management of the subscriptions an unsubscribe function is needed:

unsubscribeServiceLookup(SubscriptionID)
In the IoT Service Resolution functional component, the discovery of the available services based on ServiceSpecification and the ServiceDescription of the matched service results are listed in an array.

The discoverService function enables the User to make a query based on specification of a needed service and without any prior ID knowledge. The search for new or unknown services may be based on certain functionality, geographical range etc., or other service specifications.

Eg: User A would like have a functionality at a geographic range P.

Service Client calls the discovery component and after the query evaluation, the related results are returned as an array of ServiceDescription. The query has the following signature:

\[
\text{discoverService(ServiceSpecification): ServiceDescription[ ]}
\]

As the availability of services fitting the specification may change over time, the User can also subscribe for continuous notifications about services that fit the ServiceSpecification. Again the special case that a service is no longer available can be handled by providing a special ServiceDescription as explained above. When first subscribing, a notification with all currently fitting services will be sent. Further notifications will only contain changes, i.e., new services, changed service descriptions of existing services or service no longer being available.

\[
\text{subscribeServiceDiscovery(ServiceSpecification, notificationCallback):SubscriptionID}
\]

The User then has to implement the callback functionality, which is available through the provided notification callback:

\[
\text{notifyServiceDiscovery(SubscriptionID, ServiceDescription[ ])}
\]

For the management of the subscriptions an unsubscribe function is needed:

\[
\text{unsubscribeServiceDiscovery(SubscriptionID)}
\]

In the IoT-A reference model, services are registered in the IoT Service Resolution with a ServiceDescription, and internally these are stored in the Service Description storage. The necessity of the management of the service description in the Service Description storage is solved by defining functionalities “Update Service, Insert Service and Delete Service”.

i. Update Service

As changes occur in the ServiceDescription parameters, the related information in the Service Description Registry must be updated by the IoT Service to enable continuous service availability. The interface enables access to update the service functionality; the ServiceDescription argument includes the ServiceID:

\[
\text{updateService(ServiceDescription)}
\]

For the cases where devices or resources have changed their physical location, URL/addresses are included in the ServiceDescription so automatic update of ServiceDescriptions will enable continuous service availability.

ii. Insert Service

This interface has the functionality to add a new service description to the IoT Service Resolution. The IoT Service Resolution will then return the ServiceID assigning a new ServiceID, if no ServiceID is given in the ServiceDescription. This unique ID is used by the architectures beneath the discovery, lookup and resolution functionalities for an efficient IoT Service Resolution.

\[
\text{insertService(ServiceDescription):ServiceID}
\]

iii. Delete Service
The ServiceDescription of the Service with the ServiceID is deleted from the Service Description Registry.

\[ \text{deleteService(ServiceID)} \]

### 2.2.1.2 IoT Service Resolution Structure

Figure 7 shows the conceptual internal structure of the IoT Service Resolution functional component. It is meant to show the different aspects that have to be handled for the IoT Service Resolution – it does not mean that an actual implementation will have exactly these components, e.g. some components may be integrated with each other. Also, this conceptual view does not take any distribution or federation aspects into account. Depending on the setting, multiple components may be needed to implement one of the sub-components shown in the figure. These aspects will be detailed in Section 3, where specific approaches and options for discovery, look-up and name/identifier resolution will be discussed.

![Figure 7 Conceptual internal structure of IoT Service Resolution](image)

The IoT Service Resolution has an external interface for supporting name/identifier resolution, look-up and discovery. For a User, these functionalities should be accessible through a single access point, but of course there could be any number of such access points. The request is then analysed and dispatched to the fitting functional components. There may be different types of discovery that can be used, e.g. based on a geographic scope or semantic similarity. Possibly results from different functionalities may also be aggregated. The solid lines show the interactions between components to fulfil User requests. The discovery, look-up and resolution functionalities may not always store the service descriptions themselves. They need certain information, e.g. for building index structures, but from there they may point to the storage of the actual service description. This may also have the advantage that the final access control regarding who is allowed to access a certain service description could remain with the provider of the service.
The information needed for name/identifier resolution may be stored with the service descriptions, but for efficiency reasons, there may also be a Resolution DB that stores the mapping from name/identifier to locator, possibly implemented using an already existing service like DNS.

An interface for the management of service descriptions is also needed that implements the functionalities for inserting, updating and deleting service descriptions. Service descriptions are stored in the Service Description storage, but discovery, look-up and name/identifier resolution components may also need to be informed about all or parts of the service descriptions, e.g. to update their respective index structures that allow them to efficiently find the required service descriptions. The dashed lines show the necessary interactions for the management of service descriptions.

### 2.2.2 VE Resolution

![Figure 8 Functional Component VE Resolution in the functional view of IoT reference architecture](image)

A Virtual Entity (VE), which is uniquely identified, is a digital representation of a Physical Entity (PE). IoT Services expose Resources, which may provide information about a PE or enable an actuation on the PE. There may be more than one resource enabling the interaction with PE. An Association is the relation between a particular IoT Service that exposes a Resource, and a particular Virtual Entity. It contains a description of the Virtual Entity, the IoT Service and the VEServiceDescription that describes what the IoT Service provides with respect to the Virtual Entity. For example, a room and a temperature service may be related through the Association (e.g., modelled as an attribute) indoorTemperature. The Association would contain the Virtual Entity ID of the room, the entity type room, the attribute indoorTemperature, and the identifier of the IoT Service.

The VE Resolution (Figure 8) is the functional component which provides the functionalities to the IoT User to retrieve Associations between Virtual Entities and IoT Services. The functionalities needed by the Service Client in brief are:

- **Discovery functionality** discovers the Associations without any prior knowledge about the Virtual Entity. The Virtual Entity specification and the VEServiceSpecification, which describes the relation between the Virtual Entity and the IoT Service, are used as parameters of the query.

- **Lookup** is a functionality which enables the User to access Associations between the particular Virtual Entity and IoT Services fitting the VEServiceSpecification based on a known VE-ID uniquely identifying a Virtual Entity.
Those Associations are identified uniquely with an identifier upon their insertion to the information base by the insert functionality of VE Resolution functional component. The AssociationID is important for the efficient lookup and discovery functions. The other functions of the component are update and delete to manage the Association updates or deletion in the information base, respectively.

2.2.2.1 VE Resolution Interface

An Association is the relation between a VE-ID and a Service Identifier and is described by the attribute name and additional information about how the attribute of the entity is being associated with the service i.e. in case of sensor the associated service returns the information of the attribute, in case of actuator service changes the value of the attribute. When the IoT Service Client needs to find the Association based on the known (specific/particular) Virtual Entity and associated Service specification, lookup operation in the VE Resolution functional component enables the user to retrieve the Association for this specific Virtual Entity.

lookupAssociations is a function called by the IoT Service Client with the assumption that Virtual Entity identity VE-ID is known by the User. The other parameter in the lookupAssociations is VEServiceSpecification. The VEServiceSpecification contains the attribute of the Virtual Entity with which the required service needs to be associated and potentially other information, i.e., if the value of the attribute should be returned by the service or if the service should influence this value as in the case of actuation. For example actuation services change properties of entities from an initial state to a desired state. The Virtual Entity Resolution looks up fitting Associations based on the VE-ID and the VEServiceSpecification and provides the resulting array as the return value. The User can select the needed Service and use the IoT Service Resolution to have the contact information of it.

\[ \text{lookupAssociations}(\text{VE-ID}, \text{VEServiceSpecification}): \text{Association[]} \]

As valid Associations may change over time, the User can also subscribe for Associations based on the VE-ID and the VEServiceSpecification. An important change may also be that an Association is no longer valid, which can be indicated by a special Association containing the AssociationID and a “no longer available” status.

\[ \text{subscribeAssociationsLookup}(\text{VE-ID}, \text{VEServiceSpecification, notificationCallback}): \text{SubscriptionID} \]

The User then has to implement the callback functionality, which is available through the provided notification callback:

\[ \text{notifyAssociationLookup}(\text{SubscriptionID, Association[]}[]) \]

For the management of the subscriptions an unsubscribe function is needed:

\[ \text{unsubscribeAssociationLookup}(\text{SubscriptionID}) \]

For the scenarios where VE-ID is not known by the User, discoverAssociations function is called by the Service Client to enable the User still to retrieve the Associations between Virtual Entity and services. The search is based on the Virtual Entity and Service specifications.

VESpecification defines the Virtual Entity to be associated with the service defined in the VEServiceSpecification. Upon the entry of required Virtual Entity specification and the specification of the associated service parameters, the IoT Service Client calls the discovery component and the Associations that fit to the research criteria are returned in an array.

\[ \text{discoverAssociations}(\text{VESpecification, VEServiceSpecification}): \text{Association[]}[] \]
The returned result is a list of Associations. Those Associations must satisfy both the fit of Virtual Entity to the VESpecification and the description of the attribute name of this Virtual Entity in the VEServiceSpecification parameter of the discovery. The IoT Service is a part of the Association that is identified by the ServiceID. Using the ServiceID, as a parameter the User can look up the service description from the IoT Service Resolution component to have the contact information of the requested IoTService.

As the set of fitting Associations may change over time, the User can also subscribe for continuous notifications about Associations that fit the VESpecification and the VEServiceSpecification. Again the case that an Association is no longer valid can be handled by providing a special Association containing the AssociationID and a “no longer available” status. When first subscribing, a notification with all currently fitting Associations will be sent. Further notifications will only contain changes, i.e., new Associations, changed Associations or Associations no longer being available.

```
subscribeAssociationDiscovery(VESpecification, VEServiceSpecification, notificationCallback):SubscriptionID
```

The User then has to implement the callback functionality, which is available through the provided notification callback:

```
notifyAssociationDiscovery(SubscriptionID, Association[ ])
```

For the management of the subscriptions an unsubscribe function is needed:

```
unsubscribeAssociationDiscovery(SubscriptionID)
```

The VE Resolution component stores Associations in its Associations storage. The Associations are inserted, updated or deleted by the IoT Service, the VE& IoT Service Monitoring component or any other component. These functions are operated through the defined interfaces “Insert Association, Update Association and Delete Association”

i. Insert Association

Upon the call of the insertAssociation function, the association between the service identifier and virtual entity identifier, and the description of this association by the attribute name and additional information, is inserted into the Associations storage. This Association is uniquely identified with the AssociationID and the ID is returned as a result of the operation.

```
insertAssociation(Association):AssociationID
```

ii. Update Association

The update of the Association is provided by the function updateAssociation. Upon the call of the function by the IoT Service or VE& IoT Service Monitoring component, the Association is updated and the information for the lookup and discovery functions is available to the User.

```
updateAssociation(Association)
```

iii. Delete Association

If the Association is not valid anymore the related information must be deleted from the Associations storage of the VE Resolution component. The deleteAssociation function enables the deletion of the Association with the deletion of the assigned unique AssociationID.

```
deleteAssociation(AssociationID)
```
2.2.2.2 VE Resolution Structure

Figure 9 shows the conceptual internal structure of the VE Resolution functional component. It is meant to show the different aspects that have to be handled for the VE Resolution. Again, the figure does not imply that an actual implementation will have exactly these components, e.g., some components may be integrated with each other. Depending on the setting, multiple components may be needed to implement one of the sub-components shown in the figure. These aspects will be detailed in Section 3, where specific approaches and options for discovery and look-up of Associations will be discussed.

![Diagram of VE Resolution Structure]

The VE Resolution has an external interface for supporting look-up and discovery of Associations. For a User, these functionalities should be accessible through a single access point, but of course there could be any number of such access points. The request is then analysed and dispatched to the fitting functional components. There may be different types of discovery that can be used, e.g., based on a geographic scope or semantic similarity. Possibly results from different functionalities may also be aggregated. The solid lines show the interactions between components to fulfil User requests.

The discovery and look-up functionalities may not always store the Associations themselves, but need the required information, e.g., for building index structures. This may also have the advantage that the final access control regarding who is allowed to access a certain Association could remain with whoever inserted the Association into the Association storage. This could be the owner of the IoT Service or some third-party, or event, the VE & IoT Service Monitoring component. On the other hand, unlike service descriptions which may contain a lot of information, Associations are small, so a direct access through the look-up and discovery may be more efficient. The respective trade-offs have to be evaluated for a concrete system architecture.

An interface for the management of Associations is also needed that implements the functionalities for inserting, updating and deleting Associations. Associations are stored in the
Associations storage, but discovery and look-up components may also need to be informed about all or parts of the Associations, e.g. to update their respective index structures that allow them to efficiently find the Associations. The dashed lines show the necessary interactions for the management of Associations.

### 2.2.3 VE & IoT Service Monitoring
The VE & IoT Service Monitoring functional component (see Figure 10) is responsible for automatically finding new Associations, which are then inserted into the VE resolution functional component, where it is stored in the Associations storage. New Associations can be derived based on existing Associations, service descriptions and information about Virtual Entities. Different approaches for doing this are discussed in the respective subsections of Section 3. We differentiate between static Associations that typically do not change and dynamic Associations that are expected to change. An example for a static Association is the Association between a Room and an IoT Service providing the temperature from a stationary temperature sensor mounted to the wall of this room, measuring the indoor temperature of this room. A dynamic Association could be between a Room and a Service providing the noise level that is connected to a sensor in a mobile phone that is currently located in the Room. In the case of dynamic Associations, the conditions that lead to the creation of the Associations have to be monitored to determine whether the Association is still valid, e.g., if the location of the Physical Entity and the service area of the service that provides an attribute value still overlap. If these conditions no longer hold, the Associations will be deleted from the VE resolution functional component. It is also possible that aspects of the Association change, e.g., that the location of the Virtual Entity has changed.

![Functional Component VE & IoT Service Monitoring in the functional view of IoT reference architecture](image)

**Figure 10 Functional Component VE & IoT Service Monitoring in the functional view of IoT reference architecture**

It will not be feasible that the VE & IoT Service Monitoring always checks all possible combinations of Virtual Entities and services for potential Associations. Therefore, a target set of Virtual Entities and IoT Services has to be specified. This can consist of specific Virtual Entities and Services giving their identifiers or based on specifications, e.g. including the respective types.
Such a target set can be specified explicitly, e.g., through configuration, or implicitly based on the requests that are received by the VE Resolution functional component. New Associations may be discovered pro-actively, i.e. VE & IoT Service Monitoring by itself tries to find Associations which may be needed by Users, which can be done in the background. Alternatively, VE & IoT Service Monitoring could be explicitly triggered by VE Resolution for a current request for which no suitable Association could be found. Our focus will be mainly on the former, but of course the latter will also be considered, taking into account the delay that this may induce for a request.

2.2.3.1 VE & IoT Service Monitoring Interface
The following interface is internal to the VE & IoT Service Monitoring functional component (see Section 2.2.3.2 for subcomponents), but the operations reflect the important functionalities of the functional component and are listed here for better understanding.

assertStaticAssociation(Association): AssociationID
A new static Association has been found that is to be inserted into the VE resolution functional component. As the Association is static, no monitoring of conditions is necessary.

discoveredDynamicAssociation(Association): AssociationID
A new dynamic Association has been found that is to be inserted into the VE resolution functional component. As the Association is dynamic, the VE & IoT Service Monitoring will monitor the conditions that are required for the Associations to be valid.

associationNo LongerValid(AssociationID)
Dynamic Associations are monitored by the VE & IoT Service Monitoring functional component whether the conditions that lead to the creation of the Association still hold. If this is no longer the case, the associationNo LongerValid operation is called with the AssociationID as its parameter. As a result, the Association will be deleted from the VE resolution functional component.

associationUpdate(Association)
Dynamic Associations are monitored by the VE & IoT Service Monitoring functional component. If aspects of the Association change, e.g. the location of the VE, that do not invalidate the Association, the Association is updated.

2.2.3.2 VE & IoT Service Monitoring Structure
Figure 11 shows the conceptual internal structure of the VE & IoT Service Monitoring functional component. The main functionalities are Discover Static Associations, Discover Dynamic Associations and Monitor Validity of Associations.

The purpose of the Discover Static Associations functionality is to discover new static Associations, i.e. Associations between Virtual Entities and IoT Services that do not change or rarely change, so their validity does not have to be constantly monitored. Discover Dynamic Associations is for discovering Associations that may dynamically change and become invalid, e.g., because of mobile Physical Entities or Devices changing their location. Thus, the newly discovered Associations have to be monitored by the Monitor Validity of Associations functional component, to avoid that invalid Associations continue to be stored in the VE Resolution functional component.

The three discussed functionalities use the Update Associations functionality to interact with the VE Resolution functional component for inserting new Associations, updating aspects of existing Associations and deleting Associations that have become invalid.
The Monitor Validity of Associations functionality uses three functionalities for monitoring base information that can be used for discovering new Associations. The Monitor Associations functionality retrieves existing Associations that may contain important information about Virtual Entities like their location and about the relation to services that may be used for relating the service to other Virtual Entities. For example if a room is associated with a sensor providing information like the noise level, the same service may be associated to any person currently in the room, providing their ambient noise level. Associations can also be used to find IoT Services that provide additional information about specific Virtual Entities.

The Monitor IoT Service Descriptions functionality retrieves information about services from the IoT Service Resolution functional component, e.g. the service area for which the service is provided. can be used for associating the Service with Virtual Entities whose physical counterparts are located in this area.

Finally, the Monitor VE information functionality is responsible for accessing additional information that may be required for discovering new Associations or for determining whether an Association is still valid.

As mentioned in the introduction, it may not be feasible for the VE & IoT Service Monitoring functional component to discover all possible Associations in the systems as the required monitoring would be too expensive. As also discussed, there are different options to determine what kind of Associations should be discovered and derived from that what information needs to be monitored. One possibility is to explicitly configure this. For this purpose, an optional configuration interface is shown. As this configuration interface may be very specific to the approach used, e.g. a rule language may be used to specify rules or a list of Virtual Entities with attributes of interest may be provided, we have not specified this interface on a general level as we have done for the interfaces of the other functional components.
3. Discovery and Association Methodologies

An important enabler for the Internet of Things is a resolution and discovery infrastructure that can handle a very large number of data sources (represented by IoT Service descriptions and inferred Associations between Virtual Entities and IoT Services) as well as the inherent heterogeneity of possible descriptions. Another aspect is the creation of dynamic Associations between IoT Services and Virtual Entities to allow applications to identify actions to take in response to sensed changes in the environment. With the goal of multi-domain, multi-vendor interoperability, the infrastructure must structure the search space for both effectiveness and efficiency.

The previous WP4 deliverable D4.1, introduced five architectural options for the Resolution infrastructure: Geographic Location-based, Semantic Web-oriented, Federation-based, Distributed Hash Tables and Domain-oriented. In this section, we revisit these five options with the aim of providing the details of the IoT Service and VE Resolution mechanisms. This encompasses detailing how the functionalities of the various interfaces that have been described in section 2 can be achieved. The approaches differ in the way the specific functionalities are achieved and what forms of the different interfaces are supported. For instance, some are more suitable for look-up of service descriptions and inferred Associations, while others for discovery.

Each of the approaches is structured to detail the mechanisms for Discovery and/or Lookup, including both IoT Service and Virtual Entity Resolution. Following this, the Association mechanisms are presented, including supported methods for VE & IoT Service Monitoring.

3.1 Geographic Locations Approach

3.1.1 Discovery Mechanisms

3.1.1.1 Map Projection

The Geographic Locations approach for a discovery infrastructure needs a way to properly store and access spatial data. For this reason, several spatial structures are available, with different characteristics and performance considerations. In the IoT world, there is a large variety of mobile objects, with the prevalent system of coordinates being GPS coordinates. Most spatial structures, however, deal with a flat X-Y coordinate system, so it becomes essential to be able to convert between the two systems.

To be able to represent the world map in a flat surface, several map projections system exists, split into different types and different characteristics. Each type of map projection tries to preserve a different metric property. For example, conformal map projections try to preserve angle and shapes between the Earth’s sphere representation and its flat map representation; equal-area map projections try, as the name suggests, preserving the area; equidistant map projections try to preserve distances between any two points in the map.

The most common map projection system in use is the Mercator map projection, which is a cylindrical conformal map projection. It was often used for marine navigation, as all straight lines are lines of constant azimuth. Today, it is used by all the major online mapping services, such as Google Maps, Microsoft Bing Maps, MapQuest and others. As a conformal map, it preserves angles and shapes of small objects in the final map projection, so this makes it very well suited for the IoT geolocation-based discovery infrastructure, as it makes it possible to preserve VE and IoT Service areas’ shapes, as well as shapes of spatial query. However, distortion may occur in significantly large areas, especially those closer to the poles.
The Mercator map projection of the world shown in Figure 12, shows that for locations close to the equator, the projection is very accurate, but the closer we get to the poles, the larger the distortion is. For example, in this projection Greenland seems to have a greater area than Australia, when Australia is actually more than three times larger than Greenland.

