Semantic enablers for dynamic digital-physical object associations in a federated node architecture for the Internet of Things

Suparna De[1], Benoit Christophe[2], Klaus Moessner[1]

[1]*Centre for Communication Systems Research (CCSR), University of Surrey, Guildford GU2 7XH, Surrey, United Kingdom. {S.De, K.Moessner}@surrey.ac.uk*
[2] *Bell Labs Research, Alcatel-Lucent, 91620 Nozay, France. benoit.christophe@alcatel-lucent.com*

Abstract

The Internet of Things (IoT) paradigm aims to realize heterogeneous physical world objects interacting with each other and with the surrounding environment. In this prospect, the automatic provisioning of the varied possible interactions and bridging them with the digital world is a key pertinent issue for enabling novel IoT applications. The introduction of description logic-based semantics to provide homogeneous descriptions of object capabilities enables lowering the heterogeneity and a limited set of interactions (such as those with stationary objects with fixed availability) to be deduced using classical reasoning systems. However, the inability of such semantics to capture the dynamics of an IoT system as well as the scalability issues that reasoning systems encounter if too many descriptions have to be processed, necessitate that such approaches should be used in conjunction with others. Towards this aim, this paper proposes an automated rule-based association mechanism for integrating the digital IoT components with physical entities along temporal-spatial-thematic axes. To address the scalability issue, this mechanism is distributed over a federated network of nodes, each embodying a set of objects located in the same geographical area. Nodes covering nearby geographical areas can share their object descriptions while all nodes are capable of deducing interactions between the descriptions that they are aware of.

Keywords: Internet of Things; Federated architecture; SWRL rules; Smart object associations

1 Introduction

The Internet of Things (IoT) concept envisions a future where numerous physical world objects interacting with each other are engrained in the fabric of our environment [1]. Inspired by the RFID and Wireless Sensor Networks (WSNs) research areas, this concept that was initially considering RFID tags, readers and sensors as 'things', has evolved over the years to now encompass all types of devices supporting interactions between the physical and the virtual world [2]. Facilitating such interactions requires provisioning of mechanisms that enable virtualization of such objects to allow interaction with them, ultimately leading to a realization of the vision of "technology rich human surroundings that often initiate interactions" [3]. Finding sensors, actuators and other digital world objects that are relevant for interactions with any particular physical world object is a key precursor to achieving this IoT vision, which requires lowering the heterogeneity implied by the plethora of possible devices and their resulting data.

The applicability of Semantic Web technologies to create homogeneous, standardized and machine-processable representations has already been identified in the literature [1, 4] as an enabler of object interoperability. Existing research works in sensor networks [5-7] have focused on sensor (and actuator) middleware frameworks that offer sensor measurement data services on the Web and/or at the application level. Finally, standardization activities such as the Semantic Sensor Network Incubator Group (SSN-XG) [8]have resulted in the Semantic Sensor Network (SSN) ontology [9] that represents a high-level schema model to describe sensor devices, their capabilities, observation and measurement data and the platform aspects. However, using Semantic Web technologies brings at least two strong limitations that prevent building efficient and accurate provisioning systems in an IoT context. First, due to the impossibility of describing and reasoning over the dynamics of a system, the use of the Semantic Web precludes representing that objects in the IoT can evolve over time (e.g. having their access policy, availability, geo-location, etc. changing over time). Secondly, almost all the works on Semantic Web reasoning still assume a centralized approach where the complete terminology has to be present on a single centralized system and all inference steps are carried out on this system. While this assumption is acceptable when considering a small set of described entities, the highly dynamic nature of envisioned IoT systems – composed of a very large number of smart objects producing and consuming information – requires adopting a different approach to avoid scalability issues. Moreover, this requirement is strengthened by the fact that disregarding IoT systems dynamics may lead to the computation of meaningless interactions (e.g. an association being asserted between two objects based only on their functionalities without considering their respective geo-locations).

We believe that the use of Semantic Web in the context of the IoT must be coupled with additional processes addressing these two limitations. More precisely, temporal and spatial reasoning must be added on top of classical semantic reasoning in order to accurately reflect the behaviour of the considered IoT systems. This overall reasoning process must also be distributed to cope with computation spikes without having to maintain and administer the computing, network and storage resources each time a reasoning step is performed.

Towards this aim, this paper presents a federated distributed framework of nodes for an IoT architecture. Within this framework, the contributions proposed are focussed on two aspects: inferring automated associations that integrate the IoT digital components with physical entities and a notification algorithm to share knowledge between a determined set of nearby nodes. Each node of the framework refers to a managed geographic location that encompasses reasoning capabilities enabling associations (applicable to the objects contained in the location managed by the node) to be derived. Determining these associations is achieved by a novel rule-based mechanism along temporal-spatial-thematic axes. This mechanism builds upon our earlier work [10] on semantic models that capture the components of the IoT domain and provide a formal representation to the interactions. In line with the identification by Miorandi et al. [1] that architectures may make use of proximity communications whenever possible, each node of our framework is capable of selecting a set of geographically nearby nodes to share the knowledge about the IoT digital components that it manages. As a consequence, each node always uses a well delineated set of IoT digital components – i.e. those attached to or nearby the geographic location managed by the node – to derive associations. The consequent reduced size of the set enables reducing the computation cost implied by the reasoning process while elements composing the set still allow almost all associations to be derived. Though the proposed approaches are focussed towards IoT systems in indoor environments, the contributions can be applied to other conceivable IoT deployments as well.

We evaluate the proposed mechanisms by testing the applicability of the implemented association mechanisms for indirect inference in an entity mobility scenario and show the feasibility of the approach by quantitatively evaluating the scalability of the proposed framework.

The rest of this paper is organized as follows. The federated architecture concept and the embodiment of semantically-enabled nodes are presented in Section 2. Section 3 presents the description of the semantic models supporting both the association mechanism detailed in Section 4 and the knowledge sharing algorithm explained in Section 5. An implementation of the framework is detailed in Section 6, with a scenario validation and evaluation results discussed in Section 7. Related state of the art is presented in Section 8 and 9 concludes the paper and discusses future work.

2 Federated architecture of nodes

In the literature, federated network systems refer to shared resources among multiple loosely coupled nodes [11] in order to optimize the use of those resources, improve the quality of network-based services, and/or reduce costs. Widely used in scenarios involving information sharing between different tiers [12], such distributed systems can cope with storage and computation limitations and offer efficient – i.e. fast – search processes using optimization techniques [13]. Due to these advantages, federated systems are particularly suited to interconnecting heterogeneous physical world objects with the surrounding environment, which relies on the capability to store, retrieve and process a high number of semantic descriptions of IoT digital components.

Supporting the aforementioned IoT paradigm through a federated system is achieved by considering each loosely coupled node as the digital representation of a place hosting physical world objects. In this paper, we define a place as an indoor premise (e.g. a building, a room, etc.) and propose a model allowing such places to be described. However, nothing precludes adapting our architecture to address other kinds of places such as outdoor areas (e.g. a crossroad, a district, etc.). An example of a node (say N) presented in this paper may represent a meeting room equipped with a webcam, a presence sensor and other equipment. Embedding storage and computing capabilities, each node manages a pool of semantically described IoT digital components and can determine all possible associations between such components and the surrounding environment (following our previous example, a node *N* computes and stores the semantic descriptions of the digital interfaces of the webcam, the presence sensor and all other equipment present in the meeting room). Interconnecting these nodes allows a communication scheme where descriptions of IoT digital components can be exchanged to maximize the aforementioned determination process of associations (e.g. the node *N* sharing semantic descriptions with another node *M*).

The following sub-sections describe the building blocks composing a node of our federated system as well as an indoor location model enabling to define how nodes are interconnected.

2.1 Architecture of a node

Each node of a federated system has been designed to provide the following three capabilities:

1. The storage and the processing of semantic descriptions of IoT digital components.

2. The association process determining all possible interactions.
3. The propagation of aforementioned descriptions to other nodes in order to maximize the set of associations that they will (re)compute.

Fig. 1 details the design of each node composing the federation. Although different implementations of such a node may be investigated, a possible embodiment – that will be presented in Section 6 – can be a Semantic Web application running on a Personal Computer equipped with an Internet connection.

In our vision, two kinds of resources are managed by a node. The first type of resource embraces any physical entity that can be sensed, measured or actuated: people, tables/chairs as well as connected physical world objects. The second type of resource comprises the IoT digital components offering some services (such as measuring a temperature) which can provide information on or actuate upon a physical entity. In the remainder of this paper, we consider this second type of resources as IoT Services. In other words, the IoT Service represents the set of functionalities of an IoT digital component and the corresponding offered APIs.

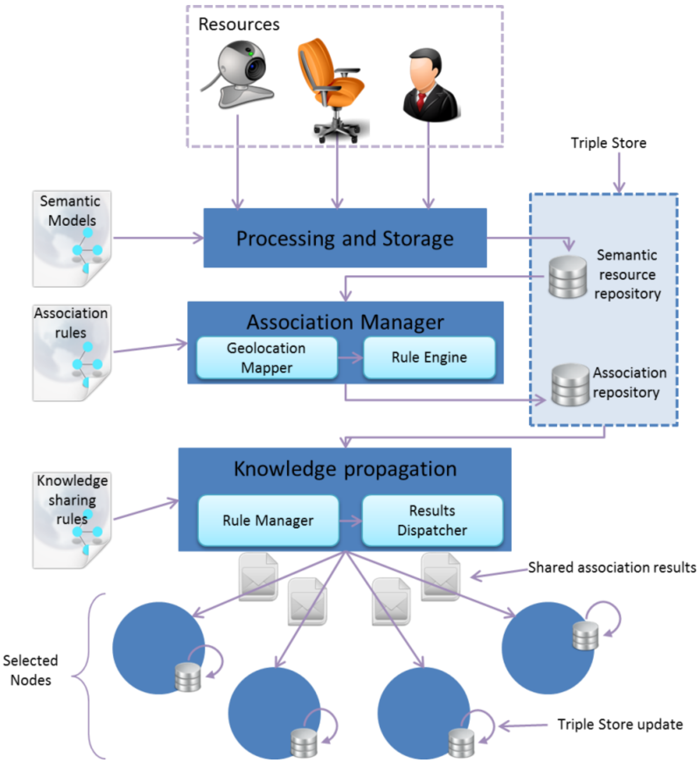

Figure 1: Building blocks of a node

We recall that any considered resource can be mobile and therefore can enter or exit from a geographic place. We assume in this paper the existence of a trigger process that notifies a node about such a join/exit event and provides it with the semantic description of the corresponding resource.

