IoT-O, a Core-Domain IoT Ontology to Represent Connected Devices Networks

Nicolas Seydoux[1,2,3(✉)], Khalil Drira[2,3], Nathalie Hernandez[1], and Thierry Monteil[2,3]

[1] IRIT Maison de la Recherche, University of Toulouse Jean Jaurès, 5 allées Antonio Machado, 31000 Toulouse, France
{nseydoux,hernande}@irit.fr

[2] CNRS, LAAS, 7 avenue du Colonel Roche, 31400 Toulouse, France

[3] Univ de Toulouse, INSA, LAAS, 31400 Toulouse, France
{nseydoux,khalil,monteil}@laas.fr

Abstract. Smart objects are now present in our everyday lives, and the Internet of Things is expanding both in number of devices and in volume of produced data. These devices are deployed in dynamic ecosystems, with spatial mobility constraints, intermittent network availability depending on many parameters (e.g. battery level or duty cycle), etc. To capture knowledge describing such evolving systems, open, shared and dynamic knowledge representations are required. These representations should also have the ability to adapt over time to the changing state of the world. That is why we propose IoT-O, a core-domain modular IoT ontology proposing a vocabulary to describe connected devices and their relation with their environment. First, existing IoT ontologies are described and compared to requirements an IoT ontology should be compliant with. Then, after a detailed description of its modules, IoT-O is instantiated in a home automation use case to illustrate how it supports the description of evolving systems.

1 Semantic Interoperability, a Challenge for the IoT

The Internet of Things (IoT) is gaining more and more traction: some projectionists predict up to 50 billion devices connected in the next five to ten years [1]. The Things of the IoT allow to connect the physical world and virtual representations. IoT applications are based on very heterogeneous devices and technologies, and are deployed in domains as diverse as agriculture, domotics[1], smart cities or e-health. Two types of interoperability issues can be identified: syntactic and semantic, brought by the variety of domains and data models [2]. This paper focuses on semantic interoperability, the ability of systems to attribute the same meaning to the data they exchange.

Semantic interoperability is based on shared, unambiguous, machine-understandable vocabularies, which is why semantic web principles and technologies are seen as semantic interoperability providers, as [3] expresses for the specific

[1] Home automation.

domain of IoT. Knowledge expressed in open formats can be shared and reused, and ontologies can evolve to adapt to new contexts or usages. To ensure the reusability of semantic models across projects and domains, good practices in ontology design have been proposed. In IoT projects, many ontologies have been built, but not always according to these guidelines, hence limiting their reusability (see Sect. 3). This is why we propose IoT-O[2], an IoT core-domain modular ontology engineered for reusability and extensibility. IoT-O is also available on the LOV[3], and based on the initial contribution of [4].

IoT systems are strongly bound to their environment, because they are composed of devices in contact with the physical world. Sensors collect data about their environment, and actuators are devices performing actions that have a direct impact on the world: light bulbs, motors, air conditioning, etc. This paper aims at showing how IoT-O can be used as an ontology to semantically describe devices and data in order to make systems aware of their environment, its evolution, and the changes they can bring to it. Such a description allows smart agents to transform their environment thanks to connected actuators, according to the perceptions they have of it through connected sensors.

In the remainder of this paper, Sect. 2 introduces a motivating use case that will serve to instantiate portions of IoT-O. Section 3 presents the design process of IoT-O, and gives an overview of the ontology. Finally, Sect. 4 details how IoT-O is instantiated in the use case.

2 Motivating Use Case

IoT technologies can have a direct impact on the everyday life of citizens, since it connects their physical environment to virtual applications. That is especially relevant in the case of domotics, where the home can be equipped with multiple low-power devices to provide new services. At LAAS-CNRS, the ADREAM project[4] aims at conducting research on smart buildings thanks to an instrumented, energy-positive building. It is equipped with more than 4500 sensing devices, producing up to 500,000 measures a day. Inside the building, there is a mock-up apartment equipped with commercial devices from various vendors. Deployed devices include sensors (temperature, luminosity, humidity, pressure), actuators (fan, space heater, multiple lamps), which communicate using different technologies (phidget, ethernet, zigbee) with gateways connected to a server.

However, small highly distributed devices usually have a limited processing power, which restrict the range of applications they can support. More complex agents can interact with these devices to collect their data and perform advanced processing to provide a higher level service. In our use case, centered on an elderly healthcare scenario, the complex agent is a robot. It is present in the house, and performs tasks such as helping the person in case of fall, moving heavy objects, pushing a wheelchair, fetching objects and bringing medications. Some of these

[2] http://www.irit.fr/recherches/MELODI/ontologies/IoT-O.

[3] http://lov.okfn.org/dataset/lov/vocabs/ioto.