### 3.1.1.2 Virtual Entity Resolution

In IoT-A, the Virtual Entity Resolution infrastructure handles the discovery and lookup of dynamic Associations between Virtual Entities (VE) and IoT Services. Virtual Entities are composed of an ID and multiple Attributes, and it is the combination of the pair VE ID and VE Attribute that forms an Association with an IoT Service.

There are two types of interactions with the VE Resolution, synchronous request-response and asynchronous publish-subscribe interactions.

Virtual Entities are composed of several Attributes, one of which is a spatial attribute, e.g. a geographical location given in GPS coordinates. To allow for efficient location-based discovery, suitable spatial structures, such as R-Trees and Quad-Trees, are used to store this information. As detailed in D4.1, there are mainly two types of spatial structures: Binary Space Partitioning (BSP) trees, such as Quad-Trees and KD-Trees, which are mainly oriented towards indexing point data; and Data Partitioning trees, such as R-Trees, R+-Trees and R*–Trees, which besides indexing point data support also indexing of regions and areas. For the VE-Resolution, the Virtual Entities can be represented as point data, using BSP trees (Quad-Trees, KD-Trees). If, on the other hand, VEs must be represented as regions and not as points, DP trees (R-Trees, R*-Trees) can be used.

To minimize the amount of information these spatial structures store and to provide better efficiency, the Associations’ non-spatial information may be stored separately, for example in a simple hash map (we will call this **main** storage, as opposed to **spatial** storage). This would mean that changes to an Association’s location can be updated quickly in the spatial structure.
and changes to other attributes do not need to change the spatial representation of the Association.

To support both request-response and publish-subscribe discovery operations, two index structures may be needed. For request-response Discovery, the location of the objects need to be stored; for publish-subscribe, the spatial area of the continuous query also needs to be stored, preferably separately. There are multiple reasons to store the separately. The biggest one is that if we store both the location of objects and the subscriptions in the same spatial storage, spatial queries for objects will also return the subscriptions, and vice versa. Another reason is that mobile objects are represented as point data, and subscriptions as areas, so although they both have spatial data they may use different spatial structures, considering the characteristics of both point-data and region-data.

**Discovery**

Discovery of Associations is useful for scenarios where the VE ID is not known. For the Geographic Location approach, request-reply discovery of Associations is based on the geographic location attributes of the Virtual Entities.

The interface for VE discovery, defined in 2.2.2.1, is the following:

```
discoverAssociations(VESpecification, VEServiceSpecification): Association[]
```

An example of such a discovery operation is the following:

```
discoverAssociations( {"type": "parking space", "scope": <geographic scope>}, {"available": "true"} );
```

The type parameter of the VESpecification defines the type of Virtual Entity we are looking for, and the `<geographic scope>` represents a spatial region within which we want to search for. This spatial region may be given in a pair of GPS coordinates, representing two corners of the rectangle region (NE for the north-east corner, and SW for the south-west corner), or an arbitrary polygon, given in a list of GPS coordinates, each representing a polygon vertex.

In the example of Figure 13, this geographic scope is a rectangle, represented by the coordinates of the two corners, NE and SW. The parameter would then be:

```
{ "NE": [49.408321, 8.67774], "SW": [49.404216, 8.686495] }
```

![Figure 13 Example of a VE-Resolution Spatial Query](image)
Internally, the Associations are stored in suitable spatial data structures, such as R-Trees, KD-Trees or Quad-Trees. Location-based discovery of Virtual Entities consists, in a first step, in performing a spatial range query, which then returns all Associations whose associated VE’s location intersect with the query’s range area. Figure 13 shows an example of such a spatial query. The figure contains a map of Heidelberg (Germany), and all existing Virtual Entities in that area. The Query area (in red) represents the spatial region of the Discovery operation, with the objective of finding all existing VEs in that area. Figure 14 shows the representation of the same spatial query, using a suitable spatial data structure, in this case an R-Tree.

Figure 14 R-Tree representation of a VE-Resolution Spatial Query

The R-Tree is an example of a data structure suitable for spatial data, which allows Range Query operations, as well as other types of queries such as N-Nearest Neighbour queries. An R-Tree, similarly to a B-Tree, is a balanced tree structure where each node represents a geographical area, and one node’s children represents sub-areas contained inside the parent node. Objects only are stored in the leaves of the tree. This allows very fast range queries, as in every level of the tree the area of the nodes is checked to see if it intersects the query’s area. In cases where it does, the query is passed recursively to the nodes’ children, until the leaves are reached.

Figure 14 shows such an R-Tree, where the root node represents Germany, and each child represents a different city; further inside each city, there are multiple nodes for different regions of the city. The query presented only spans two areas inside Heidelberg; as such, after querying the root node, the query is passed onto the Heidelberg node, and finally onto the third and final level, the leaves, querying only the HD1 and HD2 nodes. These two nodes will then check each object’s location against the query area, and then determine which objects satisfy the spatial query.

After the first step of filtering objects according to geographical scope, the second step consists of applying further filtering criteria to the output of the spatial query, and returning only the Associations which fully match the given ServiceSpecification.

Subscribe Discovery
The subscription of discovery operation consists of performing a continuous query over the entire set of Virtual Entities. This means that besides storing the VEs themselves, the query areas also need to be stored. This requires a separate spatial index structure, which will be queried with update (to the spatial information) of VEs. This separate spatial storage (subscriptions spatial storage) will store the spatial queries to be continuously executed.
against mobile object location updates. Unlike the VE spatial structure, this structure must support spatial areas and not points, so a Data-Partitioning tree such as an R-Tree must be used. Additional information about the subscription may also be stored separately (like what happens with the VE Discovery), for example in a hash table (we will call this subscriptions main storage).

When there is an update to a VE’s spatial information, this update is reflected in the spatial storage, responsible for holding the VE’s location. However, a second event is triggered in parallel: the second spatial structure, which holds the current subscriptions for discovery updates, is queried for subscription regions that intersect the new location of the VE, and a notification is triggered for each subscription found.

**Management (Insert, Update, Delete)**

Besides discovery, management operations such as Insertion, Update and Deletion of Associations are also supported by the platform. As with discovery, the algorithms for managing Associations largely depend on the chosen structures, both for the spatial storage data and main storage data.

Inserting Associations consists of both inserting them in the main storage, as well as inserting a pointer to it in the spatial data structure. This means that an Association’s geographical information can be updated entirely in the spatial structure, without changing any other information about the Association. Likewise, information about an Association that is not related to location can also be updated without changing the Association’s spatial representation in the spatial data structure.

Main storage may use a simple hash map to store a list of Virtual Entities and its respective Associations, indexed by the AssociationID. Inserting and updating information in the main storage in this case consists of changing the value of an entry in a hash map, which is quickly accessed by being given a key (the AssociationID). For the spatial storage, it can mean changes to the underlying structure need to be made. For the R-Tree example, inserting a new entry (a new Virtual Entity) may trigger changes to the upper nodes of the R-Tree, where new upper-level nodes may be created, existing ones expanded or even split, to accommodate the new entry.

Deleting an Association means deleting both its entire information stored in the main storage, as well as the pointer to it in the spatial structure. Deleting the main storage entry in a hash map is very quick, as it is only a matter of finding the correct entry given a key, and deleting it. However, deleting an entry in a tree-like spatial structure may trigger a tree readjustment, where upper tree nodes may have to be readjusted or even merged to accommodate the new setup.

For Subscribe Discovery, the continuous query regions must also be inserted in its spatial structure (the subscriptions spatial storage), as well as other information about the subscription (Who and how to notify? Subscribe to objects entering the area, or leaving the area?). The considerations for inserting, updating and deleting entries are the same as the VE’s spatial structure: inserting, updating and deleting entries may trigger a tree readjustment, where upper tree nodes need to be split, merged or new ones created; non-spatial information may be inserted in a separate structure, such as a hash table.

### 3.1.1.3 IoT Service Resolution

The Geographical Approach for the IoT Service Resolution infrastructure handles the discovery mechanisms for retrieving service descriptions based on geographical location.

**Discovery**

Discovery of IoT Services consists of retrieving a list of service descriptions that match a given Service Specification. For the Geographical Locations Approach, this Service Specification mainly consists of a geographical area where we want to query for IoT Services, plus any additional attributes to filter services by, such as the type of service.
Similarly to the Location-based VE Resolution infrastructure, suitable spatial data structures will be used to store the location information of the IoT Services (spatial storage), while other non-spatial attributes will be stored separately (main storage). Also as it happens with the VE-Resolution infrastructure, a hash map can be used for the main storage, storing all information about the service indexed by the service’s ID. One difference, however, to the VE Resolution is the location attributes of IoT Services are mainly spatial regions, and not points. For an IoT Service, the important location information is the service’s service area, which may not necessarily be the same as the location of the device on which the resource resides (the resource is made available through the service). This requires the ability to store spatial regions, which Binary Space Partitioning (BSP) trees such as Quad-Trees and KD-Trees may not be particularly good for. For this reason, only Data Partitioning (DP) trees such as R-Trees and all its variants should be used.

Although with different structures, the procedure for discovering IoT Services is similar to the VE Resolution discovery. In a first step the spatial storage is queried for IoT Services whose service areas intersect the query area, and they are retrieved from the main storage; in a second step, further filtering criteria (if present in the query) are applied to the results of the spatial query, and the results are returned.

**Subscribe Discovery**

The subscription-based discovery operation for IoT Services Resolution is very similar to the subscription-based discovery for VE Resolution. There is a separate storage that instead of storing IoT Service descriptions and their service areas, stores instead the discovery subscriptions (subscriptions main storage), including the subscription spatial areas (subscriptions spatial storage), for which we want to be notified of changes. Subscriptions are then made according to a Service Specification, specifying the type of service we are interested in, including its service area. Every time there is an update to an IoT Service’s service area, the subscriptions spatial storage is checked for possible intersections between IoT Services’ service areas and subscriptions’ service areas; when an intersection is found, a notification is triggered for all the subscribers of that service specification.

**Management (Insert, Update, Delete)**

Also like with the VE Resolution infrastructure, the IoT Services Resolution infrastructure supports management operations such as adding, updating and deleting IoT Services. Adding a new IoT Service consists of inserting its geographical service area in the spatial storage, as well
as the service description into the main storage. Changes triggered by an update to an existing IoT Service depend on the information being changed; changes to the service’s non-spatial attributes may only trigger an update in the main storage (which can be very fast, e.g. using an hash table), but changes to a service’s spatial scope may trigger a change in the spatial storage. Deleting an IoT Service entry requires removal both from main storage and the spatial storage.

Changes to the spatial storage (either with an insertion, update or deletion) may require a tree adjustment and rebalancing, whatever the structure used. In the case of R-Tree and its variants, this tree adjustment may consist in merging, splitting or adding new upper-level tree nodes. As with the VE Resolution, support for discovery subscriptions is also required.

3.1.2 Association Methods

Location is an important and highly selective criterion for discovering information about the physical world. Typically we are not interested in finding information or resources anywhere, but in a very specific location, often close to the user. There are different ways of modelling location, i.e., in multi-dimensional geographic coordinates or as symbolic locations that may have an underlying hierarchical structure. To efficiently find entities and their associated services according to location, the resolution and discovery infrastructure needs to provide the necessary support, e.g., use location for its underlying structure or utilize (multi-)dimensional index structures for efficient access.

When we want to use geographic location as a basis for finding Associations between Virtual Entities and services, we respectively have to take the geographic location of the corresponding Physical Entity and the service area of the IoT service into account. If the service area of an IoT Service and the location of a Physical Entity overlap, it has to be checked whether the Virtual Entity corresponding to the Physical Entity and the IoT Service can be associated. For example, this is the case if the IoT Service can provide some information regarding the Physical Entity or if it can execute an actuation that affects the Physical Entity.

In the following we describe use cases where Associations can change over time and thus automatic detection and monitoring of Associations is necessary in order to enable reliable resolution mechanisms. A necessary condition for these Associations to change over time is that we are in an environment where locations are not fixed. However, this condition is not sufficient, as there are many cases where the services (e.g., fuel status sensor) are always co-located with the entity (e.g., car) they provide information about. A stronger condition for dynamic Associations can be formulated as follows: Recall that the physical counterpart of Virtual Entities is Physical Entities and the physical counterpart of IoT Services described by service descriptions are sensors and actuators. Associations are dynamic whenever in the physical world there is no fixed attachment of the corresponding sensors and actuators to the Physical Entities they provide information about.

3.1.2.1 Categorization of Use Cases for Dynamic Associations

Before going into the description of concrete use case, we give a number of parameters for categorizing these cases.

Reason for dynamic Association: As explained above, Associations are dynamic whenever there is no fixed attachment of each sensor or actuator to the Physical Entity it provides information about. One could ask the question why it is not possible to simply attach to each Physical Entity of interest the sensors and actuators that are needed for providing the desired interactions with it. This would result in a fixed bijection between the set of Entity Service Specifications and the set of available IoT Services, and the consideration of static Associations would be sufficient. Clearly, there are a number of restrictions that make this approach infeasible in many situations. We identified three categories of such restrictions:
1) **Cost effectiveness.** It would be too expensive to attach the full set of sensors and actuators to each Physical Entity of interest. These kind of restrictions can in principle be overcome in the future when the specific prices decrease sufficiently.

2) **Technical infeasibility.** One cannot attach an arbitrary set of sensors and actuators to any physical entity. This might for example be because these resources are too large, to heavy, cause health problems, or need a certain distance to the physical entity in order to interact with it. These kinds of restrictions can in principle be overcome in the future by technical progress that change the physical properties of existing sensor/actuator types.

3) **Logical infeasibility.** There are situations where there exists no sensor or actuator type that could be attached to a Physical Entity to perform the desired interaction. As an example, assume that a road section is represented as a Virtual Entity, and one attribute of that entity is the fuel consumption of cars passing along. The sensors delivering this information have to be attached to the cars, not to the road. These kinds of restrictions can in principle only be overcome by the invention of new sensor/actuator types.

**Type of moving Physical Entities:** One can divide the scenarios where the Physical Entities of interest have dynamic locations into cases with controllable motion and uncontrollable motion. Here controllability refers to the point of view of IoT Applications. Whenever there are actuation services available to control the motion of the Entities we speak of controllable motion. Note that the Association of such a motion control service to the Entity can be static, e.g. in the case of robots, of dynamic, e.g. in the case of cranes that can move around objects. We speak of uncontrollable motion when location changes of the Physical Entities are beyond the control of the IoT Applications. Typical use cases are persons or cars. In general, in any IoT system that does not support actuation it holds that any Entity motion is uncontrollable.

**Type of moving Services:** At first glance it looks like one can divide the scenarios with IoT Services having dynamic service areas into the same categories as the dynamic Entity cases. However, note that applications cannot control the motion of IoT Services directly by actuation, because services always interact with Entities, and not with other IoT Services. In order for applications to be able to control the location of IoT Services, they need the additional knowledge of a Physical Entity the IoT Service is always co-located with. This can either be an entity which directly represents the sensor/actuator corresponding to the IoT Service, or an entity the sensor/actuator is attached to or part of.

Table 1 shows a number of dynamic Associations in scenarios belonging to different applications domains. For each dynamic Association the reason for supporting the dynamic Associations, the involved Physical Entities and their movement types as well as the IoT Services and their movement types (or more precisely the movement of the Devices hosting the Resource that is exposed through the IoT Service).

**Table 1 Dynamic Associations in different scenarios**

<table>
<thead>
<tr>
<th>Domain</th>
<th>Scenario</th>
<th>Reason for dynamic Association</th>
<th>Entities and their movement types</th>
<th>Services and their movement types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Industry</td>
<td>The state of products and product parts moving along the assembly line is monitored by stationary sensors.</td>
<td>Cost effectiveness and technical infeasibility</td>
<td>Product parts (controllable or uncontrollable movement)</td>
<td>Various sensors (stationary)</td>
</tr>
<tr>
<td>Industry and Agriculture</td>
<td>All equipment has a card reader, where employees authenticate themselves before the equipment is usable. As employees carry a GPS device, the location of any piece of</td>
<td>Cost effectiveness</td>
<td>Equipment (uncontrollable movement), GPS devices (uncontrollable movement, associated with employees)</td>
<td></td>
</tr>
</tbody>
</table>
### Logistics and Retail

| Logistics and Retail | Products are constantly monitored on their way from the production site to the end consumer in order to guarantee that they are not exposed to inappropriate conditions (e.g. temperature, humidity, light level, radiation) for too long. | Cost effectiveness | Products (uncontrollable or partially controllable movement) | Various sensors (stationary and/or moving, controllable and/or uncontrollable) |

### Traffic

| Traffic | Traffic jams are detected in real time based on the location and speed information received from sensor-equipped cars. | Logical infeasibility | Road sections (stationary) | Speed sensors (uncontrollable movement) |

### Traffic

| Traffic | Automatic countermeasures in case of detected accidents: set traffic lights to red for blocking the road until the scene has been secured. Detect mobile phones and stationary cameras in the area so that further information can be requested. | Logical infeasibility | Accident (stationary but location unforeseen) | Cameras (stationary but detected on-demand), Mobile phones (uncontrollable movement) |

### Smart City

| Smart City | Use mobile phones of volunteering users to determine noise level in city districts. | Cost effectiveness | City districts (stationary) | Mobile phones (uncontrollable movement) |

### Generic

| Generic | Sensor replacement: When a stationary sensor stops working, use the measurements from the closest sensor of the same type instead. | Cost effectiveness, logical infeasibility | Broken sensor (stationary, modelled as Virtual Entity) | Next available sensor (stationary, but detected only on demand) |

### 3.1.2.2 Different Cases of Finding New Associations

Figure 16 shows the dynamics with which Physical Entities and Devices that may be associated with each other move. Rarely or very slowly moving Physical Entities and Devices may be considered stationary for our purposes, e.g., a sensor fixed to the wall of a room or a table inside a room.

So simplifying the scenario a bit, we can consider four different cases.

1. Both Physical Entity and Device are stationary. If we detect an Association between them, the Association can be registered without having to be constantly monitored.
2. Only the Physical Entity is stationary, but the Device hosting the Resource that is exposed through the IoT Service is mobile. In this case, the service area of the IoT Service, which is correlated with the location of the Device, has to be monitored as a change in the service area may render the Association invalid.
3. Only the Device is stationary, but the Physical Entity is mobile. In this case the location of the Physical Entity has to be monitored as the Physical Entity may leave the service area of the IoT Service, rendering the Association invalid.
4. Both the Physical Entity and the Device are mobile. In this case both the location of the Physical Entity and the service area of the IoT Service exposing the Resource hosted...
on the Device have to be monitored as a change in any of these may render the Association invalid.

![Diagram of Physical Entity and Device dynamics](image)

**Figure 16 Dynamics of Spatial Relations between Physical Entities and Devices**

So we use geographical location to find candidates for Virtual Entities and IoT Services that can be combined in an Association. Of course, geographical co-location is not sufficient. The acceleration sensor in a mobile phone may not provide useful information about a table close-by, but a connected temperature sensor could be used to measure the temperature in a room. Thus, a core aspect is the type of information or actuation that can be provided by an IoT Service. Finding an attribute of a Virtual Entity that matches what the IoT Service can provide is necessary, but again not sufficient. If we find a temperature sensor that is geographically located in a room, it is not immediately given that it provides the indoor temperature of the room. It could also provide the temperature of a fridge located in this room. Therefore additional information is needed, e.g. an existing Association with a different Virtual Entity that clarifies the semantics of the information.

The knowledge regarding what conditions in addition to co-location and have to be fulfilled as the basis for creating a new Association can be captured in rules. We expect that these rules will be relatively specific to an application or a scenario. We will further investigate this, using concrete examples like the ones identified in Table 1.

The monitoring of location is required for all cases where new dynamic Associations between Virtual Entities and IoT Services are to be found or the validity of existing ones is to be monitored, assuming the mobility of Physical Entities, Devices or both. In the following we will therefore elaborate on this monitoring.

### 3.1.2.3 Location and Tracking Systems

The location of Physical Entities can be modelled in the Virtual Entity part of Associations, i.e., the hasLocation property shown in Figure 5. The service area is part of the service description
shown in Figure 3. In the case of stationary Physical Entities or stationary service areas, the information can be provided when the Association or service description are inserted. In the dynamic case, this information changes continuously, so the information needs to be updated.

The geographic location can be determined using different technologies. A distinction can be made here between location systems and tracking systems. With location systems, the device to be located determines its own location, using external information. A GPS sensor is a typical example, where the sensor determines its location by receiving signals from different satellites and calculating its location based on the differences in the reception times (trilateration). With tracking systems, the environment can determine the location of a device.