That being said, upon an incoming resource, the *Processing and Storage* functionality block of a node performs management functionalities including checking the validity of the semantic description of such resource. This check uses the semantic models defining an IoT Service and a physical entity –

presented in Section 3. If compliant, the semantic description is translated to a set of RDF triples and inserted into the triple store of the node.

The stored semantic descriptions of the resources are then employed by the *Association Manager* that makes use of *Association rules* to derive associations between a physical entity and the IoT Services that can actuate or provide information about it. The association mechanism is detailed in Section 4.

Finally, the *Knowledge Propagation* block – detailed in Section 5 – uses *Knowledge sharing rules* defining the strategy of information sharing. Defined by a node manager (e.g. someone with administrative rights, managing the node by accessing to its configuration), examples of such rules can be the sharing of all semantic descriptions of incoming IoT Services or physical entities. However, as this can lead to the generation of a high number of messages between nodes, we believe that a good trade-off is to limit the sharing of information to the descriptions of incoming IoT Services.

The *Knowledge Propagation* algorithm also uses an indoor location model – implemented in each node and described in the following Section 2.2 – in order to share the information with nearby nodes (recall that a node is mapped to a geographic area). This indoor location model allows localizing a place relatively to others (e.g. Chemistry lab is next to Computer Science lab) and serves as a basis to initialize and keep updated the federation system by defining how nodes are interconnected.

2.2 Interconnecting nodes and creating the federation system

To build a federated system composed of aforementioned nodes, we propose to create interconnections based on a 'container' approach, meaning that a place 'containing' other places results in as many interconnections as number of contained places (see for instance the curved arrows in Fig. 2 interconnecting N_2 to N_4 and N_5 as a consequence of having the Chemistry lab and the Computer Science lab located in the 2nd floor of a given building). In our vision, the place containing other places acts as a 'manager' of the places it 'contains'. As a consequence, the resulting federated system has a 'top-node' i.e. having no manager. By following this simple placement of rooms relatively to corridors, floors, etc. we enable a federated system to be quickly deployed and extended, i.e. when a room is newly mapped to a node, such a node only needs to contact its 'manager' in order to declare itself as a new node of the federated system. This approach must however be used in conjunction with another process, enabling information acquired by a given node to be shared only with relevant nodes, i.e. those mapped to places nearby the place managed by the given node. As an example, Fig. 2 presents the nodes of the Computer Science lab and the Chemistry lab as being interconnected to the node mapped to the 2nd floor of a University Building. However, it is not because both labs are in the 2nd floor that they should exchange knowledge (consider for instance the case of a floor being 300m long, with both labs localized at the opposite corners. Exchanging knowledge may, in this case, be irrelevant as the distance separating both labs seems too high).

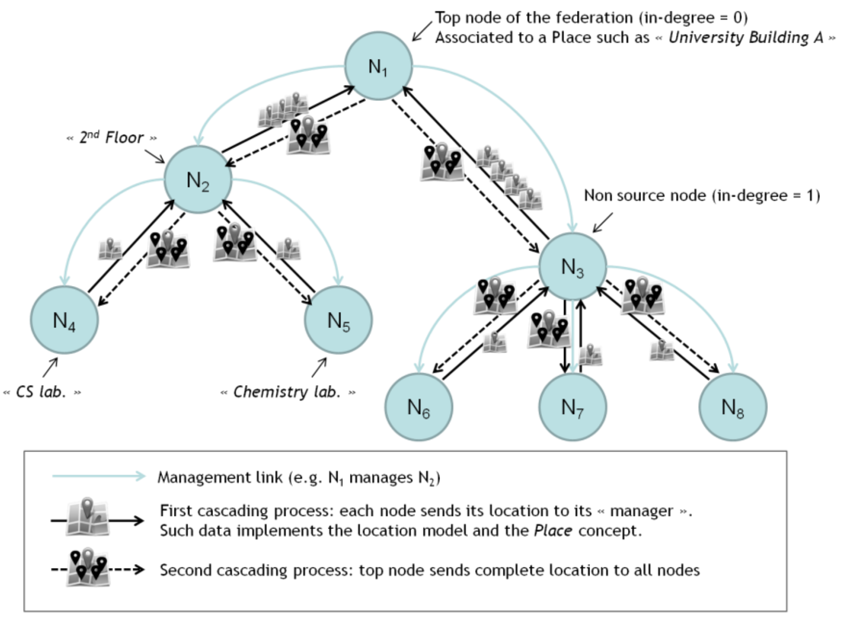

Figure 2: gathering overall nodes' location of a given federation network

To address this issue and to ensure sharing knowledge with the right nodes, it is necessary to be able to describe a place relatively to others, in order to decide whether a place is 'close' enough to another to share information with. Although vocabularies such WGS-84[1] or GeoNames [14] allow describing outdoor places based on their GPS coordinates, describing indoor location places requires a more granular description of the location concept. Towards this aim, we use Semantic Web technologies and in particular the Web Ontology Language (OWL) [15] due to its ability of providing richer descriptions for any kind of resource. The resulting model, depicted in Fig. 3, contains indoor location concepts gathered under a Place concept and representing structures of buildings, rooms, or other premises.

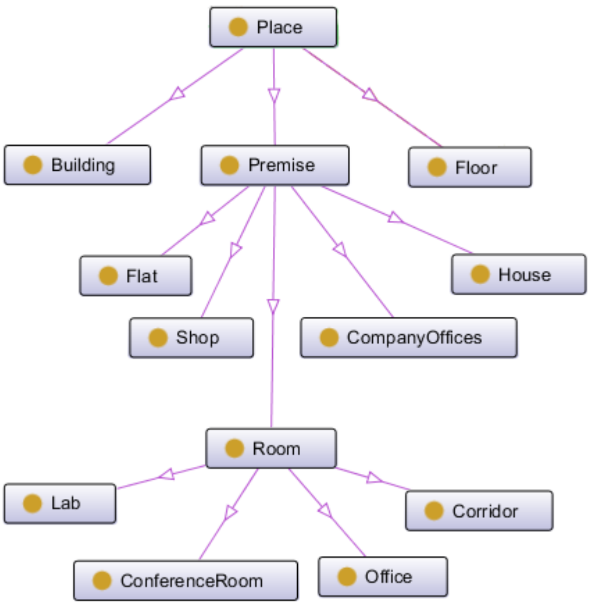

Figure 3: Indoor location concepts

[1] Basic RDF Geo Vocabulary, http://www.w3.org/2003/01/geo/wgs84 pos#

Due to the various types of places that may be described, the Place concept has a broad meaning that can be narrowed to a Building, a Floor, a Premise or other kind of structures[2]. Some of these concepts are formally defined (based on logical predicates), allowing reasoning tasks to be performed. As an instance, a Building concept is modelled as an entity not contained by another Place but that contains at least one Floor and its formal definition is given by the following equation:

$$Building \equiv \{\neg\, ContainedPlace \land contains \text{ some } Floor\} \quad (1)$$

We complete this model by defining the Region concept. Mapped to each place, a Region is defined as a geographical area (i.e. built from coordinates and distances of a place) enabling spatial associations to be derived (see Section 4).

Finally, along with these concepts, we define some OWL properties allowing different places to be interlinked and localized relatively to others (e.g. a Room can 'give access' to another Room). This set of properties, summarized in Table 1, provides a small but necessary core of relations between different places enabling to define knowledge sharing rules (see Section 5).

Note that although this model contains a small set of premises and properties, the import mechanism tied to OWL allows extending it. Consequently, other types of premises can be modelled. Besides, more complex relationships between places may be envisioned. Finally, note that the current proposed model assumes that places have a simple geometrical form (we only consider rectangular or circular places) to compute their Region and describe their relative localizations. Additional properties and concepts may therefore be defined in order to take into account places with more complex geometrical form (e.g. torus, L-shaped structures, etc.).

Table 1: OWL Properties interlinking places

Property Name	**Domain**	**Range**
Description		
Contains	Place	Place
Allows a place to contain other places (e.g. a floor containing some rooms)		
isAdjacentTo	Place	Place
Models that two places are separated by some boundaries		
inEast	Place	Place
inWest	Place	Place
inNorth	Place	Place
inSouth	Place	Place
Refinement of isAdjacentTo, including the cardinal direction(s) of a place relatively to another		

[2] Indoor location model, http://webofdevices.appspot.com/models/owl/complex/indoor location.owl

givesAccessTo	Place	Place
Means that a door exists in the boundary separating two places connecting them		
isIncludedIn	Place	Place
Inverse property of 'contains'		
isPrivate/isPublic/isSemiPrivate	Place	Boolean
Allows to know if a place can be used or not when computing associations		

By implementing this model, each node can be aware of all its 'neighbours' i.e. the ones it will share information with. This is made possible through a double cascading process (represented by straight and dashed arrows in Fig. 2) executed by each node when 'initializing' (recall that a node is a piece of software that is mapped to a place. Equipping a place with a node consists of starting this piece of software). Hence, at initialization, each node communicates the description of the place it manages to the top node using a cascading process. The top node uses a semantic engine to merge this data from all nodes to obtain the overall distribution of nodes in the federation. The same cascading process is then used to relay this inferred distribution data to all nodes. When a new node (i.e. a place implementing some indoor location model concept and containing some connected objects) is added, the above cascading process is performed again. The new node can then begin sharing knowledge about the IoT Services it manages.

3 Models for physical entities and IoT Services

This section presents the ontology models that we have used in this paper to allow associations to be discovered between IoT Services and physical entities and correspond to the Semantic Models block in Fig.1. These models have been proposed as part of our work done in the EU FP7 project IoT-A[3] and are presented in detail in [10]. Here, we briefly present the important concepts and properties of the models which are pertinent to forming associations.

A physical entity can have certain attributes which are its observable or actionable features. These attributes can be related to the domain of the entity and hence be specified in terms of a domain ontology, e.g. temperature attribute in the environmental domain. The domain attribute name is specified as a string, whereas the attribute type could link to other models, for instance, a vocabulary of physical phenomena, such as the Ontology for Quantity Kinds and Units (QU)[4] . The value itself has a literal 'value' and associated metadata information (ValueMetadata). The entity location is defined in terms of a modelled WGS-84 Location concept (hasLatitude, hasLongitude, has Altitude). The location concept also has properties that link to global (hasGlobalLocation) location models and to our proposed indoor location (hasLocalLocation) model. To specify the global location, an instantiation of the Entity Model could specify a URI from existing standards such as GeoNames that models well-known location aspects such as cities, districts, countries and

3 IoT-A: Internet of Things – Architecture (http://www.iot-a.eu/public) contract number: 257521
4 http://www.w3.org/2005/Incubator/ssn/ssnx/qu/qu-rec20.html#Section_dim

universities. Also captured are optional temporal features and links to known vocabularies (e.g. FOAF[5]) for specifying ownership. Part of the entity ontology is shown in Fig. 4.

