Towards an Ontology-Based Framework for a Behavior-Oriented Integration of the IoT

Domenico Cantone, Carmelo Fabio Longo, Marianna Nicolosi-Asmundo,
Daniele Francesco Santamaria, Corrado Santoro
University of Catania
Department of Mathematics and Computer Science
Viale Andrea Doria, 6 - 95125 - Catania, ITALY
{domenico.cantone, fabio.longo}@unict.it, {nicolosi, santamaria, santoro}@dmi.unict.it

***Abstract*—We present a prototype version of an ontology-based framework, called PROF-ONTO, that integrates IoT devices and users with domotic environments. PROF-ONTO is based on a novel OWL 2 ontology, called OASIS (*Ontology for Agents, Systems, and Integration of Services*), modelling behaviors of agents such as IoT devices and users, and other information concerning user requests, their executions, restrictions and authorizations. User requests are performed by automatically selecting compatible devices: agents expose their behaviors and are invoked accordingly to what they are able to do on specific categories of components. OASIS is also used to build semantic knowledge bases that operate as transparent communication and information exchange systems among agents.**

INTRODUCTION

Automation refers to the capability of devices to act on behalf of users in specific environments, with a little effort from their part. Automation is not confined only to factories, farms, and cities, but also small environments such as homes can take advantage of it. An ecosystem comprising interrelated devices capable to transfer data over a network and cooperate without human control is widely called *Internet of Things* (IoT). IoT is massively used in smart environments, especially in home automation systems. Domotic systems typically connect web-enabled devices through internet to a central hub or assistant, which is responsible of managing them.

Currently, many domotic systems are available on the market. Often assistants and devices are strictly bound to their providers and thus users are tied to such providers as well. It turns out that interchangeability of devices is hardly reachable without the intervention of third-part applications and, as a consequence, users may not install devices or assistants deployed by other sellers. Besides specific marketing strategies, several obstacles prevent the interchangeability of devices. Among them, the most relevant one concerns connectivity, networking, and communication protocols whose usage largely depends on the specific IoT applications deployed, which are to be regarded as black boxes. Moreover, it is almost impossible to determine *a priori* what a device is capable to do within the environment and how its functionality can be controlled by users. Such problems could be solved if devices were selected on the basis of what they are able to do through open and shared knowledge bases and if they communicated via a common, transparent protocol.

Semantic web is a vision of the web in which machine-readable data allows software agents to query and manipulate information on behalf of human agents. In such a vision, web information carries explicit meaning, so it can be automatically processed and integrated by agents, and data can be accessed and modified at a global level, thus resulting in increased coherence and dissemination of information. Moreover, with the aid of reasoners, it is possible to infer and process also implicit information present in the data, thus gaining a deeper knowledge of the domain. Automated reasoning systems allow one to also verify the consistency of the model and query the data-set. The definition of a specific domain is widely called **ontology** [1], [2].

In this paper, we present a prototype version of PROF-ONTO,[1] an ontological framework for the integration of users and IoT devices with domotic environments, which acts as a home assistant. PROF-ONTO exploits a novel ontology, called OASIS (*Ontology for Agents, Systems, and Integration of Services*),[2] which models user requests together with restrictions and scheduling, device behaviors, device authorizations implemented by smart contracts [3], and information concerning the execution of user requests by devices. With respect to other *Agent System* paradigms such as *Artifacts&Agents* [4], OASIS adopts a general approach where agents are entities able to perform actions, whereas components are subjected to actions carried out by agents.

Transcriptions of user requests are mapped by a BDI rule-based system, called PROFETA, in OASIS knowledge bases. In the current version of the assistant, user requests are satisfied by automatically selecting devices whose behaviors fulfill such requests. Communication between the assistant and the devices depends on a specific knowledge base of OASIS, used as a communication protocol, that specifies the device to be activated, the action to be performed, and the recipient of the action.

The paper is structured as follows. Section I provides an overview of the semantic web, introducing ontologies and their related modelling languages. Section II deals with related works. Section III presents our proposed ontological

[1]https://github.com/dfsantamaria/ProfOnto.git
[2]http://tiny.cc/OASIS-Ontology

framework and the way in which it is exploited to extract meaningful data. Section IV describes a case-study that shows how our proposed framework can be used in a real application. Finally, Section V concludes the paper with some hints for future work.

I. Preliminaries

Applications that automatically process information, instead of just presenting it, and exchange information with other applications need appropriate languages, with formally defined syntax and semantics. This issue turns out to be particularly relevant for the web and, in general, for any distributed environment. The *Word Wide Web Consortium* (W3C) recommends the *Web Ontology Language* (OWL), a family of knowledge representation languages relying on *Description Logics* (DLs) [5], as a solution to this problem and indicates it as a standard for representing ontologies. Ontologies are formal descriptions of the domain of interest, defined by combining three basic syntactic categories: *entities*, *expressions*, and *axioms*.

OWL, currently in version 2.1, provides users with constructs useful for the design of ontologies in real-world domains that are not available in the basic semantic web model *Resource Description Framework* (RDF) and in the basic semantic web language *RDF Schema* (RDFS). As RDF, OWL 2 is grounded on the idea of triples or statements, each one representing an atomic unit. Triples are ways to connect two entities or an entity and a data-value, each one represented by an *Internationalized Resource Identifier* (IRI), i.e., a sequence of characters that unambiguously identifies a resource within a specific context. Entities represent the primitive terms of an ontology and are identified in a unique way. They are individuals (actors), object- and data-properties (actions), and classes (sets of actors with common features). In order to provide a formal description of the domain, OWL 2 triples can be organized into two main categories: expressions and axioms. Expressions are obtained by applying OWL 2 constructs to entities to form complex descriptions, whereas axioms describe what is true in the domain.[3]

To retrieve and manipulate semantic knowledge, the W3C recommends the SPARQL query language as the standard query protocol for RDF. Like SQL, SPARQL is a declarative query language to perform operations on data represented as a collection of RDF triples. A SPARQL query has a *head* and a *body*: the *head* comprises a modifier identifying the corresponding type of query, whereas the *body* consists of an RDF triple pattern. The reader is referred to [7] for a detailed overview of SPARQL.