Location and tracking systems have different coverage areas and quality properties. For example the GPS system works worldwide, but only outdoors and provides an accuracy of a few meters. Using the signal strength of different wireless base stations may also work indoors, but often provides lower accuracies. RFID-based system may provide a high accuracy whenever the identifying tag is close to a reader, but no information at other times. Indoor positioning system using UWB or ultrasound may provide a high accuracy, but are typically very expensive and provide only a small coverage area.

The actual technology of the underlying location or tracking system used is not important for our purposes, as long as the provided location can be compared to other location, potentially by converting it first, and of course the quality of the location provided has to be taken into account.

### 3.1.2.4 Location information in Service Descriptions, Associations and from Location Resources

In order to determine whether there is a relation between a Virtual Entity and an IoT Service that could be captured as a new Association, the location of the Physical Entity modelled by the Virtual Entity and the service area of the IoT Service have to be checked for overlap. This means that for candidate Virtual Entities and IoT Services the location information has to be retrieved.

For IoT Services, the required location information can be found as a service area in the service description. Given a location scope in which services of interest can be found, the service discovery with a location scope as described in Section 3.1.1.3 can be used. In the VE & IoT Service Monitoring functional component, the Monitor IoT Service Descriptions sub-component shown in Figure 11 is responsible for the respective interaction with the IoT Service Resolution functional component. If services areas can be assumed to be stationary, a simple synchronous discovery is sufficient, in the general case, a subscription to the discovery is necessary. If changes of service areas on average happen on a moderate scales, updates to the service areas in the service descriptions can be considered reasonable. For known services with very high update rates a different approach described later has to be used.

For Virtual Entities, the location information can be made available as part of an existing Association, using the hasLocation property of the Virtual Entity part. In this case, the discovery with a location scope as described in Section 3.1.1.2 can be used. In the VE & IoT Service Monitoring functional component, the Monitor Associations sub-component shown in Figure 11 is responsible for the respective interaction with the VE Resolution functional component. The approach of storing location information in Associations is reasonable, if the location of the Physical Entity does not change so quickly that it leads to high update rates. It has to be taken into account that the update to the location property of a single Virtual Entity has to be reflected in all Associations pertaining to this Virtual Entity.

In case of high update rates, it is more reasonable to leave the management and monitoring of location information to a specialized resource that is exposed as an IoT Service. Such a resource can provide the location information of a single Physical Entity or for all Physical Entities in a whole area. For example, the latter may easily be realized in the case of a tracking service that can determine the location of Physical Entities in its service area. In this case, we have Associations between the Physical Entities and the location or tracking...
service based on a hasLocation attribute. In the VE & IoT Service Monitoring functional component, the Monitor VE Information sub-component shown in Figure 11 is responsible for the respective interaction with the respective IoT Service providing the location information.

In the case of the dynamically changing service area, we can follow the same approach, given that we have enough additional information. If we know the device hosting the resource that is exposed as a service and we know how the service area can be derived from the location as a device, we can monitor the location of the device – again assuming that the device as a Physical Entity is modelled by the Virtual Entity for which we can retrieve the hasLocation attribute as in the case above.

### 3.1.2.5 Discovering new Associations

Discovering Associations is based on retrieving or subscribing to location information as described in the previous subsection. Permanently monitoring the location of all Physical Entities and all IoT Services in the system and checking for possible Associations may not be feasible. This means that first candidate sets of Virtual Entities and IoT Services have to be determined. As described in Section 2.2.3.2, the candidate set may be restricted to certain types or even instances of Physical Entities and IoT Services. In the case of finding Associations based on geographic location, the selection of location scopes within which to look for Associations is of major importance. The VE & IoT Service Monitoring functional component may thus be distributed according to location, where each instance is responsible for a particular area, as shown in Figure 17.

![Figure 17 Discovering new Associations](image-url)

On the left side, Figure 17 also shows that a moving IoT Device or a moving Physical Entity may result in a new dynamic Association. To determine whether an Association is static or dynamic, it has to be checked whether the Physical Entity and the service area of the IoT Service is stationary. In the case of the Physical Entity this can be modelled by setting the isMobile flag of the corresponding Virtual Entity accordingly. The main difference between a static and a
dynamic Association is whether it later has to be monitored, i.e. it has to be checked whether the location of the Physical Entity is still within the service area of the IoT Service.

3.1.2.6 Monitoring existing Associations

Depending on the cases identified in Section 3.1.2.2, the location of the Physical Entity, the service area of the IoT Service, or both have to be monitored. In this case, we know the identifiers of the Physical Entity and of the Services, so we can use the respective look-up operations of the VE Resolution and the IoT Service Resolution respectively. As we have to continuously monitor, it makes sense to use the subscription mechanism to be updated on relevant changes. Depending on the expressiveness of the subscription conditions, we can monitor on changes, on a periodic basis or, ideally when the location has changed by a relevant distance, e.g., leaving a certain geographic scope. If only periodical subscriptions are possible, there is a certain trade-off between monitoring in short periods which is more expensive, but leads to a timely detection that the Association has become invalid, or using longer periods resulting in invalid Associations being kept in the VE Resolution for some time.

3.1.3 Conclusions

A Geographic-Locations based approach for a Discovery infrastructure needs both the ability to efficiently discover IoT Entities based on geographical information, as well as the means to manage and maintain the dynamic Associations between these Entities and their respective IoT Services. Although discovering Entities and Services and monitoring their Associations are different roles, they are closely related to each other.

For discovery, both the VE Resolution and IoT Services Resolution present identical architectural options: they both consist of Resolution Servers (which contain information about the IoT Services or Virtual Entities’ locations) and Catalogue Servers which group different Resolution Servers together, as well as RSs from different domains. The main difference between them resides in the data structures used for indexing. Virtual Entities which are mainly represented by point data can use suitable spatial structures oriented for points, and IoT Services, which are mainly represented by regions, can use spatial structures oriented for areas. However, VEs can also be represented by areas. The choice of data-structures is then very dynamic, and when deploying the architecture it is possible to choose different structures according to these characteristics.

For maintaining the dynamic Associations between IoT Services and Entities, several attributes need to be checked, one of them being the location information of both. The overlap of the location between Entities and Services’ service areas (in conjunction with other attributes) will create these Associations; depending on the nature of the IoT Services and/or Entities (stationary, mobile) this will also dictate the need for monitoring of location updates and respective adaptation of the Associations.

3.2 Semantic Web Approach

This section presents an approach for IoT Service Resolution, VE Resolution and Association methods using Semantic Web techniques. The use of Semantic Web techniques requires that the meaning of information is formally defined. These formal definitions could be captured in ontologies, making the data machine-interpretable and data content relations more effective. The principal technologies of the Semantic Web are the ontology representation languages RDF Schema and Web Ontology Language (OWL). Thus, the approaches presented in this section make use of a semantic implementation of the Entity, Resource and Service Models that have been presented in section 2.1. For the purpose of discovering new association, it also defines an additional ontology, capturing indoor location concepts. The use of semantic methods allows managing the heterogeneity of possible services making up the IoT-A domain and matchmaking based on both functional and non-functional attributes.
3.2.1 Discovery Mechanisms

3.2.1.1 IoT Service Resolution

The Semantic Web approach for the IoT Services Resolution infrastructure handles the discovery mechanisms for retrieving IoT Service Descriptions that have been defined using Semantic Web technologies. Since the IoT Service Resolution component seeks to find suitable services for a given user request, semantic techniques can counter the heterogeneity of possible service descriptions and perform discovery based on both functional and non-functional techniques. In this section, we present two methods for performing IoT Service Resolution; one based on machine learning techniques and the second which performs semantic distance similarity measurements for service matching.

The machine learning method is used to cluster the semantically described services by deriving latent factor indexes from the semantic descriptions. The indexes are then employed in a semantic framework to provide matchmaking to discover the IoT Services. Once the services are clustered, the discovery request for IoT services can be scoped to the relevant cluster. Matching between a query and the clustered services is then performed through cosine similarity measurements.

The semantic distance similarity method is based on matching semantic concepts from a defined ontology that describes IoT Services. Service discovery is then performed by measuring semantic similarity between semantic concepts in the query request and semantic concepts in the available services within the system (in a service repository for example). The most similar services are returned to the requester.

**Machine Learning Techniques for IoT Service Resolution:**

**IoT Service Clustering:**

The clustering mechanism starts by extracting concepts from the semantic service descriptions. To illustrate the working of the Machine Learning techniques for IoT Service Resolution, we have created a test dataset of 150 IoT Service descriptions, based on the IoT-A Service Model detailed in section 2.1.2. In this dataset, each service describes the capabilities of a sensor and exposes an IoT-A resource. An example of one of these services is shown in Figure 18. This service describes a light sensor in room 01_BA_02.

![Diagram of IoT-A Service instance example](image-url)
Concept extraction is done by a parsing mechanism that augments together the name of a concept and its role (property name) in order to express what role a concept plays in the service description. The service descriptions conform to the Service Model however, different parsers can be written for services conforming to another semantic model, e.g. OWL-S. The concept and role are augmented by simply attaching them together as a string. For example, from the service description shown in Figure 18, the concept luminousIntensity which is described by the property hasOutput is converted to hasOutput_luminousIntensity. Using this approach, a service in which luminousIntensity is defined as input will be distinguished from a service in which luminousIntensity is defined as output. The whole repository of service instances is subjected to the concept extraction method in order to create the Service Transaction Matrix (STM). The STM has as many rows as the number of services and as many columns as the number of concepts in the service dataset. Each row represents a service as a vector of N dimensions where each column represents the occurrence of a concept c in the description of that service. A vector \( \overrightarrow{s_i} \) describing a service \( i \) is denoted as:

\[
\overrightarrow{s_i} = \{c_1,c_2,...,c_N\}, \text{ where } \forall \ c \in \mathbb{I}^+.
\]

The STM represents the joint probability distribution \( P(s,c) \) of services and concepts, where \( P(S,C) \) denotes the joint probability of Concept ‘c’ and Service ‘s’, which is given by the value at (column c, row s) in the service transaction matrix. It is assumed that concepts are independent from each other.

The concepts extracted from the service shown in Figure 18 would be:

\{exposes http://purl.oclc.org/net/unis/iots/resources/resource_01_BA_02_light_sensor.owl, hasOutput http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity, http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity, hasObservationArea http://www.surrey.ac.uk/ccsr/IoT-A/entityInstance.owl#BABuildingLocation\}

Similarly, concepts are extracted from all the other services in our dataset which includes temperature sensors, power sensors, presence sensors, and sound sensors.

STM generation is followed by training the model, which is based on Latent Dirichlet Allocation (LDA), (Blei, Ng, & Jordan, 2003) an unsupervised machine-learning technique that maps high-dimensional count vectors (such as the ones expressed in the STM) to a lower-dimensional representation in Latent Factor Space. This is shown in Figure 19.
This step discovers a hidden dimension behind the vector of concepts describing a service, i.e. topics that include concepts in the service descriptions. The concepts are observable variables and their occurrence in a service description can be described in a vector as defined in Equation 1. Topics on the other hand are latent factors which are not directly observable through examining a service description. These latent factors are learned through statistical inference and during training, the model associates these unobserved latent factors \( \{z_1, z_2, \ldots, z_k\} \) with the distribution of concepts over service descriptions. The probabilistic model is defined by Equation 2.

\[
P(s, c) = \sum_{z=1}^d P(z_j) P(s | z_j) P(c | z_j)
\]

Once the latent factors are identified, services can be described as a multinomial probability distribution \( P(s | z) \) where every service \( s \) is described by a vector of latent factors. Training consists of using Gibbs Sampling to find the optimized distribution of latent variables over service descriptions \( P(s | z) \).

The following tables show four of the latent factors extracted after training the LDA model on the dataset of concepts taken from the 150 IoT Service descriptions. The model was set to extract ten latent factors in total. Each table shows the three most relevant concepts assigned to the latent factor and by what probability. Note that a concept can be assigned to more than one latent factor.

### Latent Factor #1

<table>
<thead>
<tr>
<th>Concept</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://purl.oclc.org/NET/ssnx/qu/quantity#noiseLevel">http://purl.oclc.org/NET/ssnx/qu/quantity#noiseLevel</a></td>
<td>0.219</td>
</tr>
<tr>
<td><a href="http://purl.oclc.org/NET/ssnx/qu/quantity#noiseLevel">http://purl.oclc.org/NET/ssnx/qu/quantity#noiseLevel</a></td>
<td>0.219</td>
</tr>
<tr>
<td><a href="http://www.surrey.ac.uk/ccsr/IoT-A/entityInstance.owl#BABuildingLocation">http://www.surrey.ac.uk/ccsr/IoT-A/entityInstance.owl#BABuildingLocation</a></td>
<td>0.198</td>
</tr>
</tbody>
</table>

### Latent Factor #2

<table>
<thead>
<tr>
<th>Concept</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity">http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity</a></td>
<td>0.277</td>
</tr>
</tbody>
</table>
After training the algorithm, new services can be folded in (i.e. added or published) by using Gibbs Sampling with fixed service description to concept probabilities and sampling the assignments of concepts to latent variables in the new service description (Wei, Barnaghi, & Bargiela, 2010).

Once the model is trained, the distribution of concepts for each latent factor is known and all the services in the dataset can be described as a distribution of latent factors (i.e. a vector where every dimension k reflects the probability by which that service belongs to latent factor $Z_k$). We create K clusters, where K is the number of generated latent factors (i.e. a cluster for each latent factor). The vector of latent factors describing each service is used to determine which latent factor best describes the service. The service is then assigned to the cluster corresponding to that latent factor. An abstraction of this mechanism is shown in Figure 20.

![Figure 20 IoT Service Clustering](image)

The distribution of latent factors for the service shown in Figure 18 is as below:

<table>
<thead>
<tr>
<th>Latent Factor</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
The service has a high probability of belonging to latent factor #2, therefore it is assigned to cluster #2.

**IoT Service Discovery:**

The discovery process accepts a query with an IoT Service Specification as input, as shown below:

\[
\text{discoverService(ServiceSpecification): ServiceDescription[]}
\]

The service specification can include any number of concepts from the IoT Service Description model. The query can be expressed in terms of latent factors by computing the distribution of latent factors for the concepts that describe the query. Thus, the query consists of a list of concepts that are required to be matched and –the query, expressed in terms of latent factors, is submitted to the search engine. The search engine then converts the concepts into a vector of latent variables. With LDA, this process is called folding-in, which is performed by using Gibbs Sampling with fixed service description to concept probabilities \(P(c|z)\) and sampling the assignment of concepts to latent factors in the query. Once the query has been described in terms of a vector of latent factors, the degree of match between the service dataset and the query can be determined using vector similarity measures. The similarity measure used is the Cosine Similarity function, which uses the cosine of the angle between two vectors. Equation 3 computes the Cosine Similarity between a vector containing the query's distribution of latent factors \(q\) and a vector containing the service's distribution of latent factors \(p\), where both vectors are of dimensionality \(f\).

\[
\cos(p, q) = \frac{\sum_{i=1}^{f} p_i q_i}{\sqrt{\sum_{i=1}^{f} p_i^2} \sqrt{\sum_{i=1}^{f} q_i^2}}
\]

For efficiency purposes, the degree of match is computed between the query and every cluster in the dataset, instead of matching the query with each individual service, which would be impractical in IoT scenarios. When the best matching cluster is identified, the query is matched only to the services that belong to that cluster, making the discovery process more refined. This can be illustrated by the following query example, which has the concepts stated below:

\{hasOutput http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity, http://purl.oclc.org/NET/ssnx/qu/quantity#luminousIntensity\}

The query is converted to a vector of latent factors:

\{0.038, 0.916, 0.006, 0.006, 0.0, 0.006, 0.01, 0.008, 0.003, 0.0\}

As we can see from the vector of latent factors, the query belongs to latent factor #2 with the highest probability. Therefore, the query is forwarded to the service registry responsible for cluster #2 and the query is matched within that cluster. The first five results obtained are:

- Rank #1: Service_54_BA_01_light_sensor.owl with a score of 0.99847953076855
- Rank #2: Service_44_BA_01_light_sensor.owl with a score of 0.99847953076855
- Rank #3: Service_01_BA_02_light_sensor.owl with a score of 0.99847953076855
- Rank #4: Service_37_BA_01_light_sensor.owl with a score of 0.99847953076855
- Rank #5: Service_18_BA_02_light_sensor.owl with a score of 0.99847953076855
Note that all five services providing a light sensor scored equal probability as they all have the same degree of similarity to the query vector.

**Semantic Distance Similarity for IoT Service Resolution:**

In the IoT-A context, service discovery consists in finding suitable IoT Services descriptions that match some User expectations. Assuming that in IoT-A, IoT Services may be semantically described and annotated with concepts defined in a considered ontology; a method to find suitable IoT Services is to measure the similarity between concepts expressed in a User request as well as in IoT Service descriptions.

This section will then focus on the work done by (Bramantoro, Krishnaswamy, & Indrawan, 2005), which presented a method based on the notion of semantic distance, to measure similarities between semantic concepts in an ontology.

**a. Calculation of the Semantic distance and similarities**

Amongst the different methods having been developed to measure semantic distances between different concepts in an ontology, some like (Paolucci, Kawamura, Payne, & Sycara, 2002) used a qualitative approach by conceptualizing degrees of similarities to categorize semantic distances. Others privileged a quantitative one (M, 1993), notably focusing on the impact to affect a weight on each concept of a considered ontology, based on its depth compared to the top node of such ontology (e.g. OWL:Thing). Hirst and St-Onge (G & D, 1998) first used a formula to measure the semantic distance between two nouns having a path connecting them in WordNet\(^3\), that (Bramantoro et al., 2005) tuned in order to define a semantic distance measure for matching Web Services annotated with ontological concepts. We have re-adapted this last work by defining a semantic distance such as the following Equation 4.

\[
SD = \max \left\{ SD_{C_i, C_j} \right\}, \quad SD_{C_i, C_j} = \frac{G - (W_i + W_j) \times PathLength - NumberOfDownDirection}{G} \times 100
\]

Where:
- \(C_i\) is a concept contains by User request and \(C_j\) is the concept annotating a candidate service
- \(G\) is a constant, usually set to 8 (see (G & D, 1998) for more details).
- \(W_i\) is a weight assigned to the position of the concept \(C_i\) in the ontology and calculated based on (5).

\[
W_i = \frac{n - \ln(C_i) + 1}{n}
\]

Where:
- \(\ln(C_i)\) is the level of the node representing \(C_i\) in the ontology and \(n\) is the total number of levels.
- The PathLength is the number of arcs relating the two concepts in the ontology.
- The NumberOfDownDirection is number of arcs having a down direction from \(C_1\) to \(C_i\) in the ontology.

---

\(^3\) http://wordnet.princeton.edu/
As an example, Figure 21 displays an ontology with \( n = 3 \) levels of concepts. Calculating the semantic distance between two concepts such as \( C_5 \) and \( C_3 \) would then lead to the following parameters:

- \( G = 8 \) (fixed)
- \( W_5 = 1/3 \) (obtained when considering \( C_5 \))
- \( W_3 = 2/3 \) (obtained when considering \( C_3 \))
- \( \text{PathLength} = 3 \)
- \( \text{NumberOfDownDirection} = 1 \) (the number of arcs having a down direction from \( C_5 \) to \( C_3 \))

Which, once inserted in Eq. 4, would compute the following semantic distance:

\[
SD = \max \{SD(C_5, C_3), SD(C_3, C_5)\} = 50 \%
\]

b. Example of applications in the context of IoT-A to discover services

Based on the Service Model presented in section 2.1 of this document, an example of a sensor ontology model is presented below. It represents the different types of Outputs related to different kind of sensors. This ontology uses concepts from SSN ontology\(^4\) and Qudt ontology\(^5\).

The semantic distance method will be applied on semantic concepts from the ontology depicted in Figure 22 to measure their semantic similarities.

\(^4\) http://www.w3.org/2005/Incubator/ssn/wiki/Main_Page
\(^5\) http://qudt.org/
Calculation of depth weight:

The example of Sensor Ontology above has four levels \((n=4)\). Therefore by using Eq. 5, we weight the vertices in the top most level as 1; the second level as \(\frac{1}{4}\), the third level as \(\frac{1}{2}\) and the fourth one as \(\frac{1}{4}\).

Calculation of semantic distance and semantic similarity:

In the scope of IoT-A, to discover an IoT Service, a user sends a request containing a service specification to the system, example: *User A would like to have F functionality at a geographic range P*. In this case, after a first selection based on the geographic range \(P\), we extract the semantic concept from the functionality \(F\) (example in its hasOutput parameter) and try to match it with the semantic concepts from the existing service descriptions.