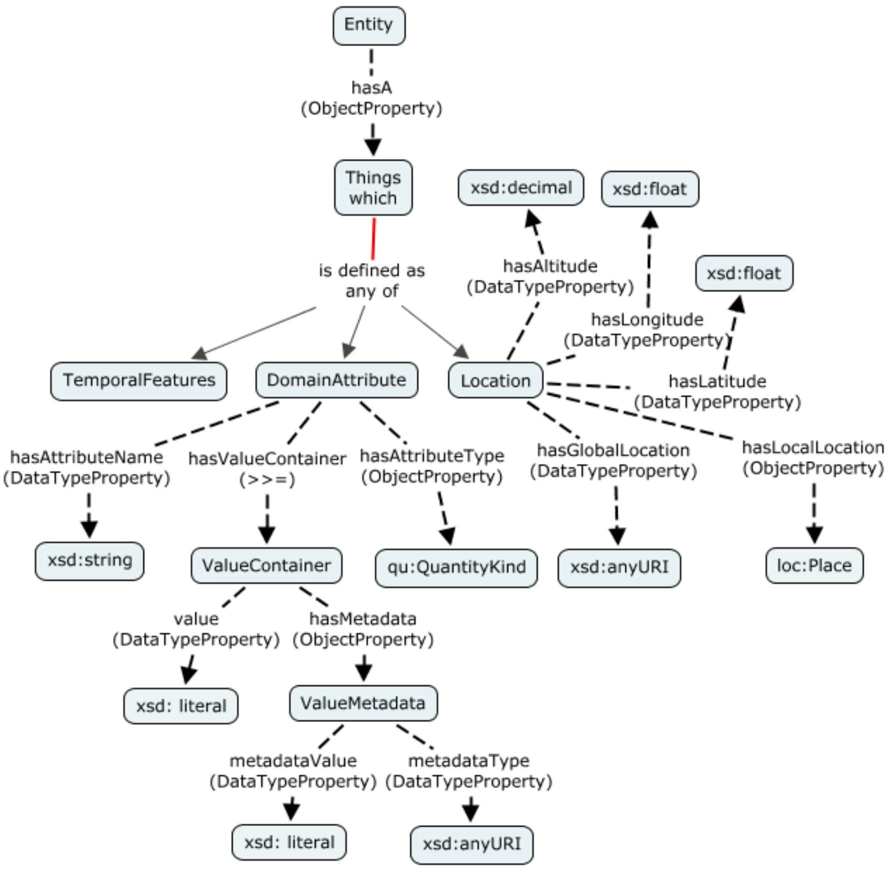

Figure 4: Model describing semantics of a physical entity

The IoT digital component may be a sensor (including RFID tag), actuator or a storage device that stores information obtained from other sensors. Such components can be abstracted as ‘resources’, as detailed in [10]. Many ontologies already exist to detail such devices, e.g. SSN ontology for sensors. Due to the different types of digital components possible in the IoT domain and the resulting hardware and software heterogeneity, the IoT Service model has been designed to provide a uniform abstraction for exposing the functionalities provided by them. Fig. 5 depicts the main properties of the IoT Service model. The ‘exposes’ property represents the mapping of the IoT Service to the corresponding digital component, which could be of different types (rm:hasType property) depending upon the kind of digital component. The resource abstraction allows for both hardware (e.g. sensor, actuator) and software specification (e.g. in the case of storage device) of the digital component.

[5] The Friend Of A Friend project, http://www.foaf-project.org/

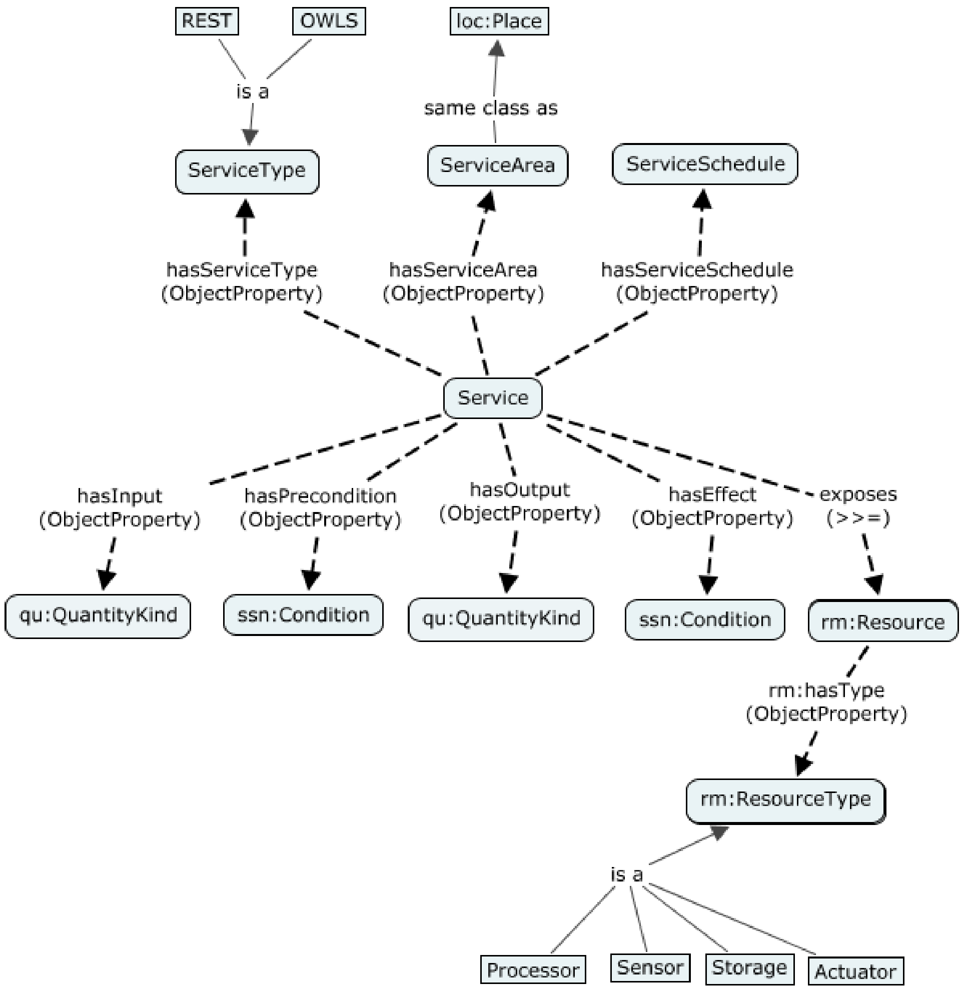

Figure 5: IoT Service Model

The IoT Service model provides the capability to gather information about entities that can be associated with the digital components or to manipulate physical properties of the associated entities. This is modelled using the IOPE (input, output, preconditions and effects) parameters. The functionality of the digital component is captured by the hasOutput (e.g. for sensor services) and hasInput (e.g. for actuator services) properties. The input and output parameters can be specified in terms of the generic instance quantities from the QU ontologies, such as 'temperature' or 'luminosity'. This is then employed for deriving associations. For instance, a physical entity can have an attribute that represents its 'indoorTemperature'. The generic type of this particular attribute is 'temperature'. Then, if there is a service that measures temperature, specified as the service's hasOutput parameter, the corresponding service can be a candidate for a possible association to the relevant entity. For actuating services, the impact on the entity attribute being controlled after the service execution 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. 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 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 service area is defined in terms of the indoor location model 'Place' concept. The possibility of specifying time constraints on service availability is captured through the ‘hasServiceSchedule’ property. The IoT Service also has ID (‘hasID’) and name (‘hasName’) properties.

4 Associations along thematic-spatial-temporal Axes

The concept of a Semantic Sensor Web with thematic, spatial and temporal information was first introduced by Sheth et al. [16], wherein the authors aimed to provide web accessible semantic descriptions of sensor networks and archived sensor data. The sensor data had temporal and location information embedded within the descriptions. There are well-defined thematic or domain-specific ontologies for a number of domains and applications. Specifically, in the sensor domain, different ontologies cover sensor descriptions, sensor site information and sensor observation and measurements. Along with these thematic models, temporal and spatial models are increasingly employed for capturing meaning from data [3]. These can then aid semantic computations, inference and rule-based reasoning that enable semantic search and other IoT applications.

The Association Manager of a node specifies forming the associations between physical and IoT digital objects along the thematic-spatial-temporal axes. Associations between a physical entity and an IoT Service link an attribute of the physical entity to either the IoT Service’s input or output. Thus, according to the IoT Service model detailed in Section 3, the service may either provide information about a physical entity, in which case the service output is of interest, or the service may bring about a change in the physical entity, when we are interested in the service input. In this section, we discuss forming the associations between IoT Services and physical entities through a first set of rules that can be applied when a node’s triple store is updated with new IoT Service instances.

An association is defined along thematic (feature), location and temporal axes, as depicted in Fig. 6. The feature dimension is defined as an intersection between an entity’s domain attribute and the IoT service’s input or output properties. The location axis takes into account the concept of place as defined in the indoor location model. For the location match, the entity needs to be in the IoT service’s service area to allow an association between them. Whenever the location and feature dimensions meet at the same time, associations can be established automatically.

