Semantic Web 0 (2017) 1
IOS Press


# Social Internet of Things for Domotics: a Knowledge-based Approach over LDP-CoAP

Michele Ruta [a,*], Floriano Scioscia [a], Giuseppe Loseto [a], Filippo Gramegna [a], Saverio Ieva [a], Agnese Pinto [a], and Eugenio Di Sciascio [a]

[a] *Dept. of Electrical and Information Engineering, Polytechnic University of Bari, via E. Orabona 4, I-70125, Bari, Italy*
*E-mail: name.surname@poliba.it*

**Abstract.** Ambient Intelligence aims at simplifying the interaction of a user with her surrounding context, minimizing the effort needed to increase comfort and assistance. Nevertheless, especially in built and structured environments, current technologies and market solutions are often far from providing the required levels of automation, coordination and adaptivity. Devices should interact autonomously, should reach opportunistic decisions and take actions accordingly. In particular, user activities and profiles should be among the manifold *implicit* factors to be taken into account in that process. Home and Building Automation (HBA) is one of the most relevant scenarios suffering from the poorness of the allowed system intelligence, automation and adaptivity. Devices are logically associated through static profiles defined at system deployment stage. The large majority of solutions are proprietary and centralized, and require manual configuration.

This paper proposes a novel semantic-based framework complying with the emerging Social Internet of Things paradigm. Infrastructured spaces can be intended as populated by device agents organized in social networks, interacting autonomously and sharing information, cooperating and orchestrating resources. A service-oriented architecture allows collaborative dissemination, discovery and composition of service/resource descriptions. The Semantic Web languages are adopted as knowledge representation layer and mobile-oriented implementations of non-monotonic inferences for semantic matchmaking are used to give decision capabilities to software agents. Finally, the Linked Data Platform (LDP) over the Constrained Application Protocol (CoAP) provides the knowledge organization and sharing infrastructure underpinning social object interactions. The framework has been implemented and tested in a home automation prototype integrating several communication protocols and off-the-shelf devices. Experiments advocate the effectiveness of the approach.

Keywords: Semantic Web of Things, Objects Social Networks, Building Automation, Linked Data Platform, Constrained Application Protocol

> Eventually everything connects – people, ideas, objects. The quality of the connections is the key to quality per se.
>
> *Attributed to Charles Eames [1]*

## 1. Introduction

In the *Ambient Intelligence* (AmI) vision, built environments interact with their inhabitants in an "unobtrusive, interconnected, adaptable, dynamic, embedded and intelligent" way [2]. Personal requirements and preferences are grasped, deciphered and formalized as well as the environment can adapt to them, and even anticipate people's needs and behaviors. The AmI idea leverages technological progress in the *Internet of Things* (IoT), where large numbers of everyday objects are augmented with communication and computation capabilities. People in their usual environments are increasingly surrounded by networks of micro-devices, endowed with embedded sensors for data capture as well as processing units for deriving context information. To create real cohesive AmI, such devices should communicate and coordinate autonomously, making

*Corresponding author. E-mail: michele.ruta@poliba.it.

1570-0844/17/$35.00 © 2017 – IOS Press and the authors. All rights reserved

decisions dynamically based on manifold factors, including the state of surrounding objects and places as well as user activities and profiles. While traditional human-computer interaction has been explicit and mediated by input peripherals, in AmI implicit, effortless interaction paradigms predominate, where relevant information about users' goals and intentions is inferred automatically by analyzing their actions and context, through sensors integrated in the environment or in wearable things.

Current solutions for Home and Building Automation (HBA) are still far from the above levels of intelligence, automation and adaptivity. They grant limited flexibility, as devices are logically associated at the application level by means of static profiles, defined at system deployment stage. With most established HBA standards, changing the set of possible configurations or introducing new devices require the intervention of qualified practitioners. Recently, product manufacturers and system integrators have proposed more user-friendly "smart home" devices and platforms, leveraging the IoT [3]. Unfortunately, solutions are proprietary and centralized, and they still require manual configuration. This seemingly improved usability comes at the price of providing only very basic automation [4], typically using Event-Condition-Action (ECA) rules on simplistic threshold or on/off conditions.

Hence, significant technological advances are needed to fully accomplish the AmI vision. Flexible and meaningful relationships among devices in a given environment should be possible, established automatically to support articulate orchestration and coreography patterns. Recent research in the so-called *Social Internet of Things* (SIoT) [5] is starting to define models and architectures to reach this goal. Paradigms are often borrowed from Social Networking Services (SNS) for human users. If properly adapted to the peculiarities and requirements of Multi-Agent Systems (MAS), they can support powerful approaches. SIoT offers several benefits and interesting perspectives for the IoT. The adoption of a *social model* for object interchange makes structured (to some extent) the intrinsically unpredictable interaction in the IoT and therefore it gives an unquestionable added-valued in terms of interoperability, autonomicity, versatility and coordination. This is done in such a way not imposing intolerable constraints to the fundamental flexibility and variability of Internet of Things scenarios and then not limiting a priori their peculiarities. This is not enough, however, for true AmI: versatile cooperation, organization and integration can be achieved only if connected things can represent, discover and share information and services described in an articulate way by means of high-level formalisms. Semantic Web technologies are natural candidates for such a role, as they provide interoperable languages and tools grounded on formal logic semantics [6]. Semantic Web standards enable knowledge modeling, assertion, organization, querying and inference in distributed systems, but technologies and tools require proper adaptation to work efficiently in resource-constrained environments like the IoT. The *Semantic Web of Things* (SWoT) [7] aims at the convergence of the Semantic Web and IoT visions, endowing environments with intelligence by means of semantic metadata dynamically produced by ubiquitous micro-devices to characterize sensor data, detected events and phenomena, objects, places and other relevant entities in a context. Due to the volatility and unpredictability of mobile and IoT environments, device and service discovery are two major challenges in the SWoT. Achieving acceptable performance also requires attention, as Semantic Web tools, protocols and languages are typically too resource-consuming for current IoT devices. Application-level protocols and reasoning tools for the (Semantic) Web must be properly adjusted, tailoring their feature set to pervasive computing contexts.

This paper presents a possible approach for a Semantic Social Internet of Things grounded on Ambient Intelligence scenarios. According to the SWoT paradigm, standard technologies were adapted to provide a cohesive knowledge and service discovery architecture. The proposal leverages: i) the Linked Data Platform (LDP) [8] to annotate and organize information resources and ii) the Constrained Application Protocol (CoAP) [9] –a proposed IETF[1] standard RESTful protocol– for resource exchange in constrained environments, as it is more efficient than HTTP. Above this knowledge/service interoperability layer, a semantic-enhanced application level enables social networking among "agentified" things. Entities and operations in domotics and IoT scenarios are mapped to a novel social multi-agent service-oriented architecture (SOA), supported by semantic-based capabilities. It should be noticed that the proposed framework addresses the specific needs of IoT systems both in terms of communication and computing standpoints, but also in terms of functional and/or

[1] Internet Engineering Task Force, https://www.ietf.org

architectural aspects. The proposed MAS is intrinsically general purpose and platform-independent in order to comply with a possible exploitation in different scenarios and contexts. Anyway, IoT constraints set the choice of a lightweight application protocol like CoAP (satisfying minimum functional requirements of a small device-oriented SN without excessive computational/bandwidth load at the application level), as well as they impose architectural and implementation choices devoted to keep under control the framework deployment in real-world micro-devices. This has been pursued through the adoption of compression strategies for language verbosity control, through the selection of a compact OWL syntax and via the targeted usage of both memory and data structures. Finally, the architectural approach followed aims to make modular and componentized the micro-SN fundamental elements in order to allow functional (high-level) properties to be enrolled on-demand following (low-level) device capabilities.

Borrowing core relationships and structure from popular SNSs, devices enable specific interaction patterns for information sharing and cooperative decentralized service/resource discovery. Such selective choreography is triggered autonomously, based on the kind of managed resources and other contextual factors; this capability enhances interoperability across heterogeneous platforms and scalability in dense multi-agent environments. Resource discovery exploits semantic matchmaking between ontology-based annotations which describe requests and available resources. Non-standard, non-monotonic inferences [10] implemented in the *Mini-ME* mobile matchmaker and reasoner [11] allow supporting approximated matches, resource ranking and aggregation for covering complex requests. The framework also supports basic and legacy devices, which do not have computational power enough for on-board reasoning, by allowing them to select a more capable friend as inference facilitator.

The general framework outlined above has been focused on smart HBA, to provide AmI experiences in residential and workplace settings. It was implemented and evaluated in a real prototypical testbed, encompassing diverse device types, communicating across different wired and wireless HBA protocols. Experimental evidences are reported and assess framework feasibility and effectiveness.

The remainder of the paper is as follows. Section 2 discusses the state of the art, while the framework is described in detail in Section 3. Section 4 introduces basics of the Linked Data Platform over CoAP and details the developed testbed. Section 5 presents a case study to further clarify the proposal and its benefits. Experimental evaluations in Section 6 provide an assessment of both practicability and efficiency of the proposed approach, before conclusion.

## 2. State of the art: pervasive computing in the social networks epoch

In latest years, social networking services have changed personal interaction habits and relationships management on a global scale. Members of SNSs create personal profiles with basic information about themselves; connect with other users in either bidirectional (*e.g.*, *friendship*, *group*) or unidirectional (*e.g.*, *follower*) relationships; post text and/or multimedia items on their *wall* (*i.e.*, log) for sharing with their contacts; flag (*tag*) some contacts to associate them and draw their attention to a certain element; respond to content published by other users with comments and reactions (*e.g.*, *like*). SNS adopters generally manifest an intention to continue using them [12], because SNSs provide both utility (*extrinsic* value) and gratification (*intrinsic* value). Their usefulness also grows as they connect more users, and particularly *complementary* ones [12], since opportunities increase for discovering interesting information and services.

A social evolution of pervasive computing [5] envisions objects acting as independent agents, capable of establishing relationships and using them to share information and services more effectively. This may allow to reap the above benefits in advanced IoT scenarios; actually, it is reasonable to expect them to be higher in large and heterogeneous networks, such as in HBA. An in-depth analysis of object social networks can be found in [13], which discussed key metrics about nodes and links by adapting from and expanding upon the social network analysis literature. Definitions were formalized in an ontology objects can use to manage their policies, friends and reputation. The following differences exist w.r.t. the approach proposed in this paper: (i) both a symmetrical relationship (friend) and an asymmetrical one (follower) were modeled, whereas [13] includes only an asymmetrical friendship model; (ii) a reference ontology referring to several well-known Resource Description Framework (RDF) vocabularies was developed in order to improve and facilitate the interoperability among different systems; (iii) non-standard inference services were ex-

ploited to support a semantic-enhanced resource discovery and composition in social environments. Further ontology proposals exist to formalize models of the social networking domain, *e.g.*, [14]. Particularly, in [5], things engage with one another in social networks independently from human SNSs and from user interactions. A relevant case [15] included social object capabilities in control networks, aiming at distributed Web Ontology Language (OWL) Knowledge Base (KB) management and inference. When connecting to the network, every object proactively exchanged information with other devices in a handshake process. "Requester" devices, equipped with reasoning facilities, could then distribute queries automatically among "known" devices. Unfortunately, the adopted query language supported only very simple inferences, limiting the practical usefulness of the proposal. Another research direction has been focusing on the integration of the IoT into the social context of human users [14], either to improve adaptivity in AmI [16] or to monitor users and assist them in personalizing their SNS experience and interactions [17]. In [18] semantic-based situation detection and goal retrieval were used for matchmaking with task profiles to recommend activities to users, based on their current context. Unlike our approach, social interactions occurred only between devices and users; furthermore, adopted rule-based reasoning could not retrieve approximate matches when exact ones did not exist. A further effort to achieve social capabilities is *object blogging*, defined as an object's capability of annotating and publishing its history and context on the Web and/or in a mobile ad-hoc network, supporting intelligent machine-to-machine interactions. Some proposed approaches required user intervention [19], while others aimed at autonomous self-description and decision-making [20].