[4] http://www.laas.fr/public/en/adream.

tasks require the robot to know **where the person is** in the apartment. To have this information, the robot can move around the apartment, scan it with its embedded cameras, and through image processing figure out where the person is. However, it requires the robot either to follow the person around all the time, or to scan the apartment completely each time it has to find the person. To make the robot more acceptable to the person, the house can be equipped with an IoT system, collecting information useful to the robot, such as information given by **presence sensors**. Moreover, the connected devices can provide new functionalities to the robot: he can easily interact with connected light switches or sensors. Our use case is composed of two scenarios: the robot must bring pills at fixed hours to the person using the presence sensors to locate her, and the robot must control the temperature in the apartment using temperature sensors and connected fans to improve the comfort of the person (Fig. 1).

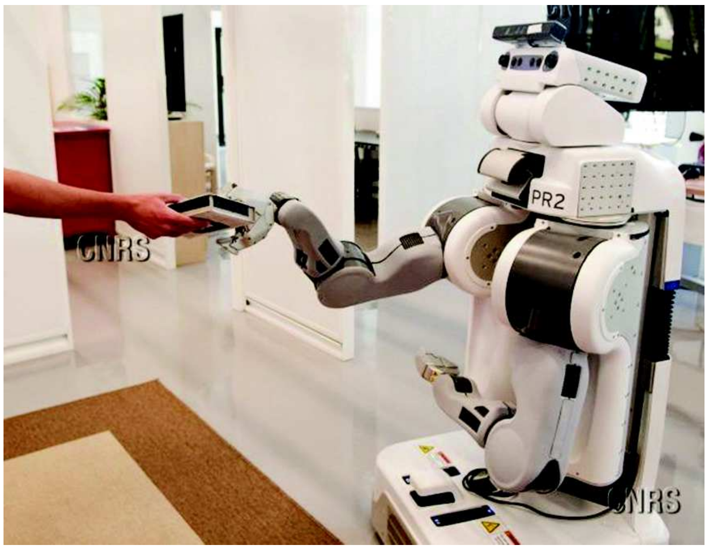

Fig. 1. PR2, the companion robot

In this use case, both syntactic and semantic interoperability are required, among the devices and between the devices and the robot. Syntactic interoperability is ensured using OM2M[5], an open-source horizontal integration platform implementing the oneM2M[6] standard. On top of OM2M, another platform, SemIoTics, enriches the collected data with semantic descriptions, and makes them available to the robot through a REST interface. SemIoTics is driven by a knowledge base capturing knowledge about the devices of the system represented according to our core-domain IoT ontology, and about the environment shared by the robot and the devices (here, the apartment). It is a Java software developed to showcase the role of semantic web technologies in IoT data management, based on Apache Jena. The robot itself is also a semantically enabled agent, it uses a "common sense" ontology and a knowledge base to reason about its 3D environment, as described in [5]. The knowledge specific to the robot relies on

[5] om2m.org.
[6] http://onem2m.org/.

ontologies out of the scope of this paper, but its knowledge base can be extended with any ontology, including IoT-O (as it is done in this paper).

That is why the knowledge described in this paper is implemented in a dedicated knowledge base using IoT-O, ADREAM-Robot[7]: the ontology is shared by the robot and SemioTics, and each system has its own knowledge base. The synchronization between the knowledge bases of the different agents is out of the scope of this paper. The use case focuses on home automation, but IoT-O and our approach are generic enough to be adapted to other domains. For instance, it could be used to support an air quality monitoring system in a smart city, by describing the sensors that collect the data and the services the citizens can subscribe to. The usage of IoT-O and its module in the use case is double: it is used to model the observations about modifications of the apartment, allowing the robot to keep an up-to-date representation of its environment, but also to model the changes the robot wants to make into the apartment through its actions and through the connected devices.

3 IoT-O, Not Just Another IoT Ontology

The design of IoT-O is compliant with the NeOn methodology, presented in [6].

The first step of the NeOn process is to define requirements. We split them in two types: **conceptual**, regarding the concepts that should be present in the ontology (detailed in Sect. 3.1), and **functional**, regarding the ontology structure and design principles (detailed in Sect. 3.2).

These requirements are used to analyze existing IoT ontologies: Semantic Sensor Network (SSN)[8], Smart Appliance REFerence (SAREF)[9], iot-ontology[10], IoT-lite[11], Spitfire[12], IoT-S[13], SA[14] and the oneM2M base ontology[15]. These ontologies are IoT ontologies for which we have found information on the web. Further details are available on the Linked Open Vocabularies for the IoT (LOV4IoT)[16], a recent initiative that lists IoT ontologies, even if they are not referenced on the LOV because they fail to comply with its requirements recalled in [2]. Ontologies related to specific domains impacted by IoT (domotics, agriculture, smart cities...) are out of the scope of this study.