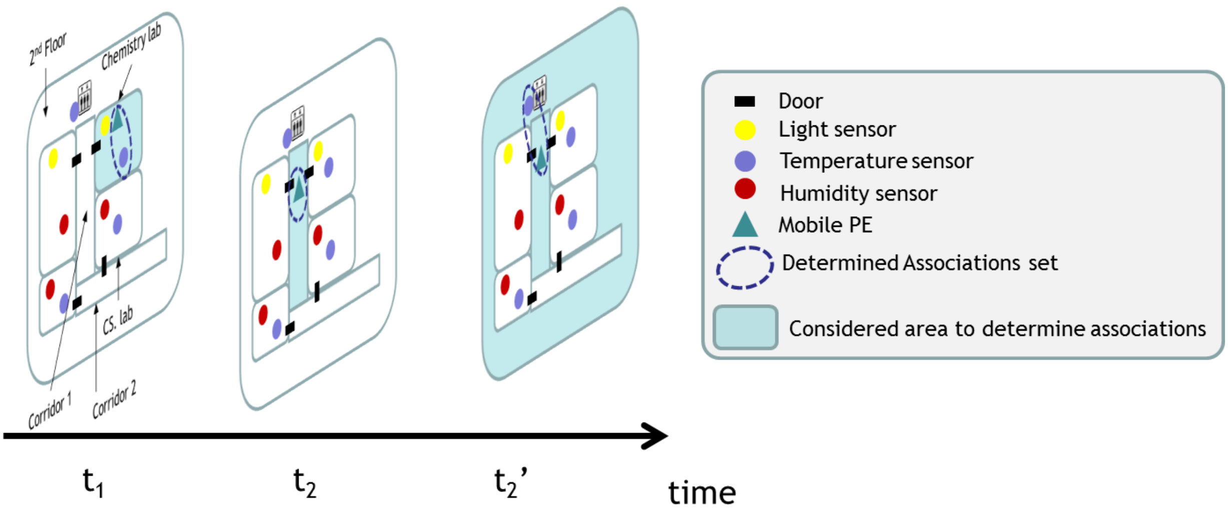

Figure 6: Derivation of Associations along thematic-temporal-spatial axes

Fig. 6 shows a floor of a building with a number of rooms and corridors, with each room having multiple sensors (and hence IoT Services) deployed in it. The placement and boundaries around each depicted sensor corresponds to its service area. A mobile physical entity is situated in the Chemistry Lab on this floor at time t_1 and having a temperature attribute, is thus associated to the IoT Service exposed by the temperature sensor in this room. At time t_2, the entity has moved to corridor 1 and since there are no sensors with a service area matching this corridor, the entity is no longer associated with any service. However, the association mechanism then considers the next higher level space in the indoor location ontology and finds a temperature sensor with service area specified as the floor 2. Thus, the entity is then associated to its IoT Service (shown as t_2' in Fig. 6). As a consequence, we propose the following rule as typified in the Rule Manager block:

A thematic association is asserted if there is a non-empty intersection between the output (or input) of a service and the attribute types of the entity.

4.1 Spatial analysis

Following a match along the thematic attributes, the next step of the association logic is to consider various levels of spatial relations. The location-specific rules follow an incremental approach and make use of the knowledge inferred by the thematic association rules, i.e. only entity-IoT Service pairs matched along the thematic axis are considered for location matching. Since the indoor location ontology allows specifying logical locations for entities as well as the area served by an IoT Service, this can then serve as the basis for deriving spatial associations. However, the current logical location may not be known in all scenarios, e.g. in unfamiliar environments. In such cases, the current location according to the indoor location model needs to be ascertained first. Thus, the Geolocation Mapper block considers the nearest known geographical coordinate and defines an inference mechanism for determining the logical location of a mobile entity. We follow a top-down approach for the inference mechanism as follows:

a) Consider all known ‘place’ concepts from the location ontology (i.e. premises/building/room) and their corresponding ‘regions’. We assume that a region is defined as a polygon including geo-coordinate information (e.g. a sphere, with the coordinate as its centre and a known radius).
b) Starting from the top-node of the federation, i.e. considering a Premise instance, determine its area. Then calculate if the entity’s known coordinate is within the area defined by the Premise instance.
c) If the entity is within the Premises, then consider all Building instances. Similarly, if it is determined that the entity is within the area of a building, then consider individual rooms with asserted dimension properties.
d) If the physical entity is inferred to be within a particular room’s area, its ‘haslocalLocation’ property is asserted to be that of the ID of the room. If the entity is not within any room, but within a building, then the ‘haslocalLocation’ property is set to be the building location and so on.

Once the local location is known, the matching of the physical entity and the IoT Service along the spatial dimension can be defined. The following rules consider four levels of spatial association, depending upon the proximity of the physical entity and the IoT Service:

a) **sameLocation**: the entity's current logical location, as denoted by the 'localLocation' attribute falls within the service's service area.
b) **nearby**: the proximity of the connected device 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 entity's location is adjacent to the IoT Service area, or the device is in a corridor that gives access to the room the physical entity is in, the association is then annotated as 'nearby'.
c) **samePremise**: if the adjacency and access properties yield no valid spatial associations, the association derivation process looks at the next higher level in the location model, i.e. employing the place containment captured in the indoor location model. This can be, for instance, co-location within company offices or houses. The association is then labelled to be within the same premise.
d) **sameRegion**: the resource location matches the global location of the entity, e.g. same city, or county or geographically defined regions.

The temporal logic for the association derivation process follows an event driven strategy tied to the federation framework, i.e. we assume that the rules are triggered based on some context change (e.g. IoT Service/physical entity added to the triple store of a node). Thus, the associations are automatically kept up-to-date regarding the physical entities and IoT Services known to the node at that instant of time and as a result, we do not explicitly employ any temporal variables in the rule-set.

5 Knowledge propagation between nodes

As mentioned in the introduction of this paper, we believe that sharing information between nodes of the federated system can optimize the set of associations obtained by the process described in the previous section. In other words, we believe that a given node will be able to extend the associations it can compute by knowing the IoT Services and the physical entities that 'live' in neighbour nodes. To realize this sharing of information, we design a knowledge sharing process implemented by the Knowledge Propagation block of each node. Triggered each time the triple store of a node is modified (e.g. when adding or removing IoT Service descriptions), this process consists of using the aggregated location information (described in Section 2.2) as well as a list of knowledge sharing rules (Section 5.1). Based on the semantic models defined in Section 3, the rules use Semantic Web technologies. Depending on the rule results, messages are sent to all 'neighbours' of the node with the information to be shared (Section 5.2).

5.1 Knowledge sharing rules

Sharing knowledge between federated nodes is about extending the knowledge of nodes to allow them to derive more associations. Resulting in sharing descriptions of IoT Services or physical entities, this process make use of Semantic Web technologies and is specified in the Rule Manager component of a node. Although many rules could be defined, this section focuses on six particular rules forming a basic strategy about the way a node could exchange knowledge with others. These rules use the generic term resources to refer to semantically described physical entities or IoT Services. Note however that in our vision, the sharing knowledge strategy should be defined by the node manager as being the only one able to decide whether he wants to share information or not. Consequently, the six following rules may be adapted in each node.

The two first rules, trigger a message when an IoT Service (or physical entity) joins or left a place.

1) When a resource has joined a place P, notify all the places accessible from P about this fact.
2) When a resource has left a place P, notify all the places accessible from P that the resource could reach them.

The two following rules, replace the two first ones by ’adjacency‘ concept. Compared to the two first rules, applying these two ones results in sharing information with more nodes (i.e. not only the ones that can be accessed but also the one that have a boundary in common).

3) When a resource has joined a place P, notify all the places adjacent to P about this fact.
4) When a resource has left a place P, notify all the places adjacent to P that the resource may reach them.

The final two rules take into account mobility of resources by associating a learning process allowing nodes to notify other selected nodes that a resource should join them in the near future. In detail, the fifth rule consists of notifying a place P2 that a resource may reach it soon. P2 can then discover beforehand the associations between this resource and the other resources it currently manages. As such associations are predicted, P2 “locks” them (i.e. makes them not retrievable from search) by tagging them as being “prepared”. The sixth rule, finally, consists of unlocking these aforementioned associations by tagging them as being “available” (i.e. retrievable if searched). Note that although not described in this paper, such learning process associates a confidence score to each of these two rules. The more this process has learnt, the higher the confidence score is.

5) When it has been learned that any mobile resource always reaches a place P2 after having reached P1 and if a resource has just joined P1, notify P2 that such resource will join.
6) When the previous pattern has been learned and that a resource leaves P1, notify P2 that a resource joins.

The benefit of using SWRL rules to define how knowledge between nodes has to be exchanged is twofold. First, it allows any node manager to define additional rules, processable by a Semantic Web engine without requiring code to be developed (as long as the rules do not contain calls to custom built-ins unassociated with the engine). Second, SWRL allows custom built-ins to be developed. In particular, some built-ins have been developed (see Section 6) to enable notification features to the ‘head’ of a rule. Therefore, assuming someone having access to the implementation of the Sharing knowledge process, allows developing specific exchange protocols and rules. This flexibility allows policies to be associated to a strategy of knowledge sharing. As an example, two different place managers may decide two different strategies to share knowledge between nodes of the same federated network. Two different federated networks could also lead to different knowledge exchange models. Finally, different policies may be applied depending on their associated business models.

5.2 Notification mechanism

Having selected a set of nodes with which to share some knowledge, a given node needs to send appropriate messages so that its ‘neighbours’ will be notified of new content. Towards this aim, the Result Dispatcher component of the Knowledge Propagation block of a node specifies a notification mechanism. This mechanism leads to generating messages composed of a payload containing results to share and a header containing the appropriate routes that the messages have to follow to reach their respective recipients. Knowledge to share arises from the execution of aforementioned rules (Section 5.1) and is therefore a set of triples.

Determining the path between a given node and the recipient of a message relies on the organizational aspect of the federation (recall Section 2.2 and Fig. 2). Such a path is exactly the list of nodes that need to be crossed, in order to find a ’common manager’of both considered nodes. Computing this path relies on the gathered and inferred location of all nodes and involves the anonymous property ’inverse of contains‘ (with contains – a defined property – and its inverse provided by a Semantic Web engine). This property allows finding the ancestors of both the issuer and the recipient nodes. Hence, with this property, we build two sub-graphs, one starting with the issuer and the other one starting with the recipient. Each time we find ancestors, we check if the 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. Because the nodes cannot have more than one ‘manager’ the federation has no undirected cycles, which ensures that the algorithm converges to one unique solution. For a given result to share the notification mechanism consists then of the generation of K messages (assuming K neighbours). Each message contains a payload composed of a simple envelope to be routed properly as well as the result to. Once having received a result, a selected node processes it and updates its triple store.

6 Implemented framework

This section presents the prototype that we have realized to assess the processes described in Sections 4 and 5. Section 6.1 presents our implementation of the architecture components described in Section 2, while Section 6.2 presents the implementation of the notification process that allows sharing knowledge between nodes.

6.1 Implementation of architecture components

6.1.1 Implementation of a node

Our implementation considers that a node of the federated system is embodied in a Java Web application deployed in a servlet container such as Tomcat. This Web application orchestrates the three blocks presented in Fig. 1 that have been implemented as follows.