Many of the above works combine social networks of pervasive objects with semantic technologies. Indeed, semantic-based approaches have wide adoption in pervasive MAS, and smart building automation is one of the most relevant areas [21, 22]. Ontologies have been used in all stages of the lifecycle of HBA systems, including design and deployment, infrastructure description, data modeling and access, and device control [23, 24]. In [25] an ontology-based building automation system delivered context-aware information in a customized way to different kinds of users, *e.g.*, upkeep and healthcare operators in a clinic. OWL device and user descriptions were matched through SPARQL queries and SWRL rules were used to combine logical constraints with context-dependent temporal and numerical ones, achieving capabilities similar to classical Complex Event Processing (CEP) systems. Nevertheless, the solution was affected from poor maintainability, because installing new devices required not only manual configuration, but also changes to the reference ontology. The proposed architecture in [24] included a reasoning module exploiting rule-based inferences. Unfortunately, the system state should fully match the rule head in order to trigger its body. Full matches seldom occur in realistic scenarios, whose entities are featured by detailed, heterogeneous and often contradictory information, unless one uses very basic rules. In our approach, non-monotonic inference services allow supporting approximate matches, which can yield "good enough" results whenever full matches are not available.

The proposed distributed Knowledge Base management and service discovery methods can be a foundation for developing further semantic SOA platform capabilities [26], including automated clustering [27], negotiation [22], composition [28] and substitution [29]. Finally, semantic alignment is often a problem in heterogeneous systems such as HBA and IoT. Several works, *e.g.*, [30], propose mappings. This work leverages Linked Data principles to limit the issue, instead, by importing and reusing meaningful parts of other vocabularies in a larger social HBA modeled.

## 3. A social framework for smart linked objects

In what follows the proposed framework, architecture and technologies are described.

### *3.1. Knowledge-based architecture*

The approach proposed here aims at object coordination in purposely infrastructured environments and particularly in domotics scenarios through interaction paradigms borrowed from social networks. The main goal is allowing devices (a.k.a. nodes) to gain wide agency and autonomy in sharing information and services, enabling them to distribute requests and obtain responses through fully decentralized peer-to-peer (P2P) interactions also assuming decisions. Table 1 outlines basic correspondences of concepts and features in the IoT and domotics domain with the proposed service-oriented social environment and the semantic capabilities devised to support it. They are discussed in detail in what follows.

Table 1
IoT/HBA entities, social features and semantic capabilities

| IoT/HBA feature | Social environment | Semantic capability | Detailed description |
|---|---|---|---|
| Object / Device / Application | Social agent | LDP-CoAP server & client | Section 3.1.1 |
| Functional profile | Service | OWL service description | Section 3.2 |
| Object pairing | Social relationship (friend/follower) | LDP-CoAP interface | Section 3.1.3 |
| Object communication | Social interaction | LDP-CoAP interface | Section 4.1 |
| Object configuration update/adaptation | Distributed service discovery | Semantic matchmaking | Section 3.1.2 & Section 3.1.4 |
| Object log | Wall | LDP Basic Container | Section 3.2 |
| Object command | Post | LDP Basic Container | Section 3.2 |
| Object reply | Comment | LDP RDF Resource | Section 3.2 |
| Functionality activation/deactivation | Tag & Like | LDP PATCH method | Section 3.1.4 |

### *3.1.1. Service-oriented architecture*

Each object is a *social agent*, which exposes an individual profile, describing its basic features (device type, location, hardware details) and the resources/services it can provide, *e.g.*, its possible configurations and functional profiles. agent is able to become *friend* and/or *follower* of other agents. It makes *posts* on either its wall or friend's wall (according to the different types of interaction described later) when its settings or capabilities change, and also when it produces new or updated information through context sensing and analysis. Each post contains all sensed perceptions and events observed by the social agent and it is considered as a request for system reconfiguration through a distributed semantic service discovery process. Posts can be exploited by: (i) sensor agents, only able to observe the surrounding environment having no actuating capabilities (*e.g.*, a weather station) with the goal of sharing observations with other agents on the network; (ii) actuator agents, only able to react to the environment change but presenting limited or absent sensing capabilities (*e.g.*, a lamp or a fan). Reading the posts, they can be aware of current conditions and activate/deactivate some services; (iii) smart agents, presenting both sensing and actuating capabilities. If the agent is not able to satisfy autonomously the perceived changes, a discovery process is started to find potential agents providing further suitable services.

Agent profiles, service descriptions and requests are expressed as semantic annotations referred to concepts modeled within ontologies in Web Ontology Language (OWL 2) [31], formally grounded on Description Logics (DLs) semantics, resulting both machine understandable and human readable at the same time. Devices such as computers or smartphones can run multiple applications concurrently: each application participating in the social service-oriented environment will behave as an autonomous social agent. Functional profiles of agent instances running on the same physical device will typically have common elements – related *e.g.*, to hardware capabilities– expanded with specific application-oriented information. The social relationship, interaction and distributed service discovery models outlined hereafter involve single-purpose physical objects (*e.g.*, lamps, printers) as well as applications deployed on multi-purpose devices, integrating them in a single cohesive social space without conflicts. A decentralized service-oriented architecture (SOA) underlies the whole proposed social network model, where shared knowledge fragments about devices, functional profiles and context represent annotated service/resource advertisements.

### *3.1.2. Semantic matchmaking*

Service/resource discovery conveys decision capabilities of social agents. As stated before, this collaborative process leverages semantic matchmaking, *i.e.*, the overall process allowing the retrieval and ranking of the most relevant resources for a given request, where both resources and requests are satisfiable concept expressions w.r.t. a common ontology $\mathcal{T}$ in a DL $\mathcal{L}$. This paper refers to the OWL 2 subset corresponding to the $\mathcal{ALN}$ (Attributive Language with unqualified Number restrictions) Description Logics, as it is supported by an embedded matchmaking and reasoning engine which provides the required inference services [11]. Before applying any inference service, the loaded knowledge base is preprocessed performing *unfolding* and *Conjunctive Normal Form* (CNF) *normalization*, as detailed in [10]. Unfolding expands terminological axioms in concept expressions, allowing the reasoner to disregard the ontology in subsequent inference procedures. Normalization enables structural algorithms for both standard and non-standard reasoning tasks, with polynomial complexity for acyclic Terminological Boxes (TBoxes) under practical assumptions [11].

Standard reasoning tasks for matchmaking include *Subsumption* and *Satisfiability*. Given a request *R* and a

Table 2
Example descriptions for non-standard reasoning tasks

| | |
|---|---|
| **TBox $\mathcal{T}$ axioms** | $LargeCapacity \sqsubseteq \neg RegularCapacity$<br>$WasherDrier \equiv WashingMachine \sqcap Drier$ |
| **Request** | $R_1 \equiv WasherDrier \sqcap LargeCapacity$ |
| **Resources** | $S_1 \equiv WashingMachine$<br>$S_2 \equiv WasherDrier \sqcap RegularCapacity$ |
| **Abduce**$(R_1, S_1, \mathcal{T})$ | $H \equiv Drier \sqcap LargeCapacity$ |
| **Contract**$(R_1, S_2, \mathcal{T})$ | $\langle G, K \rangle = \langle LargeCapacity, WasherDrier \rangle$ |

resource $S$, Subsumption verifies whether all features in $R$ are included in $S$: its outcome is either *full match* or not. For example, consider a TBox $\mathcal{T}$ including axioms and individuals shown in Table 2. In case of $R_1$ and $S_1$, Subsumption returns *false* since $S_1 \not\sqsubseteq R_1$, so $S_1$ is not deemed useful, even though it provides a washing machine which is a part of the request. Satisfiability checks whether any constraint in $R$ contradicts some specification in $S$, hence it divides resources in *compatible* (a.k.a. *potential matches*) and *incompatible* (a.k.a. *partial matches*) w.r.t. the request. In the above example, $S_2$ is incompatible with $R_1$ and should be discarded, even though the requester might accept a smaller capacity if a larger one is unavailable. These binary outcomes are inadequate for advanced scenarios, because full matches are rare and incompatibility is frequent when dealing with articulated concept expressions from heterogeneous sources.

In order to produce a finer resource ranking and a logic-based explanation of outcomes, the framework proposed here extends the basic subsumption/satisfiability approach by exploiting the following non-standard inference services [11]:

– *Concept Abduction*: whenever $R$ and $S$ are compatible, but $S$ does not imply $R$, Abduction allows to determine what should be hypothesized in $S$ in order to completely satisfy $R$. The solution $H$ (for *Hypothesis*) to Abduction can be interpreted as *what is requested in R and not specified in S (adopting an Open World Assumption)*. In the above example, Abduce($R_1$, $S_1$, $\mathcal{T}$) returns the $H$ expression in Table 2. CNF allows defining a *norm* on concept expressions, so enabling a logic-based relevance ranking of a set of resources w.r.t. a given request based on the norm of their $H$s.

– *Concept Contraction*: if request $R$ and resource $S$ are not compatible, Contraction determines which part of $R$ is conflicting with $S$. If one retracts conflicting requirements in $R$, $G$ (for *Give up*), a concept $K$ (for *Keep*) is obtained, representing a contracted version of the original request, such that $K \sqcap S$ is satisfiable w.r.t. $\mathcal{T}$. The solution $G$ to Contraction represents "why" $R$ and $S$ are not compatible. In particular, a conflict occurs when concepts in the two descriptions clash; in the above example, Contract($R_1$, $S_2$, $\mathcal{T}$) produces the output shown in Table 2. Resource ranking is possible also in this case, based on the CNF norm measured on $G$.

– *Concept Covering*: pervasive computing scenarios often require relatively large numbers of resources to be aggregated in order to satisfy a complex request. To this aim, a further non-standard reasoning task based on the solution of *Concept Covering Problem* (CCoP) has been defined. It allows to: (i) satisfy features expressed in a request as much as possible, through the conjunction of one or more small instances of a KB – seen as elementary knowledge blocks– and (ii) provide explanation of the uncovered part of the request itself. Given a request $R$ and a set of available resources $S = \{S_1, S_2, \ldots, S_n\}$, all satisfiable in the reference ontology $\mathcal{T}$, *Concept Covering* aims to find a pair $\langle S_c, H \rangle$ where $S_c \subseteq S$ contains concepts covering $R$ w.r.t. $\mathcal{T}$ and $H$ is the residual part of $R$ not covered by concepts in $S_c$. Concept Covering exploits Abduction to find at each step a concept to include in $S_c$, which is the one producing the minimum residual uncovered part $H$. Also in this case, a CNF-based *score* is associated to the result $S_c$ as the obtained covering percentage. A Covering example can be found in the case study in Section 5.