II. Related work

In the last decade, integration of agent systems and ontologies has been deeply studied in several contexts [8], [9], [10].

Concerning IoT, ontological approaches have been focused mainly on sensors, with the purpose of collecting data for generating perceptions and abstractions of the world [11].

[3]For a detailed explanation of axioms and expressions introduced in OWL 2, the reader is referred to [6].

In [12], a comprehensive ontology for representing IoT services is presented, together with a discussion on how it can be used to support tasks such as service discovery, testing, and dynamic composition, taking into account also parameters such as *Quality of Services* (QoS), *Quality of Information* (QoI), and IoT service tests.

A unified semantic knowledge base for IoT, capturing the complete dynamics of IoT entities and where their heterogeneity is hidden and semantic searching and querying capabilities are enabled, is proposed in [13].

Unification of the state-of-the-art architectures, as put forward by the scientific community of the *Semantic Web of Things* (SWoT), by means of an architecture based on different abstraction levels, namely *Lower*, *Middle* and *Upper Node* (LMU-N), is described in [14]. The LMU-N architecture provides a reading grid used to classify processes, to which the SWoT community contributes, and to describe how the semantic web impacts the IoT.

In [15], an ontology providing an elementary approach to modelling agents' behaviors and artifacts is proposed together with a tool that uses the ontology to generate programming code for agent-oriented software engineering.

None of the aforementioned work deals in depth with agent behaviors or tries to formalize the interaction mechanism among them. As far as we know, this paper represents the first attempt of applying semantic web technologies as a communication protocol among IoT devices and as a representation system for their behaviors and interactions.

III. The Framework Prof-Onto

In this section, we describe the most important features of the ontology OASIS, and the software architecture implementing the prototype version of the ontology-based domotic assistant called Prof-Onto.

A. *The ontology* OASIS

OASIS is an OWL 2 ontology consisting of about 120 classes, more than 100 object-properties, 2 data-properties, and more than 900 axioms. Among other information, it currently models: a) behaviors of agents (i.e., users and devices) in terms of the operations that they are able to perform; b) agent configurations; c) agent requests; d) executions of operations and their related status.

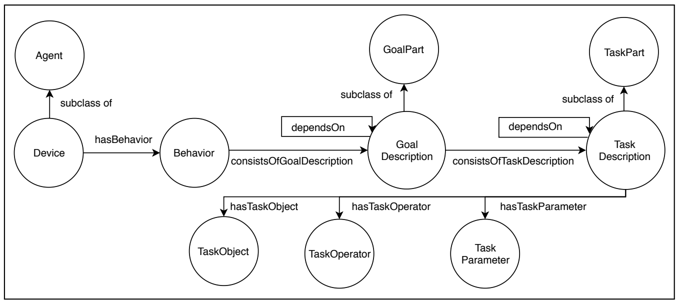

Figure 1. Ontology schema of device behaviors

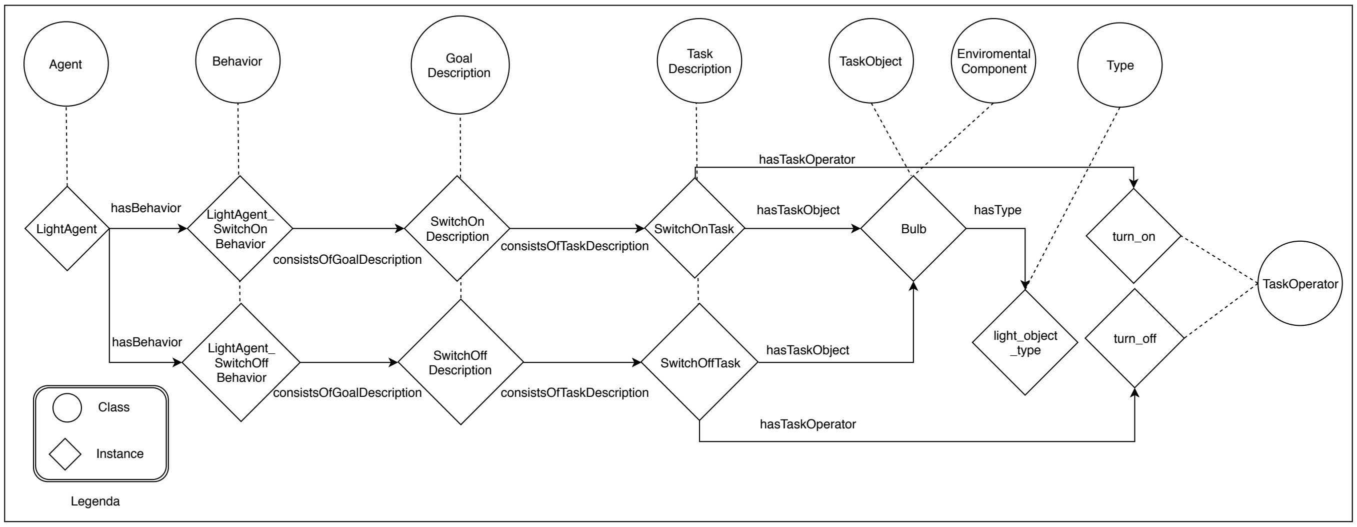

Figure 2. A representation of a smart bulb device in OASIS

Despite the fact that OASIS is able to represent behaviors of generic agents, the ontology is mainly focused on devices entrusted by users to perform certain operations within a domotic context. Behaviors of such agents are described by the schema in Figure 1.