The *Processing and Storage* functionality block uses an *RDF-based API* capable of processing semantic descriptions. Reading and processing these descriptions is performed using the OWL API [17] coupled with Pellet [18], a semantic engine capable of reasoning on OWL ontologies. Once checked, these descriptions are inserted into OWLDB [19], acting as the triple store of a node.

The *Geolocation Mapper* of the *Association manager* determines if an entity’s geographical coordinates lies within the area defined by a known location (premise/building/room).This is implemented by using the JTS Topology Suite [20] APIs. The steps are as follows: (a) create an object of class jts.geom.Polygon for the relevant Place instances, (b) take the physical entity’s geographical coordinate and create an object of class jts.geom.Point and (c) determine if the Polygon covers the Point. If it is true, then the entity is within the area defined by the matching place instance. Since this functionality is only executed in certain specific conditions as specified in Section 4.1, the associated complexity does not impact the federated system working.

The *Rule Engine* then implements an expert system using the SWRL Factory Java APIs and the Jess inference engine. It is worth noting that the rules are independent of the inference engine used, allowing the SWRL-Jess bridge to be replaced with another implementation of an inference engine that can execute SWRL rules. The derived property assertions are not inserted into the actual service or entity models, thus avoiding violating OWL's monotonicity. However, the inferred knowledge is held within the rule engine, so that subsequent rules and queries can make use of the inferred associations. The derived associations are stored in a triple, with the entity-ID and the IoT service ID associated by the corresponding entity attribute. These triples are then written into the *Association Repository* in the node for subsequent queries. Table 2 shows a SWRL realization of some of the association rules:

Table 2: SWRL association rules

Rule-1:

srv:Service(?s) ∧ srv:hasOutput(?s, ?out) ∧ em:Entity(?et) ∧ em:hasA(?et, ?da) ∧ em:hasAttributeType(?da, ?atype) ˚ sqwrl:makeSet(?sr, ?out) ∧ sqwrl:groupBy(?sr, ?s) ∧ sqwrl:makeSet(?se, ?atype) ∧ sqwrl:groupBy(?se, ?et) ˚ sqwrl:intersection(?in, ?sr, ?se) ∧ sqwrl:size(?n, ?in) ∧ swrlb:greaterThan(?n, 0) → assoc:sameFeatureAs(?s, ?et)

Rule-2:

assoc:sameFeatureAs(?s, ?et) ∧ srv:hasServiceArea(?s, ?sa) ∧ em:Entity(?et) ∧ em:hasA(?et, ?l) ∧ em:hasLocalLocation(?l, ?loc) ˚ sqwrl:makeSet(?rsa, ?sa) ∧ sqwrl:groupBy(?rsa, ?s) ∧ sqwrl:makeSet(?eloc, ?loc) ∧ sqwrl:groupBy(?eloc, ?et) ˚
sqwrl:intersection(?in, ?rsa, ?eloc) ∧ sqwrl:size(?n, ?in) ∧ swrlb:greaterThan(?n, 0) →
assoc:isAssociatedWith(?s, ?et)

Rule-3:

assoc:sameFeatureAs(?s, ?et) ∧ srv:hasServiceArea(?s, ?sa) ∧ em:Entity(?et) ∧ em:hasA(?et, ?l) ∧ em:hasLocalLocation(?l, ?loc) ∧ loc:givesAccessTo(?sa, ?loc) → assoc:isAssociatedWith(?s, ?et)

Rule-4:

assoc:sameFeatureAs(?s, ?et) ∧ srv:hasServiceArea(?s, ?sa) ∧ em:Entity(?et) ∧ em:hasA(?et, ?l) ∧ em:hasLocalLocation(?l, ?loc) ∧ loc:isAdjacentTo(?sa, ?loc) → assoc:isAssociatedWith(?s, ?et)

Rules in Table 2 use the namespaces referring to the use of the service (srv prefix), entity (em prefix) and location models (loc prefix) defined in Sections 2 and 3, the defined association model (assoc prefix) and the SWRL (swrlb prefix) and SQWRL (sqwrl prefix) built-in libraries.

Rule-1 implements the feature association, expressed as a 'sameFeatureAs' property. It infers a match between sensor services and entities, if there is a non-null intersection between the output of a service, ('hasOuput' object property) and the attribute types of the entity ('hasAttributeType' property), made possible since both property ranges map to the QU ontology instances. Both being object properties, rules out a literal string matching operation through SWRL built-ins for string comparison. Moreover, an entity may have multiple domain attributes and thus, multiple attribute types. Thus, we use the SQWRL collection operators for set theory operations to derive a non-null intersection. First, the 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 a non-null intersection associates the relevant service-entity pairs through the same feature property. A similar rule can be written for actuating services, with the 'hasInput' property of the service being considered.

The rules to derive location association build upon the feature association rule results, i.e. the service and entity instances considered in these rules is the subset that are already associated along the feature axis. Thus, 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. It asserts an association when the physical entity's current location and the IoT service's service area intersect. Rules 3 and 4 implement the 'nearby' association where the service area is adjacent to, or gives access to (as known from the indoor location model properties) the entity's current location. Other rules can be formulated along similar lines to derive 'sameArea' association by matching the premises of the service areas and entity locations. The 'sameRegion' association matches the service area with the global location of the entity; this can be the case when the service area covers the same city where the entity is located.

Finally, the *Rule Manager of the Knowledge Propagation* block extends the features offered by SWRL and makes use of customized built-ins to create rules containing directives that initiate the exchange of information messages between different nodes. These built-ins implement an interface of Pellet (com.clarkparsia.pellet.rules.builtins.GeneralFunction), are packaged in a library and are loaded when the node starts. Custom built-ins are further registered to Pellet through a *BuiltinRegistry* class. Only once all built-ins have been registered, an instance of Pellet is created enabling rules using such custom built-ins to be processed by the semantic engine.

Table 3 denotes a SWRL realization of rules (1) and (5) detailed in Section 5.1. These rules make use of prefixes referring to the indoor location model described in this paper (loc prefix), the service models (the srv prefix), SWRL built-ins connected to machine learning processes (the pattern prefix) or notification mechanisms (alert, notify and pnotify patterns). They involve concepts, properties and constants that can be found in the aforementioned semantic models.

Table 3: SWRL expressions of rules 1 and 5 presented in section 4.2

loc:Place(?p1) ∧ loc:Place(?p2) ∧ loc:givesAccessTo(?p1, ?p2) ∧ srv:IoTService(?s) ∧ alert:notify(?p1, ?s, loc:JOIN) → notif:notify(?p2, ?p1, ?s, loc:JOIN)
loc:Place(?p1) ∧ loc:Place(?p2) ∧ srv:IoTService(?s) ∧ srv:isMobile(?s, xsd:true) ∧ pattern:isNext(?p1, ?p2) ∧ alert:notify(?p1, ?s, loc:JOIN) → notif:pnotify(?p2, ?p1, ?s, loc:WILL_JOIN)

About developed patterns, the features mentioned in these rules act as follows:

- *pattern:isNext* checks if the next node that a resource will join is a given node and returns a probabilistic score.
- *alert:notify* simply checks if an entity has joined or left a given node.
- *notif:notify* sends messages to nearby nodes about a fact that has (or will) happen. Its associated probability score is equal to 1.

- *notif:pnotify* sends messages to nearby nodes about a fact that may happen with a certain probability. Getting such probability information is outside the scope of this paper. Thus, the overall idea is to return a score taking into account the number of nodes that are accessible from or adjacent to a considered node.

6.1.2 Interconnecting nodes as a federated system

As mentioned in Section 2, interconnection of nodes is realized by a double cascading process. In our implementation, this process is achieved by attaching configuration parameters to each node. Amongst these parameters, one is an accessible endpoint of the manager of a given node (recall N_2 managing N_5 in Fig. 2). As our nodes are embodied in Web applications, this accessible endpoint is a URL mapped on a piece of code able to process incoming requests. The following shows an extract of a *web.xml* document used to configure our Web application. Note that a node without the ‘manager’ parameter is supposed to be the top node of the federated system (see Listing 1).

```
<context-param>
  <param-name>manager</param-name>
  <param-value>192.168.1.21:8888/SecondFloor</param-value>
</context-param>
```

Listing 1 : Context parameter given the endpoint of the manager of a node

At initialization, a node is configured with the values of these parameters and becomes capable of contacting its manager. Thus, it enables the implementation of the curved arrows shown in Fig. 2. Initialization of a node continues by reading a second parameter giving a pointer to the semantic description of the place this node supervises. This step is justified by the fact that we assume that a node may not have explicitly said who all its neighbours are.

Computation of the neighbours of a node is described by Algorithm 1 and starts by a node sending the description of its indoor location to its manager. This message is forwarded between different managers until reaching the top node of the federated system (first cascading process). By receiving this message, the top node aggregates this new amount of location data with those it is already aware of (e.g. location data previously sent by other nodes). It then recomputes all neighbours of all known nodes by calling a semantic engine and passing this aggregated information. Finally, this manager notifies all nodes it has previously received location information with this updated location model. The process is repeated until all nodes of the federated system received a notification message.

```
// initialization variables
indoor_location_desc ← Config.get_parameter("indoor_location_desc");
semantic_engine← Pellet.get_reasoner("OWL_reasoning");
manager ← Config.get_parameter("manager");
managed_descriptions ← []

// double cascading process triggered when a node starts
Procedure: start()
  send_message("UPDATE_DESCRIPTION", manager, indoor_loc_desc);

// the following procedure handles incoming messages, e.g. issued from other nodes
Procedure: handle_incoming_message(type, content)
  if content ≠ <> then
    if type = "UPDATE_DESCRIPTION" then
      // keep track of all nodes this one manages
      managed_descriptions ← content;
      // update description of this node by merging the received info
      indoor_location_desc.add_triples(content);
      // if this node is the top node of the federated system, infer on the merged location
      if manager = <> then
        // update the ontology used by the semantic engine
        semantic_engine.update_ontology(indoor_location_desc);
        // (re)infer relationships between places
        semantic_engine.infer();
        // send inferred triples back to all managed nodes
        foreach managed_node in managed_descriptions do
          send_message("DESCRIPTION_UPDATED", managed_node.endpoint,
                              semantic_engine.get_inferred_ontology());
      else
        send_message("UPDATE_DESCRIPTION", manager, indoor_location_desc);
    else if type = DESCRIPTION_UPDATED then
      // updates all managed nodes with the updated description
      foreach managed_node in managed_descriptions do
          send_message("DESCRIPTION_UPDATED", managed_node.endpoint,
semantic_engine.get_inferred_ontology());
```