The reader is referred to [11] for algorithms and further considerations on Concept Abduction and Contraction, as well as to [32] for Concept Covering.

### *3.1.3. Social entities and relationships*

The above inference services are used to regulate the interactions between social agents. They are grouped in two families:

- *Smart*: devices able to perform reasoning tasks exploiting non-standard deductions;
- *Basic*: low-memory, low- (or no)-computing power devices. They can only provide sensing/acting services, but do not perform autonomous reasoning.

A pair of agents can establish a social relationship following the basic interaction pictured in Figure 1. Two schemes are implemented:

a) *Friend*: a bidirectional relationship where nodes $N_i$ and $N_j$ can exchange both information and services. In particular, a device $N_j$ sends a *friendship* request; since the receiver $N_i$ accepts it, they became able to: (i) read and write on each other's wall; (ii) request the friend's service descriptions; (iii) activate or deactivate the friend's services. A basic node, when becoming friend with a smart node, can select it as semantic fa-

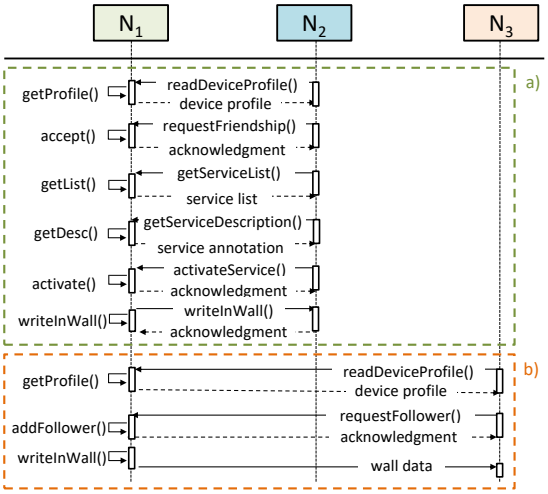


Fig. 1. Sequence diagram for social agent relationships: a) friend; b) follower

cilitator *i.e.*, reasoning supporter.

b) *Follower*: a unidirectional relationship where a node $N_k$ is interested only in receiving the updates published by $N_i$ on its wall. In other words, if $N_k$ sends a *follower* request to $N_i$, $N_k$ becomes an observer of $N_i$'s behavior.

Following/Friendship criteria are automatically verified by means of a matchmaking process involving the device profile. Differences from being follower or friend of a device are related to the device characteristics which in turn reflect the adoption of proper concepts in the reference taxonomy. Basically, semantics of profiles refers to functional and non functional properties of devices influencing the compatibility among them. In other words, $N_j$ decides to become a friend or a follower of $N_i$ according to information in $N_i$'s profile. It contains all data about the device, *e.g.*, location and type (according to the *M3-lite* taxonomy [33]), as described in Section 3.2. In particular, after retrieving the profile of the nearby object, $N_j$ evaluates the following relationship conditions –first outlined in [34]– to decide whether to send a friendship request to $N_i$: (i) strong *co-location*, *i.e.*, both devices are placed in the same room/area. The granularity of co-location can be enhanced at will in the profile definition: a co-location can be set for devices in the same wall or posed in the same shelf of a furniture item. This could restrain the friendship relation according to objects requirements and functions; (ii) *parental* or *co-ownership*, *i.e.*, they are from the same manufacturer or belong to the same owner; (iii) *co-work*, *i.e.*, nodes are able to cooperate closely as they share annotations referred to the same ontology and provide functionalities related to the same activity (*e.g.*, room lighting) or observation (*e.g.*, indoor temperature). Annotations are embedded at start-up on each social agent according to the memory availability. These can be either static, *i.e.*, the same for the whole agent's lifetime, or dynamic, *e.g.*, changing after a sensor data gathering phase or when the agent's internal state changes. On the other hand, $N_j$ asks to become a follower of $N_i$ if the following conditions are met: (i) weak or sporadic co-location, such that information produced by $N_i$ can still be useful to $N_j$ to characterize its own context but at the same time $N_j$ needs/prefers to start a discovery process completely independent from $N_i$; (ii) weak co-work relationship, *i.e.*, there is low utility in a direct interaction, *e.g.*, the two devices are deeply different. In this case, they could be defined as concepts belonging to different device categories or present partially incompatible definitions in the reference ontology; (iii) no co-ownership. For example, if a *printer* and a *scanner* are in the same room, they become friends because they share the same location, but also as they were both defined within the reference ontology as devices providing services for document management. In this case, friendship is preferred to a following relationship because the printing functionalities must be explicitly required by the scanner, *e.g.*, after doing a document scan. The framework also allows $N_j$ to be both a friend and a follower of $N_i$; this enables a broader variety of interaction patterns between the two devices, which is useful in highly dynamic scenarios. In any case, $N_i$ can refuse the friendship/follow request if: (a) relationship constraints described above are not respected; (b) the maximum number of friends/followers has been reached. Every device sets limits according to its processing and memory capabilities. In practice, however, refusing a new friend or follower is rare, as a device increases its opportunities for useful cooperation by expanding its social network.

Like in SNSs, in the framework proposed here the object's wall is the main channel for sharing knowledge. Both *push* and *pull* models are supported, through the above relationships. In a nutshell, if a node $N_i$ wants to receive updates from a node $N_j$ automatically, it will ask to become a follower. In this case, the follower $N_i$ can start a semantic matchmaking session when it receives a notification from the followed device $N_j$ (as in Figure 1(b)); instead, if $N_i$ wants to be able to access the wall of $N_j$ on demand, it will ask to become a friend (and in doing so it will also grant access to its own wall). Then $N_i$ will start a semantic matchmaking process only if $N_j$ writes a post on $N_i$'s

wall, *e.g.*, during a distributed covering as reported in Figure 2(b). Every agent will select either model –or even both– depending on its application requirements.

### *3.1.4. Collaborative adaptivity*

When a node detects some change in internal or contextual conditions requiring adaptation *i.e.*, a functional configuration update of itself and/or nearby devices, it writes a post on its own wall. A post $P$ is modeled as a pair $\langle R, L \rangle$. $R$ is the request issued by the node; $L$ is the *like* value. The idea of the *like* reaction to a post is mutuated from human-oriented SNS, but with two important differences: it is a real value in $[0, 1]$ rather than a binary value; it represents the percentage of coverage and completion of the request $R$, as obtained from Concept Covering in the collaborative service discovery triggered by the post to reconfigure the environment. Specifically, if $H$ is the uncovered part resulting from Concept Covering of $R$ with a set of available services, the corresponding like value will be

$$L = 1 - \frac{norm(H)}{norm(R)}$$

using the CNF-induced norm on concept expressions. The overall process is exemplified in Figure 2. It consists of the following steps:

1) When a node $N_i$ detects a reconfiguration is needed, it writes a post $P_i$ on its own wall. Initially, $L_i$ is set to 0.

2) If $N_i$ is a basic device, go to step 3. Otherwise, $N_i$ executes the Concept Covering task on the local set of service descriptions $S$ (section a) in Figure 2). Upon completion, $N_i$ activates the selected services and adds a *comment* $C_i$ to $P_i$ as a pair $\langle U_i, T_i \rangle$, where $U_i$ is the uncovered part of $R_i$ and $T_i$ *tags* the local selected services/resources. Moreover, the value of $L_i$ is updated to the obtained covering score, as reported in Figure 2(a).

3) If $R_i$ is not completely covered, $N_i$ selects a friend $N_j$ and writes a post $P_j$=$\langle R_j, L_j \rangle$) on its wall. Particularly, if $N_i$ has executed step 2, $R_j$ is set to the uncovered part $U_i$, otherwise $R_j$ is equal to $R_i$ and $L_j$ is reset. Furthermore, $N_i$ requests to be notified when a comment is added to $P_j$. $N_j$ recursively executes the steps 2) - 3). This is illustrated in section b) of Figure 2.

4) When $N_i$ receives the notification of $P_j$, it reads the comment from the friend's wall, which is appended to $P_i$ to update the status of the request. Finally, $N_i$ updates the *like* value according to the overall covering score.

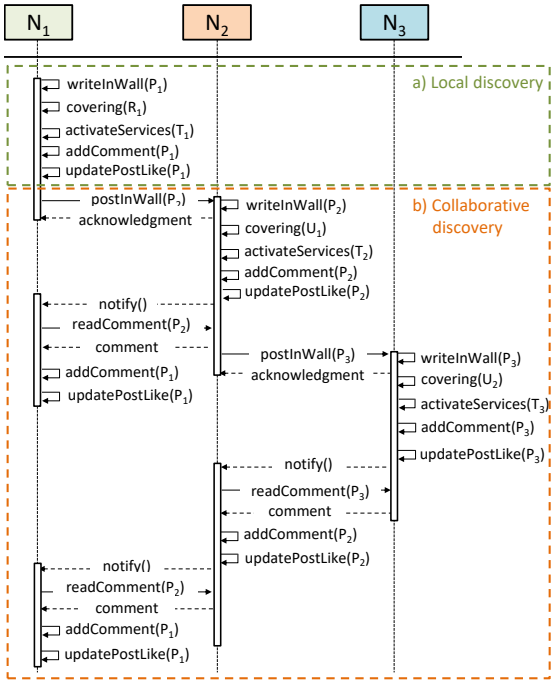


Fig. 2. Sequence diagram of distributed reconfiguration

A full example is in the case study reported in Section 5. Some remarks may be useful:

– The recursive discovery procedure can be applied in a depth search with no theoretical bounds. Practically, discovery can be limited by means of the following threshold values, modeled as device parameters: (i) *max_depth*, representing the maximum depth of the discovery w.r.t. the device starting the process, where a direct friend has depth 1, a friend of a friend has depth 2, *etc.*; (ii) *minimum like value*, to identify a satisfactory coverage from the discovery process: when a device reaches this like value, the covering procedure can be halted, also avoiding to forward the (possible) uncovered part of the request to further friends. Each device can use these parameters to prevent nodes over the network from being flooded with multiple and/or useless messages. Agents manage heuristics to decide the values of both parameters.

Message flooding is also managed by exploiting a simple caching mechanism implemented on each node. A request is identified by means of a unique key saved on the device cache when the message is accepted and processed. If the request message was previously received, it is discarded by the device.

– The choice of friend(s) to call in the above step 3 also depends on heuristic preference criteria, such as the number and type of services exposed by the friend

(known at friendship establishment time), network latency or friend's computational resources.
– Main purpose of comments is to keep track of the progressive fulfillment of an adaptation request, exploiting tagging to avoid duplication of service/resource selection.