The main classes of OASIS and their characteristics for describing behaviors of agents are summarized next.

- *Agent*: comprises all the individuals capable of executing actions on the associated components. This class includes, among others, the classes *HumanAgent*, *SoftwareAgent*, and *Device*, mapping physical people, software or programs, and physical devices, respectively. The class *HumanAgent* contains, in its turn, the class *User*, representing users that usually access the system.
- *TaskDescription*: describes atomic operations (e.g., turn on, turn off, wipe, and so on) that an agent performs as a result of some requests made by other agents. An atomic operation O may depend on other atomic operations, whose execution is mandatory in order to perform the operation O. Dependencies of atomic operations are modelled through the object-property *dependsOn*, which relates instances of the class *TaskDescription*.
- *GoalDescription*: models a set of atomic operations (represented by the class *TaskDescription*), whose executions are subject to no order. Execution dependencies of goals (non-atomic operations) are modelled through the object-property *dependsOn*, relating instances of the class *GoalDescription*. Instances of the class *GoalDescription* are linked to the related task descriptions by means of the object-property *consistsOfTaskDescription*.
- *Behavior*: represents the behavior of a single agent in terms of what the agent is able to do. It comprises a set of non-atomic operations (described by instances of the class *GoalDescription*) whose execution is subject to no order. Instances of the class *Behavior* are linked to the related goal descriptions through the object-property *consistsOfGoalDescription*.

The core of OASIS revolves around the description of atomic operations introduced by the instances of the class *TaskDescription*. Atomic operations are the most simple actions that agents are able to perform and that they expose to other agents. Hence, atomic operations represent what agents can do and what they can ask other agents to do by means of request submissions. Instances of the class *TaskDescription* are related with three elements that uniquely identify the operation. The first element is an instance of the class *TaskOperator*, characterizing the action to perform. Instances of *TaskDescription* are related with instances of *TaskOperator* by means of the object-property *hasTaskOperator*. The second element is an instance of the class *TaskObject*, representing the object recipient of the actions (described by the instances the class *TaskOperator*) performed by devices. Instances of *TaskDescription* are related with instances of *TaskObject* by means of the object-property *hasTaskObject*. The third element is an instance of the class *TaskParameter*. It is conceived for complex actions requiring a specific input parameter in order to accomplish the requested operation, such as the temperature of air conditioners or the light intensity of bulbs. Instances of *TaskDescription* are related with instances of *TaskParameter* by means of the object-property *hasTaskParameter*.

Objects and parameters are related by means of the object-property *hasType* with instances of the class *Type*. The latter class comprises a set of categories that specialize the type of elements introduced in OASIS. For example, floors of buildings are of type "floor" while bulbs are of type "light object".

Figure 2 describes a smart bulb device through its behavior. In particular, the device shows two behaviors, each one consisting of a single goal that, in its turn, consists of a single task. The device is able to perform the actions of turning on and off (i.e., the task operators) the connected bulb (i.e., the task object). The latter component is represented by the individual *bulb*.

Devices are configured in order to fit the needs of some agents such as users. A device configuration comprises a set of optional features that users associate with a single component (recipient of some device operations) or with a single device. There are many features that can be associated with objects or devices such as virtual collocation, physical position, nickname, and so on. A specific configuration providing information about how devices communicate can be assigned to devices when they are automatically detected by the system.

Configurations are represented in OASIS according to the schema in Figure 3, and make use of the following classes:

- *Configuration*: models configurations admitted in OASIS.
- *ComponentConfiguration*: is a subclass of the class *Configuration* and models configurations associated with the components of the environment involved as objects in some operations (i.e., instances of the class *TaskObject*).
- *DeviceConfiguration*: models configurations associated with devices.
- *Connection*: specifies how to physically communicate with a device, e.g., the protocol used and the communication address associated with the device.

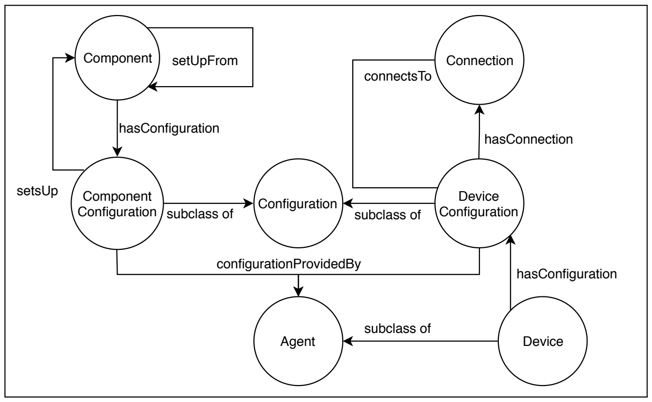

Figure 3. Ontology schema of device configurations

Instances of the class *Device* are associated with instances of the class *DeviceConfiguration* by means of the object-property *hasConfiguration*. Instances of *DeviceConfiguration* are related with instances of the class *Connection* by means of the object-property *hasConnection*. As stated above, instances of the latter class are exploited for creatiing an entry-point that puts devices in communication. In such a case, the object-property *connectsTo* is used to link instances of *DeviceConfiguration* with instances of *Connection*.