Algorithm 1: Getting all the neighbours of a node with a double cascading process

6.2 Implementation of the notification process

The Results Dispatcher of the Knowledge Propagation block uses the JGraphT[6] open source library that has features to build graphs to determine the path between two nodes willing to share knowledge. To establish a graph between two nodes *A* and *B*, we fed JGraphT with data retrieved from the aggregated and inferred location data. Considering that the knowledge has to be sent from *A* to *B*, our implementation uses the property loc:givesAccessTo – loc being the prefix used to refer to the location model of Section 2.1 – to build two subgraphs (see Algorithm 2), respectively called left subgraph (starting with node *A*) and right subgraph (starting with node *B*). Building the left subgraph consists of asking a Semantic Web engine to provide all nodes $\{N_i\}$ such that "A loc:givesAccessTo N_i" and to reiterate this request on the nodes having been found. The right subgraph uses the inverse of loc:givesAccessTo property and therefore returns the list of nodes N_j

[6] JGraphT a Java graph library providing mathematical graph-theory objects and algorithms, http://jgrapht.org/

such as “B inverseOf(loc:givesAccessTo) N_j”. Having no undirected cycles in the federated system allows us to ensure that our algorithm terminates (i.e. as givesAccessTo is a symmetric property, iterations on such a property may have led to infinite loops). Having obtained the two subgraphs, we search if both contain common vertices. Searching for common vertices in these graphs is possible due to the fact that each vertex is associated with a unique URI, as representing a Place, defined using the indoor location model presented in Section 2.2. Finally, in the case of common elements found, we merge both graphs and apply the Djikstra algorithm [21] to find the shortest path between A and B.

Once the path between the two nodes is determined, the developed SWRL built-ins fire HTTP messages containing the customized HTTP Request header (referred to as X-nodes in Listing 2) containing the ordered list of nodes retrieved when establishing the path between the nodes. The content of this HTTP message consists of a SPARQL Update query containing the triple(s) to push in the triple store of the recipient node. This message is sent to the first node to cross and then goes through all the other nodes appearing in X-nodes. Each time the message is forwarded by a given node, its IP address appears in the standardized “via” header while it is removed from the X-nodes one. The following Fig. 7 summed up this notification process.

```
POST <N3_ip_address>/store/update HTTP/1.1
Content-Type: application/sparql-update
X-nodes: <N4_ip_address>, <N5_ip_address>, <N6_ip_address>
Via:
Content-length: 340
Prefix assoc: <http://models.iot-a.eu/association.owl>
DELETE DATA { <http://[N1_ip_address]/service/webcam1234.rdf>
        assoc:isAssociatedWith <http://dblp.l3s.de/d2r/resource/authors/
        Benoit_Christophe> }
INSERT DATA { <http://[N1_ip_address]/service/webcam1234.rdf>
        assoc:isAssociatedWith <http://dblp.l3s.de/d2r/resource/authors/
        Suparna_De> }
```

Listing 2: Message sent between two nodes

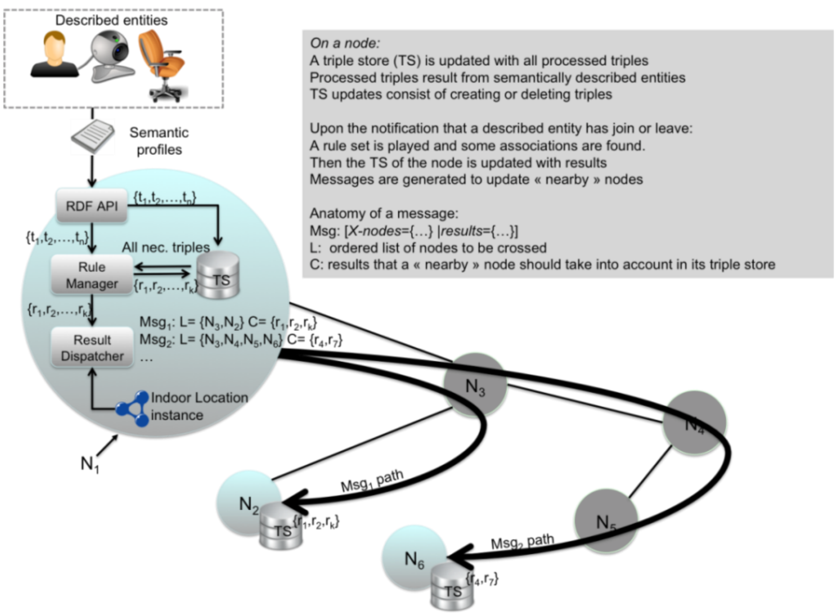

Figure 7: Sharing associations between nearby nodes

```
// Create a DAG using JGraphT library
SG ← JGraphT.create_DAG(Node, DefaultEdge);

Procedure: create_subgraph(n):
Require: n ≠<> and n typeof Node
        JGraphT.add_node(SG, n);
        analyze(n, direction);

Procedure: analyze(node, direction):
        // Analyze node to build its subgraph SG
Require: node ≠<> and (direction = "left" or "right")
        subnodes ← [ ];
        predicate ← "";
        if direction="left" then
                predicate ← "loc:givesAccessTo";
        else
                predicate ← "inverseOf(loc:givesAccessTo)";
        end if
        subnodes ← get_rdf_objects(node, predicate);
        if subnodes ≠<> and subnodes.length ≥ 1 then
                for all sn in subnodes do
                        if sn ≠<> then
                                add_node(sn, node);
                                add_node(sn, node);
                                analyze(sn);
                        end if
                end for
        end if

Procedure: add_node(node, parent):
        // Add a node in the DAG
Require: node ≠ <> and parent ≠ <>
        if node ∉ SG and parent ∈ SG then
                JGraphT.add_edge(SG, parent, node);
        end if

Procedure: get_rdf_objects(subject, predicate):
        //Get a collection of objects objet such that (subject, predicate, object) exists in the
        knowledge base
Require: subject ≠ <> and predicate ≠ <>
        objects ← [ ];
        objects ← Reasoner.get_objects(subject, predicate);
        return objects;
```

Algorithm 2: Compute the left or right subgraph SG of a given node n

7 Evaluation and discussion

To evaluate our implemented framework, the indoor location model has been instantiated with different types of premises, namely, floors, corridors and various types of rooms (offices, meeting rooms and labs) across different buildings. A node has then been deployed in each described premises to build up a federated architecture, comprising of four levels of management (i.e. the maximum distance between the root and the leaf node). Our evaluation approach consists of testing the applicability of the implemented mechanisms through a scenario validation and showing the feasibility of the approach by quantitatively evaluating the scalability of the proposed framework.

7.1 Scenario validation

The proposed mechanisms have been applied to a scenario that is representative of dynamic IoT systems. The testbed consists of a number of sensors deployed in rooms in a university building, with four floors in the building. We limit the service areas of the IoT Services to the room location. We organized the testbed into a federated network of nodes, comprising up to four management levels (i.e. university premise, building, floor and room). The distribution on a given floor is as shown in Fig. 8 (blue circles represent sensor locations). The deployment of the IoT Services in each node triggers its Processing and Storage block which processes the corresponding semantic descriptions and stores them in the triple store. Once this is done for each node, the double cascading process allows the information related to the distribution of the nodes to be shared within the federation.

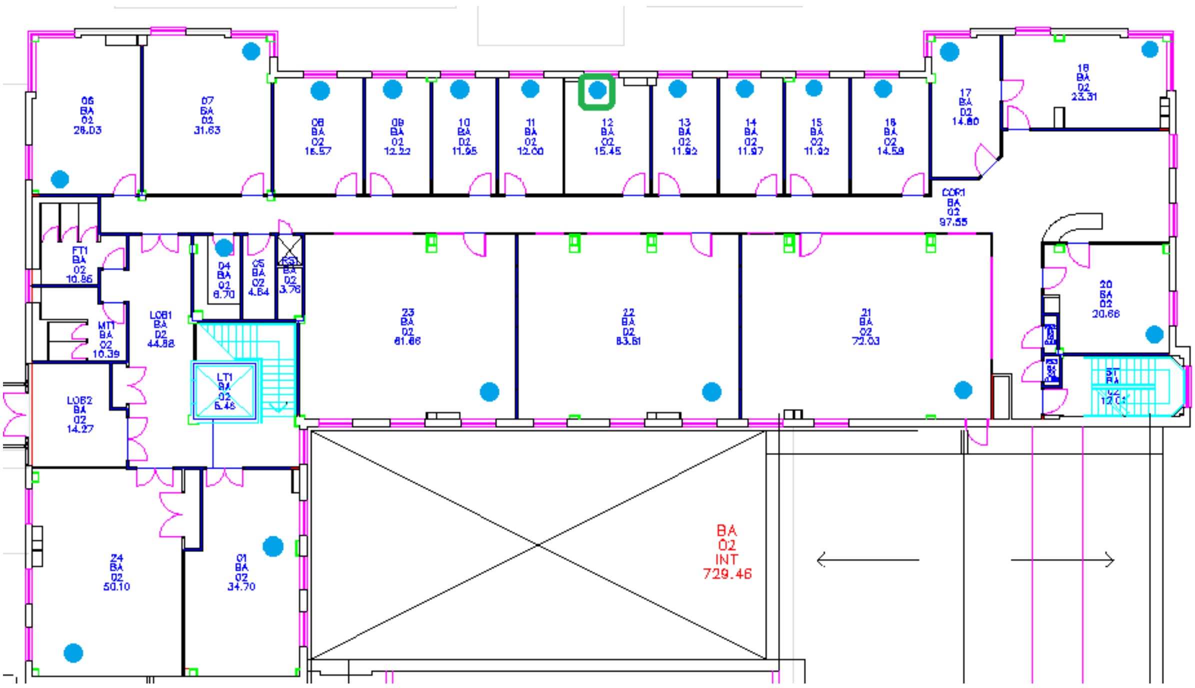

Figure 8: Dataset visualization on a floor plan

The first case of the scenario consists of an entity, John, who moves around the university premises and is interested in finding the relevant sensors that can give him an idea of his ambient temperature at any given location. John's current location is known in terms of geographical coordinates. A user application allows this request to be received and triggers insertion of the entity description (i.e. FOAF profile and temperature attribute) into the node's triple store. This then feeds the Geolocation Mapper which translates the received latitude, longitude pair to an indoor location model instance, which is asserted to be John's 'localLocation' property. In this case, this is determined to be a room, corresponding to 12BA01 in Fig. 8. Since the room contains a temperature sensing service (circled in green in Fig. 8), it is associated to John by the association rules executed by the Association Manager's Rule Engine.