Below, we illustrate some examples of the calculation method. We consider that the descriptions of the requested service and the candidate service (candidate for discovery) contain concepts from the ontology.

We calculate the semantic distance between concepts related to the service output parameter.

**Case 1: Same Concept**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasOutput</td>
<td>Temperature_value</td>
</tr>
</tbody>
</table>

**Case 2: Subsumption relationship: parents - child**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>A candidate service</td>
<td>Temperature_value</td>
</tr>
</tbody>
</table>
hasOutput Temperature_value
A service
hasOutput InfraredSensor_value
Result from matching
SD (%) \((8 - \frac{1}{4} + \frac{1}{2}) \times \frac{1}{8} \times 100 = 78\%\)
There is 78% of matching between the two services

**Case 3: Subsumption relationship: grandparent - grandson**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasOutput</td>
<td>Weather_value</td>
</tr>
<tr>
<td>A service</td>
<td>Barometer_value</td>
</tr>
<tr>
<td>Result from matching</td>
<td></td>
</tr>
<tr>
<td>SD (%)</td>
<td>((8 - (\frac{3}{4} + 1/2) \times 2 - \frac{2}{8} \times 100 = 44%)</td>
</tr>
</tbody>
</table>

There is 47% of matching.

**Case 4: No Subsumption relationship**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasOutput</td>
<td>Humidity_value</td>
</tr>
<tr>
<td>A service</td>
<td>Led</td>
</tr>
<tr>
<td>Result from matching</td>
<td></td>
</tr>
<tr>
<td>SD (%)</td>
<td>((8 - (1/2 + \frac{1}{2}) \times 4 - \frac{2}{8} \times 100 = 25%)</td>
</tr>
</tbody>
</table>

There is 25% of matching.

**Case 5: Children from same parents in lower level (level 4)**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasOutput</td>
<td>Thermometer_value</td>
</tr>
<tr>
<td>A service</td>
<td>InfraredSensor_value</td>
</tr>
<tr>
<td>Result from matching</td>
<td></td>
</tr>
<tr>
<td>SD (%)</td>
<td>((8 - (1/4 + 1/4) \times 2 - \frac{1}{8} \times 100 = 75%)</td>
</tr>
</tbody>
</table>

There is 75% of matching.

**Case 6: Children from same parents in higher level (level 2)**

<table>
<thead>
<tr>
<th>Requested Service</th>
<th>Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>hasOutput</td>
<td>Weather_value</td>
</tr>
<tr>
<td>A service</td>
<td>Motion_value</td>
</tr>
<tr>
<td>Result from matching</td>
<td></td>
</tr>
<tr>
<td>D (%)</td>
<td>((8 - (3/4 + 3/4) \times 2 - \frac{1}{8} \times 100 = 50%)</td>
</tr>
</tbody>
</table>

There is 50% of matching

Figure 23 presents a summarization of the IoT Service resolution process.
Discussion and critics:

- In this part, this method has been applied on a single ontology (weights were set according to the whole graph of classes/subclasses). However it can be applied to other ontologies with a different graph of classes/subclasses.

- Based on the SD score value, we can deduce if two services are similar or not. For that a threshold needs to be determined. Through experimentation and different calculation cases, results show that two services are considered as similar if \((SD \geq 70\%\)).

- In this method, only concepts related to the same ontology are considered. The case of matching concepts related to different ontologies is not considered here.

- According to what has been defined in section 2.2.1.1, the method that we have presented is an implementation of the “discoverService” interface, with a “ServiceSpecification” parameter containing one single concept defined in a considered ontology. However, the case of a “ServiceSpecification” containing \(k\) concepts would lead to applying our method \(k\) times, with for a given step \(i\), the number of services having been found at the step \((i-1)\). The “discoverService” functionality would in any case have the following signature: discoverService(URI[] someConcepts)→Triple[] were the result is the set of RDF triples constituting the service description and where the ServiceSpecification is a set of ontological concepts represented by their URIs.

Discovery Interface Specification:

The Semantic Web approach for discovering IoT Services can enable a number of forms for specifying the service specification part that constitutes the input to the discovery interface. This assumes that the semantic instantiations of the Service Model are stored in a store, either as RDF triples or OWL serialised as RDF. The query interface can accept a text string containing the concepts, or this can be specified through an interactive interface which suggests concepts to the user as the words are being typed in. However, when the search engine converts the query to a vector of latent factors, it can only recognize concepts that the model has seen in the original training dataset. Concepts that have not been seen before will be simply disregarded. It is suggested that the query interface would have a list of the concepts that the search engine has been trained on so that it can suggest them to the users when they are typing in the query. Alternatively, the search interface could provide a dropdown menu of concepts from which the users can choose.
3.2.1.2 Virtual Entity Resolution

In IoT-A, the Virtual Entity Resolution infrastructure handles the discovery and lookup of Associations between Virtual Entities (VE) and IoT Services. In the Semantic Web approach, the derived Associations are stored in a triple, with the VE ID and the IoT Service ID associated by the VE attribute. The associated VE and IoT Service are related along the VE attribute, about which the IoT Service either provides information on, or is actionable by the IoT Service. These triples could either be stored as RDF triples or as OWL property assertions in a RDF serialisation. The VE Resolution interface could be realised through a semantic query engine that allows discovery of the Associations. The interface itself can be implemented through a Web-based mechanism. Such a mechanism can provide a standardized interface to provide a semantic query service and does away with the need for a user to know the underlying ontological terms.

Association Look-up:
The look-up interface returns all Associations for a particular VE ID and a given VE Service specification. The Semantic Web approach for Association look-up accepts the associating attribute in the service specification part. The format for an interface could then be as follows: http://localhost:8188/api/association/lookup?Attribute={VEServiceSpecification}&veid={VE_ID}

Association Discovery:
The discovery interface returns all Associations for a given VE and VE Service Specification. The Semantic Web approach for Association discovery accepts the associating attribute in either the VE or the Service specification part. Other possible VE Specification inputs could be the entity type. An interface for this would specify the entity or the service attribute as the query parameter. The interface format is as follows: http://localhost:8188/api/association/discover?vespecification={VESpecification}&servicespecification={VEServiceSpecification}

3.2.2 Association Mechanisms

Associations between a Virtual Entity and an IoT Service link a VE attribute to either the IoT Service’s input or output. Thus, according to the Service Model detailed in section 2.1.2, the service may either expose a sensing resource’s capabilities, in which case the service output is of interest, or the service may correspond to an actuating resource, when we are interested in the service input. In this section, we discuss forming the Associations between IoT Services and Virtual Entities along the thematic-spatial-temporal axes, using Semantic Web techniques. This relates to semantically deriving and annotating the relationships between the IoT services exposing resource capabilities that can be relevant to the entity. The thematic dimension is captured in the Entity and IoT Service models (section 2.1) through an entity’s domain attribute and the IoT Service’s input or output properties. For the spatial dimension analysis, we first introduce a semantic model for an indoor environment that can facilitate the varied granularity of spatial Associations possible, notably by “defining” places (e.g. rooms, buildings, etc.) relatively against others (e.g. a room ABC is next to another room DEF, this room ABC has some dimensions, etc.).

3.2.2.1 Indoor Location Model

Although basic inclusion of places within other places can be captured through structures such as directed trees (e.g. places being nodes of such tree and regarding edges, the place mapped to the head of the edge including the place mapped to its tail), a significant one remains not captured: getting knowledge about closeness of different places (e.g. knowing how a room “is close to” another; if a room gives access to another), knowing that some places are gathered into a virtual area (e.g. two rooms belonging to the “R&D division” area), etc. Yet, capturing
such semantics may help in inferring which resources belong to which areas (e.g. a WiFi hotspot “xyz” belonging to “R&D division” area) as well as building and maintaining Associations (e.g. knowing that an entity is still measureable by a sensor even if it has moved to another but nearby room). To get as much knowledge as possible regarding situations where some places could be gathered into a virtual area (e.g. two rooms belonging to the “R&D division” area), we have described an ontological model defining the concepts (and their relationships) involved in the representation of an indoor environment (e.g. composed of rooms, corridors, etc.).

This model contains geo-location concepts and defines common types of building as well as spatial shapes. It represents physical environments (e.g. house, building, etc.) through geometrical forms such as rectangular cuboids or spheres. With properties like “included in” or “nearby”, several smart spaces can be relatively located.

The following Figure 24 gives an overall view of the concepts setting up the definition of a place, as we envision it.

![Figure 24 Indoor location model (edges denotes relations between terms)](image)

In this ontology, three main concepts allow to semantically define how places are organized, how a physical area (i.e. composed of places, etc.) can be virtually cut into pieces and finally allow to give an indication about distances and dimensions of considered places. Such concepts are respectively called “Place”, “Region” and “IndoorPoint”. 
The main relationships shown on Figure 25 can be summed up by the following:

- Places or Regions may contain (or be contained) by other Places or Regions
- Places can be adjacent to other places (note that we have also define sub-properties of such adjacency, given indication whether a place is in the north, south, east, west of another one, but that we have not represented them on the figure for readability)
- A Region has a “base” point defined as an “IndoorPoint”

First and second set of relationships allows to relatively localize places or regions against others. The third relationship allows certain computations to be performed, especially to find distances between different places or between a resource and an entity. Note however that this last kind of computation requires both resource and entity models to import this indoor location model in order to define properties linking an “Entity” or a “Resource” to an “IndoorPoint”.

The concept of “IndoorPoint”:
Before describing concepts such as “Place” or “Region”, it is necessary to describe what we mean here when speaking about the “IndoorPoint” concept. In our vision, an IndoorPoint allows to spatially define regions and places (i.e. to attribute them some coordinates), by taking as a reference one GPS coordinate (e.g. form the entrance of a Building) and by applying a translation vector. It is formally defined by the following predicates:

\[
IndoorPoint = \begin{cases} 
\text{refersToOutdoorLongitude} = 1 & \forall \text{refersToOutdoorLongitude}.\text{Float} \\
\text{refersToOutdoorLatitude} = 1 & \forall \text{refersToOutdoorLatitude}.\text{Float} \\
\text{refersToOutdoorAltitude} = 1 & \forall \text{refersToOutdoorAltitude}.\text{Float} \\
\text{isTranslatedInXBy} = 1 & \forall \text{isTranslatedInXBy}.\text{Float} \\
\text{isTranslatedInYBy} = 1 & \forall \text{isTranslatedInYBy}.\text{Float} \\
\text{isTranslatedInZBy} = 1 & \forall \text{isTranslatedInZBy}.\text{Float} 
\end{cases}
\]

In other words, an indoor point refers to one GPS coordinate from which we apply a translation vector.
Figure 26 shows an illustrative example of a Building composed of regions (here we assume that each represented room has been mapped to a region).
By feeding a semantic engine with such knowledge and by extending it with the sufficient piece of code (e.g. translating GPS coordinates into Cartesian ones and applying the translation vector) one can obtain the base “IndoorPoint” of each room (assuming they are mapped to a Region) and can therefore know how far rooms are from each other. More information about adding such computation can be found in the following reference\(^6\).

Finally, by adding Cartesian coordinates to entities or resources (see Figure 27), relatively to a room they live, a reasoner could also deduce their positions. Note that putting such details may require an additional amount of time to create an instance of this ontology. Indeed, considering the example of a map displaying \(N\) places would imply the following assertions:

\[\begin{align*}
\text{N base points to create (meaning } 3N \text{[x,y,z] coordinates to enter),} \\
\text{Assuming places have all a rectangular form, } 2N \text{ dimensions should be entered} \\
\text{Finally, as mentioned in figure 10, } N \text{ theta angles may be entered as well} \\
\text{This, obviously would also require for the “Place owner” to know such thetas and to be} \\
\text{able to provide coordinates compliant with the chosen referential}
\end{align*}\]

Note finally that such detail is not mandatory and tightly depends on the kind of associations that we want to be obtain. Associations may still be obtained without these details, but with less accuracy. As an instance, a system could logically associate a “temperature” resource and a “fireman” entity because it would have found that both temperature resource and fireman are in the same room and because no other information would have avoided such association to be established (e.g. coordinate of the temperature resource being not in the ontology, area of coverage may not be established as well). Therefore, although this Association is logically right, it may be inaccurate in the reality because of having not taken into account distances separating one from the other (e.g. even in the same room, the temperature resource may be too far from the fireman entity to obtain accurate measures).

\(^6\) [http://www.ontotext.com/owlim/geo-spatial](http://www.ontotext.com/owlim/geo-spatial)
Figure 27 Computing coordinates of a resource (the same would work for an entity)

Note that we assume here that the base reference regarding the Earth was defined by the two following rules:
- Oz axis directed to the center of the Earth
- Oy axis directed to the North and tangent to the Earth

The concept of “Place”:
In our ontology (and as shown in Figure 25), “Place” has a broader meaning which can be narrowed to a Building, a Floor, a Premise or other kind of structures. Some of these concepts are formally defined (based on logical predicates), allowing reasoning to be performed. Some others can again be re-specialized.

As an instance, the following two concepts are formally defined by a set of terminological axioms (i.e. axioms defining the “vocabulary” of an Indoor environment):

\[
\begin{align*}
\text{Building} &= \exists \text{ContainedPlace} \land \exists \text{containsFloor} \\
\text{Floor} &= \exists \text{containsPremise} \land \neg \text{Corridor} \land \neg \text{House} \land \text{isIncluded} = 1 \land \forall \text{isContained}. \text{Building} \lor \text{Premise} \land \neg \text{Room} 
\end{align*}
\]

What the above definitions mean is that:
- A Floor is an entity described as containing some corridors as well as some premises other than corridors or houses, which is included in exactly 1 building or premise (other than of type room).
- Finally, a Building is any entity that can be inferred as being not a place contained by some wider area and as place containing some entity of type Floor.
Obviously above representation implies the definition of other entities, which will not be detailed in this document but that can be found on the Web, where such indoor location model\textsuperscript{7} can be found.

**The concept of “Region”:**
Another major concept of this ontology is “Region”. In our view, a Region allows to either partition or aggregate physical spaces (e.g. creating an “R&D area” gathering 2 rooms, or creating an area “Ben’s office” that includes only a piece of the open office where Ben is working in). In addition, a region can be mapped to one place, allowing a place to have dimensions and coordinates. Finally, one may extend the use of such a concept; to define the area a sensor is active in (e.g. a spherical region centered around a pressure sensor, indicating that within this range, the sensor can produce a result).

The formal definition of a Region says that it has a base point being an “IndoorPoint”, a theta angle (see section explaining “IndoorPoint” concept) and some dimension. It is expressed as follows:

\[
\text{Region} = \left\{ \exists \text{hasDimension}.\text{Float} \land \text{hasBasePoint} = 1 \land \forall \text{hasBasePoint}.\text{IndoorPoint} \land \text{hasTheta} = 1 \land \forall \text{hasTheta}.\text{Float} \right\}
\]

Regions can be specialized in either rectangular cuboids or spheres. Although more complex geometrical forms may exist, we decide to limit our study to these two forms especially to not introduce too much computation when deducing intersections between different regions.

By taking back the properties as defined in Figure 25, Figure 28 displays the following kind of regions that can be created (Note that on this figure, we have chosen to present rectangular forms, but the same can be applied with spheres).

\textbf{Some notes on using OWL:}
Using OWL to create a model comes with benefits but also drawbacks, especially if misused. The first important consideration relies on that OWL is based on Open World Assumption

\textsuperscript{7} Indoor location model for the IoT,
http://webofdevices.appspot.com/models/owl/complex/indoor_loc.owl

Internet of Things Architecture © - 54 -
stating that something not asserted is not necessarily wrong (but may be defined somewhere else).

A direct consequence is that “Region”, “Place” or “IndoorPoint” can be created without respecting their formal definitions; allowing one to instantiate the indoor location model by focusing on interlinking his Places and Regions instead of entering precise coordinates.

A second consequence is that such precise coordinates could then be entered “a posteriori”, typically resulting from the computation of an external program, and be automatically taken into account.

Another aspect of OWL is that it allows reasoning, especially as based on first order logic. The associated drawback, however, relies on that it may require expensive computation to draw such inferences.

3.2.2.2 Associations along thematic, spatial, temporal axes

Figure 29 below illustrates the three dimensions of space, time and theme in which a match is needed in order to trigger Associations.

At first, there is the location (spatial) dimension: the entity needs to be in the service area of a resource to allow an Association between them. In the figure above, an entity is shown to be in Room A (the rectangles at time 1 and time 3). The thematic dimension is interpreted in terms of the domain attributes or the features of the entity, which are observable or actionable by a resource. Thus, the ‘feature’ or thematic link, as can be seen from the Entity (section 2.1.3) and Service Models (section 2.1.2) is defined as an intersection between the entity ‘attributes’ and the input or output features of an IoT service. In the figure, we have shown three observable features outlined with different colours: noise (green), light (yellow), and temperature (blue). The entity located in Room A provides three attributes that can be observed by the resources deployed in Room A. Whenever the dimensions location and feature meet at the same time,
Associations can be established automatically. Thus, in Figure 29, Associations can be established at time 1 between 1) the entity’s noise attribute and the service exposing the noise sensor in Room A, 2) entity’s light attribute and the light sensor in Room A, and 3) the entity’s temperature attribute and the temperature sensor in Room A. The points where all three conditions match are depicted as red circles in the figure. Later on at time 3 the noise and light sensing resources in Room A are not available anymore. Thus the only valid Association that can be made is that between the entity and the temperature sensor in Room A.

Following a match along the feature and temporal attributes, the Association is further refined by considering various levels of spatial relations. Thus, the location matching process is further delineated to reflect the varied granularity of spatial Associations possible. The underlying location model that enables reasoning about spatial relationships has been detailed in section 3.2.2.1. As defined in section 3.2.2.1, the top-level Place concept has been further sub-classed into Building, Floor, FloorSwitcher (e.g. stairs, lift etc.), Premise and Region. Also, a Place can ‘contain’ an instance of another Place. In the model it is specified that premises such as company offices and shops could be composed of corridors, floors and rooms. Rooms are included in floors and premises, while corridors provide access to certain premises (Room or Flat or Shop as well as any extension or other sibling concepts not mentioned in 3.2.2.1). Another object property from the location model that helps to annotate Associations along the spatial axis is the symmetric ‘isAdjacentTo’ property between two Place instances. Furthermore, this property has transitive sub-properties ‘isInEastOf’, ‘isInWestOf’, ‘isInNorthOf’ and ‘isInSouthOf’ to depict the spatial arrangement of two place instances with respect to each other.

For the spatial dimension, Associations are inferred when a VE is within the service area of a resource, as specified in the corresponding serviceArea attribute of the IoT Service exposing the resource capabilities. These Associations can be further categorised into four levels, depending upon the proximity of the entity and the resource:

- **sameLocation**: the entity’s current logical location, as denoted by the ‘local_location’ attribute falls within the area served by a resource’s capabilities (as expressed by its service’s service area). Furthermore, the serving resource’s location matches exactly the local location of the entity.

- **nearby**: the proximity of the resource location to the local location of the entity is not an exact match, but can be inferred by the location model that outlines spatial relationships between locations. For instance, if the VE and the resource are on the same floor as well as the entity’s location being adjacent to the resource location, or the resource is in a corridor that gives access to the room the VE is in, the Association is then annotated as ‘nearby’. If no local location model is available, e.g. in unpopulated places, the proximity may be calculated based on the GPS coordinates of resource’s and entity’s location (assuming GPS coordinates can be obtained).

- **samePremise**: if there is no resource within the floor where the entity is currently located, the Association derivation process looks at the next higher level in the location model. i.e. by considering the concept of Premises from section 3.2.2.1. This can be, for instance, co-location within company offices or houses. The Association is then labelled to be within the same premise.

- **sameRegion**: the resource location matches the global location of the entity, e.g same city, or county or geographically defined regions.

In Figure 30 below, the blue tube depicts an entity placed in a building that contains two floors with three rooms on each floor. The entity is currently located in room U38. Every room is equipped with three resources shown as cubes in each room. It is assumed that the service areas of all resources are limited to the room they are located in. For the resources in room U38 the ‘sameLocation’ property is applicable to express the location matching to the entity; for the other resources deployed on the first floor in the BA building the property ‘nearby’ is appropriate. Any resources located in the rooms on the second floor of the same building would be associated with the annotation ‘samePremise’. The proximity can be inferred through the location model of BA building that states that rooms U8, U38, and U53 belong to 1st floor and

---

Internet of Things Architecture © - 56 -
rooms E12, E22, and E23 belong to 2nd floor and furthermore that 1st floor and 2nd floor are accommodated by BA Building.