The object-property *hasConfiguration* is also used to link instances of the class *Component* with instances of the class *ComponentConfiguration*. Once a component has been configured, the instance of *ComponentConfiguration* is related by means of the object-property *setsUp* with a fresh individual representing the configured component. The latter individual, in its turn, is related with the non-configured component defined by the device in its behavior description by means of the object-property *setUpFrom*.

An example of a component configuration is shown in Figure 4. The example extends the smart bulb device illustrated in Figure 2 by providing a configuration, introduced by the user Alan, of the single bulb controlled by a device. Alan specifies the kitchen as physical position of the bulb; in this case, the object-property *hasSpaceSpan* is used to link the configured object to an instance of the class *Space* representing Alan's kitchen. In this specific smart bulb example, the device is physically indistinguishable from the controlled bulb. In such a case, the user configuration should be provided for the bulb as shown in Figure 4, leaving to the device configuration only the management of the connection information.

Agents that create the configuration of a device or of a component are specified by the object-property *configurationProvidedBy*, linking an instance of the class *Configuration* to an instance of the class *Agent*.

As stated above, OASIS models user requests consisting in entrusting some devices to do something within the domotic environment, in accordance with the restrictions imposed by their configurations. User requests are introduced by exploiting classes and properties used to model device behaviors, except that the desired sets of user goals to be accomplished are related with a user plan. The ontology schema for user requests is depicted in Figure 5.

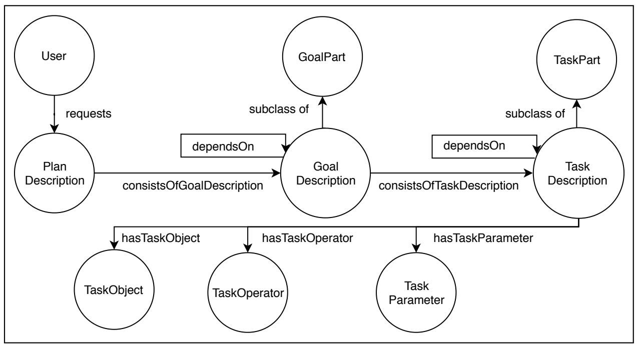

Figure 5. Ontology schema of user requests

User requests are introduced by instances of the class *PlanDescription* related to instances of the class *GoalDescription*. The user requesting the action is modelled by instances of the class *User* that are linked to the plan through the object-property *requests*. Descriptions of goals from user requests are modelled analogously to the descriptions of goals from device behaviors.

In Figure 6, we show an example of a user request consisting of turning off a light, performed by the user Alan.

Finally, a user request is associated with an attempt of finding a device that is able to execute it. OASIS uses the class *TaskExecution* for representing the attempt of executing an agent request. Instances of the class *TaskDescription* defined by the user requests are related with instances of the class *TaskExecution* through the object-property *hasTaskExecution*. An instance of *Device*, representing the device responsible

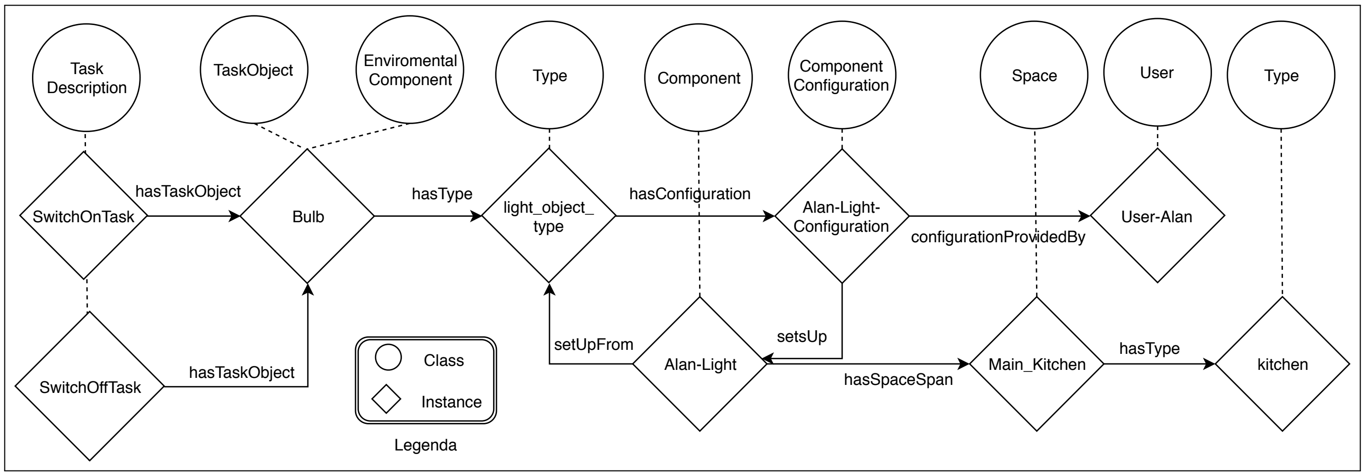

Figure 4. Example of configuration in OASIS

of satisfying the request, is related with the instance of *TaskExecution* by means of the object-property *performs*. Once the selected device attempts to fulfill a user request, the status of the execution task is updated by linking the instance of *TaskExecution* to an instance of the class *ActionStatus*. The most important instances of *ActionStatus* are the individuals *succeeded_status*, representing the correct execution of the action, and the individual *failed_status*, representing the status of those actions that have not been accomplished. The modelling of the attempt of fulfilling agent requests in OASIS is summarized in Figure 7.