The second case of the scenario showcases relocation of a sensor from one room to another, and thus a change in the semantic description of its IoT Service. The generated event (IoT Service joining a place) triggers the Rule Manager of the Knowledge Propagation block which executes the relevant knowledge sharing rules to determine the set of nodes to be updated. The Results Dispatcher then employs the notification algorithm to determine the path to the selected nodes and the IoT Service's semantic description is sent to these nodes.

7.2 Performance measurements

Our evaluation approach consisted of a number of performance related experiments. The first experiment we performed was to assess the time taken to compute associations, by varying the number of IoT Services to be taken into account by the Association Manager, from 20 to 2000. We run this experiment on a Personal Computer with a standard configuration (Intel Core 2 Duo processor – 2.26 GHz frequency – 2 GB RAM – Ethernet connection). We used a centralized triple store containing all the semantic descriptions of the IoT services considered. To determine associations, we also used a fixed set of five described physical entities. Associations were then derived using the logic of the Association Manager. The results displayed in Figure 9 show the exponential growth of the time required to derive associations, in function of the number of IoT Services.

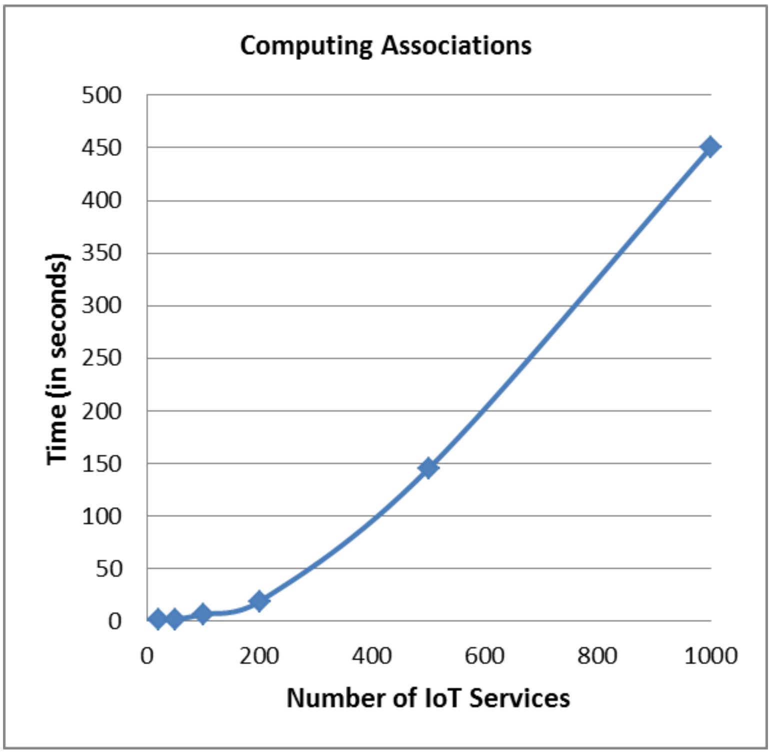

Figure 9: Association computation measurements

This experiment highlights the computationally expensive task of recomputing associations and validates the inappropriate use of a centralized approach to do so. As an example, Fig. 9 shows that 20s are required to recompute associations involving 200 IoT Services, a number that may however be quickly reached when deploying sensors in a whole building. This conclusion bolsters our belief that a federated architecture would be a more feasible deployment option in IoT scenarios, where each node would manage only a limited number of IoT Services.

We assess the scalability of the federated framework by a second experimentation quantifying the number of messages exchanged with different nodes sharing information as well as the time taken to process these messages. For this experimentation, we used the 20 nodes of the federated system associated to the Building displayed in Fig. 8 and deployed 50 IoT Services in each of them (i.e. the overall system was managing 1000 IoT Services). We then simulated the relocation of groups of sensors to evaluate how the number of sensors relocated was impacting the federated system compared to a centralized approach. Tests involved respectively the relocation of 1, 20 and finally 50 IoT Services. For this experimentation, we used a node sharing knowledge with only one other node. Consequently, respectively 1, 20 and 50 messages were generated. Upon receptions of these messages, semantic descriptions of relocated sensors were retrieved by the node and, finally, associations were derived. Fig. 10 summarizes the overall times that we have obtained.

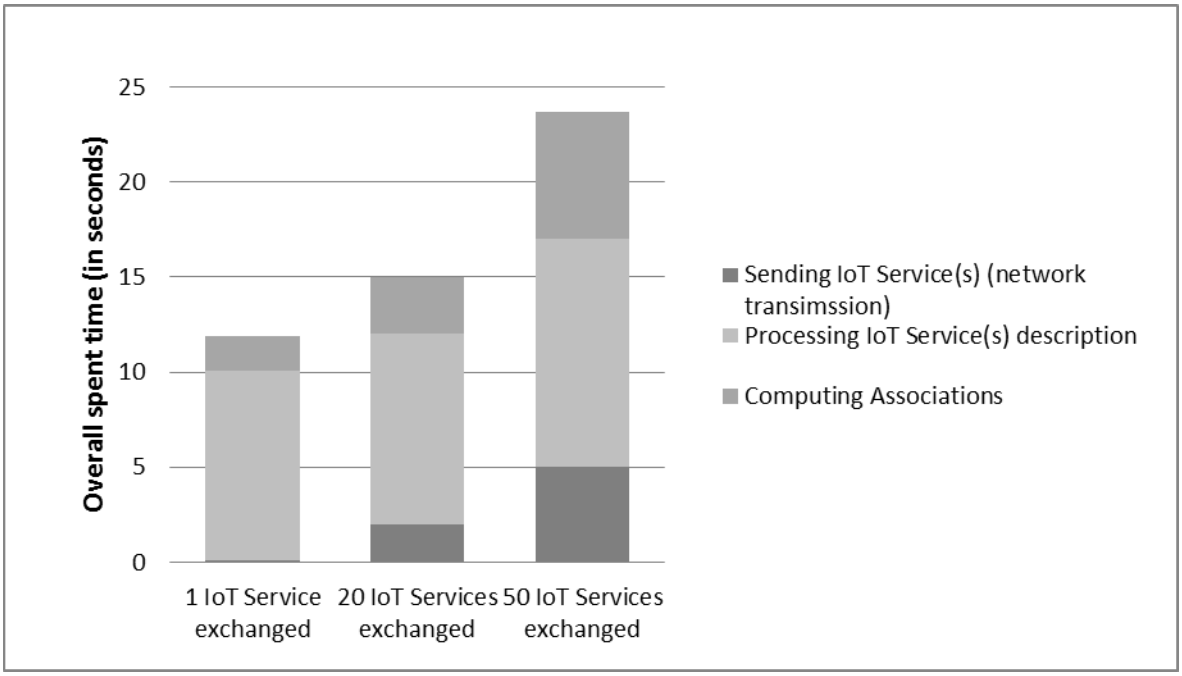

Figure 10: Measurements for maintaining the federated system when IoT Services are relocated

These times are decomposed in the time taken to send the set of messages, the time taken to load the semantic descriptions associated to these messages and the time taken to recompute associations. This figure indicates that the time spent in sending messages follows a linear growing (function of the number of messages to send) resulting in a significant amount of time added by the knowledge sharing process. Besides, this figure shows that the time taken to load semantic profiles of sensors was constant. Finally the time to compute associations follows a similar curve than what was presented in Fig. 9. Compared to a centralized approach deriving associations with 1000 IoT Services, these times stay however much more acceptable (see Fig. 9 showing a time of 645s to derive associations with 1000 IoT Services).

Finally, we did a third experimentation checking whether the number of nodes crossed by a knowledge sharing message was impacting the federated system or not. We then run the scenario of the relocation of one sensor multiple times; varying the route of this relocation by changing the recipient room. Such scenario provided us with a set of messages, each having been propagated differently (i.e. having crossed up to 5 nodes). Although the time increased linearly with the number of nodes having been crossed, the results displayed in Fig. 11 shows that it could be disregarded compared to others (i.e. time to load the semantic description of the relocated sensor and time to recompute associations using 51 IoT Services).

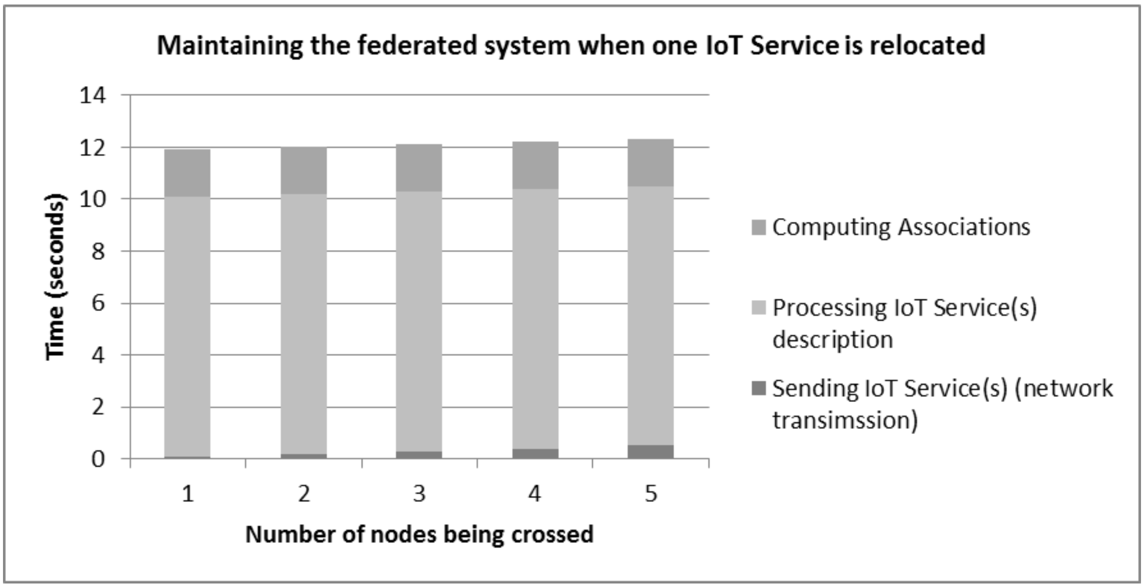

Figure 11: Maintaining the federated system when one IoT Service is relocated

8 State of the art

Due to the nascent IoT paradigm, it is relevant to look at on-going research in allied areas such as the broad sensor Web community. In this section, we first review other research works that have looked at linking sensor descriptions or data to existing data sources. An ontology-based event detection system for wireless sensor networks by Danieletto et al. [22] automatically classifies any sensing device based on its capabilities and any event based on its source and detection place. The device classification method categorizes sensor types based on the detected data. The presented event classification algorithm distinguishes between general, focused and outlier events based on the number of sensors detecting the event values and agreed threshold values. Yu et al. [23] use the Linked Data approach to integrate sensor Web data with geospatial, streaming and event data sources in the context of integrated water resource decision support. The thematic-spatial-temporal concept for annotating sensor Web observation data was first proposed by Sheth et al. [16]. This concept was extended with the Linked Data concepts by Barnaghi et al. [24] to allow users to publish linked sensor data for sensor site information that is associated to existing resources that are already a part of the Web of data. In this proposed work, we take the theme, time and space concept and extend it to the IoT world to associate physical world objects with digital world objects that can provide information or mediate interaction with the physical objects.