### 3.2. Interoperability layer

All social features reported in Table 1 are modeled as RDF resource following the *Linked Data Platform* (LDP) guidelines in order to make the proposed approach general-purpose and independent from the particular protocol used at the application layer. The LDP W3C Recommendation [8] provides standard rules for accessing and managing Linked Data on the Web. Basically, it defines a set of communication patterns based on HTTP methods and headers for CRUD (Create, Read, Update, Delete) operations as well as different types of LDP *Resources* (LDPRs): *RDF Source* (LDP-RS), whose status corresponds to an RDF graph and can be fully represented in an RDF syntax; *Non-RDF Source* (LDP-NR), not represented in RDF (*e.g.*, a binary or text document without useful RDF annotation); *Basic* (LDP-BC), *Direct* (LDP-DC) and *Indirect* (LDP-IC) containers, defining collections of LDP resources according to specific membership patterns.

This subsection, along with figures from 3 to 6, describes the models of core entities in the social framework: device profiles, service profiles, walls, posts and comments.

**Device profiles.** The *profile* resource exposes main device features as an RDF-based annotation. An example is reported in Figure 3. In addition to well-known RDF vocabularies, a so-called *Semantic Web of Social Things* (SWST) ontology[2] has been defined to model basic elements of a social device. In particular, each profile contains the following properties:
– type of device, according to the classification proposed by the *M3-lite* taxonomy [33], a lightweight version of the *Machine-to-Machine Measurement* (M3) ontology used to describe sensor measurements and observations;
– device name, using the *dcterms:title* property of the *DCMI Metadata Terms* vocabulary [35];
– supported ontologies (*dcterms:requires*) used as reference vocabularies to define the OWL-based annotations of the functionalities exposed by the device;

[2] Available at http://sisinflab.poliba.it/swottools/onto/swst/

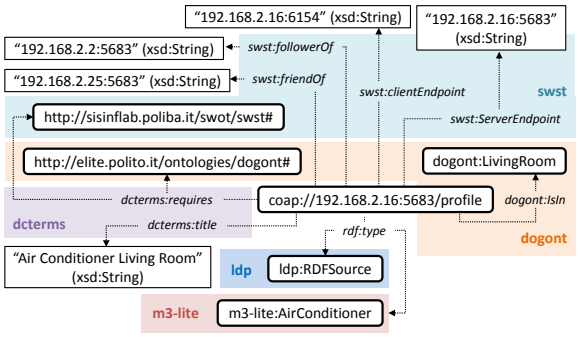


Fig. 3. Ontology-based modeling of a device profile

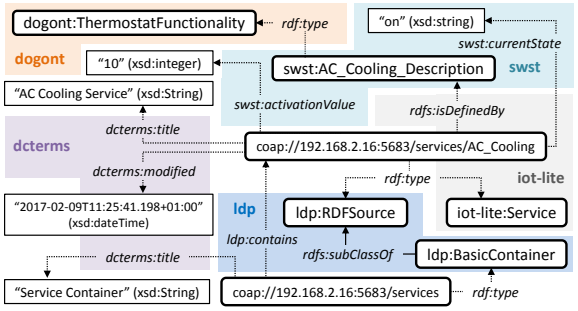


Fig. 4. Service container and device functionalities

– location of the device (*e.g.*, in an area, building, department, apartment), exploiting the *dogont:isIn* property of *DogOnt* [24], a reference ontology proposed to model intelligent domotic environments. DogOnt also contains several concepts related to indoor and outdoor locations;
– address of the device endpoints, on both the server (*swst:serverEndpoint*) and client (*swst:clientEndpoint*) side. Both properties were defined as sub-properties of *iot-lite:endpoint* contained in the *IOT-lite* ontology [36], a lightweight vocabulary based on SSN-XG [37] proposed to describe IoT concepts and relationships;
– (possible) friend and followed devices exploiting the *swst:friendOf* and *swst:followerOf* relations, respectively.

*Friendship* is an LDP-BC listing the friend devices of a social object. Sub-resources are identified by the name of the friend and are connected to the container through an *ldp:contains* property, according to the LDP guidelines [8]. Each of them corresponds to the object profile retrieved after the friendship was established.

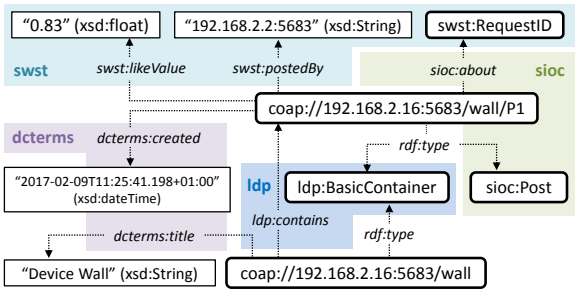


Fig. 5. Wall and posts

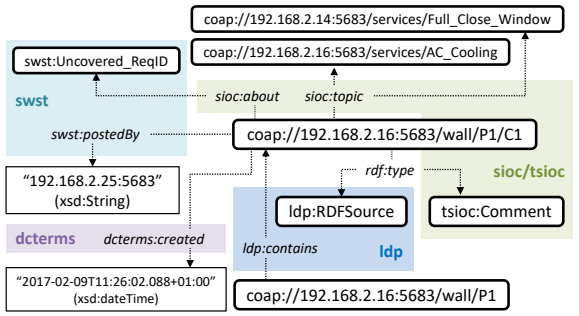


Fig. 6. Comment to a post

**Service profiles.** As depicted in Figure 4, the functionalities exposed by a device are described by means of a resource named *services* and characterized by a set of RDF properties: *dcterms:title* specifies the service name; *rdfs:isDefinedBy* indicates the IRI of the OWL individual modeling the service within the reference KB; *dcterms:modified* reports the timestamp of the last modification applied to the individual description; *swst:currentState* and *swst:activationValue* identify the service current state and the specific value to be used to activate the functionality (set point), respectively.

**Wall and posts.** Figure 5 shows the modeling of a device *wall*. It lists one or more *posts* defined as nested RDF resources and each post can include several *comments*. Post descriptions include: creation date (*dcterms:created*); creator device (*swst:postedBy*); content of the post (*sioc:about*), defined in the *Semantically-Interlinked Online Communities* (SIOC) Core Ontology [38], as IRI of the individual representing the received OWL annotation; like value (*swst:likeValue*).

**Comments.** Finally, a comment annotation (Figure 6) consists of: creation date; creator device; content of the comment, corresponding to the part of the post the friend device is not able to cover; tagged (*i.e.*, activated) services (*sioc:topic*), selected through the covering process.

## 4. Implementation

At the application layer, the above reference architecture is implemented on a LDP-CoAP framework [39]. Each agent in the social network is modeled as an LDP-CoAP node exposing the interface reported in Table 3. In what follows, basics of Linked Data Platform for the CoAP protocol will be introduced along with a detailed description of the developed prototypical testbed.

### *4.1. Framework implementation with LDP-CoAP*

LDP specification only supports the HTTP protocol, which requires not negligible bandwidth, processing and memory resources for most IoT devices. LDP-CoAP variant, on the contrary, aimed to integrate LDP in resource-constrained devices and networks just leveraging CoAP [9], a compact counterpart of HTTP conceived for machine-to-machine (M2M) communication. Some CoAP options are derived from HTTP header fields (*e.g.*, content type, headers and proxy support), while some other ones have no analogous in HTTP. In any case, the HTTP-CoAP mapping, included in the LDP-CoAP framework, can be exploited to support all LDP features with CoAP.

In the present case, social devices communicate over the network through CoAP messages. Basically, each message is composed of: (i) a 32-bit *header*, containing the request method code or response status; (ii) an optional *token* value, used to associate replies to requests, (iii) a sequence of *option* fields (containing information such as resource URI and payload media type), (iv) the *payload* data. CoAP adopts the CoRE Link Format specification [40] for resource discovery. A client accesses the reserved `/.well-known/core` URI on the server via `GET` to retrieve available resource entry points. Further `GET` requests will include `URI-query` options to retrieve only resources with given attributes. Standardized query attributes include resource type (`rt`), interface usage (`if`), content-type (`ct`), and MIME (Multipurpose Internet Mail Extension) `type` for a resource. Further non-reserved attributes can be freely used. CoAP also provides push notifications without polling [41], a useful feature when data have to be monitored

Table 3
LDP-CoAP interface of a social device

| Resource URI | Resource Type | LDP-CoAP Method | Description |
|---|---|---|---|
| **/profile** | LDP-RS | GET | Returns the device profile |
| **/friendship** | LDP-BC | GET | Returns the list of friend devices |
| | | POST | Receives a friendship request |
| **/friendship/<device-name>** | LDP-RS | GET | Returns the profile of a specific friend device |
| **/services** | LDP-BC | GET | Returns the list of the functionalities exposed by the device |
| **/services/<service-name>** | LDP-RS | GET | Returns the RDF-based description of a specific functionality |
| | | PATCH | Updates the status of a functionality according to the received command |
| | | HEAD | Returns the LDP-CoAP headers (*e.g.*, *Etag* value) to verify the presence of updates |
| **/services/<service-name>/owl** | LDP-NR | GET | Returns the OWL annotation related to a specific functionality |
| **/wall** | LDP-BC | GET | Returns the device wall containing the list of published posts |
| | | POST | Publishes on the wall a post received from a friend device |
| **/wall/<post-id>** | LDP-BC | GET | Returns the detailed description of a specific post |
| | | POST | Publishes on the wall a comment related to a post |
| **/wall/<post-id>/owl** | LDP-NR | GET | Returns the content of a post as OWL annotation |
| **/wall/<post-id>/<comment-id>** | LDP-RS | GET | Returns the description of a comment |
| | | PATCH | Tag a device functionality on a comment |
| **/wall/<post-id>/<comment-id>/owl** | LDP-NR | GET | Returns the content of a comment as OWL annotation |

over time (*e.g.*, in case of *follower* relationship). CoAP also supports proxies, enabling Web applications (*i.e.*, HTTP clients) to transparently access the resources hosted in devices based on CoAP.

In order to clarify the proposed approach, some reference examples are shown in Figure 7, reporting RDF annotations in Turtle syntax [42]. Alternately, they can be retrieved in JSON-LD [43], by setting the *Accept* header appropriately. In particular, it is possible to notice that:

- GET requests are used to retrieve data (*e.g.*, wall content, profile or service description) from a device. As shown in the example Nr. 8, OWL annotations are treated as LDP-NR resources, in order to support any OWL concrete syntax, not only RDF-based ones;
- POST method allows to send data to a device, *e.g.*, send friendship requests (example Nr. 2) or write posts/comments to the wall (example Nr. 5);
- PATCH requests are used to update data, *e.g.*, to tag a new device on an existing comment (example Nr. 6);
- HEAD method is exploited to verify if a device service description has changed (example Nr. 9). It exploits the *ETag* value, defined in [44], *i.e.*, a resource identifier differentiating representations of the same resource that vary over time. In the proposed framework, it is based on a given OWL annotation and changes every time the description is modified.