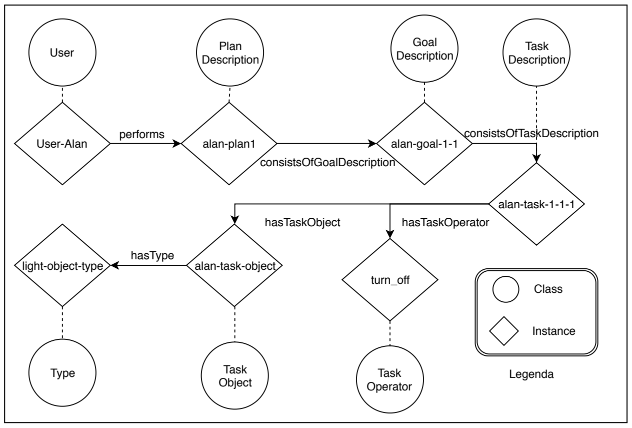

Figure 6. Example of modelling of user requests

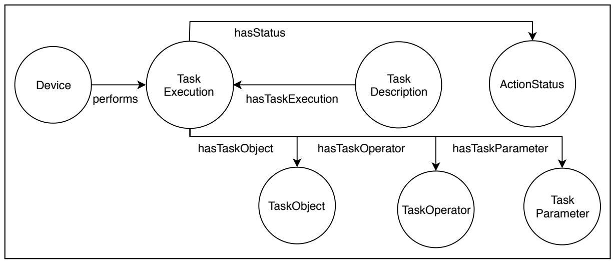

Figure 7. Ontology schema of execution of user requests

In Figure 8, we show an example concerning the execution of Alan's request depicted in Figure 6. In the example, Alan's request of turning off the light, described in Figure 6, is accomplished by the smart bulb (represented in Figure 2) that sets to *succeeded* the execution status of the action.

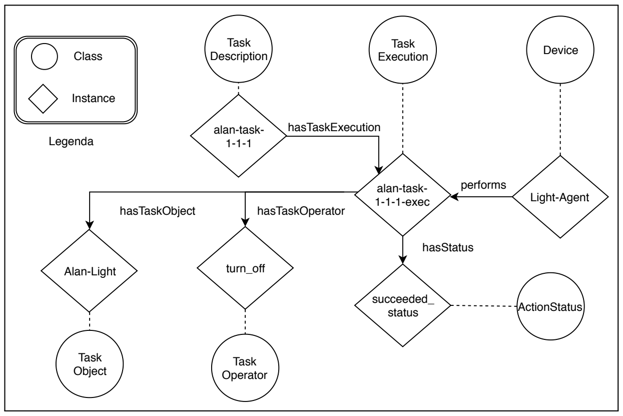

Figure 8. Ontology schema of a task execution

B. Software Architecture

In this section, we briefly summarize the main features of the architecture of PROF-ONTO, illustrated in Figure 9.

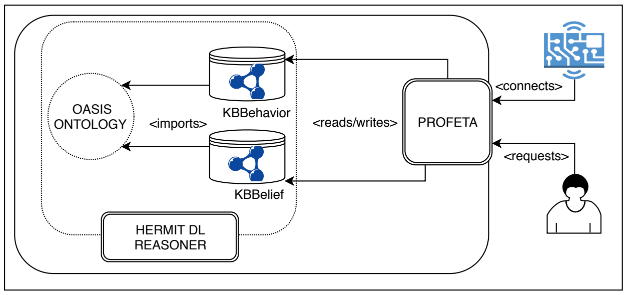

Figure 9. Architecture of PROF-ONTO

The core of PROF-ONTO comprises by OASIS (the ontologies illustrated in Figures 1, 3, 5, and 7), the dataset *KBBe-*

havior collecting behaviors and user defined configurations of devices (RDF graphs of the types depicted in Figures 2 and 4), and the data-set *KBBelief* containing user requests together with their execution information (RDF graphs of the types reported in Figures 6 and 8).

PROF-ONTO knowledge bases are implemented in JAVA by exploiting the OWL API [16] and Apache Jena [17], both used to manipulate the ontological information and to perform SPARQL queries. Consistency of PROF-ONTO knowledge bases is checked by means of the HermiT DL reasoner [18].

PROF-ONTO also exploits PROFETA [19], a *Belief-Desire-Intention* (BDI) rule-based system able to trigger rules representing computations to be performed, in order to parse, interpret, and manage user requests and device connections. BDI rules are passed to the PROF-ONTO core as OASIS knowledge bases.

Since most interaction among common users and IoT domains occur through a natural spoken language, PROF-ONTO manages user utterances via the PROFETA interface. Given a domotic command either by *Speech-To-Text* (STT) services or *Chatbots*, PROFETA processes the string representing the transcription of the user intention in natural language by implementing a robust dependency parser able to deal very satisfactorily with imperative verbs (without raising the issues treated in [20]), within a so-called *Translation Service* (TS). A domotic command usually has the form of an imperative verbal phrase; consequently, the relations generated by the dependency parser and providing information about user intentions have the form $dobj(arg_1, arg_2)$, namely a direct object, where arg_1 and arg_2 are verb and object related to the intention, respectively. For example, given the sentence *wipe the floor*, PROFETA produces the relation *dobj(wipe, floor)*.

Additionally, PROFETA supports other types of relation such as *pobj* (preposition object), which provides additional information about the physical location of an object, and *compound/prt* (compound/particle), which provides the compositionality of phrasal verbs such as “turn off” or of nouns such as “living room”. The reader can refer to [21] for further details concerning the process of extraction of intentions from natural spoken language.

The result of the TS module consists of a set of beliefs representing user intentions together with related parameters. Such beliefs are processed by the PROFETA engine through the use of rules of the form:

$$\textbf{+Int(Verb,Obj,Loc)} \gg \textbf{generate\_request(Verb,Obj,Loc)},$$

which produce the desired plan from the set of beliefs obtained from previous steps. The plan *generate_request* consists in sending user requests to the ontological core of PROF-ONTO through an appropriate wrapper that maps plans in OASIS knowledge bases of the forms described in Section III-A. For the sake of conciseness, details about the syntax of *PROFETA* are omitted and can be found in [19].