Among the middleware approaches proposed for the IoT, some have applied semantics to objects to leverage the benefits of interoperability that Semantic Web technologies provide. Katasonov et al. [25] propose coupling of ontologies with agents, interconnected with the FIPA[7] specification, to develop a middleware allowing heterogeneous devices to cooperate. They employed Semantic Web Service ideas [26] to create a Semantic Web of Things composed of agents presenting semantic

[7] FIPA Specification, http://www.fipa.org/specifications/index.html

profiles of devices that they were monitoring. The agents process incoming semantic requests by triggering appropriate device functionalities. Boussard et al. developed a Web of Things (WoT) framework exposing smart environments and their constituents as Web resources [27]. This framework relies on the concept of Virtual Object (VO) and makes use of semantic profiles [28] coupled with reasoning mechanisms to propose locally relevant objects [29]. A middleware to couple the envisioned IoT architecture with enterprise applications has been proposed in [6]. The proposed SOCRADES middleware architecture enables enterprise-level applications to interact with and consume data from a wide range of networked devices, including sensors. Device abstraction is achieved by device proxies that integrate low-capacity devices to the platform and expose the offered functionalities as services on the middleware. It relies on Web Services for all communication interfaces. The middleware supports composition of IoT-level services. It implements a service implementation repository that stores all services that are available for composition of new services, orchestration of business process or deployment. Pfisterer et al. [30] have proposed an architecture allowing enhanced integration of sensor data and services. Their approach includes defined vocabularies that facilitate integration of descriptions of sensors and things with Linked Open Data (LOD) cloud[8] and the search mechanisms take into account sensor states (e.g. availability). User queries were answered by querying a triple store with SPARQL.

All of the middleware approaches reviewed here contain similarities with the one presented in this paper. However, our approach differs in the fact that we integrate the geographical distribution of objects (sensors, actuators etc.) into a federated architecture of nodes allowing efficient distribution of knowledge. The above approaches consider a unique registry where all user requests are processed. Although some approaches have mentioned that the registry could be implemented across distributed servers, none of them have addressed the benefits of distributing the knowledge gathered by a node with a selected set of geographically nearby peers.

9 Conclusions

This paper presents an exploratory, development-oriented approach for associating physical and digital world objects forming part of the Internet of Things. The associations are defined in an automated way, along the concepts of theme, time and space. We have also proposed a scalable, distributed framework of nodes organized in a federated architecture, with each node capable of processing the semantic descriptions of the objects comprising the IoT and their associations. Though other approaches have also applied Semantic Web technologies for achieving interoperability between the connected objects in the IoT domain, our approach additionally considers a particular deployment infrastructure, with each node been mapped to an indoor physical environment. This facilitates local reasoning capabilities and makes use of proximity knowledge for inter node communication, thus allowing a solution to the scalability issue of IoT. Our approach also takes into account mobility of entities or devices within the infrastructure, making use of SPARQL 1.1 update support. Our future initiatives involve expanding the temporal dimension for associations, for alignment with the SWRL temporal ontology. Integration of the service model with on-going initiatives like SSN and Linked USDL[9] are also envisaged.

[8] Linked Open Data Cloud, richard.cyganiak.de/2007/10/lod/
[9] http://www.linked-usdl.org/

10 Acknowledgements

This paper describes work undertaken in the context of the IoT-A project, IoT-A: Internet of Things – Architecture (http://www.iot-a.eu/public) contract number: 257521.

11 References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, I. Chlamtac. Internet of things: Vision, applications and research challenges, Ad Hoc Networks, 10 (2012) 1497-1516.

[2] A. Gluhak, S. Krco, M. Nati, D. Pfisterer, N. Mitton, T. Razafindralambo. A Survey on Facilities for Experimental Internet of Things Research, IEEE Communications Magazine, 49 (2011) 58 - 67.

[3] A. Sheth. Computing for human experience: Semantics-empowered sensors, services, and social computing on the ubiquitous Web, IEEE Internet Computing, 14 (2010) 88-91.

[4] L. Atzori, A. Iera, G. Morabito. The Internet of Things: A survey, Computer Networks, 54 (2010) 2787–2805.

[5] H. Abangar, P. Barnaghi, K. Moessner, R. Tafazolli, A. Nnaemego, K. Balaskandan. A Service Oriented Middleware Architecture for Wireless Sensor Networks, in: Proceedings of Future Network & Mobile Summit 2010, Florence, Italy, 2010.

[6] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, D. Savio. Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services, IEEE Transactions on Services Computing, 3 (2010) 223-235.

[7] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. Souza, V. Trifa. SOA-based integration of the internet of things in enterprise services, in: Proceedings of IEEE ICWS, Los Angeles, CA, USA, 2009.

[8] W3C Semantic Sensor Networks Incubator Group (SSN-XG), 2011. <http://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/> (online, accessed August, 2012)

[9] M. Compton, P. Barnaghi, L. Bermudez, R.G. Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, V. Huang, K. Janowicz, W.D. Kelsey, D.L. Phuoc, L. Lefort, M. Leggieri, H. Neuhaus, A. Nikolov, K. Page, A. Passant, A. Sheth, K. Taylor. The SSN Ontology of the Semantic Sensor Networks Incubator Group, Journal of Web Semantics, (2012).

[10] S. De, T. Elsaleh, P. Barnaghi, S. Meissner. An Internet of Things Platform for Real-World and Digital Objects, Journal of Scalable Computing: Practice and Experience, 13 (2012) 45-57.

[11] D. Heimbigner, D. McLeod. A federated architecture for information management, ACM Trans Inf Syst, 3 (1985) 253-278.

[12] M. Balazinska, H. Balakrishnan, M. Stonebraker. Contract-based load management in federated distributed systems, in: Proceedings of the 1st conference on Symposium on Networked Systems Design and Implementation - Volume 1, San Francisco, California: USENIX Association, 2004, pp. 15-15.

[13] S. Ternier, D. Olmedilla, E. Duval. Peer-to-Peer versus Federated Search: towards more Interoperable Learning Object Repositories, in: Proceedings of World Conference on Educational Multimedia, Hypermedia & Telecommunications, 2005, pp. 1421-1428.

[14] GeoNames. GeoNames ontology, 2011. [Online]. Available: http://www.geonames.org/ontology/documentation.html. Accessed: June, 2012.

[15] OWL Web Ontology Language, W3C Recommendation, 2004. [Online]. Available: www.w3.org/2004/OWL. Accessed: June, 2011.

[16] A.P. Sheth, C. Henson, S.S. Sahoo. Semantic sensor web, IEEE Internet Computing, 12 (2008) 78-83.

[17] M. Horridge, S. Bechhofer. The owl api: A java api for working with owl 2 ontologies, in: R. Hoekstra, P.F. Patel-Schneider (Eds.) CEUR Workshop Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED), 2009.

[18] E. Sirin, B. Parsia, B. Grau, A. Kalyanpur, Y. Katz. Pellet: A practical owl-dl reasoner, Web Semantics: Science, Services and Agents on the World Wide Web, 5 (2007) 51-53.
[19] J. Henß, J. Kleb, S. Grimm, J. Bock. A Database Backend for OWL, in: R. Hoekstra, P.F. Patel-Schneider (Eds.) CEUR Workshop Proceedings of the 6th International Workshop on OWL: Experiences and Directions (OWLED), 2009.
[20] V. Solutions. JTS Topology Suite, Developer’s Guide, 2003.
[21] E.W. Dijkstra. A short introduction to the art of programming, Aug. 1971. [Online]. Available: http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD316.PDF.
[22] M. Danieletto, N. Bui, M. Zorzi. An Ontology-Based Framework for Autonomic Classification in the Internet of Things, in: IEEE International Conference on Communications Workshops (ICC), Kyoto, 2011.
[23] L. Yu, Y. Liu. Using the Linked Data Approach in a Heterogeneous Sensor Web: Challenges, Experiments and Lessons Learned, in: Proc Sensor Web Enablement (SWE) Workshop, Banff, Alberta, Canada, 2011.
[24] P. Barnaghi, M. Presser, K. Moessner. Publishing Linked Sensor Data, in: Proc 3rd International Workshop on Semantic Sensor Networks (SSN), in conjunction with the 9th International Semantic Web Conference (ISWC 2010), 2010.
[25] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, V. Terziyan. Smart semantic middleware for the Internet of Things, in: J. Filipe, J. Andrade-Cetto, J.L. Ferrier (Eds.) 5th International Conference Informatics in Control, Automation and Robotics (ICINCO’08), 2008.
[26] T.R. Payne, O. Lassila. Guest Editors Introduction: Semantic Web services, IEEE Intelligent Systems, 19 (2004) 14-15.
[27] M. Boussard, B. Christophe, O. Le Berre, V. Toubiana. Providing user support in Web-of-Things enabled Smart Spaces, in: Proceedings of the Second International Workshop on Web of Things (WoT '11), San Francisco, USA: ACM, 2011.
[28] B. Christophe. Semantic Profiles to Model the "Web of Things", in: of the 2011 Seventh International Conference on Semantics, Knowledge and Grids (SKG '11), Washington, DC, USA: IEEE Computer Society, 2011.
[29] B. Christophe, V. Verdot, V. Toubiana. Searching the ‘Web of Things’, in: Proc Fifth IEEE International Conference on Semantic Computing (ICSC'11), Palo Alto, CA, 2011, pp. 308 - 315.
[30] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, T. Cuong, H. Hasemann, A. Kroller, M. Pagel, M. Hauswirth, M. Karnstedt, M. Leggieri, A. Passant, R. Richardson. SPITFIRE: toward a semantic web of things, Communications Magazine, IEEE, 49 (2011) 40-48.