Figure 30 Spatial Matching

3.2.2.3 Semantic Web Rules for Associations

The methods for deriving Associations, as specified in the preceding section, have been formalised by defining rules on the semantic representations of the Entity, Resource and Service models (section 2.1). Existing rule languages that harness the expressivity of ontology languages by allowing rule expressions containing ontology classes and properties have been defined in the literature. The most cited of these is the Semantic Web Rule Language (SWRL). SWRL is a W3C submission aimed at combining Web Ontology Language (OWL) and an inference rules language based on RuleML (Rule Markup Language). SWRL rules can be written in terms of OWL classes, properties and individuals. SWRL makes use of pattern-directed invocation of procedures from assertions. It also provides a common language for the data models and the inference mechanism. A SWRL rule contains an antecedent part, called the body and a consequent part called the head. This implies that if all the atoms in the antecedent are true, then the consequent must also be true, i.e., $\text{antecedent} \Rightarrow \text{consequent}$

The rules are defined in an Association model that imports the resource, service and entity instances that form the data set for the rules along with the underlying domain models. The Association model also defines a number of properties reflecting the derived Associations, for instance, ‘sameLocationAs’ that are asserted between the relevant IoT Service-Virtual Entity pairs after rule execution. It is worth noting that the derived property assertions are not inserted into the actual resource or entity models, thus avoiding violating OWL’s open world assumption. However, the inferred knowledge is held within the rule engine, so that following rules can make use of the inferred Associations.

**Thematic Rules:**

The feature similarity rule to infer a match along the thematic axis is defined as follows:
Rule 1 asserts a feature Association, expressed as a ‘sameFeatureAs’ property between matching sensor services and entities, if there is a non-empty intersection between the output of a service, as modeled through its ‘hasOutput’ object property and the attribute types of the entity. In the Semantic Web realisation of the Entity and Service Models, both the entity’s domain attribute type and the service’s hasOutput property ranges map to the cf-feature (climate feature) ontology\(^8\) of the SSN ontology. Thus, a non-null intersection implies that the service output and the type of the domain attribute refer to the same feature. Both the ‘hasOutput’ property in the resource model and the ‘hasAttributeType’ property in the entity model are object properties, which necessitates more than a literal string matching operation, as is available through SWRL built-ins for string comparison, between the possible instance property ranges. Moreover, an entity may have multiple domain attributes and thus, multiple attribute types. In such a case, to infer the intersection between the possible instance sets, the collection operators, prefaced with the sqwrl namespace, are used to bring set theory operations to the rule. First, the class instances of the ‘hasOutput’ and ‘hasAttributeType’ property ranges are grouped into their respective sets using the makeSet operator. Then, each set is grouped by the services and entities, respectively, through the ‘groupBy’ operator. This constructs a new set for each service matched in the service-related query and all the instances of the ‘hasOutput’ property are added to that set. The standard set theoretic intersection operation is then employed to find the intersection between the two grouped collections and the matching service-entity pairs are then associated through the same feature property. A similar rule can be written for actuating services, with the ‘hasInput’ property of the service being considered.

Spatial Association Rules:
The rules to derive location Association build upon the feature Association rule results, i.e. the service and entity instances considered in these rules are the subset that are already associated along the feature axis. Since we are looking at attribute and location similarity at any given point of time, we can consider the temporal variable to be constant. Thus, we consider a plug-and-play architecture where if the resource is mobile, the corresponding service coverage property is updated in the instance data, triggering the rules to be executed and thus the Associations remain relevant. For a mobile entity monitoring case, notification mechanisms have been considered in section 3.3.3.2.

The location rules thus infer different levels of Associations depending on the location proximity of the resources and entities that are already ‘feature associated’ to arrive at the final Association results.

Before the location proximity of resources and entities can be determined, we must ascertain if the VE is currently within the service area of the IoT Service. The following rule associates the service instances whose service area matches exactly the local location of the entity:

\(^8\)http://www.w3.org/2005/Incubator/ssn/ssnx/cf/cf-feature.owl

Internet of Things Architecture © - 58 -
Rule-2 starts by considering only the service-entity pairs that are already inferred to have a feature match, through the sameFeatureAs property, as a result of Rule-1 execution. Then, the service areas of the relevant services are put into a set, grouped by the corresponding service. Similarly, the local location instances of the entities' location property are made into another set, grouped by each relevant entity. Finally, an intersection of the service areas with the local locations results in the matching service-entity pairs to be annotated with the 'locatedinSameCoverageArea' property of the Association model.

Following Rule-2, the sameLocation Association can be derived when the resource and VE are in the same logical Place, as known from the indoor location model. Since results of preceding rules are taken into account in succeeding ones, Rule-3 considers only the subset of service-entity pairs where the VE is within the service coverage area. By transitivity, such pairs have already been associated along with domain attribute axis.

Rule-3:

\[
\text{assoc:locatedinSameCoverageArea}(\text{?s, ?et}) \land \text{sm:exposes}(\text{?s, ?r}) \land \\
\text{rm:hasResourceLocation}(\text{?r, ?rl}) \land \text{rm:hasLocalLocation}(\text{?rl, ?loc}) \land \text{em:Entity}(\text{?et}) \land \\
\text{em:hasA}(\text{?et, ?l}) \land \text{em:hasLocalLocation}(\text{?l, ?loc}) \land \\
\text{swrlb:greaterThan}(\text{?n, 0}) \rightarrow \text{assoc:sameLocationAs}(\text{?s, ?et})
\]

Considering the VE that fall within the service area of an IoT Service, Rule-3 then determines the corresponding local location of the resource, whose capabilities are exposed by the IoT Service. Similar to previous rules, a non-null intersection of the local locations of the VE and the resource then associates the relevant IoT Services and VEs to be in the same location.

Other rules can be formulated along similar lines to derive ‘nearby’ and ‘sameArea’ Associations by using the proximity properties defined within the indoor location model. The ‘sameRegion’ Association matches the resource area with the global location of the entity; this can be the case when the resource’s service area covers the same city where the entity is located.

### 3.2.3 Conclusions

This section has employed Semantic Web techniques for discovering IoT Services and Associations, as well as inferring about new Associations. The machine learning-based IoT Resolution approach leverages the advantages offered by semantic service descriptions for automated search and discovery. Due to the probabilistic nature of this mechanism, it is more suited to performing discovery, than for lookup. However, the machine-learning method as well as the semantic distance-based mechanism could be used for IoT Service look-up. This would involve a query that would take a single ServiceID as input parameter and return a URL corresponding to the "access point" of a Service, assuming that in the triple, the given information about such URL has been updated as many times as required (e.g. if the IoT Service
is redeployed to another platform). The latent factor level automatically captures the semantic relationships between concepts using probabilities. Also, once the model is trained, the derived latent factors can be sent to all the service registries and each service registry can automatically fold-in each new service description. Similar services can then be assigned to the same cluster. Latent Dirichlet Allocation also captures cases of synonyms where logic based techniques would fail. If different description models are used, i.e. not conforming to the data models described in section 2.1, similar concepts might be defined differently. If in the training set, these different definitions appear within service descriptions surrounded by similar concepts, LDA will capture that similarity which indicates synonymy. The service specification input types supported for the discovery interface include simple string descriptions or a template. Any service specification template can be used as long as a parser is provided which can extract the concepts from it. Once the concepts are extracted from the interface, either machine learning or semantic distance based method can be applied.

In the case of machine learning approach, a Gibbs Sampling algorithm is used to generate the distribution of latent factors for that query. The vector describing the input specification in terms of latent vectors can then be used to match it with relevant services by using vector matching techniques.

In the case of semantic distance computation, comparisons between obtained concepts and concepts of services are performed, allowing to return the service having the highest similarity score. The Association mechanism has been defined to discover Associations between VEs and IoT Services in an automated way, along the concepts of theme, time and space. The proposed rule-based Association mechanism builds upon the semantic entity and service data models that have been presented in section 2.1. Thus, the rules can be formulated in terms of the concepts defined in the models. Due to the incremental reasoning performed through the rules, each succeeding rule assertion can take into account the results of the preceding one. Queries can then be formulated to query data asserted by the rule engine.

### 3.3 Federation-based Approach

This section discusses the specific case of a federation based approach using Semantic Web technologies (as presented in section 3.2). In particular, it highlights how nodes composing such federated architecture couple “local” reasoning process with efficient request routing mechanisms to speed up the discovery of IoT Services upon the reception of an incoming query as well as to help in the management of associations between VEs and IoT services.

This section is then divided into three sub-sections. The first one describes in more details the kind of architecture intended behind the term “federation” and in particular the fact that nodes composing the federation denote places. It also presents an overview of the approach used to discover services as well as to discover and maintain Associations, taking into account the particular organization of nodes composing the architecture. Finally it highlights some components required by all resolution frameworks mentioned in the introduction of this document. The second section details the dynamic request routing mechanisms enabling an efficient IoT Service discovery.

Finally, the last section presents how aforementioned indoor location ontology (see section 3.2.2.1) is processed to help in creating, maintaining or rebuilding Associations.

#### 3.3.1 Federation based architecture

Our approach concentrates on a federated architecture to discover IoT Services as well as to discover, create and maintain Associations between such services and Virtual Entities (VEs) of a physical environment. As mentioned in D4.1, “the distributed nature of such architecture has typically to be considered for problems with the root cause of unmanageable complexity that could be caused by heterogeneous environments and where a central one-fits-all approach could not be applied”. In the scope of IoT-A, a very high number of devices (each one offering
some resources further mapped to services) are typically hosted in different places (considered in this case as heterogeneous environments), leading then to a centralized management hardly feasible.

3.3.1.1 Conceptual vision

In our approach, such a federation is conceptually represented as a directed acyclic graph (DAG) with no undirected cycles (some also calling such structure, a “polytree”). We also add the requirement that each non source vertex has an in-degree strictly equals to 1 and an out-degree above or equal to 0 (see Figure 31).

![Figure 31 Federation based architecture as a “polytree” with all non source nodes having an in-degree equals to 1](image)

A graph with vertices labeled (in-degree, out-degree)

Each vertex (or node) composing such architecture represents then a place hosting devices (and therefore IoT Services). A directed edge between two nodes, expresses the fact that a place contains other places. As an illustration, consider a mall that would contain a coffee shop as well as a jean store. In this illustration, three nodes would represent the mall, the coffee shop and the jean store respectively. One edge would go from the mall to the coffee shop and another one from the mall to the jean store.

Although such representation enables a certain amount of semantics to be obtained (basic inclusion of places in another place), a significant one remains not captured: getting knowledge about closeness of different places (e.g. knowing how a room “is close to” another; if a room gives access to another), etc. Yet, capturing such semantics may help in inferring which resources belong to which areas (e.g. a WiFi hotspot “xyz” belonging to “R&D division” area) as well as building and maintaining Associations (e.g. knowing that an entity is still measureable by a sensor even if it has moved to another but close room). To get as much knowledge as possible regarding such situations, we employ the Indoor Location Model as defined in section 3.2.2.1.

3.3.1.2 Proposed approach regarding IoT Service discovery and Association management

We propose to work on each node of this architecture (recall that a node represents a place here) by instantiating the indoor location model of section 3.2.2.1 and by processing the descriptions of both IoT Services and VEs that the place is currently aware of, to establish the following:
• Clusters of services behaving similarly.

• A routing table redirecting a service discovery request (SDR) to the connected node having the maximum likelihood of handling it properly.

• Associations between an IoT Service and a VE: Inferring or maintaining the fact that an IoT Service can provide information or actuate on some attributes of a VE (e.g. humidity of a room, size of a table, temperature of a body, etc.) by feeding a rule based mechanism with their respective descriptions as well as the ones describing places where both entity and resource live.

In addition, we propose to speed up the discovery of Associations, by enabling each node of our architecture to share what it has learned (e.g. new Associations discovered) between selected nodes. We assume here that pieces of description of both VEs and IoT Services can be processed in order to extract their meaning. In particular, in the scope of IoT-A, we have considered semantic descriptions, based on the set of standards as defined by the Semantic Web community (in particular OWL). Consequently – and because the Federation approach is having an overlap with the pure Semantic Web based approach – first capability expected for each node of our architecture will use some methods described in section 3.2. The mechanism consisting of feeding a rule based system with Semantic Web descriptions must also be taken as tuning the one presented in section 3.2 with peculiarities of a federated architecture.

Next sections will then rely on federation specificities (mainly related to architecture as well as to the knowledge for each node about proximity of other nodes) to detail the following:

• Process coupling some methods presented in section 3.2.1.1 with a clustering algorithm.
• Process enabling to forward a request from nodes to nodes to discover an IoT Service.
• Process enabling to monitor Associations.
• Process enabling to share Association discovered between selected nodes

### 3.3.1.3 Necessary components to deploy on each node of the architecture

**Deploying a triple store on each node**

As described in sections 2.2.1 to 2.2.3, all frameworks need to keep track of information about IoT Services, VEs and Associations. Although implementing such functionalities may differ between frameworks, providing information about IoT Services and VEs and Associations as well as managing Associations in a federated architecture implies the ability for each node to maintain a database containing the current state of affairs about:

• The IoT Service and the VEs currently in the place,
• The set of determined (asserted or inferred) Associations.

This implies a database to be deployed on each node, keeping track of such “local” knowledge (i.e. local to the node). To be compliant with an approach based on Semantic Web descriptions, a triple store\(^9\) has to be setup on each node, allowing RDF triples – the atomic unit of Semantic Web descriptions – to be stored and updated. Triple store management systems already exist and some of them propose free versions, most of the time limited by the number of triples that they can manage. However, because of the distributed aspect of a federated architecture, one will note that most free versions are far sufficient. As an instance, the free version of AllegroGraph\(^10\) can manage until 50 million triples.

---


Once deployed on each node, attention must be paid regarding CRUD\(^\text{11}\) operations proposed by a triple store.

In a general case, reading a triple in the store is a quite fast operation that does not entail high computation time. Creating, updating or deleting a triple can however require the full set of inferences to be recomputed. Fortunately in the scope of IoT-A, some strong assumptions can be taken into account to consider that no reasoning will be performed “at runtime” (i.e. when resources or entities join or leave a place):

- Resource and entity model does not allow reasoning (meaning DL reasoning).
- Indoor location allows reasoning, but such reasoning will be performed at design time (i.e. when federation nodes are created).
- All decision process regarding Association management will be performed through rule sets.
  - Adding a new VE description in a node (e.g. when a VE joins a node) does not involve any inference computation,
  - Removing VE triples from a node will imply to remove all Associations between this VE and any IoT Services, but keeping the rest unchanged
- All produced results will be triples associating a VE to an IoT Service and therefore, not requiring any sort of inference to be recomputed (the rest of the knowledge base keeping unchanged).

Considering all of the above, a triple store can be setup to explicitly not recompute inferences after having performed any CRUD operations. If not recomputed, CRUD operations can then be compared to ones of a classical database system.

**Deploying a Semantic Web engine on each node**

In addition to a triple store, a second component that must be deployed on each node of our federated architecture is a Semantic Web engine – such as Pellet\(^\text{12}\) or HermiT\(^\text{13}\) – in order to extract RDF triples from descriptions of either VE or IoT Services. Note however that some stores already come with an embedded one.

### 3.3.2 Discovery Mechanisms

#### 3.3.2.1 IoT Service Resolution

As already mentioned in the previous subsection, each node composing our federated architecture handles VEs and IoT Services having been described using Semantic Web technologies. However, because of peculiarities (see above section 3.3.1) tied to our federated approach, this section extends what has been presented in 3.2.1.1, by:

1. Applying the semantic distance approach on each node of the Federated architecture.
2. Using an algorithm doing hierarchical clustering\(^\text{14}\) on the results previously obtained.
3. Creating tables – using obtained clusters – on each node of the architecture, to route an incoming Service Discovery Request (SDR) within the architecture, in order to reduce the time spent in searching an IoT Service matching a query.

**Clustering mechanism within a vertex and routing table construction**

Our clustering technique is coupling the semantic distance introduced in section 3.2.1.1 with an algorithm doing hierarchical clustering. Clustering is applied to create groups of IoT Services related to similar semantic concepts within a node (see Figure 32).

---


\(^{12}\) Pellet, OWL2 Reasoner for Java, [http://clarkparsia.com/pellet/](http://clarkparsia.com/pellet/)

\(^{13}\) HermiT, OWL Reasoner, [http://hermit-reasoner.com/](http://hermit-reasoner.com/)

Clustering IoT Services avoids performing matching with each individual service presents in each node. The query is matched only to the services that belong to the best matching cluster within a node. This makes the discovery process more accurate. Hierarchical clustering is used to produce hierarchies of clusters related to the semantic concepts of the used ontology. In this approach the bottom – up¹⁵ algorithm is used.

The following Figure 33 is an example of what could be envisioned by applying our approach, where IoT Service descriptions are linked to semantic concepts of a considered ontology (such as the ones presented in Figure 22):

<table>
<thead>
<tr>
<th>Service (S_i)</th>
<th>Semantic Concept</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>Thermometer_value</td>
</tr>
<tr>
<td>S2</td>
<td>Temperature_value</td>
</tr>
<tr>
<td>S3</td>
<td>Weather_value</td>
</tr>
<tr>
<td>S4</td>
<td>Motion_value</td>
</tr>
<tr>
<td>S5</td>
<td>Led_LuminosityValue</td>
</tr>
<tr>
<td>S6</td>
<td>Altimeter_value</td>
</tr>
<tr>
<td>S7</td>
<td>Position_value</td>
</tr>
<tr>
<td>S8</td>
<td>Humidity_value</td>
</tr>
</tbody>
</table>

Figure 33 Example of IoT-A services and their related concepts

In this example, semantic distance similarities between each pair of services could be calculated based on Eq. 4, leading to the following similarity matrix: SimMatrix_{service} = (SD_{ij})_{n+1 x n+1} built as follows:

Based on the semantic distance values calculated in the SimMatrix above, IoT Service descriptions can be clustered hierarchically. Applied on the example, the bottom-up algorithm would start by creating one separate cluster for each service. Repeatedly, the two most similar clusters (values in red in the SimMatrix, see Figure 34) would then be determined and merged together into a new cluster. The algorithm would finally terminate with one large cluster containing all others. Figure 35 provides such hierarchical clusters applied with values of aforementioned example.

In level 1 of the clustering process, the most similar IoT Services are clustered together resulting in the creation of 3 clusters: CL1 = {S1, S2}, CL2= {S3, S8} and CL3= {S6, S7}. After this first step, S4 and S5 do not belong to any cluster. CL1 contains IoT Services related to concepts (thermometer value and temperature value) which are close concepts in the Sensor Ontology (see Figure 22).

In level 2 of the clustering process {S1, S2} and {S3,S8} have closed semantic distances to each other, and therefore are merged together in the same cluster CL4 = {S1, S2, S3, S4}. The same applies between S4 and {S6, S7} resulting in a cluster CL5= {S6, S7, S4}, while S5 still does not belong to any cluster. Finally, and still using the Sensor Ontology, CL4 and CL5 are
merged together as containing respectively service descriptions linked to the concept of **Weather** and **Motion**.

The hierarchical clustering applied to each node of the federated architecture, leads to mostly distinct clusters. We assume that each generated cluster has a unique ID that is managed internally. The clusters can contain one or several IoT Services. Thus, each cluster contains concepts from the defined ontology.

To know to which cluster the request needs to be sent, a routing table is associated with each node. It associates each cluster to its semantic concepts and its ID as shown in the figure below (see Figure 36).

When a node receives a user request for a service description, the semantic concept is extracted from the request and matched with the concepts in the routing table using semantic distance. In the case of a match i.e. if compared concepts have a similarity distance above a given threshold set to 70% (see section 3.2.1.1) the request is redirected to the suitable cluster based on its ID. A local matching can then be re-performed using the semantic distance similarity.

This method allows looking inside the most suitable cluster of services and avoids scanning each IoT Service description contained in the node. Such a mechanism would be practical in IoT scenarios where service repositories related to a location can contain millions of services. Matching every of them each time a query is submitted would require too much processing power and time, making the solution unscaleable. Clustering services based on their semantic concepts and creating related routing tables would consequently scale the discovery process while still keeping the same accuracy.

**Figure 36 Routing table construction in each vertex**

**Routing tables to forward requests in the federated architecture**

To discover IoT Services matching User expectations in the whole federated architecture, we build and use routing tables. One such table is present on each node composing the federation to enable a User request containing semantic concepts received by a given node, to be directed to the node having the highest probability to contain a matching IoT Service.