Table 4
KNX devices in the testbed

| Product ID | Description | Social Device |
|---|---|---|
| GW90707 | KNP/IP router | — |
| GW90740 | Switch actuator 4 channels | Air conditioner (2 ch.) + Garden watering system (2 ch.) |
| GW90740 | Switch actuator 4 channels | n.4 Simple on/off lamp |
| GW12782 | Push button 4 channels | n.4 Basic on/off button |
| GW90800 | Weather Station with GPS | Weather Station |
| GW90746 | Dimmer actuator | Dimming lamp |
| GW10948 | Burglar alarm system interface | Alarm system |
| GW90754 | Roller shutter actuator | Shutter controller |

### *4.2. Developed testbed*

A prototypical testbed was developed following the proposed social framework described in Section 3. The core goal was to address the interoperability problem among multiple IoT platforms and standards. Therefore, the testbed implements a basic environment (similar to the one described in the case study in Section 5) consisting of a subset of home areas, *i.e.*, a hall door, a living room, a kitchen, and a small outdoor space. An IEEE 802.11 network was exploited as a fast backbone including 3 *smart* nodes, each implementing a single social device. The smart nodes were developed on three reference platforms with different processing capabilities. Moreover, a KNX sub-network was connected to the main area by means of an additional smart node, acting as *gateway* toward the social net-

**Example 1. Read the profile of a social device**

```
[REQ] GET coap://192.168.2.16:5683/profile    Accept: text/turtle
[RES] 2.05 Content    Content-Format (ct): text/turtle    ETag: W/’1234’
      <coap://192.168.2.16:5683/profile> a ldp:RDFSource, m3-lite:AirConditioner ;
          dcterms:requires <http://sisinflab.poliba.it/swottools/onto/swst>, <http://elite.polito.it/ontologies/dogont> ;
          dcterms:title "Air Conditioner LR" ; dogont:isIn dogont:LivingRoom ;
          swst:serverEndpoint <coap://192.168.2.16:5683> ; swst:clientEndpoint <coap://192.168.2.16:61655> ;
          swst:friendOf <coap://192.168.2.25:5683> ; swst:followerOf <coap://192.168.2.2:5683> .
```

**Example 2. Send a friendship request**

```
[REQ] POST coap://192.168.2.32:5683/friendship?title=AC   Accept: text/turtle
      ...payload (RDF device profile)...
[RES] 2.01 Created    Location-Path: coap://192.168.2.32:5683/friendship/AC
```

**Example 3. Read the wall of a social device**

```
[REQ] GET coap://192.168.2.16:5683/wall    Accept: text/plain
[RES] 2.05 Content    Content-Format (ct): text/turtle    ETag: W/’4567’
      <coap://192.168.2.16:5683/wall> a ldp:BasicContainer ; dcterms:title "Device Wall" ;
      ldp:contains <coap://192.168.2.16:5683/wall/P0>, <coap://192.168.2.16:5683/wall/P1> .
```

**Example 4. Read a post on the wall**

```
[REQ] GET coap://192.168.2.16:5683/wall/P0      Accept: text/turtle
[RES] 2.05 Content    Content-Format (ct): text/turtle    ETag: W/’a235’
      <coap://192.168.2.16:5683/wall/0> a ldp:BasicContainer, sioc:Post ;
          dcterms:created "2017-02-09T16:02:56.993+01:00"^^xsd:dateTime ; sioc:about swst:WS_Request ;
          swst:postedBy <192.168.2.15:5683> ; swst:likeValue "87,15"^^xsd:float ;
          ldp:contains <coap://192.168.2.16:5683/wall/P0/C0>, <coap://192.168.2.16:5683/wall/P0/C1> .
```

**Example 5. Write a post on the friend wall**

```
[REQ] POST coap://192.168.2.16:5683/wall
      ...payload (OWL annotation)...
[RES] 2.01 Created    Location-Path: coap://192.168.2.16:5683/wall/P2
```

**Example 6. Tag a device functionality on a comment**

```
[REQ] PUT coap://192.168.2.2:5683/wall/P0/C1?ldp=patch
      If-Match: W/"a872"     Content-Format (ct): application/rdf-patch
      A <coap://192.168.2.2:5683/wall/P0/C1> sioc:topic <coap://192.168.2.16:5683/services/AirCondition_Cooling> .
[RES] 2.04 Changed
```

**Example 7. Read the RDF-based description of a device functionality**

```
[REQ] GET coap://192.168.2.16:5683/services/AirCondition_Cooling    Accept: text/turtle
[RES] 2.05 Content    Content-Format (ct): text/turtle    ETag: W/’bd72’
      <coap://192.168.2.16:5683/services/AirCondition_Cooling> a ldp:RDFSource, iot-lite:Service ;
          dcterms:title "Air Condition Cooling" ; dcterms:modified "2017-02-09T16:02:48.698+01:00"^^xsd:dateTime ;
          rdfs:isDefinedBy swst:AC_Cooling ; swst:activationValue "10"^^xsd:integer ; swst:currentState "on" .
```

**Example 8. Read the OWL annotation of a device functionality**

```
[REQ] GET coap://192.168.2.16:5683/services/AirCondition_Cooling/owl
[RES] 2.05 Content    Content-Format (ct): text/plain    ETag: W/’bd72’
      ...payload (OWL annotation)...
```

**Example 9. Read LDP-CoAP headers to check (possibile) modifications in the description of a device functionality**

```
[REQ] GET coap://192.168.2.16:5683/services/AirCondition_Cooling?ldp=head
[RES] 2.03 Valid    Content-Format (ct): text/turtle    ETag: W/’bd72’
```

**Example 10. Activate/deactivate a functionality**

```
[REQ] PUT coap://192.168.2.16:5683/services/AirCondition_Cooling?ldp=patch
      If-Match: W/"bd72"     Content-Format (ct): application/rdf-patch
      D <coap://192.168.2.16:5683/services/AirCondition_Cooling> swst:currentState "on" .
      A <coap://192.168.2.16:5683/services/AirCondition_Cooling> swst:currentState "off" .
[RES] 2.04 Changed
```

Fig. 7. Examples of basic device interactions over LDP-CoAP

work, allowing KNX-based devices to interact with the other social objects in a transparent way. As detailed in Table 4, the KNX installation consisted of 8 off-the-shelf devices, produced by Gewiss Inc.[3], connected through a twisted pair bus in a hierarchical network. Each device (except the KNX router, used only for the communication over IP) corresponds to one or more LDP-CoAP *endpoints*, exposed by the gateway node, representing the social objects in Table 4. In this way, all devices in the home can interact through the proposed LDP-CoAP interface independently from the specific HBA protocol. Particularly, multi-channel devices can manage several functionalities, *e.g.*, switch actuators can handle up to 4 push buttons whereas the dimmer actuator can provide several functionalities according to the different light level. Therefore, more than 30 services were exposed overall by the KNX sub-network toward the social framework.

According to Figure 8, a Java-based management software was implemented to run on each home device. The main Java package *it.poliba.sisinflab.swst* was partitioned in the following sub-packages to separate developed classes in well-defined sections each providing the following specific functionality:
- `core`: contains the *HomeDevice* reference implementation. It extends the *CoAPLDPServer* provided by the LDP-CoAP library[4] and exposes one or more *DeviceEndpoint* managing the resources described in Table 3. At network level, LDP-CoAP provides also a modified version of the *Californium* CoAP framework [45] supporting LDP features over CoAP;
- `resources`: several Java classes model the different device resources. LDP-CoAP *RDFSource*, *NonRDFSource* and *BasicContainer* base classes were extended, providing common attributes and methods to save, retrieve and update the home data. All information is stored within an RDF repository based on the *RDF4J 2.1.3* library[5];
- `rdf.vocabulary`: contains RDF ontology files mapped as Java classes to simplify creation and querying of RDF triples. Ontologies cited in Section 3.2 (*e.g.*, SIOC, IoT-lite, M3-lite) were mapped through the *Sesame Vocabulary Builder*[6] tool and included in the package;
- `owl`: provides basic functionalities to load the reference KB, manage all generated OWL annotations

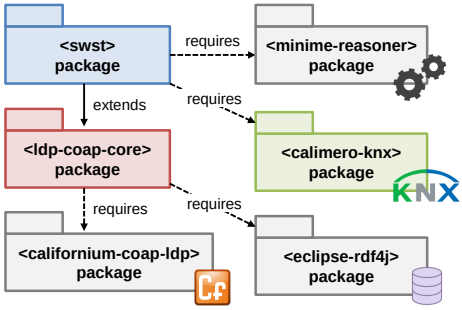


Fig. 8. Reference software modules

through the *OWL-API 3.4.10* library [46] and invoke the *Mini-ME* reasoner [11] implementing inference services;
- `knx`: an additional package implemented to support the communication over the *ISO/IEC 14543-3 EIB/KNX* protocol stack [47]. *Calimero-core* library[7] was exploited for network management and to exchange data with KNX devices (*e.g.*, read state values or send commands). An import utility was also implemented to parse data from an XML-based project file exported from ETS4[8], the official software tool used to design and configure home installations based on KNX systems, and to model the same device features within the home social network.

The following embedded boards were used to implement the social devices:
(a) *Raspberry Pi Model B*[9], equipped with a single-core ARM11 CPU at 700 MHz, 512 MB RAM (shared with GPU), 8 GB storage memory on SD card, Raspbian Wheezy OS;
(b) *Intel Edison Kit*[10] equipped with an Intel Quark x86 CPU at 400 MHz, 1 GB RAM, 4 GB eMMC flash storage and Yocto Poky Linux OS (32-bit kernel 3.10.98);
(c) *UDOO Quad*[11] equipped with quad-core ARM Cortex A9 at 1 GHz clock frequency, ARM Cortex M3 coprocessor, 1 GB DDR3 RAM, 32 GB storage memory on SD card, UDOObuntu 2.0 Minimal Edition OS. All platforms included a 32-bit Java 8 SE Runtime Environment (JRE, build 1.8.0-b121).

[3] http://www.gewiss.com
[4] https://github.com/sisinflab-swot/ldp-coap-framework
[5] http://rdf4j.org/
[6] http://github.com/tkurz/sesame-vocab-builder
[7] http://github.com/calimero-project
[8] http://www.knx.org/knx-en/software/ets/about/
[9] http://www.raspberrypi.org/products/model-b/
[10] http://www.intel.com/content/www/us/en/do-it-yourself/edison.html
[11] http://www.udoo.org/udoo-dual-and-quad

## 5. Case study: Semantic Web of (social) Things for building automation

This section presents a case study, devoted to clarify the social and collaborative features of the proposed framework in terms of orchestration of smart devices in a complex HBA context. Let us consider the example scenario depicted in Figure 9. *Two apartments on the same floor of a building, $H_1$ and $H_2$, include a set of semantic-enabled devices forming a home social network.* In particular, $H_1$ is configured with an alarm system (AS), a rolling shutter controller ($SC_1$), an air conditioner ($AC_1$) and a dimmer lamp ($L_1$). A weather station (WS), a rolling shutter controller ($SC_2$), an air conditioner ($AC_2$) and a dimmer lamp ($L_2$) are installed in $H_2$ instead. The blue arrows in Figure 9 specify the existing friendship relations between the different devices. In particular, according to the criteria reported in Section 3.1.3, each pair of devices in the same apartment establishes a friendship relation because they are in the same location and share functionalities useful to improve comfort or security in the house. As said, when a friendship relation is established, each friend is able to directly read the wall of an object, write a post on its wall and use the services of the device. Within the two apartments, each object has sensing and/or actuating capabilities and exposes a set of features to its friends. According to the resource interface described in Section 4.1, all social network interactions can be implemented as request/response messages over LDP-CoAP.