PROF-ONTO takes care of the following two activities:

- *Device connection and configuration*. When a device tries to connect to PROF-ONTO, PROFETA receives the request and generates the corresponding knowledge base of OASIS. This is passed to the ontological core of PROF-ONTO, which in turn updates *KBBehavior* accordingly and executes the HermiT reasoner to check its consistency.
- *User requests*. When a user requests an action, *PROFETA* analyzes the command and produces the corresponding RDF graph, which is submitted to the ontological core of PROF-ONTO. PROF-ONTO builds and performs the related SPARQL queries. The result is then sent to PROFETA, which activates the selected device. If the request can be accomplished, the data-set *KBBelief* is updated accordingly. Once the device has performed the action, it updates PROFETA by sending the execution status. Then, PROFETA, in its turn, sends a request to the ontological core of PROF-ONTO, which updates the data-set *KBBelief*. Finally, the HermiT reasoner is called by PROF-ONTO to check the consistency of *KBBelief*.

IV. CASE-STUDY

We describe a simple case-study illustrating the working basics of the framework PROF-ONTO. In our example, the environment consists of (a) a smart bulb, called *light-device*, (b) a user, called Alan, and (c) a domotic assistant running PROF-ONTO.

As a first step, *light-device* is connected to the assistant and suitably configured by the user. The device exposes its behavior by submitting the RDF graph illustrated in Figure 10 to the assistant, which integrates it with the PROF-ONTO knowledge base as described in Section III. We denote with *prof* the prefix of OASIS and with *dev* the prefix of the smart bulb ontology.

Before fully connecting the smart bulb to the assistant and integrating it with the domotic environment, the user is summoned to configure the device. In the particular case of the *light-device* specification, the user may provide a friendly name and indicate the physical collocation of the device. In our example, the user Alan sets the name of the device to *main kitchen light* and collocates it in the main kitchen. Once such information is provided, the configuration manager maps Alan’s request into an RDF graph (see Figure 11) and transmits it to the assistant.

Subsequently, information provided by the smart bulb behavior and the user configuration is merged with the *KBBehavior* knowledge base. Such task is carried out by adding the triples to the *KBBehavior* knowledge base and by executing the HermiT reasoner for consistency checking.

Once a device is connected to the system, namely once RDF triples describing its behavior and configuration are integrated in the *KBBehavior* knowledge base, the system is ready to accept user requests, which are formalized as RDF graphs as well. Request RDF graphs are then merged with the *KBBelief* knowledge base and the HermiT reasoner is executed. In the smart bulb example, Alan sends the command “turn off the light in the kitchen" to the assistant which generates the RDF graph illustrated in Figure 12.

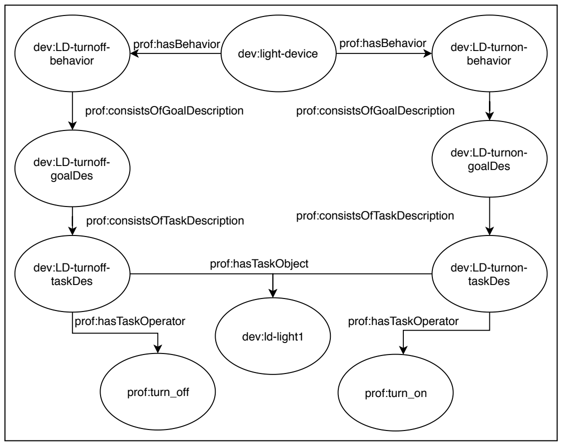

Figure 10. The RDF graph mapping the smart bulb behavior

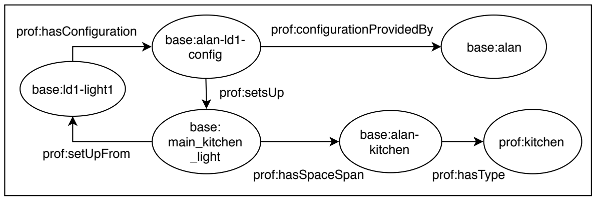

Figure 11. The RDF graph representing the configuration of the smart bulb

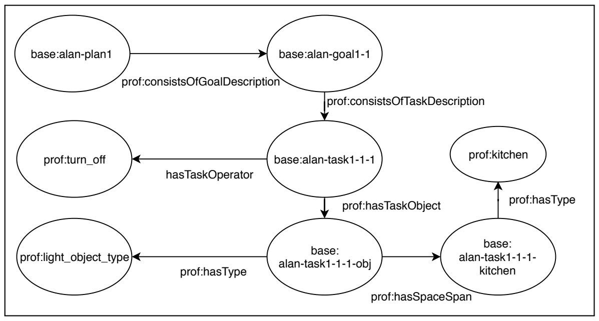

Figure 12. The RDF graph of the request of turning off the kitchen light

The RDF graph representing Alan's request is analyzed by PROF-ONTO in order to produce a SPARQL *CONSTRUCT* query to be executed over the knowledge base *KBBehavior* and over Alan's request graph. The *CONSTRUCT* query produces an RDF graph that associates Alan's request with a specific device able to fulfill his request. The SPARQL query obtained from Alan's command, consisting in turning off the kitchen light, is illustrated in Figure 13. The body of the query consists of three parts. The first one is taken from the specifications of Alan's request. The second part, obtained from the ontology reported in Figure 3, explores all the user configurations in order to discover the type of the device and the place the device has been installed in. The last part, constructed from the ontology in Figure 1, selects an available device fulfilling the conditions specified in the first two parts.