The figure below (See Figure 37) illustrates the process of discovery and routing table construction in the federated architecture through an example. In this example, **N1** and **N2** are managed nodes and **N3** and **N4** are managing nodes as explained in the federation concept (see section 3.3.1.1).
In N1 and N2, clusters of similar services are created and identified with an ID as explained previously. A routing table is created on each node and contains two parts:

- **A request treatment part:** Receives the user request, extracts the concept based on the ontology and then, matches the concept with the existing concepts in the node. If a concept is found then the request is sent to the suitable cluster based on its ID. A local matching within the cluster is performed.

- **The forwarding part:** If the resource related to the user request is not found in the node (N1 here) then the request is forwarded to the managing node (N3 in this case).

The managing nodes N3 and N4 contain the Lowest Common Ancestor (LCA\(^{16}\)) semantic concepts extracted from each cluster in N1 and N2. Following the example,

- LCA of C1, C2, C3 in N1 returns C1′
- LCA of C4, C5 in N1 and C7, C8 in N2 returns C3′ in N3.

The same process is done as well for the other nodes of both N1 and N2 (C9, C10 and C11).

Finally, a routing table is created in N3 which contains two parts:

- **The Request treatment part:** Extracting the semantic concept from the request and sending it to the suitable node according to the table values (e.g. a request linked to concept C′3 would be sent to N1 and N2).

- **The Forwarding part:** If no concept in the table matches with the requested concept then the request is forwarded to its father (N4 in this case).

---

16 Lowest Common Ancestor in the ontology (LCA): gives the lowest common ancestor of two classes or more in a given ontology. This kind of function is found within the Jena API ontology.
From the interface definition point of view, discovering a service through our approach is exactly the same as what has been said in section 3.2. Therefore, the interface defined in section 2.2.1.1: `discoverService(ServiceSpecification) → ServiceDescription[]` could be mapped like the following:

- `ServiceSpecification`—some ontological concept(s) defined in the considered IoT Service model and its transitive import closure (i.e. the other ontologies that this model uses, such as the SSN ontology or the Resource Model, see Figure 3).
- `ServiceDescription[]`—an array of a set of RDF triples, each set constituting the RDF or OWL file of a service

### 3.3.2.2 Virtual Entity Resolution

While considering a particular architecture (federation), our approach relies on Semantic Web technologies to process descriptions of both IoT Services and VEs, in order to derive Associations. Consequently, whether statically asserted or discovered, Associations are expressed in the form of an RDF triple:

- If statically asserted, then an Association has to conform to our approach using triple stores and has to be expressed using RDF triples.
- If discovered, then the Associations result from a set of rules having been passed in a semantic engine (see next section 3.3.3 for details) and are then expressed through a set of RDF triples.
In any case, such triples are registered in a triple store – as introduced previously in subsection 3.3.1.3 – that can be queried using either a SPARQL endpoint\(^\text{17}\) or a front-end program based on a different protocol (as long as incoming requests conform to this protocol), to either look up or discover Associations.

Working with Semantic Web descriptions offers also the benefit that all entities are named with an URI. Therefore and according to the interfaces and their parameters defined in section 2.2.2.1, the following mapping could be established:

- **VE_ID**→**URI** (defined in the semantic description of the considered VE)
- **VEServiceSpecification**→Ontological concept(s) defined in the considered IoT Service model and its transitive import closure (SSN ontology or Resource model)
- **VESpecification**→Ontological concept(s) defined in the considered Entity model and its transitive import closure
- **Association**→A set of RDF triples containing at least the three first statements:
  - **VE_ID isAssociatedWith ServiceID** linking a Virtual Entity to an IoT Service
  - **AssocID hasDomainAttribute URI** where AssocID refers to the current RDF entity containing this fragment and where URI points to a DomainAttribute. This triple asserting around which attribute the Association has been established.
  - **AssocID provides [INFORMATION | ACTUATION]** where INFORMATION and ACTUATION would be two constants, defined by a URI and contained by a semantic representation of the Association model. This triple enabling to know whether the IoT Service of this Association provides information or actuate on the associated VE.
- **Other statements** may also complete the RDF fragment, to express additions that has been discussed in section 2.1.4.
- **ServiceID**→**URI** (defined in the semantic description of the considered IoT Service)

This would then imply the following functionalities offered by the VE Resolution Framework:

- **lookupAssociation**(URI a**VE_ID**, URI a**Concept**)→**Triple**, where **Triple** means an RDF triple
- **discoverAssociation**(URI[] some**VE_Attributes**, URI[] some**ServiceAttributes**)→**Triple**

Functionalties allowing an Association to be inserted, updated or deleted will be kept internal to the federated architecture and mostly used according to what has been obtained through the process described in next section 3.3.3. However, for statically asserted Associations, such functionalities may be offered to an “administrator” of the architecture. In our approach, such functionalities would rely on the update\(^\text{18}\) mechanism of SPARQL 1.1, allowing triples to be inserted, updated or deleted. A simple SPARQL update query could then be issued to manage triples (and therefore Associations) in the nodes composing our architecture. The following SPARQL code is what could be sent to a triple store in order to insert an Association.

Prefix assoc: <http://models.iot-a.eu/association.owl>

INSERT DATA

{
    <http://[node_ip_address]:[port]/service/sensor_service1234.rdf>
    assoc:isAssociatedWith
    <http://[node_ip_address]:[port]/ventity/ve5678.rdf>
}

\(^{17}\) [http://semanticweb.org/wiki/SPARQL_endpoint](http://semanticweb.org/wiki/SPARQL_endpoint)

\(^{18}\) SPARQL 1.1, W3C working draft, [http://www.w3.org/TR/sparql11-update/#insertData](http://www.w3.org/TR/sparql11-update/#insertData)
Deleting an Association from the store would consist of replacing the keyword \texttt{INSERT} of the previous code by \texttt{DELETE}. Finally, and according to the SPARQL 1.1 specification, an update would consist of applying both operations (\texttt{DELETE} then \texttt{INSERT}).

Subscription, notification and unsubscripton functionalities do not need to be detailed as involving an ID of subscription managed internally as well as an Association parameter presented while detailing the mapping between interfaces defined in section 2 and our approach (see previous page).

### 3.3.3 Association Mechanisms

Creating and maintaining Associations in a Federation based architecture relies on implementing for each node of this architecture, a rule-based mechanism processing Semantic Web descriptions of both resources and entities. It therefore, refers to what has been presented in section 3.2 of this document.

However, while section 3.2 does not assume a particular deployment approach, we have tuned the management of Associations in the particular case of a federated architecture, by using the ontological model previously defined (section 3.2.2.1). By processing such model, we enable local (i.e. either per node or involving closely located nodes) Association monitoring. Finally by coupling such model with a notification mechanism, we allow a global mechanism updating (and therefore maintaining) Associations.

In detail, to manage Associations in a federation, we execute on each node a set of customized (i.e. using instantiations of aforementioned location model) Semantic Web rules. These rules are triggered upon the reception of an alert giving information on a Physical Entity (or Resource) – considered in IoT-A –that would have joined (or left) this node. We assume here that such notification message is out of the scope of this document but that it would consist of obtaining the VE description associated to the PE (resp. the IoT Services exposing the resource). Then, based on what has been obtained by passing the rule set, we populate results (consisting in some RDF triples) to closed nodes that update their local triple stores (i.e. databases for RDF triples).

**On each node, when notified of a resource or entity change:**

- **Get triples from description**
- **Execute rules known by a node**
- **Filter results by closed nodes and populate**

**On each node, when notified about results computed by another node:**

- New message reaching a node
- Msg content: some selected results

**Figure 38 Executing rules when a resource/entity joins/leaves a place then updating triple store of closed nodes**

This section is then divided as follows. First we detail the use of customized Semantic Web rules as well as the necessity to tune classical Semantic Web rule engines with additional features allowing notification messages to be initiated. Then, we present a simple notification mechanism and how notification messages are conveyed through our architecture.
### 3.3.3.1 Customized Semantic Web rules

In the literature, existing Semantic Web rule languages combining OWL and rules have been defined. The most well known is SWRL\(^{19}\), enabling the use of OWL constructs (classes, individuals or properties) to form “processable” rules.

Our approach, complying with section 3.2, is based on such language and uses a classical and open-source semantic engine, to process SWRL rules. However our approach brings complements to such rules, to enable:

- A degree of confidence to be associated to each found Association
- Association results to be shared between nodes, by inserting specific rule built-ins throwing the initiation of notification messages.

Both points come with the idea of being able to detect behavioural patterns in order to anticipate the creation of Associations, avoiding a set of rules to be replayed. As an instance, assume that we have detected the following pattern: “Many entities go by Room A, then Room B then Room C”. Knowing that an entity has moved from Room A to Room B may be sufficient to associate the VE of such entity with IoT Services of some resources in room C, upon the reception of a message notifying that the entity has left Room B. This would allow the node attached to Room C to not replay its rule set. Such pattern detection would typically come with an associated score, increasing or decreasing depending on what has been learned.

The following describes – using SWRL syntax – some customized rules, extensively relying on the properties provided in the indoor location model as well as on different built-ins, later explained.

#### Table 2 Some customized SWRL rules

<table>
<thead>
<tr>
<th>Rule ID</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>res: Re source(?r) ∧ loc: Place(?p) ∧ loc: givesAccessTo(N, ?p) ∧ alert: notify(?r, loc : JOIN) → iota: notify(NODE, ?r, loc : JOIN);</td>
</tr>
<tr>
<td>2</td>
<td>res: Re source(?r) ∧ loc: Place(?p) ∧ loc: givesAccessTo(NODE, ?p) ∧ alert: notify(?r, loc : LEAVE) → iota: notify(?p, ?r, loc : JOIN);</td>
</tr>
<tr>
<td>3</td>
<td>res: Re source(?r) ∧ loc: Place(?p) ∧ isAdjacent(NODE, ?p) ∧ alert: notify(?r, loc : JOIN) → iota: notify(NODE, ?r, loc : JOIN);</td>
</tr>
<tr>
<td>4</td>
<td>res: Re source(?r) ∧ loc: Place(?p) ∧ isAdjacent(NODE, ?p) ∧ alert: notify(?r, loc : LEAVE) → iota: notify(?p, ?r, loc : JOIN);</td>
</tr>
<tr>
<td>5</td>
<td>res: Re source(?r) ∧ loc: Pattern: isNext(NODE, ?p) → iota: notify(?p, ?r, loc : WILL _ JOIN);</td>
</tr>
</tbody>
</table>

\(^{19}\) SWRL, A Semantic Web Rule Language combining OWL and RuleML, [http://www.w3.org/Submission/SWRL/](http://www.w3.org/Submission/SWRL/)

Internet of Things Architecture © - 71 -
Above

Table 2 shows the use of different prefixes, concepts, properties, individuals and built-ins. Some explanations for each of them are given hereafter.

**Prefixes:**
- "res" is a prefix used to refer to the resource model.
- "loc" is a prefix used to refer to the indoor location model.
- "alert" is a prefix used to refer to a model describing the action for an IoT Service to join or leave a place (as mentioned in this section, such mechanism is not presented here).
- "iota" is a prefix used to refer to customized built-ins triggering notification messages.
- "pattern" is a prefix used to refer to customized built-ins searching for behavioural patterns.

**Concepts:**
- "res:Resource" defines a resource in the Resource model.
- "loc:Place" refers to the same concept than the one explained in section 3.2.2.1.

**Properties:**
- "loc:givesAccessTo" is defined in the indoor location model and means that a Place can give access to other places.
- "alert:notify" means the action of a resource (or an entity) to join or leave a place.
- "loc:isAdjacent" is defined in the indoor location model and means that a Place can have adjacency places (e.g. a room on the North of the considered one, etc.).

**Individuals:**
- "loc:LEAVE", "loc:JOIN" and "loc:WILL_JOIN" are constants defined in the indoor location model.
- N only refers to an Individual declared in the local "indoor location" model instantiation and representing the current node executing the rule set.

**Built-ins:**
- "iota:notify" results in sending messages to closed nodes about a fact that has (or will) happen. The probability score is equal to 1.
- "iota:pnotify" results in sending messages to closed nodes about a fact that may happen with a certain probability. Getting such probability will not be described in this document. Thus, the overall idea is to return a score taken into account the number of nodes that are "accessible from" or "adjacent with" a considered node.
- "pattern:isNext" checks if the next node that a resource will join is a given node and returns a probabilistic score issued from what has been learned.

**Some remarks:**

Such built-ins require additional programs to be developed in order to be handled properly. Details about their implementation will however not be given in this deliverable.

Assuming a deployment where the different built-ins and models are known (or present), allows to envision other rules to be created and processed by the Semantic Web rule engine.
### 3.3.3.2 Notification mechanism

In order for nodes to share knowledge with other nodes in their vicinity – by vicinity, we meant here what can be obtained by processing the description of a node implementing the indoor location model – a notification mechanism has to be defined and implemented on each node. As shown by Figure 38 (bottom layer), a node updates its triple store upon the reception of a set of selected results. Such results have previously been obtained – using a classical RDF API\(^{20}\) – then filtered – see above subsection 3.3.3.1 – by another node, upon the execution of a set of rules. Results are then a set of triples (recall that a triple is the base unit in the Semantic Web). To enable these triples to be taken into account by a store, we rely on SPARQL 1.1 specification that defines a dedicated section\(^{21}\) to update (through “insert” or “delete” operations) a graph. The consequence is that the content of a notification message consists of a SPARQL query.

Regarding how to convey messages from a node to another, we rely on the architecture brought by the Federation (recall Figure 31 as well as D4.1). Therefore, whatever the recipient of a message issued by a given node, the route followed by such message is exactly the list of nodes that need to be crossed, in order to find a “common manager” of both nodes (see following picture).

- \( F = \{ N_i \} \) denotes the nodes of this Federation
- In the picture, different levels of nodes are represented. Some (the grey ones) manage other nodes. Others (the blue ones) do not.
- Arrows drawn in this picture represent a path between \( N_2 \) and \( N_6 \). Such path is equivalent to the list of ancestors of either the issuer or the recipient.
- Only one node is a common ancestor to both nodes.

![Figure 39 A path between two nodes, using Federated approach](image)

Such route is obtained by processing the indoor location instance of the whole Federation and by building a graph relying on the transitive nature of the property “contains” (see Figure 25). The overall idea to build such graph is to use the anonymous property “inverse of contains” (provided by a semantic engine), allowing to find the ancestors of both the issuer and the recipient nodes. Hence, with this property, we build 2 sub-graphs, one starting with the issuer and the other one starting with the recipient. Each time we found ancestors, we check if two sub-graphs have a common node. If so, we merge them into a single graph, which gives the shortest – and only – path between both nodes. Due to the particular nature of a federated infrastructure (recall that in Figure 31 we said that the Federation was seen as a DAG with no undirected cycles), we are sure that the algorithm converges to a unique solution.

As a consequence, an envelope wraps each message’s content by such obtained list, and then enables any node receiving a message to convey it properly until the final recipient. Another consequence of routing messages by this approach is the ability for a managing node to be aware of the Associations found by its managed nodes. It therefore has a more global view of the system and may even be used to answer queries on behalf of its managed nodes (these details however, will not be described in this document).

The result is a simple notification mechanism consisting of the generation of \( K \) messages, each one containing a payload composed of a simple envelope to be routed properly as well as a

\(^{20}\) The OWL API, http://owlapi.sourceforge.net/
\(^{21}\) SPARQL 1.1, W3C working draft, [http://www.w3.org/TR/sparql11-update/#insertData](http://www.w3.org/TR/sparql11-update/#insertData)

Internet of Things Architecture © - 73 -
content using an “almost” standardized language (SPARQL 1.1 specification is currently a working draft) to update triple stores.

3.3.3.3 Overall picture

Figure 40 describes all the aforementioned points. We assume here that there is a notification system warning a place if an entity or an IoT Service has come or is leaving.

![Diagram of message flow](image)

**Figure 40 Sharing Associations between nearby nodes: Overall process**

From the semantic description of a VE tied to a Physical Entity (resp. an IoT Service related to an exposed resource), some triples are extracted (if joining the place) or retrieved from the Triple Store (if leaving). Using classical semantic engines generates triples, while querying the store with VE (resp. IoT Service) URI (in our approach, such URIs would respectively be VE_ID for a VE and ServiceID for an IoT Service) allows to retrieve triples. Once this step is done, triples feed a Rule Manager (that can also be found in most of semantic engines) that (re)plays the set of rules to check whether some Associations should be created, updated or deleted. This leads to a list of resulting triples that represents the “current state of affairs” for this node. Such resulting triples are then sent to the store for update purpose. Finally, resulting triples go to a Result Dispatcher. For each one, the dispatcher processes the indoor location model local to the node and deduces a list of nodes supposedly interested by the result. It then creates a message with the list of nodes to cross, obtained as mentioned in section 3.3.3.2 and with the resulting triple as content. A message is then forwarded from nodes to nodes, using the list of nodes provided by the message issuer and following routes created by the federated architecture, until it reaches the message recipient. Once reaching its recipient, the message’ content consisting of a triple is extracted and pushed to the node’ store for update purpose.
3.3.4 Concluding remarks

3.3.4.1 Generalities and similarities with Semantic Web approach
This section has presented a hybrid approach using Semantic Web technologies and routing mechanisms in a federated architecture. Such mechanisms allow speeding up the discovery of IoT Services upon the reception of an incoming query as well as efficiently managing the Associations between VEs and IoT Services. Triple stores are also an easy way to retrieve information about Associations between VEs and IoT Services.

This approach has primarily been designed with the discovery of IoT Services as well as the management of Associations in mind. Through the different section presenting our approach, we have seen that discovering IoT Services, looking up, discovering and managing Associations were possible features. Also, according to the definition of the interface allowing to look up for an IoT Service, this seems feasible by our approach. Resolving services, however, is not tackled by our approach.

On each node of the architecture, local reasoning mechanisms are used to perform IoT Service discovery or Association management. From this point of view, mechanisms are similar to what has been presented in section 3.2:

- Computing a semantic distance between ontological concepts associated to different IoT Services starts the process of service discovery in both sections.
- Executing a set of SWRL rules against a set of triples in a knowledge base (or store) is the overall idea in both sections.

This implies that both sections 3.2 and 3.3 will use the same interface signatures to interact with third parties querying for information. Also, and by only considering one node of the architecture, advantages and drawbacks of using Semantic Web technologies in our approach are the same than what has been said in section 3.2.

3.3.4.2 Specificities of the federation approach
Our approach, however, can overcome some drawbacks linked to a pure Semantic Web approach, due to that we consider a particular deployment infrastructure, resulting on a federation of nodes, each one mapped on a physical environment (e.g., buildings, rooms, etc.). The following sums up the important points described above and the benefits that an overall system would get in considering our hybrid approach instead of a pure Semantic Web based one.

**Distributed knowledge bases:**
Considering a set of nodes allows reasoning mechanisms local to each node to be setup. Consequently, it enables knowledge (i.e., what a local semantic engine has deduced regarding service discovery or Association management) to be locally stored, avoiding a single and centralized database (or triple store) to be flooded by queries (and data).

**Sharing knowledge between nodes:**
Due to the particular mesh composing a federation, simple notification message mechanism can be setup, to share common interesting knowledge across different nodes (see section 3.3.3). Coupled with specific rules aiming to detect patterns of “mobility” for entities or resources, the process of sharing knowledge between nodes may allow some Associations to be predicted instead of deduced from a rule engine.

**Dealing with mobile entities or resources:**
Our approach uses all necessary technology (SPARQL1.1 Update functionality as well as triple stores deployed in each place), to handle the fact that some entities or resources can move across different places, requiring Associations to be updated, inserted or removed. This “in-
motion” attribute is reflected in the triple stores that can be updated thanks to SPARQL 1.1 Update support.

**Customizing clusters of services for discovery:**
To discover IoT Services, our approach couples semantic distance with a cluster technique tuned to take into account the hierarchical distribution of nodes in a federated architecture. On a given node, the established clusters gather similar – as the term is defined in section 3.2.1.1 – Services and are labelled with a set of ontological terms. From these terms, we retrieve their last common ancestor (LCA) that we send to the node managing this one. By processing so, the managing node has a partial knowledge about the kind of services that are handled in the nodes it manages (directly or indirectly). With such knowledge, a routing mechanism is associated to efficiently select a subset of nodes, determined as good candidates to serve the query with appropriate services.

### 3.3.4.3 Additional remarks
What has not been discussed in this section, however, relies on the fact that places (and therefore nodes) can change over time: a new “room” created, a new “shop” opening, etc. Although it would require more details, below is summed up how such a change could be handled:

- Adding a node in the infrastructure with its own instantiation of the indoor location model, by hanging up to another node (its “manager”)
- On the manager side, getting the indoor location instantiation and notifying all concerned nodes (i.e. all nodes mentioned in this instantiation) that a new “neighbour” node has just joined.
- On each node concerned by this notification, updating its own indoor location instantiation to further send interesting knowledge to the new one.