*It is evening, there is no one in the apartment $H_1$ and the AS detects an intrusion in the house*. Immediately the AS writes a new post on its wall representing what it has sensed as an OWL annotation. Figure 10 shows a possible formalization of the post (reported in OWL2 Manchester syntax [48] for the sake of readability) w.r.t. the reference ontology. Service requests and descriptions are modeled in a general way, by expressing the context conditions suitable for the activation of a given service.

The AS starts a *Concept Covering* process using the post content as request, whereas available resources are represented by the functionalities exposed by all the devices directly involved into a friendship relation. The AS verifies if the services of the connected objects, $SC_1$ and $L_1$ in our example, were modified by performing a simple check: (i) AS sends a lightweight request (as shown in Example 9 of Figure 7) to the service resources exposed by each friend; (ii) only if the resource has changed (*i.e.*, the OWL annotation was updated), the new service description will be retrieved (Example 8 of Figure 7). Otherwise, the AS can directly use the cached service annotation. This procedure ensures that the covering task is performed using the latest descriptions of all available services.

According to the semantic service descriptions, listed in Figure 11 and 12 respectively, the matchmaking procedure highlights the shutter should be *fully closed* and the dimmer lamp *turned on* to completely satisfy the request. This is due to the fact the AS detects a low luminosity, no presence of humans and an intrusion event. All device perceptions were modeled with the related concepts described within the reference ontology. In particular, the `Intrusion` class was defined as more specific than `IntrusionForLamp` and `IntrusionForShutter` (*i.e.*, it should require services both from a lamp and a shutter controller), as shown in Figure 13. These two concepts belong to the annotation of the *Lamp_On* and *Full_Close* service annotations, respectively; moreover, *Lamp_On* is also useful in case of low luminosity. Due to this reason, they are selected during the covering process as suitable functionalities to be activated. With a modestly expressive DL like $\mathcal{ALN}$, such a modeling pattern allows activating functionalities of different devices that are fired when the same event is detected. In the first step of Concept Covering, all services in Figure 11 and Figure 12 undergo Concept Abduction with the request: *Full_Close* is selected because it yields the smallest uncovered part, reported in Figure 14. This becomes the new request for the subsequent Abduction round, when *Lamp_On* is selected and *AS_Request* is fully covered. Therefore, the AS writes on its wall a comment to the post, containing only a tag for each service to activate, *i.e.*, *Full_Close* ($SC_1$) and *Lamp_On* ($L_1$). The uncovered part of the request is empty because the request was completely satisfied. Finally, according to the covering results, the like value of the post is automatically updated to 1 and no further operations are required.

Simultaneously, devices within the apartment $H_2$ could exploit the knowledge shared by the home social network in $H_1$ to adapt their configuration according to the detected conditions. As shown in Figure 9, a *follower* relation exists between the weather station and the alarm system. As explained in Section 3.1.3, the WS decided to follow (*i.e.*, continuously observes) the wall of the AS because: (i) they are in different houses; (ii) AS is an alarm device providing services not directly useful for a sensing device as WS; (iii) at the same time, AS provides further sensing capa-

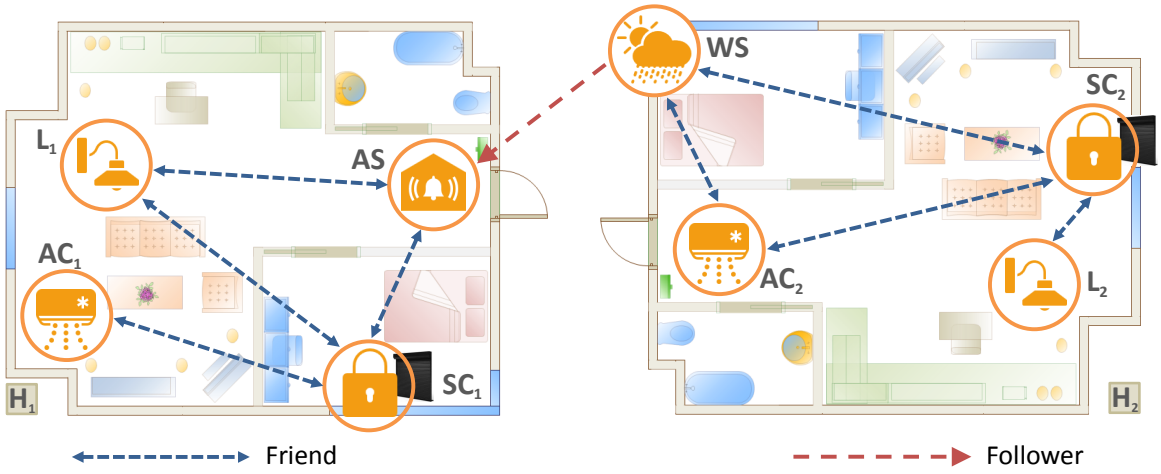


Fig. 9. Case study scenario

*AS_Request* ≡ (detectsOutdoorLuminosity **some**) **and** (detectsOutdoorLuminosity **only** LowLuminosity) **and** (detectsIntrusion **some**) **and** (detectsIntrusion **only** Intrusion) **and** (detectsOccupancy **some**) **and** (detectsOccupancy **only** (**not** Presence))

Fig. 10. Post content (*i.e.*, request) written by the alarm system

*Full_Close* ≡ (detectsPrecipitation **some**) **and** (detectsPrecipitation **only** Rain) **and** (detectsWindSpeed **some**) **and** (detectsWindSpeed **only** StrongWind) **and** (detectsIntrusion **some**) **and** (detectsIntrusion **only** IntrusionForShutter) **and** (detectsOccupancy **some**) **and** (detectsOccupancy (**not** Presence))

*Half_Close* ≡ (detectsPrecipitation **some**) **and** (detectsPrecipitation **only** (**not** Rain)) **and** (detectsWindSpeed **some**) **and** (detectsWindSpeed **only** ModerateWind)

*Open* ≡ (detectsPrecipitation **some**) **and** (detectsPrecipitation **only** (**not** Rain)) **and** (detectsWindSpeed **some**) **and** (detectsWindSpeed **only** LightBreeze) **and** (detectsOutdoorLuminosity **some**) **and** (detectsOutdoorLuminosity **only** HighLuminosity)

Fig. 11. Shutter controllers $SC_1$ and $SC_2$ service annotations

*Lamp_On* ≡ (detectsOutdoorLuminosity **some**) **and** (detectsOutdoorLuminosity **only** LowLuminosity) **and** (detectsIntrusion **some**) **and** (detectsIntrusion **only** IntrusionForLamp)

*Lamp_Medium* ≡ (detectsOutdoorLuminosity **some**) **and** (detectsOutdoorLuminosity **only** MediumLuminosity) **and** (detectsOccupancy **some**) **and** (detectsOccupancy **only** Presence)

*Lamp_Off* ≡ (detectsOutdoorLuminosity **some**) **and** (detectsOutdoorLuminosity **only** HighLuminosity) **and** (detectsOccupancy **some**) **and** (detectsOccupancy **only** (**not** Presence))

Fig. 12. Dimmer lamps $L_1$ and $L_2$ service annotations

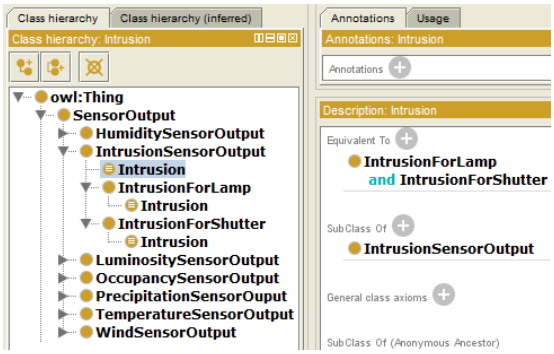


Fig. 13. `Intrusion` ontology class modeling

bilities (*i.e.*, intrusion detection) useful for WS to better characterize the surrounding environment. So when the alarm system posts the intrusion message, it is immediately notified. The WS reads the annotation and shares it on its wall as a new post. This event triggers also in $H_2$ a *Concept Covering* process involving the services exposed by the friends of WS ($SC_2$ and $AC_2$), which are listed respectively in Figure 11 and Figure 15.

Only the *Full_Close* service, provided by $SC_2$, is selected to partially satisfy the request: this is basically due to commonality with the request of concepts *(detectsOccupancy some)* and *(detectsOccupancy (not*

```
AS_Req_Uncovered ≡ (detectsOutdoorLuminosity some) and (detectsOutdoorLuminosity only LowLuminosity) and (detectsIntrusion some) and (detectsIntrusion only IntrusionForLamp)
```

Fig. 14. OWL annotation of the uncovered part of *AS_Request*

```
AC_Cooling ≡ (detectsTemperature some) and (detectsTemperature only HighTemperature) and (detectsHumidity some) and (detectsHumidity only MediumHumidity)

AC_Heating ≡ (detectsTemperature some) and (detectsTemperature only LowTemperature) and (detectsHumidity some) and (detectsHumidity only LowHumidity)

AC_Dehumidification ≡ (detectsTemperature some) and (detectsTemperature only MediumTemperature) and (detectsHumidity some) and (detectsHumidity only HighHumidity)
```

Fig. 15. Air conditioners $AC_1$ and $AC_2$ service annotations

```
WS_Req_Uncovered ≡ (detectsOutdoorLuminosity only LowLuminosity) and (detectsIntrusion only IntrusionForLamp)
```

Fig. 16. OWL annotation of the uncovered part of *WS_Request*

*Presence))*. WS comments its post including a tag to the shutter service and the uncovered part of the request as content. In this case, to further satisfy the post, the WS can forward the uncovered part to one of its friends. The WS selects $SC_2$, since it provided the highest contribution to covering in the initial step, and posts on the wall of $SC_2$ the OWL annotation of the uncovered part reported in Figure 16. Moreover, the WS starts observing the post it just sent to the friend's wall. $SC_2$ in turn receives the message, starts a covering process involving the services exposed by $L_2$ (Figure 12) and selects the *Lamp_On* functionality, which completely covers the remaining part of the initial request. $SC_2$ comments its post tagging the activated services and updates the like value with the percentage of covered features. The request is fully satisfied so the uncovered part is empty and no other posts are needed. Thanks to the observer pattern, the WS receives a notification about the post, reads the comment and understands the initial request has been completely served. As a consequence, it updates the like value of the post on its wall, not forwarding further requests.

It is useful to point out how social capabilities allowed apartment $H_2$ to compensate for the lack of an alarm system, taking advantage of the sensing capabilities of the one in $H_1$ to appropriately configure and modify the status of its devices. This is just an obvious example of the benefits of the proposed semantic-based social framework in information and service/resource sharing in complex settings and heterogeneous networks. Furthermore, request and service descriptions in the case study were kept short for easier understanding of the proposed framework, but the adopted inferences allow managing more detailed specifications with articulated constraints.

## 6. Evaluation

A performance and functionality assessment of the proposed framework and implementation are outlined hereafter. Experiments have been carried out performing the 10 reference tasks described in Figure 7, to identify and evaluate specific features characterizing their performance. Each test was repeated five times and average values were taken.