The graph returned by the *CONSTRUCT* query in Figure 13 is illustrated in Figure 14.

```
CONSTRUCT {
?selected_agent prof:performs :alan-task1-1-exec .
:alan-task1-1 prof:hasTaskExecution :alan-task1-1-exec .
:alan-task1-1-1-exec prof:hasTaskOperator ?operation .
:alan-task1-1-1-exec prof:hasTaskObject ?user_object .
?user_object prof:setUpFrom ?device_object . }

WHERE {
:alan-task1-1-1 prof:hasTaskOperator ?operation .
:alan-task1-1-1 prof:hasTaskObject ?object .
?object prof:hasSpaceSpan ?space .
?object prof:hasType ?obtype .
?space prof:hasType ?spacetype .

?config rdf:type prof:ComponentConfiguration .
?config prof:configurationProvidedBy :user .
?config prof:setsUp ?user_object .
?user_object prof:hasType ?obtype .
?user_object prof:hasSpaceSpan ?space_user .
?space_user prof:hasType ?spacetype .
?user_object prof:setUpFrom ?device_object .

?selected_agent prof:hasBehavior ?behav .
?behav prof:consistsOfGoalDescription ?goal .
?goal prof:consistsOfTaskDescription ?task .
?task prof:hasTaskOperator ?operation .
?task prof:hasTaskObject ?agent_subject .
?agent_subject prof:hasType ?obtype . }
```

Figure 13. The SPARQL *CONSTRUCT* query for Alan's request

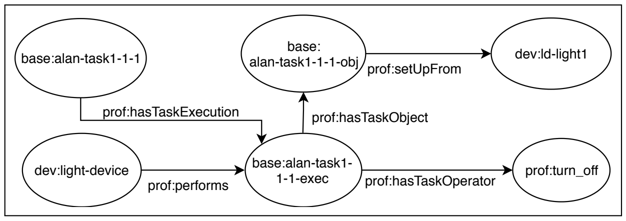

Figure 14. The RDF graph obtained by executing the query of Figure 13

By means of the graph in Figure 14, the smart bulb can execute the requested action, thus turning off the light. In fact, the agent committed to execute the action (*dev:light-device*), the operation (*prof:turn_off*), and the object of the action (*dev:ld-light1*) have been correctly spotted and the device can be activated by sending to it the requested information.

Subsequently, the device updates the status of the execution task by sending to the assistant the triple:

:alan-task-1-1-1-exec prof:hasStatus prof:succeeded_status .

if the task has been successfully accomplished, or by sending the triple

:alan-task-1-1-1-exec prof:hasStatus prof:failed_status .

if the task has not been performed for some reason. The assistant updates its belief knowledge base by adding the information relative to the execution status, as provided by the device. Afterwards, the assistant is ready to accept new requests, configurations, or to connect new devices.

V. CONCLUSIONS AND FUTURE WORK

In this paper we presented PROF-ONTO, a prototype framework that integrates users and IoT devices within domotic environments. Prof-Onto is based on a novel ontology called OASIS, modelling device behaviors, user requests, and their executions. Prof-Onto acts in two phases. The first phase consists in connecting devices to the domotic assistant. Devices share their behaviors by means of knowledge bases of OASIS, which are automatically integrated with Prof-Onto and whose consistency is checked by the HermiT DL reasoner. In the second phase, users send their requests to a BDI rule-based system, called PROFETA, that maps the transcriptions of user requests in OASIS knowledge bases. In the last phase, Prof-Onto automatically selects devices compatible with user requests by means of SPARQL queries specifically constructed. Then, the resulting action is sent to the selected device in order to perform the required action. In this phase, a suitable knowledge base of OASIS is used as a communication and information exchange system between the assistant and the selected device.

We plan to integrate with Prof-Onto temporal modifiers, action restrictions, conditionals already modelled by OASIS, and with user requests directly entrusting specific devices with the execution of actions.

We also intend to study how OASIS can be exploited by *OntologyBeanGenerator 5.0* [22] inside the JADE framework [23] to generate code for agents and artifacts and how it can exploited by *CArtAgO* [24], a framework for building shared computational worlds.

We shall integrate OASIS with the ontology for IoT services defined in [12] and the *Sensor Ontology* in [25]. In addition, we shall extend the set of actions and parameters provided by OASIS with the synset introduced by WordNet [26], in order to make the whole infrastructure multi-language and meaning-oriented as in the case of [21].

In addition, we intend to define a set-theoretic representation of OASIS in the flavour of [27]. However, since OASIS contains existential restrictions, we also need to modify the underlying set-theoretic fragment in such a way as to allow a restricted form of the composition operator. The related reasoning procedure will then be adapted to the new set-theoretic fragments exploiting the techniques introduced in [28], [29] in the area of relational dual tableaux. Finally, we intend to replace the HermiT reasoner in Prof-Onto with a suitably extended version of the set-theoretic reasoner described in [30].

References

[1] D. Oberle, N. Guarino, and S. Staab, *What is an ontology? Handbook on Ontologies*. Springer, 2009.

[2] T. Hofweber, *Logic and Ontology*. Edward N. Zalta (ed.), The Stanford Encyclopaedia of Philosophy (Summer 2018 Edition), 2018.

[3] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.

[4] A. Ricci, M. Viroli, and A. Omicini, "Give Agents Their Artifacts: The A&A Approach for Engineering Working Environments in MAS," in *Proceedings of the 6th International Joint Conference on Autonomous Agents and Multiagent Systems*, AAMAS '07, (New York, NY, USA), pp. 150:1–150:3, ACM, 2007.