### 3.4 P2P Infrastructure – DHT Approach
This section introduces distributed hash tables (DHT) as one powerful distribution paradigm of Peer-to-Peer networks that implements content-addressable data access. Traditional array-based data types require numbers as indexes and, hence, need fast sorting and binary search to access single data components. The efforts are $O(n \cdot \log n)$ and $O(\log n)$, respectively. In contrast, hash tables allow constant access effort and help users to address the desired data by itself: For example, to access a phone number of a person, the person name can be used as the index of the table component which contains a record with the person name and the related phone number.

#### 3.4.1 Hash Tables
In computer science, a hash table is an array-like data structure that organizes data as key/entry pairs. In contrast to arrays, where integer keys address the position of an entry, so-called key objects are used to determine the address where the entry object is located. Therefore the key object is mapped to the address of the entry object by the application of a hash function. Typically string objects are used as key objects.

Writing to a hash table could proceed by a function call `put(keyString, entryObject)`. The read function `get(keyString)` would return the entry object – if it would have been written to the hash table before. And the function call `remove(keyString)` would clear the value from the hash table addressed by `keyString`.

It is possible that the hash functions maps multiple key objects to the same address. In that case different strategies exist to deal with this kind of collision. Either the existing entry object at that address is replaced by the new entry object or a data structure (e.g. a list), located at that address, manages all entry objects, whose respective key objects are colliding.

The hash function applies a mapping from a key to an address. Here it is an important criteria that the mapping is evenly distributed on the set of all possible addresses, so that preferably the
whole address space is used as best as possible and that collisions are as seldom as possible. With respect to the limitation of the hardware platform used for IoT, the computational effort is another criterion for the hash function. In this context the hash function maps a character string with an arbitrary, but limited length to an address within a fixed range, which can be understood as an integer number. In the following section, properties of the hash function are described.

### 3.4.2 Hash Functions

In this context a hash function maps a character string to a natural number satisfying the following requirements:

- The hash function result is fully determined by the string being hashed, i.e. there is no further input other than the input string used for the calculation. Otherwise, there might be different results for the same input string at different points of time or on different machines.
- The hash function uses all characters of the input string. Otherwise, strings "aaa", "aza" and "a_a" could map to the same integer if the second character was ignored.
- The hash function uniformly distributes the data across the entire set of possible hash values. The full target range of the hash function is covered by corresponding input strings; i.e. the probability to hash one string to a particular number is the same as that of any other string and number. Hence, this requirement guarantees the avoidance of accumulation points – so-called hotspots – and thus selective overloads.

The hash function generates very different hash values for similar strings. The basic rule is that if two input strings differ in one single bit, then in this context a hash function maps a character string to a natural number satisfying the following requirements:

- The hash function result is fully determined by the string being hashed, i.e. there is no further input other than the input string used for the calculation. Otherwise, there might be different results for the same input string at different points of time or on different machines.
- The hash function uses all characters of the input string. Otherwise, strings "aaa", "aza" and "a_a" could map to the same integer if the second character was ignored.
- The hash function uniformly distributes the data across the entire set of possible hash values. The full target range of the hash function is covered by corresponding input strings; i.e. the probability to hash one string to a particular number is the same as that of any other string and number. Hence, this requirement guarantees the avoidance of accumulation points – so-called hotspots – and thus selective overloads.
- The hash function generates very different hash values for similar strings. The basic rule is that if two input strings differ in one single bit, the generated numbers differ in about 50% of their bits. In that sense, a hash function is a chaotic function.
- Hash functions should be collision free. This means that the probability of two different strings being mapped onto the same integer is extremely small. This requirement is about the same as that for uniform distribution.
- The same as that for uniform distribution.


Applying hash functions to array addressing leads to the following result: Let $h$ be the hash function and $A[N]$ an array of any object as component type. Then $\text{put}("aaa", \text{obj}_1)$ is implemented by $A[h("aaa") \mod N] = \text{obj}_1$ and $\text{get}("aba")$ is consequently implemented by return $A[h("aba") \mod N]$. In case of small $N$, a further data structure is needed to maintain all (key, entry)-pairs with different keys, which are mapped to the same index. Though the hash function is collision free, a small array size could destroy this property. For better relations between the key string and the entry object, the key is usually generated from the (probably more complex) entry object. Assume a composed data structure, which contains a name and a phone number.

So $\text{obj} = \{"Humphrey Bogart", "+1-555-1234"\}$ could be stored by $\text{put}(\text{obj}.$getName(), $\text{obj}$), which hashes the string "Humphrey Bogart" (as the result from the getName function call) to the appropriate index. The phone number retrieval proceeds by...
using the get function whose result is applied to a selector for the phone number:
get("Humphrey Bogart").getNumber(). According to the requirements from hash functions, values are uniformly distributed within the hash table. Internal memory management provides sufficient space to accept each key/entry pair.

Figure 41 gives an overview of how data storage and retrieval work in a hash table. The user applies the hash function to the key (name = "Humphrey Bogart") and retrieves the entry (phone number = "+1-555-1234") from the calculated array position (873).

3.4.3 Distributed Hash Tables
Having now learned about hash functions and their use in hash tables, the next step is to distribute the hash tables among a large set of networked computers. First of all, each computer involved calls itself a peer and randomly selects a unique constant number as its lifetime peer identifier. Thereby, each peer represents one component of the array \( A[N] \) within the hash table that is now distributed. Consequently, the peers can be ordered in a sequence according to their identifiers. Peers with close IP addresses could have totally different peer IDs and, thus, be far away from each other in the sequence of peers. The distributed hash table forms a so-called overlay network on top of MAC and IP with new addresses generated from the peer IDs.

Once an application applies a `put(key, entry)` function call to a DHT, the hash result of the key determines the peer according to the peer ID where the value is to be stored. As there will be (probably pretty huge) gaps between chosen peer IDs, and the hashed key does usually not hit a peer ID exactly, the next peer will be chosen – where next could mean for example the peer with the smallest ID greater than or equal to the hashed key. If peer \( X \) and \( Y \) are neighboring peer IDs with \( X < Y \), then \( Y \) is responsible for all keys in the range \([X+1, Y]\). In other words: one peer is responsible for quite a bunch of keys. The function call `get(key)` therefore addresses the peer with the identifier next to the hashed key and fetches the desired value from that peer.
In Figure 42, a Distributed Hash Table is displayed where the peer with the ID 111 takes responsibility of keys hashed to [1, …, 111], peer 555 accepts values hashed to the range [112, …, 555], etc. This approach follows the definition of next from the previous paragraph. In small networks, each peer is able to know all others. This simplifies the process of accessing peers for putting and getting. However, large Peer-to-Peer networks use an address width of 160 bits that provide an address space of $2^{160}$ (or ~$10^{48}$) entries to select peer IDs from and to map hash values to. Therefore, the idea of all peers knowing each other does not work anymore. Instead, highly sophisticated protocols like those of Chord (see (Stoica, Morris, Karger, Kaashoek, & Balakrishnan, 2001)), Kademlia (see (Maumounkov & Mazieres, 2002)), Tapestry (see (Zhao, Kubiatowicz, & A. D. Joseph, 2001)), CAN (see (Ratnasamy, Francis, Handley, Karp, & Shenker, 2001)), etc. were developed in order to address peers by a routing mechanism without knowing the whole network. They are mostly based on the assumption that knowing just $O(160)$ from $2^{160}$ peers is sufficient to reach all others in an iterative process. So the term DHTs is usually used to designate a probably huge set of peers along with a protocol for addressing them. This is the so-called peer look-up mechanism, which maps an integer (generated by hashing a key) to the IP address of the proper peer which is responsible for that key.

### 3.4.4 Peer-to-Peer Routing

The look-up protocols are mostly based on a greedy algorithm where the main difference between the protocols is the metric for a distance calculation between a hashed key and a known peer ID. To address the peer responsible for key X, the look-up selects peer Y from its local list of known peer IDs (called the routing or finger table), which has a minimum distance to X. The distance function is the XOR operation in Kademlia or $(Y-X) \mod 2^{160}$ for Chord. Tapestry detects runtime duration to known peers and selects the closest one that fits to X. If X equals Y or if Y is responsible for X, the target peer is found and Y’s IP address is returned. Otherwise, Y continues the look-up routine in the same way. The approach of managing only a small set of known peers in the local routing tables – in this case a multiple of 160 entries – extends the efforts for routing to an unknown node. As the routing routine depends strongly on the greedy algorithm, the routing efforts are directly related
to the selection of the distance function. The previously mentioned protocols a Chord, Kademlia, Tapestry, CAN, etc. use well-defined distance functions and therefore provide a maximum hop count of $O(\log N)$ where $N$ is the number of peers in the network. In practice, the maximum hop count is 160 in case the Peer-to-Peer network has gained a certain size.

![Figure 43 The Distributed Hash Table Architecture of the Chord Protocol](image)

Routing in Chord involves so-called fingers. Each peer has $n$ fingers, which are at least $2n$-1 steps apart in the binary identifier space. Figure 43 shows the fingers of the peer with key 0, where $n=4$: these are the peers with keys $1=2^0$, $2=2^1$, $4=2^2$ and $8=2^3$. When peer 0 requests data, it must first hash the meta-information with the system internal hash function in order to find the key of the meta-information. In the example, 13 be the key that is sought. The peer forwards the request to its finger table with the key closest$^{22}$ to 13, which is peer with key 8. Peer 8 then consults its own finger table to find the next closest peer to 13, which is 12, and so on until the peer that is responsible for the sought key is found. Once the meta-information about the actual location is found, the requesting peer can contact the source directly.

Other implementations of DHTs as Peer-to-Peer protocols have slightly different key-based routing algorithms. Pastry (see (Rowstron & Druschel, 2001)) and Tapestry use a routing algorithm that is based on identifier prefixes. The Kademlia protocol generates a binary tree topology, in which a XOR metric is used for routing. CAN is not based on a one-dimensional identifier space but instead uses an $n$-dimensional Cartesian coordinate space on a virtual multi-torus topology. The routing in CAN is then a greedy forwarding to the neighboring peer that is closest to the destination coordinates in the multi-dimensional information space.

### 3.4.5 Further Peer-to-Peer Features

Beside peer look-up, some other basic functions are necessary to construct a robust, highly scalable Peer-to-Peer network based on distributed hash tables.

- Data replication (or redundancy) to provide data in case peers break down or leave the network legally.
- Watchdog functions for detecting peer breakdowns
- Self-organization to re-install data replication and update routing tables when peers are gone
- Leasing/keep alive to remove outdated data when a certain time has passed without any notification from the data originator.
- An interface to the application programs which provides methods for keyword-based management of DHT entries:
  - Publish DHT entry

---

$^{22}$ Please be aware, that "closest" refers to the peer ID which is responsible. In case peer N is responsible for IDs smaller than N, too, X closest to N means the greatest X smaller than N.
3.4.6 Assessment of DHT-based P2P Infrastructures

DHTs and key-based routing are reliable and predictable: If an entry is in the system it will be found involving a maximum of \( \log n \) peers during search, where \( n \) is the number of peers in the system. Still, these rigidly structured Peer-to-Peer networks remain predictable and reliable on an Internet scale at churn rates up to 40 percent.

According to the replication of each entry and each keyword that characterizes the entry, the system is highly robust. This holds, too, for the safety because there is no single point of failure like index servers or other centralized functions. Consequently, there is no single point of attack which might weaken the security of the system.

Peer-to-Peer based systems are usually built upon standard components regarding hardware and software. Along with little maintenance efforts, Peer-to-Peer systems require a minimum TCO. On the other hand, a DHT-based Peer-to-Peer system provides keyword based data management. Since hash functions are involved, similar keywords are basically mapped to distant peers. This prevents an easy wildcard search or range queries using keywords.

If a complex query involves ranges or wildcards as in geographic locations, there needs to be at least one pure keyword. This keyword helps collecting all related data from the DHT system. In a second step, all those data that satisfy the range or wildcard queries will be selected.

3.4.7 Implementation Aspects

This section addresses the implementation aspects regarding the application of a DHT based Peer-to-Peer approach for the IoT Service Resolution. It is describes how the algorithms, data structures and interfaces are designed in principle.

3.4.7.1 Data Structures and Data Types

Besides the information about services, the IoT Service Resolution subsystem maintains also information about device addresses. For referencing this information, the key objects are 8-bit character strings in both cases. For identifying services, the service ID is used and for identifying devices the device ID is used.

The differentiation between services and devices require different types of entry objects, which are inserted in the DHT. The generic structure of entry objects is shown in Figure 44.

```c
struct entry_type{
    String key;

    List subscriberList_1;

    ...;

    List subscriberList_n;

    <Entry type specific data fields>
```
Depending on the specific entry type, different specific data fields are added. For example, the entry type for a service includes additionally the service description. For all entry objects it is required, that they include the key, with which they are referenced. Since the hash function with which a entry point is identified is not collision free, this data value is required to identify the correct entry unambiguously. It is foreseen that users are notified upon changes in entries (e.g. because a service description changes). Therefore multiple subscriber lists for different notification purposes are included in the specific entry type. In those subscriber lists the information about the users, which want to be notified, are stored.

3.4.7.2 Functions and Algorithms
The IoT Service Resolution defines internal DHT functions in order to manage the Peer-to-Peer system. The utilization of these functions is transparent for the user. They are used by the IoT Service Resolution Interface functions. There exist the following functions:

- `put(key_object, entry_object)`
- `modify(key_object, entry_object)`
- `delete(key_object)`
- `get(key_object)`

With the `put(key_object, entry_object)` an entry is inserted in the DHT. The function first determines the IP address of the peer node, which is responsible for the entry according to the hash value of the key\(^23\). After that it creates a new entry with key object and the entry object as parameter. The function `modify(key_object, entry_object)` is similar to the put function, however an existing entry is modified. Depending on the implementation, the semantic of put and modify might be different. Since a service is updated from time to time, it might be convenient to modify only a certain part of an entry. The function `delete(key_object)` deletes an entry on an existing node and the function `get(key_object)` returns the entry stored for the key object. Since each DHT function requires routing to the peer node, which has a certain complexity, it is reasonable to cache the IP address of peer nodes.

3.4.8 IoT Service Resolution
In this section it is described how the IoT Service resolution is enabled by the proposed Peer-to-Peer based approach. IoT Service Resolution covers the discovery, the lookup and the resolution of a service or a Virtual Entity. Furthermore an Association between a service and a Virtual Entity is addressed. For this purpose appropriate interface functions are defined in section 2.2.1.1. With respect to resolution and lookup the utilization of a Peer-to-Peer based approach is straight forward: Both functions require a unique service ID as input parameter. Here the respective information is accessed in the distributed hash table with the service ID as a key. However for the discovery of an unknown service the Peer-to-Peer based approach is only partially useful. If the discovery is based on a limited set of keywords, indeed one can use the keywords as the key object in order to insert data into the DHT. However if it comes to wildcard search or range search for a certain location a Peer-to-Peer approach has its limitation because even similar objects, in terms of location, are distributed in the network.

3.4.8.1 Service Entry Management
As an initial step, a service has to be inserted into the IoT Service Resolution subsystem. According to section 2.2.1.1, the function `insertService(ServiceDescription)` is

\(^{23}\) The description of the routing to a peer node is explained in subsection 3.4.4
available for this purpose. The function inserts the service description with the string representation of the service ID as a key and returns the service ID back. The pseudo code for service description insertion is shown in Figure 45.

```java
ServiceId insertService(ServiceDescriptionType ServiceDescription)
{
    String serviceId = ServiceIdManagement.createServiceId;
    put(serviceId, ServiceDescription);
    return serviceId;
}
```

**Figure 45 Inserting a service description**

In order to update a service description, the call `updateService(ServiceDescription)` is executed with the semantic of the pseudo code in Figure 46. For convenience one can define a modify function, which allows the modification of an existing entry without deletion and reinserion.

```java
void updateService(ServiceDescriptionType ServiceDescription)
{
    String serviceId = ServiceDescription.ServiceId;
    modify(serviceId, ServiceDescription);
}
```

**Figure 46 Updating a service description**

In order to delete a service, the function in Figure 47 is used.

```java
void deleteService(String ServiceId)
{
    ServiceIdManagement.deleteServiceId(serviceId);
    delete(ServiceId);
}
```

**Figure 47 Deleting a service**

### 3.4.8.2 Service lookup

The service description is retrieved with the function `lookupService(ServiceID)`. The function applies a `get(ServiceId)` call and returns the service descriptor. See Figure 48:

```java
ServiceDescriptionType lookupService(String ServiceID)
{
    return get(serviceId);
}
```

**Figure 48 Service description lookup**

### 3.4.8.3 Service URL resolution

The URL to a service is resolved with the function `resolveService(ServiceID)`. The functionality is shown in Figure 49.

```java
ServiceUrl resolveService(String ServiceID)
{
    ServiceDescriptionType serviceDescription = get(ServiceId);
    return serviceDescription.url;
}
```

**Figure 49 Resolving a service URL**

### 3.4.9 IoT Service Discovery

Keyword based service discovery is realized by inserting a service description with the keyword as identifier in the DHT. In contrast to utilization of an ID, which unambiguously identifies a service, a keyword would be descriptive in this context. There are currently two ideas how a DHT can be used for service discovery:
- A suiting descriptive keyword is used and the service description is inserted in the DHT with that keyword as key object. For example: using 'pizza' as keyword for an Italian restaurant serving pizzas. However the service couldn't be found by somebody, who is just looking for any restaurant as he might use a more generic keyword. Since the DHT is keyword based and every entry is identified by exactly one key, for every keyword an own DHT entry is required. Even combining keywords for narrowing down search results is hard to realize with DHT. One could propose to insert a service for each permutation of a set of keywords. However this doesn't scale well, even if the subsets are assumed to be alphabetical.
- Geographical information could be used as a keyword. For example the name of a place. One could also use the geo coordinate as a key word. However arbitrary range queries are not possible with this approach.

3.4.10 Virtual Entity Resolution
Virtual Entities can be managed by a DHT approach in a similar way, how services are managed. They are referenced by a unique VE-ID. Additionally to services and Virtual Entities, there exist Associations, which are relations between a particular Virtual Entity and a particular service. A functionality of the Virtual Entity Resolution is lookup and discovery of an Association. During the former is feasible with a DHT approach, the discovery of an Association requires a VE specification and a service specification, and is therefore hard to be provided with DHTs (see 3.4.9).
The function lookupAssociations(VE-ID, VEServiceSpecification) returns an array of Associations. To implement this, the pseudo code in Figure 50 is applied.

```java
Association[] lookupAssociation(String VE-ID, VEServiceSpecificationType vess) {
    VirtualEntityType virtualEntityEntry = get(VE-ID);
    return findAssociations(virtualEntityEntry.listOfAssociations, vess);
}
```

Figure 50 Association lookup

First, the function in the pseudo code of Figure 50 retrieves the entry of the Virtual Entity, which includes a complete list of all Associations of this Virtual Entity (virtualEntityEntry.listOfAssociations). After that, a list of Associations is returned, which fit to the service specification.

3.4.11 Managing Subscriptions
Upon the update of a service description or the modification of a service URL, the IoT Service Resolution component provides functionality to inform users about changes. Therefore subscription methods, according to section 2.2.1.1, are provided in the IoT Service Resolution Interface.
The subscribeServiceResolution and subscribeServiceLookup functions insert the callback information about a specific subscriber into an existing entry for the specific service in the DHT. The functionality is that the peer node, holding the entry for the respective service, is queried and the subscription information is added to the subscription list, which is part of the entry (see subsection 3.4.7.1). It is assumed that that peer nodes have a specific function, which allows to add an element to an entry’s subscription list, which however is supposed to be specified in later deliverables.
In the example of a changed service URL, which would cause a notification, the IoT Service resolution component retrieves the list of all subscribers out of the service description entry and informs them by using the respective notify function.
Regarding subscriptions to service discovery, the function subscribeServiceDiscovery is defined in section 2.2.1.1 in order to notify users about the existence of new services fitting to a certain Service Specification. However such a functionality, as well as the functionality to find associations according to VEServiceSpecification (subscribeAssociationLookup
and subscribeAssociationDiscovery), is not supported by Peer-to-Peer-Functionality, as a specification, which could include multiple keywords, categories, ranges or wildcards, is not applicable for DHTs.

3.4.12 Conclusions

This section has described the concept of the Peer-to-Peer based approach and how it can be integrated into the IoT Service Resolution. A coarse grained algorithmic description was given from which an implementation can be derived, which shall be presented and discussed in D4.4 (R. d. I. Heras, 2013).