### *6.1. Performance*

**Time.** Time results are reported in Figure 17. In particular, the *processing* time is defined as the time elapsed on the device receiving a request to process the message and send the related response, whereas the *communication* time represents the time needed to exchange request/response data (*i.e.*, CoAP packets) over the home network between the sender and the receiver device. As expected, RaspberryPi required a longer time to process the requests due to the reduced computational capabilities. On the contrary, Intel Edison was the fastest platform in case of tasks only requiring simple I/O operations, thanks to the internal flash memory. Concerning communication time, a significant variation can be noticed, due to the different hardware adopted for connecting to the home network. In particular, RaspberryPi and UDOO were equipped with a *Wi-Pi* IEEE 802.11 USB dongle, whereas Intel Edison exploited the on-board transceiver.

Test results about the Concept Covering task performed on a single node are reported in Figure 18. Experiments were conducted exploiting a shared dataset of 7 requests (see Table 5) with growing size and restriction complexity. As a consequence, a different number of services was selected (from a set of 50 instances) and tagged after the reasoning step. It can be pointed out that like value was lower for simpler requests and increased for more complex ones. This is due to the fact that ontology and service annota-

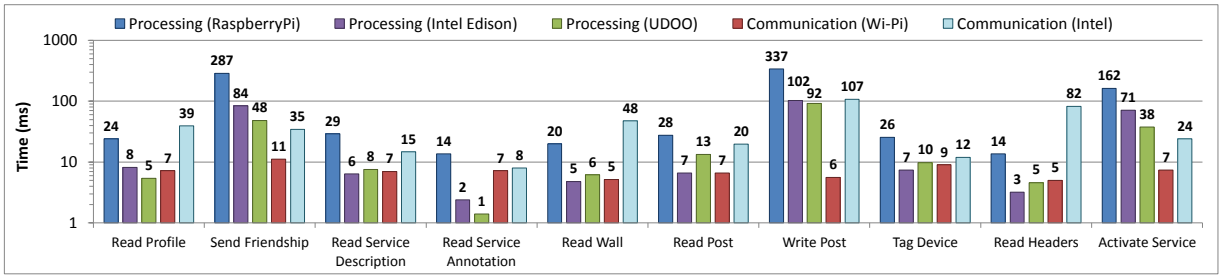


Fig. 17. Processing and communication time for the social tasks

Table 5
Request Dataset

| ID | Size (Byte) | Tagged Services | Like Value | Processing Time (ms) | | |
|---|---|---|---|---|---|---|
| | | | | Rasp. Pi | Intel Edison | UDOO |
| R1 | 1381 | 1 | 28.57 | 89.2 | 20.8 | 12.8 |
| R2 | 1967 | 2 | 61.54 | 122.2 | 30.8 | 14.2 |
| R3 | 2468 | 3 | 73.68 | 132.0 | 37.6 | 17.8 |
| R4 | 2902 | 4 | 79.17 | 140.2 | 47.0 | 18.0 |
| R5 | 3309 | 5 | 82.76 | 146.4 | 64.4 | 23.2 |
| R6 | 3651 | 6 | 84.85 | 154.6 | 78.8 | 29.8 |
| R7 | 4202 | 7 | 87.18 | 160.8 | 81.4 | 35.4 |

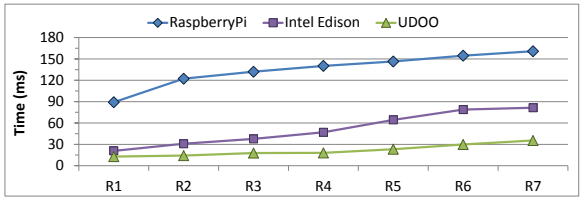


Fig. 18. Processing time for the Concept Covering task

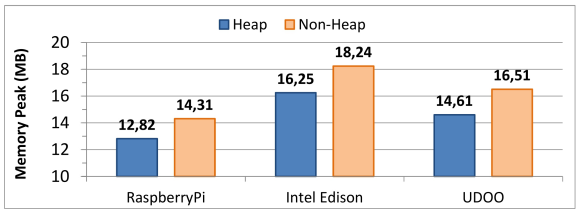


Fig. 19. Memory usage

tions are modeled to fit articulate and specific descriptions, such as device operating requirements; generic requests result less significant, leading to a relative loss of the semantic-based score. For all platforms, the complexity of the request affected time only slightly, showing a similar trend. As expected, the processing time was longer for complex annotations, because a higher number of services was retrieved to satisfy the request.

**Memory.** Memory usage values are shown in Figure 19. Framework requirements were low on all platforms, with a memory peak always under 18.5 MB for stack memory and 16.5 MB for heap memory, representing reasonable values for embedded systems.

**Data compression.** Another relevant parameter of the social framework performance is the amount of data exchanged over the home network. In order to reduce the number of packets used to transmit each message over CoAP, the LDP-CoAP implementation described in [39] was extended to support the following encoding algorithms, aiming to reduce the size of resource descriptions: (i) *GZIP* and *BZIP2* (both included within the *Apache Commons Compress* library[12]), general-purpose and suitable for annotations described with RDF Turtle [42], JSON-LD [43] and all OWL syntaxes; (ii) *Binary JSON* (BSON)[13], *Universal Binary JSON* (UBJSON)[14] and *Message Pack* (MsgPack)[15], specific for JSON-based annotations.

Selected algorithms were tested on three basic resources corresponding to the LDP-CoAP resource types available in the proposed framework: RDF Source (LDP-RS, *e.g.*, device profiles and comments): Basic Container (LDP-BC, *e.g.*, walls and posts); Non-RDF Source (LDP-NR, *e.g.*, OWL annotations of posts, comments and services). LDP-RS and LDP-BC were described with RDF Turtle and JSON-LD to test both syntaxes supported by LDP-CoAP, whereas LDP-NR was described through the OWL 2 Manchester syntax. Figure 20 reports on the size of the reference annotations with and without compression. GZIP provided better results, achieving a compression ratio of

[12] http://commons.apache.org/proper/commons-compress/
[13] http://bsonspec.org/
[14] http://ubjson.org/
[15] http://msgpack.org/

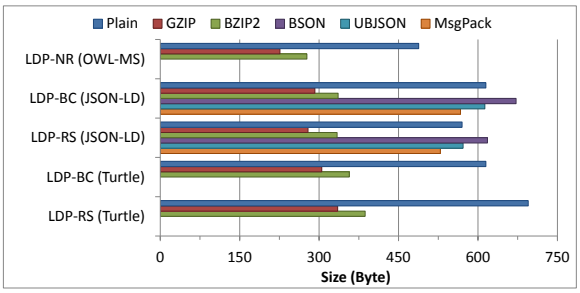


Fig. 20. Size of encoded payload messages

about 48%. In this way, each device sends on average the half of CoAP packets (which must contain a maximum of 64B as payload), so reducing the overall communication latency. On the contrary, JSON-specific algorithms were not particularly useful for short messages, being designed to encode large documents.

### 6.2. *Functionality*

Benefits of the devised semantic social platform were assessed in a comparison w.r.t. the following IoT-oriented frameworks in the HBA market: KNX IoT[16]; IzoT Platform[17], originally developed by Echelon Corporation for the Industrial IoT but also exploited for building applications; Dog Gateway[18] [24]; Eclipse SmartHome[19]. Table 6 highlights that only the proposed approach combines fitness for resource-constrained environments (by using CoAP and a P2P architecture), expressiveness of device modeling (by exploiting RDF and OWL 2) and support for both exact and approximated matches, with formally grounded service composition.

## 7. Conclusion and Future Work

This paper introduced a novel semantic-based framework for Social Internet of Things, particularly useful for home and building automation but inherently general-purpose. The proposal adopted a decentralized service-oriented architecture to manage, publish, discover and compose semantically annotated service/resource descriptions. It adopted LDP-CoAP to join the benefits of efficient RESTful machine-to-machine communication and structured Linked Data organization. Non-standard, non-monotonic inferences enabled semantic matchmaking for discovery with support for approximate matches, logic-based ranking and composition via request covering. The framework was developed on a multi-protocol HBA testbed with single-board computers and embedded home devices, exhibiting effectiveness in AmI scenarios.

Future work includes a wider testbed implementation and experimentation, to validate scalability of the proposal in very large object networks. Moreover, heuristics governing decisions about the creation and removal of friend/follower relationships will be explored, including behaviors based on agents' past experience, possibly by means of machine learning techniques. A similar approach can be adopted to endow social objects with proactive adaptivity to environmental modifications.

A not negligible aspect to be further investigated is related to cybersecurity risks of the proposed approach: trust elements are strongly coupled to the need for a device to prove its own identity to their potential friends. This becomes even more relevant when devices networks increase their dimension, functions and population.

Finally, additional message propagation models based on different event priorities will be investigated along with further object interaction schemes according to the *Linked Data Notifications* protocol [49].

[16]http://www.knx.org/knx-en/Landing-Pages/KNX-IoT
[17]http://www.echelon.com/izot-platform
[18]http://dog-gateway.github.io/
[19]http://www.eclipse.org/smarthome/index.html

## References

[1] G. Bizios, *Architecture Reading Lists and Course Outlines*, Eno River Press, 1998.
[2] F. Sadri, Ambient intelligence: A survey, *ACM Computing Surveys (CSUR)* **43**(4) (2011), 36.
[3] W. Kastner, M. Jung and L. Krammer, Future Trends in Smart Homes and Buildings, in: *Industrial Communication Technology Handbook*, 2nd edn, R. Zurawski, ed., CRC Press/Taylor & Francis, 2014, pp. 1483–1502, Chap. 59.
[4] W. Kastner, L. Krammer and A. Fernbach, State of the Art in Smart Homes and Buildings, in: *Industrial Communication Technology Handbook*, 2nd edn, R. Zurawski, ed., CRC Press/Taylor & Francis, 2014, pp. 1415–1434, Chap. 55.
[5] L. Atzori, A. Iera and G. Morabito, From "Smart Objects" to "Social Objects": The next evolutionary step of the Internet of Things, *Communications Magazine, IEEE* **52**(1) (2014), 97–105.
[6] N. Shadbolt, T. Berners-Lee and W. Hall, The semantic web revisited, *IEEE intelligent systems* **21**(3) (2006), 96–101.

Table 6
Comparison with current IoT-oriented frameworks for HBA

| Features | Proposed Approach | KNX IoT | IzoT Platform | Dog Gateway | Eclipse SmartHome |
|---|---|---|---|---|---|
| Home Area Network reference protocol | multi-protocol over CoAP | EIB/KNX | LonTalk | multi-protocol over HTTP | multi-protocol over HTTP |
| Network architecture | full P2P | centralized | centralized | centralized | centralized |
| Network/devices configuration | autonomous social agents configuration | via ETS software | via LonBuilder software or XML configuration files | XML configuration files | Domain Specific Language (DSL) configuration files |
| Add/remove devices | agents self-configuration | edit network/devices configuration | edit network/devices configuration | edit network/devices configuration | edit network/devices configuration |
| Multi-protocol communication | smart node acting as gateway | KNX gateway | IzoT gateway | Dog gateway | node acting as gateway |
| Device binding | dynamic, based on friendship relationships | static, defined during network configuration | static, defined during network configuration | dynamic, based on device profile | static, defined during device configuration |
| Scenarios configuration | dynamic, exploiting non-standard inferences | static, defined during network configuration | static, defined during network configuration | dynamic, exploiting rule-based reasoning | static, based on an ECA rule engine |
| Service composition | yes, through distributed covering | no | no | no | no |
| Message data format | RDF Turtle, JSON-LD, OWL 2 | proprietary, according to KNX spec. | proprietary, XML-based according to LonTalk spec. | OWL 2 | XML |
| Standardised framework interface | LDP-CoAP RESTful interface | KNX IoT Web Services | HTTP RESTful API | WebSocket and HTTP RESTful API | HTTP RESTful API |


[7] M. Ruta, F. Scioscia and E. Di Sciascio, Enabling the Semantic Web of Things: Framework and Architecture, in: *Semantic Computing (ICSC), 2012 IEEE Sixth International Conference on*, IEEE, 2012, pp. 345–347.