[5] F. Baader, I. Horrocks, C. Lutz, and U. Sattler, *An Introduction to Description Logic*. Cambridge University Press, 2017.

[6] D. Allemang and H. J., "Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL," *Elsevier*, 2011.

[7] B. DuCharme, *Learning SPARQL*. O'Reilly Media, Inc., 2011.

[8] J. Hendler, "Agents and the semantic web," *IEEE Intelligent Systems*, vol. 16, no. 2, pp. 30–37, 2001.

[9] M. Hadzic, E. Chang, and P. Wongthongtham, *Ontology-Based Multi-Agent Systems*. Springer Publishing Company, Incorporated, 2014.

[10] D. Fritzsche, M. Grüninger, K. Baclawski, M. Bennett, G. Berg-Cross, T. Schneider, R. Sriram, M. Underwood, and A. Westerinen, "Ontology Summit 2016 Communique: Ontologies within semantic interoperability ecosystems," *Applied Ontology*, vol. 12, no. 2, pp. 91–111, 2017.

[11] G. Bajaj, R. Agarwal, P. Singh, N. Georgantas, and V. Issarny., "A study of existing Ontologies in the IoT-domain," *hal-01556256*, 2017.

[12] W. Wang, S. De, R. Toenjes, E. Reetz, and K. Moessner, "A Comprehensive Ontology for Knowledge Representation in the Internet of Things," in *11th International Conference on Trust, Security and Privacy in Computing and Communications*, IEEE, 2012.

[13] S. N. Akshay Uttama Nambi, C. Sarkar, R. V. Prasad, and A. Rahim, "A Unified Semantic Knowledge Base for IoT," in *World Forum on Internet of Things (WF-IoT)*, IEEE, 2014.

[14] N. Seydoux, K. Drira, N. Hernandez, and T. Monteil, "Capturing the Contributions of the Semantic web to the IoT: a Unifying Vision," in *Semantic Web technologies for the Internet of Things*, ISWC, 2017.

[15] A. Freitas, R. H. Bordini, and R. Vieira, "Model-driven engineering of multi-agent systems based on ontologies," *Applied Ontology*, vol. 12, pp. 157–188, 2017.

[16] M. Horridge and S. Bechhofer, "The OWL API: A Java API for Working with OWL 2 Ontologies," in *Proceedings of the 6th International Conference on OWL: Experiences and Directions - Volume 529*, OWLED'09, pp. 49–58, CEUR-WS.org, 2009.

[17] Apache Software Foundation, "Apache Jena, A free and open source Java framework for building Semantic Web and Linked Data applications," 2011. `https://jena.apache.org`.

[18] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, and Z. Wang, "HermiT: An OWL 2 Reasoner," *Journal of Automated Reasoning*, vol. 53, no. 3, pp. 245–269, 2014.

[19] L. Fichera, F. Messina, G. Pappalardo, and C. Santoro, "A Python Framework for Programming Autonomous Robots Using a Declarative Approach," *Sci. Comput. Program.*, vol. 139, pp. 36–55, 2017.

[20] T. Hara, T. Matsuzaki, Y. Miyao, and J. Tsujii, "Exploring Difficulties in Parsing Imperatives and Questions," in *Proc. of the 5th International Joint Conference on Natural Language Processing*, AFNLP, 2011.

[21] C. F. Longo, C. Santoro, and F. F. Santoro, "Meaning Extraction in a Domotic Assistant Agent Interacting by means of Natural Language," in *28th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises*, IEEE, 2019.

[22] D. Briola, V. Mascardi, and M. Gioseffi, "OntologyBeanGenerator 5.0: Extending Ontology Concepts with Methods and Exceptions," in *Proceedings of the 19th Workshop "From Objects to Agents", Palermo, Italy, June 28-29, 2018.*, pp. 116–123, 2018.

[23] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. Wiley, 2007.

[24] A. Ricci, M. Piunti, M. Viroli, and A. Omicini, "Environment Programming in CArtAgO," in *Multi-Agent Programming: Languages, Tools and Applications*, (Boston, MA), pp. 259–288, Springer US, 2009.

[25] Word Wide Web Consortium, "Semantic Sensor Network Ontology." `https://www.w3.org/TR/vocab-ssn/`.

[26] Princeton University, "WordNet, A Lexical Database for English," 2010. `https://wordnet.princeton.edu`.

[27] D. Cantone, C. Longo, M. Nicolosi-Asmundo, and D. F. Santamaria, "Web Ontology Representation and Reasoning via Fragments of Set Theory," in *Cate, B. and Mileo, A. (eds) Web Reasoning and Rule Systems. Lecture Notes in Computer Science, vol. 9209. Springer*, 2015.

[28] D. Cantone, M. Nicolosi-Asmundo, and E. Orłowska, "Dual tableau-based decision procedures for some relational logics," in *Proceedings of the 25th Italian Conference on Computational Logic, CEUR-WS Vol. 598, Rende, Italy, July 7-9, 2010*, 2010.

[29] D. Cantone, M. Nicolosi-Asmundo, and E. Orłowska, "Dual tableau-based decision procedures for relational logics with restricted composition operator," *Journal of Applied Non-Classical Logics*, vol. 21, no. 2, pp. 177–200, 2011.

[30] D. Cantone, M. Nicolosi-Asmundo, and D. F. Santamaria, "A set-based reasoner for the description logic $\mathcal{DL}_{\mathbf{D}}^{4,\times}$," *Proceedings of SETS 2018, Southampton, United Kingdom, 5 June 2018. CEUR Workshop Proceedings, ISSN 1613-0073*, vol. 2199, 2018.