Concluding it is to say, that Peer-to-Peer based approaches can apply lookup or resolution for services, devices, Associations or Virtual Entities. Here information is referenced by an identifier, which can be used as a key to reference the respective information in a DHT. Everything, which goes beyond a simple keyword based query, e.g. a wildcard search, is not supported by this approach.

3.5 Domain-based Approach

3.5.1 Introduction

This architecture is based on a hierarchical structure, built on the idea that, usually in the IoT world, the information source has its local scope (city, building, etc.) designed in such way that directly reflects the native organization of real thing’s interaction. This local scope is what we call “area of interest”.

As already stated in D4.1, this architecture proposal presents many similarities with the familiar DNS (Domain Name System), the hierarchical naming system built on a distributed database for computers, services, or any resource connected to the Internet.

In this section we will describe how domain-oriented infrastructure fits and supports all the elements of the IoT Resolution infrastructure (discovery, look-up, and name/ID resolution for a Service or a Virtual Entity).

The paramount elements supporting this architecture are the Directory Servers, in charge of managing Services, Virtual Entities, and associations, between VEs and services and the Master Index Catalogue that maintains an updated index with all the information available in the scope of each low level Directories Servers.

Theoretically from an architectural point of view, we differentiate between the management of service descriptions and the management of associations. It is up to the actual architecture instances to choose to handle them together or to keep them separately.
In the following we will describe how Master Index and directory servers are composed and how discovery and look-up functions could be implemented using this proposed architecture.

### 3.5.2 Directory Servers
In this approach to build an IoT architecture, Directory Servers are the main components to store service and Virtual Entity descriptions, and to keep associations between services and Virtual Entities.

There are two types of Directory Servers: Masters and Slaves: Masters are directory servers that store information to relate zone areas to each of the directory servers of the system. Meanwhile Slave Servers use automatic updating mechanisms in communication with master servers to maintain an identical copy of the records that are related to that zone.

Directory server embeds an index searching algorithm, organizing information using this index. The Index is built using position and localization information both of services and entities. This position and localization must be bounded to a certain scope, called area of interest. (This area of interest could be defined by geographic coordinates or can be related to a certain logical unit as company premises). Deep description of the domain underlines two levels of information: network level and host level. To perform this task of managing and hold information for both services and entities they rely on the area of interest associated to each of them. These zones are the area of interest that every Directory Servers can see. Zones can be defined just as a unique root zone, or can also have sub-domains attached to it.

For the discovery and look-up operations Directory servers manage data using several databases. These databases communicate among themselves to resolve Service/VE ID and to register the Association between entities and services.
3.5.2.1 Directory Server Components
As it is depicted in Figure 52 a directory server could be split into the following components:
- Frontend
- Domain Manager
- Subscriber
- Association monitor
- Synchronizer.

Frontend component is in charge of receiving requests and dispatch them to the rest of the elements of the Directory Server. It is also responsible to serve all the requests related to discovery and lookup operation related to its coverage.

Domain Manager is the management component, so its task is to insert, update or delete any Service Description or Association.

Subscriber is the piece is charge of deal with subscription, so it receives new subscription, delete old ones and monitor living subscription firing a notify operation each time the information an application is subscribed changed.

Association monitor periodically checks associations between Services and Virtual Entities. To do it the process has to check if either the VE or the associations are still valid.

Finally the Synchronizer component is in charge of update Master Index Catalogue information of the updates but without collapsing the Master Index.

![Diagram of Directory Server Components](image-url)

**Figure 52 Directory server components**
3.5.3 Master Index catalogue

The Master Index is the component that is in charge of storing information about the area of interest of the directory servers and their addresses. The area of interest is used for building position and localization that is bounded in a certain scope and that is the area that a Directory server can observe. Master Index searches the area to which send the service Specification request and there too, to its correspondent Directory server. Storing indexed information of directory servers will permit Master Index to delegate an income request. Information of services and the Associations are stored in the Directory servers.

As it is shown in Figure 53 in the Master Index are stored references to Virtual Entities and to Services, so some requests are easily served and relayed to the directory server where the real bunch of information is stored.

![Figure 53 Distributed Hierarchical Dynamic Resolution](image)

The idea of having this two layers approach is to ease the whole resolution process

Zooming the Master Index itself (Figure 54) we can identify the following components:

- Frontend
- Resolution
- Dispatcher

Frontend component is in charge of receive requests and dispatch them to the rest of the elements of the Master Index.
Service Resolution takes as input Service specification and try to match this information against any of the stored descriptions related to services. VE Resolution takes as input VE id and VEServiceSpecification and try to match this information against any of the stored associations.

In case it is not possible to find a match because the information is not included in this Directory server as resynchronization between the directory servers and the master index is not made in real time (Due to the calculation made by the resolution component, of the matching between the description input and the different areas of interests covered by each Directory Server), this request is transferred to one or many Directory Servers to finish the Discovery/Look-up process.

**3.5.4 Discovery mechanisms**

In this section we are going to describe discovery mechanisms for both IoT services and VE Resolution. We will describe synchronous requests where the answer follows the request and synchronous ones (subscriptions) where the client is interested to be notified when there are changes in the subscription request (either service description for services or association for VEs).

**3.5.4.1 Discovery Mechanisms for Services**

Discover method has the following signature:

\[
\text{discoverService(ServiceSpecification): ServiceDescription[ ]}
\]

Once a request comes to the Master Index (Service Specification) it is decided if the Master Index is able to serve the request or it has to ask related Discovery Servers according to the matching between the specification and one or more associated areas of interest.
In case a match with any of the references updated in the Master Server is not achieved it is transferred to the Directory Servers that can finish the discovery process (this could happen as stated above due to loss of synchronization between master and directory servers).
3.5.4.2 Asynchronous Discovery mechanisms

subscribeServiceDiscovery(ServiceSpecification, notificationCallback):SubscriptionID

When users are interested in changes in a service description, better than use the normal discovery method, they can use the subscribeServiceDiscovery method. For each of this subscription the directory server affected has to store the service specification, the notification callback endpoint (implemented by the caller's user) and a subscription ID (user by end the subscription by the user). It returns the correspondent subscription-ID.

The Directory Servers whose area of interest includes this service, store in one of its database subscription-ID, notificationCallback and service specification fields. An autonomous process inside each of the DS look for each of its subscription and for each of them a discovery service process as the described above is executed, and the results notified to the user.

This mechanism allows querying functions only to be executed when changes occurs, and not always, allowing updating any changes with periodic notifications.

3.5.4.3 Association Discovery mechanisms

discoverAssociations(VESpecification, VEServiceSpecification):Association[]

When a given service specification enters IoT domain-based approach infrastructure, the first issue is to determine the address of the Directory Server to which send the service specification request. For that we use a special software element we call f-package functions and discovery operation in Master Index. The request is sent to a concrete Directory Server. Directory servers dispose of another software element (k-package functions) for searching and storing algorithms, that given a service specification it returns the virtual entities with that service specification. After this preliminary process, then it is able to discover the group of associations, by calling discoverAssociations function. The association, result of the call to discoverAssociations function, is stored in a database (called database-3 in Figure 58).
3.5.4.4 Subscription Association Discovery mechanisms

```
subscribeAssociationDiscovery(VESpecification, VEServiceSpecification, notificationCallback): SubscriptionID
```

Subscription to an association change is another process that is triggered each time there is a change in any of the subscriptions active at any time. The information needed for this process is stored in another database where we have notification callback url and association ID.

3.5.5 Lookup mechanisms

3.5.5.1 Lookup mechanisms for services

```
lookupService(ServiceID): ServiceDescription
```

The systems' scenario when doing look-up is a bit different from that of Discovery. When making a look-up, service ID is known. Incoming Service Id request enters Master Index. Master Index delegates the service ID request to the Directory Server where service description for this ID is stored.
In lookup the result that our system provides, is that given a certain Service ID, it returns the corresponding service description. To know where request lookup must be delegated, g-functions search in database two of Master Index which returns the area of interest of the Directory Server that provides that service. Request Service ID is passed to the corresponding Directory Server when g-functions search the address of the directory server given the area of interest of Directory Server. We use a database (database-3) for storing pairs Service ID and service descriptions as alphabetically ordered, and using a dedicated software (Package g-functions) a query is executed to return the service specification field which corresponds to that service-ID.
Figure 60 VE lookup process at the directory server

3.5.5.2 Association Look-up

`LookupAssociations(VE-ID, VEServiceSpecification):Association[]`

When an association look-up is requested, first we have to determine the address of the Directory Server to which to send the look-up (For this we use VE-ID and VE Service Specification). In the selected directory server the look-up is made and the fitting associations are returned as answer to the client.

3.5.6 Management functions

For each of the elements used (Services and Association) the allowed operations are:

- **Insert**
  - `insertService(ServiceDescription):ServiceID`
  - `insertAssociation(Association):AssociationID`

- **Update**
  - `updateService(ServiceDescription)`
  - `updateAssociation(Association)`

- **Delete**
  - `deleteService(ServiceID)`
  - `deleteAssociation(AssociationID)`

Each time an element to be managed is enter into the system (In principle these operations will be inputted manually) there is a process that using the element ID and the areas assign the new item to the related Directory Servers according to the following figure:
Element to be registered
(Serv., VE or assoc.)

DOMAIN
DIRECTORY
MANAGER
D.S. X

1.- Insert new element in Information DB’s

DATABASE-1
INDEX
AREA OF D.S
ELEMENT ID

2.- If needed send info to others DS’s involved

InsertElement(DS_N, element)

3.- Inform DM scheduler of updating

InsertElement(ELEMENT ID)

Figure 61 Manager process at a directory Server
4. Conclusions

This deliverable has provided descriptions of the different mechanisms and algorithms that are feasible for designing and implementing a framework for an IoT Resolution Infrastructure. Based on the particularities of the different approaches, these may be applicable to different requirements and scenarios.

The Geographic Location approach is targeted at the discovery functionality of both the IoT Service Resolution and VE Resolution components. Moreover, the discovery interface specification, as asserted by this approach, requires the geographic coordinates to be specified in the input. Both the Semantic Web-based and Federation approaches are also best suited for IoT Service and Association discovery, but can handle lookup as well. Name/identifier resolution is however, not handled by these two approaches. Both these approaches allow arbitrary keywords to be specified in the discovery interface implementation, including logical location specification of VEs and IoT Services. For name/identifier resolution and lookup, the Peer-to-Peer DHT-based infrastructure seems to be the most appropriate approach. It, however, does not go beyond a simple keyword-based query.

The three approaches, Geographic Location-based, Semantic Web-oriented and the Federation-based approach, also define various methods for inferring new Associations between Virtual Entities and IoT Services; while the Geographic Location and Federation approaches also define mechanisms and conditions to be taken into account for VE and IoT Service Monitoring. While location is a driving factor for inferring associations for the Geographic Location approach, both the Semantic Web-based and Federation approaches follow a rule-based mechanism for deriving and managing Associations.

Since the functionalities provided by the different approaches are not mutually exclusive, a combination of these will have to be implemented in order to realise the IoT-A Resolution infrastructure that will also be used in WP7. Figure 62 presents a high-level pictorial representation of how an integration of the different approaches could be achieved.
different in an actual implementation scenario. Based on the methods employed for the analysis and dispatching function, different approaches would be instantiated in different scenarios. As a next step, we will refine the integration and define a specific architecture instance, choosing options that best fit the IoT-A project requirements, taking into account the use cases in WP7. Then the required components will be implemented by the partners and integrated, first within WP4 and then with WP7 that currently uses dummy components with interfaces matching our high-level specification. Apart from using the components in WP7, we will also further evaluate the implemented instance of the IoT Resolution Infrastructure. The implementation and the evaluation results will be documented in our final deliverable D4.4.
5. References


SySML, O. Library for Quantity Kinds and Units: schema, based on QUDV model. 1.2. Retrieved from doi:http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu


## 6. Terminology and Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actuators</td>
<td>“An actuator is a mechanical device for moving or controlling a mechanism or system. It takes energy, usually transported by air, electric current, or liquid, and converts that into some kind of motion.”</td>
</tr>
<tr>
<td>Address</td>
<td>An address is used for locating and accessing – “talking to” – a Device, a Resource, or a Service. In some cases, the ID and the Address can be the same, but conceptually they are different.</td>
</tr>
<tr>
<td>Architectural Reference Model</td>
<td>The IoT-A architectural reference model follows the definition of the IoT reference model and combines it with the related IoT reference architecture. Furthermore, it describes the methodology with which the reference model and the reference architecture are derived, including the use of internal and external stakeholder requirements.</td>
</tr>
<tr>
<td>Architecture</td>
<td>“The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution”.</td>
</tr>
<tr>
<td>Association</td>
<td>An association establishes the relation between a service and resource on the one hand and a Physical Entity on the other hand.</td>
</tr>
<tr>
<td>Business Logic</td>
<td>Goal or behaviour of a system involving Things serving a particular business purpose. Business Logic can define the behaviour of a single Thing, a group of Things, or a complete business process.</td>
</tr>
<tr>
<td>Device</td>
<td>Technical physical component (hardware) with communication capabilities to other IT systems. A device can be either attached to or embedded inside a Physical Entity, or monitor a Physical Entity in its vicinity.</td>
</tr>
<tr>
<td>Digital Entity</td>
<td>Any computational or data element of an IT-based system.</td>
</tr>
<tr>
<td>Virtual Entity</td>
<td>Computational or data element representing a Physical Entity. Virtual Entities can be either Active or Passive Digital Entities.</td>
</tr>
<tr>
<td>Discovery</td>
<td>Discovery is a service to find unknown resources/services based on a rough specification of the desired result. It may be utilized by a human or another service. Credentials for authorization are considered when executing the discovery.</td>
</tr>
<tr>
<td>Domain Model</td>
<td>“A domain model describes objects belonging to a particular area of interest. The domain model also defines attributes of those objects,</td>
</tr>
</tbody>
</table>

Internet of Things Architecture © - 100 -
such as name and identifier. The domain model defines relationships between objects such as "instruments produce data sets". Besides describing a domain, domain models also help to facilitate correlative use and exchange of data between domains.

<table>
<thead>
<tr>
<th>Entity of Interest (EoI)</th>
<th>Any physical object as well as the attributes that describe it and its state that is relevant from a user or application perspective. The term is obsolete in the IoT-A reference model: the term Physical Entity should be used instead.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human</td>
<td>A human that either physically interacts with Physical Entities or records information about them, or both.</td>
</tr>
<tr>
<td>Identity</td>
<td>Properties of an entity that makes it definable and recognizable.</td>
</tr>
<tr>
<td>Identifier (ID)</td>
<td>Artificially generated or natural feature used to disambiguate things from each other. There can be several IDs for the same Physical Entity. The set of IDs is an attribute of an Physical Entity.</td>
</tr>
<tr>
<td>Information Model</td>
<td>“An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context. The information model is an abstract representation of entities which can be real objects such as devices in a network or logical such as the entities used in a billing system. Typically, the information model provides formalism to the description of a specific domain without constraining how that description is mapped to an actual implementation. Thus, different mappings can be derived from the same information model. Such mappings are called data models.”</td>
</tr>
<tr>
<td>Interface</td>
<td>“Named set of operations that characterize the behaviour of an entity.”</td>
</tr>
</tbody>
</table>
| Internet                 | “The Internet is a global system of interconnected computer networks that use the standard Internet protocol suite (TCP/IP) to serve billions of users worldwide. It is a network of networks that consists of millions of private, public, academic, business, and government networks of local to global scope that are linked by a broad array of electronic and optical networking technologies. The Internet carries a vast array of information resources and services, most notably the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail. Most traditional communications media, such as telephone and television services, are reshaped or redefined using the technologies of the Internet, giving rise to services such as Voice over Internet Protocol (VoIP) and IPTV. Newspaper publishing has been reshaped into Web sites, blogging, and web feeds. The Internet has enabled or accelerated the creation of new forms of human interactions through instant messaging, Internet forums, and social networking sites. The Internet has no centralized governance in either technological implementation or policies for access and usage; each constituent
network sets its own standards. Only the overreaching definitions of the two principal name spaces in the Internet, the Internet-protocol address space and the domain-name system, are directed by a maintainer organization, the Internet Corporation for Assigned Names and Numbers (ICANN). The technical underpinning and standardization of the core protocols (IPv4 and IPv6) is an activity of the Internet Engineering Task Force (IETF), a non-profit organization of loosely affiliated international participants that anyone may associate with by contributing technical expertise."

<table>
<thead>
<tr>
<th>Internet of Things (IoT)</th>
<th>The global network connecting any smart object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interoperability</td>
<td>&quot;The ability to share information and services. The ability of two or more systems or components to exchange and use information. The ability of systems to provide and receive services from other systems and to use the services so interchanged to enable them to operate effectively together.&quot;</td>
</tr>
<tr>
<td>Look-up</td>
<td>In contrast to discovery, look-up is a service that addresses exiting known service descriptions (previously: resources) using a key or identifier.</td>
</tr>
<tr>
<td>Physical Entity</td>
<td>Any physical object that is relevant from a user or application perspective.</td>
</tr>
<tr>
<td>Perspective (also referred to as architectural perspective)</td>
<td>&quot;Architectural perspective is a collection of activities, checklists, tactics and guidelines to guide the process of ensuring that a system exhibits a particular set of closely related quality properties that require consideration across a number of the system's architectural views.&quot;</td>
</tr>
<tr>
<td>Reference Architecture</td>
<td>A reference architecture is an architectural design pattern that indicates how an abstract set of mechanisms and relationships realises a predetermined set of requirements. It captures the essence of the architecture of a collection of systems. The main purpose of a reference architecture is to provide guidance for the development of architectures. One or more reference architectures may be derived from a common reference model, to address different purposes/ usages to which the Reference Model may be targeted.</td>
</tr>
<tr>
<td>Peer-to-Peer (P2P)</td>
<td>Peer-to-Peer refers to a computer network in which each computer in the network can act as a client or server for the other computers in the network, allowing shared access to files and peripherals without the need for a central server. [...] Peer-to-peer systems often implement an abstract overlay network, built at Application Layer, on top of the native or physical network topology. [...]</td>
</tr>
</tbody>
</table>
| Reference Model          | "A reference model is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete
architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details. A reference model may be used as a basis for education and explaining standards to non-specialists.”

| Resolution | Query-response process by which a given ID is associated with a set of Addresses of information and interaction Services. Information services allow querying, changing and adding information about the thing in question, while interaction services enable direct interaction with the thing by accessing the Resources of the associated Devices. Based on a priori knowledge. |
| Resource | Computational element that gives access to information about or actuation capabilities on a Physical Entity. |
| Requirement | “A quantitative statement of business need that must be met by a particular architecture or work package.” |
| RFID | “The use of electromagnetic or inductive coupling in the radio frequency portion of the spectrum to communicate to or from a tag through a variety of modulation and encoding schemes to uniquely read the identity of an RF Tag.” |
| Sensor | A Device identifying or recording features of a given entity. |
| Service | Software component enabling interaction with resources through a well-defined interface, often via the Internet. Can be orchestrated together with non-IoT services (e.g. enterprise services). |
| Stakeholder (also referred to as system stakeholder) | “An individual, team, or organization (or classes thereof) with interests in, or concerns relative to, a system.” |
| Storage | Special type of Resource that stores information coming from resources and provides information about Entities. They may also include services to process the information stored by the resource. As Storages are Resources, they can be deployed either on-device or in the network. |
| System | “A collection of components organized to accomplish a specific function or set of functions.” |
| Tag | Label or other physical object used to identify the Physical Entity to which it is attached. |
| Thing | Generally speaking, any physical object. In the term ‘Internet of
<table>
<thead>
<tr>
<th>User</th>
<th>A Human or some Active Digital Entity that is interested in interacting with a particular physical object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>View</td>
<td>“The representation of a related set of concerns. A view is what is seen from a viewpoint. An architecture view may be represented by a model to demonstrate to stakeholders their areas of interest in the architecture. A view does not have to be visual or graphical in nature”.</td>
</tr>
<tr>
<td>Viewpoint</td>
<td>“A definition of the perspective from which a view is taken. It is a specification of the conventions for constructing and using a view (often by means of an appropriate schema or template). A view is what you see; a viewpoint is where you are looking from - the vantage point or perspective that determines what you see”.</td>
</tr>
<tr>
<td>Wireless communication technologies</td>
<td>“Wireless communication is the transfer of information over a distance without the use of enhanced electrical conductors or &quot;wires&quot;. The distances involved may be short (a few meters as in television remote control) or long (thousands or millions of kilometres for radio communications). When the context is clear, the term is often shortened to &quot;wireless&quot;. Wireless communication is generally considered to be a branch of telecommunications.”</td>
</tr>
</tbody>
</table>