[8] S. Speicher, J. Arwe and A. Malhotra, Linked Data Platform 1.0, W3C Recommendation, W3C, 2015, https://www.w3.org/TR/ldp/.

[9] C. Bormann, A. Castellani and Z. Shelby, CoAP: An Application Protocol for Billions of Tiny Internet Nodes, *IEEE Internet Computing* **16**(2) (2012), 62–67.

[10] M. Ruta, E. Di Sciascio and F. Scioscia, Concept abduction and contraction in semantic-based P2P environments, *Web Intelligence and Agent Systems* **9**(3) (2011), 179–207.

[11] F. Scioscia, M. Ruta, G. Loseto, F. Gramegna, S. Ieva, A. Pinto and E. Di Sciascio, A mobile matchmaker for the Ubiquitous Semantic Web, *International Journal on Semantic Web and Information Systems* **10**(4) (2014), 77–100.

[12] K.-Y. Lin and H.-P. Lu, Why people use social networking sites: An empirical study integrating network externalities and motivation theory, *Computers in Human Behavior* **27**(3) (2011), 1152–1161.

[13] O. Voutyras, P. Bourelos, S. Gogouvitis, D. Kyriazis and T. Varvarigou, Social Monitoring and Social Analysis in Internet of Things Virtual Networks, in: *Intelligence in Next Generation Networks (ICIN), 2015 18th International Conference on*, IEEE, 2015, pp. 244–251.

[14] G. Biamino, A semantic model for socially aware objects, *Advances in Internet of Things* **2**(3) (2012), 47–55.

[15] B. Ramis Ferrer, S. Iarovyi, L. Gonzalez, A. Lobov and J.L. Martinez Lastra, Management of distributed knowledge encapsulated in embedded devices, *International Journal of Production Research* (2015), 1–18.

[16] B. Jadhav and S. Patil, Wireless Home Monitoring using Social Internet of Things (SIoT), in: *Automatic Control and Dynamic Optimization Techniques (ICACDOT), International Conference on*, IEEE, 2016, pp. 925–929.

[17] E. Papadopoulou, S. Gallacher, N.K. Taylor, M.H. Williams, F.R. Blackmun, I.S. Ibrahim, M.Y. Lim, I. Mimtsoudis, P. Skillen and S. Whyte, Combining pervasive computing with social networking for a student environment, in: *Proceedings of the Twelfth Australasian Symposium on Parallel and Distributed Computing-Volume 152*, Australian Computer Society, Inc., 2014, pp. 11–19.

[18] D. Hussein, S.N. Han, G.M. Lee, N. Crespi and E. Bertin, Towards a dynamic discovery of smart services in the social internet of things, *Computers & Electrical Engineering* (2016).

[19] D.N. Crowley, E. Curry and J.G. Breslin, Leveraging Social Media and IoT to Bootstrap Smart Environments, in: *Big Data and Internet of Things: A Roadmap for Smart Environments*, Springer, 2014, pp. 379–399.

[20] E. Bove, Object (b)logging: semantically rich context mining and annotation in pervasive environments, in: *Advances in Sensors and Interfaces (IWASI), 2015 6th IEEE International Workshop on*, IEEE, 2015, pp. 210–215.

[21] S.N. Han, G.M. Lee and N. Crespi, Semantic context-aware service composition for building automation system, *IEEE Transactions on Industrial Informatics* **10**(1) (2014), 752–761.

[22] M. Ruta, F. Scioscia, G. Loseto and E. Di Sciascio, Semantic-based resource discovery and orchestration in Home and Building Automation: a multi-agent approach, *IEEE Transactions on Industrial Informatics* **10**(1) (2014), 730–741.

[23] H. Dibowski and K. Kabitzsch, Ontology-based device descriptions and device repository for building automation devices, *EURASIP Journal on Embedded Systems* **2011**(1) (2011), 623461.

[24] D. Bonino, E. Castellina and F. Corno, The DOG gateway: enabling ontology-based intelligent domotic environments, *IEEE Transactions on Consumer Electronics* **54**(4) (2008), 1656–1664.

[25] Y. Evchina, J. Puttonen, A. Dvoryanchikova and J.L.M. Lastra, Context-aware knowledge-based middleware for selective information delivery in data-intensive monitoring systems, *Engineering Applications of Artificial Intelligence* **43** (2015), 111–126.

[26] K.P. Joshi, Y. Yesha and T. Finin, Automating cloud services life cycle through semantic technologies, *Services Computing, IEEE Transactions on* **7**(1) (2014), 109–122.

[27] B.T. Kumara, I. Paik and W. Chen, Web-service clustering with a hybrid of ontology learning and information-retrieval-based

term similarity, in: *Web Services (ICWS), 2013 IEEE 20th International Conference on*, IEEE, 2013, pp. 340–347.

[28] C. Chang, S.N. Srirama and J. Mass, A Middleware for Discovering Proximity-based Service-Oriented Industrial Internet of Things, in: *Services Computing (SCC), 2015 IEEE International Conference on*, IEEE, 2015, pp. 130–137.

[29] M. Ruta, G. Zacheo, L.A. Grieco, T. Di Noia, G. Boggia, E. Tinelli, P. Camarda and E. Di Sciascio, Semantic-based resource discovery, composition and substitution in IEEE 802.11 mobile ad hoc networks, *Wireless Networks* **16**(5) (2010), 1223–1251.

[30] Z. Cong and A.F. Gil, Enabling web service discovery in heterogeneous environments, *International Journal of Metadata, Semantics and Ontologies 4* **8**(2) (2013), 106–118.

[31] B. Parsia, S. Rudolph, M. Krötzsch, P. Patel-Schneider and P. Hitzler, OWL 2 Web Ontology Language Primer (Second Edition), W3C Recommendation, W3C, 2012, http://www.w3.org/TR/owl2-primer.

[32] M. Ruta, F. Scioscia, G. Loseto, F. Gramegna, S. Ieva and E. Di Sciascio, Mini-ME 2.0: Powering the Semantic Web of Things, in: *OWL Reasoner Evaluation Workshop (ORE 2014)*, S. Bail, B. Glimm, E. Jiménez-Ruiz, N. Matentzoglu, B. Parsia and A. Steigmiller, eds, CEUR Workshop Proceedings, Vol. 1207, CEUR-WS, 2014, pp. 8–15.

[33] R. Agarwal, D.G. Fernandez, T. Elsaleh, A. Gyrard, J. Lanza, L. Sanchez, N. Georgantas and V. Issarny, Unified IoT Ontology to Enable Interoperability and Federation of Testbeds, in: *3rd IEEE World Forum on Internet of Things*, 2016.

[34] L. Atzori, A. Iera, G. Morabito and M. Nitti, The Social Internet of Things (SIoT) – When Social Networks Meet the Internet of Things: Concept, Architecture and Network Characterization, *Computer networks* **56**(16) (2012), 3594–3608.

[35] Dublin Core Metadata Initiative and others, Dublin core metadata element set, version 1.1, Dublin Core Metadata Initiative, 2012.

[36] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi and K. Taylor, IoT-Lite: A Lightweight Semantic Model for the Internet of Things, in: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, IEEE, 2016, pp. 90–97.

[37] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog et al., The SSN Ontology of the W3C Semantic Sensor Network Incubator Group, *Web Semantics: Science, Services and Agents on the World Wide Web* **17** (2012).

[38] U. Bojars, J.G. Breslin and S. Decker, Porting social media contributions with SIOC, in: *Recent Trends and Developments in Social Software*, Springer, 2010, pp. 116–122.

[39] G. Loseto, S. Ieva, F. Gramegna, M. Ruta, F. Scioscia and E. Di Sciascio, Linked Data (in low-resource) Platforms: a mapping for Constrained Application Protocol, in: *The Semantic Web - ISWC 2016: 15th International Semantic Web Conference, Proceedings, Part II*, P. Groth and et al., ed., Lecture Notes in Computer Science, Vol. 9982, Springer International Publishing, Cham, 2016, pp. 131–139.

[40] Z. Shelby, Constrained RESTful Environments (CoRE) Link Format, *Request for Comments*, IETF, 2012, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc6690.txt.

[41] K. Hartke, Observing Resources in the Constrained Application Protocol (CoAP), *Request for Comments*, IETF, 2015, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc7641.txt.

[42] G. Carothers and E. Prud'hommeaux, RDF 1.1 Turtle (Terse RDF Triple Language), W3C Recommendation, W3C, 2014, http://www.w3.org/TR/turtle/.

[43] M. Lanthaler, M. Sporny and G. Kellogg, JSON-LD 1.0, W3C Recommendation, W3C, 2014, http://www.w3.org/TR/json-ld/.

[44] C. Bormann and Z. Shelby, The Constrained Application Protocol (CoAP), *Request for Comments*, IETF, 2016, Internet Engineering Task Force. http://www.ietf.org/rfc/rfc7959.txt.

[45] M. Kovatsch, M. Lanter and Z. Shelby, Californium: Scalable cloud services for the Internet of Things with CoAP, in: *Internet of Things, 2014 Int. Conf. on the*, IEEE, 2014, pp. 1–6.

[46] M. Horridge and S. Bechhofer, The OWL API: A Java API for OWL Ontologies, *Semantic Web* **2**(1) (2011), 11–21.

[47] M. Ruta, F. Scioscia, G. Loseto and E. Di Sciascio, KNX, a worldwide standard protocol for home and building automation: state of the art and perspectives, in: *Industrial Communication Technology Handbook*, 2nd edn, R. Zurawski, ed., CRC Press/Taylor & Francis, 2014, pp. 1463–1481, Chap. 58.

[48] M. Horridge and P.F. Patel-Schneider, OWL 2 Web Ontology Language Manchester Syntax (Second Edition), W3C Working Group Note, W3C, 2012, https://www.w3.org/TR/owl2-manchester-syntax/.

[49] S. Capadisli, A. Guy, C. Lange, S. Auer, A. Sambra and T. Berners-Lee, Linked Data Notifications: A Resource-Centric Communication Protocol, in: *The Semantic Web: 14th International Conference, ESWC 2017, Portorož, Slovenia, May 28 – June 1, 2017, Proceedings, Part I*, E. Blomqvist, D. Maynard, A. Gangemi, R. Hoekstra, P. Hitzler and O. Hartig, eds, Springer International Publishing, Cham, 2017, pp. 537–553.