As recommended by NeOn, reusable ontologies that are compliant with parts of the requirements are integrated in our design process. They are analyzed and presented in Sect. 3.3. The core-domain ontology we propose is then described in Sect. 3.4.

[7] https://www.irit.fr/recherches/MELODI/ontologies/Adream-Robot.
[8] http://purl.oclc.org/NET/ssnx/ssn.
[9] http://sites.google.com/site/smartappliancesproject/ontologies.
[10] http://ai-group.ds.unipi.gr/kotis/ontologies/IoT-ontology.
[11] http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite.
[12] http://sensormeasurement.appspot.com/ont/sensor/spitfire.owl.
[13] http://personal.ee.surrey.ac.uk/Personal/P.Barnaghi/ontology/OWL-IoT-S.owl.
[14] http://sensormeasurement.appspot.com/ont/sensor/hachem_onto.owl.
[15] http://www.onem2m.org/ontology/Base_Ontology/.
[16] http://www.sensormeasurement.appspot.com/?p=ontologies.

3.1 The Core Concepts of IoT

Conceptual Requirements: These requirements come from an analysis of the IoT domain, driven by the home automation use case introduced in Sect. 2, but not limited to it: the use case is not seen as an end per se, but as an instantiation of the general domain of the IoT. To be reusable in a wide scope of domains, an IoT ontology should contain a set of key concepts. These are representative of IoT systems with no regard to the application domain. This approach facilitates the merging of data collected in different domains for horizontal applications, and allows the ontology to be an extendable core-domain ontology. We distinguish namely:

– **"Device"** and **"software agent"** constitute the two basic components of an IoT system, composed of both physical and virtual elements. The devices can be of two principle types, not mutually exclusive, that are listed below.
– **"Sensor"** are devices acquiring data, and **"observation"** describe the acquisition context and the data collected by the system. These concepts capture the perception the system has of the evolutions of its environment.
– **"Actuator"** are the devices that enable the system to act on the physical world, and **"action"** represents what they can perform. These concepts capture the knowledge the system has on its own abilities to impact its environment, and to make it evolve.
– **"Service":** In many cases, the IoT and the programmable web are very close. Connected devices can be seen as service providers and consumers, and by specifying a notion of service, every aspect of an IoT system can be represented.
– **"Energy":** In the paradigm of pervasive computing, many distributed Things perform computations. Most of these Things being physical devices, a complete modelling of the system will include a description of their energy consumption. Energy management is a crucial topic in IoT systems.
– **"Lifecycle":** Be it data, devices or services, IoT components are all included in different scales of lifecycles. Devices are switched on and off, services are deployed or updated, pieces of data become outdated... The evolution through a set of discrete states representing a lifecycle is an important concept for IoT systems.

Concept Coverage by Existing Ontologies: Table 1 sums up the assessment of existing IoT ontologies regarding the presence of key concepts. One star means that the concept is superficially represented (coarse-grained specialization, few data/object properties), two stars that the requirement is covered, and stars between parentheses indicate that the requirement is met by an included ontology. IoT-O, the ontology we propose, is also included for comparison. Note that we focus on connected device ontologies, and exclude, on purpose, the ontologies SSN is based on, since they are only focused on sensors and observation, which is only a subset of the identified key concepts. We can observe that some of the IoT ontologies cover most of the key concepts but none of them covers them all. Moreover, the different concepts are not represented with the same level of expressivity. In iot-ontology and SAREF, key concepts such as Actuator or Action are present but their representation is limited. For example, an actuator is defined as a device that modifies a property. This is less expressive than what can be expressed for a sensor with SSN which proposes a deep modeling of the sensors and the property they observe, but also of the relations between the sensors and their observations, and of the observations themselves. In eDIANA[17], an ontology referenced by SAREF, some specializations of actuator are given, but the mappings from these specializations to the *saref:Actuator* concept are not available directly. This analysis highlights the fact that an ontology for Actuators and Actions is needed (c.f. Sect. 3.3). This analysis also highlights the failure of existing IoT ontologies in representing correctly all IoT key concepts. As these concepts are not limited to the IoT domain, reusing ontologies dedicated to them (such as SSN for sensor) could help gain in expressivity, as is shown in Sect. 3.2.

Table 1. Key concept coverage in IoT ontologies

	Actuator	Action	Service	Sensor	Observation	Energy	Lifecycle	Device	Software agent
iot-ontology	*	*	**	(**)	(**)		(*)	(**)	**
saref	*	*	**	*		**		**	**
OWL-IoT-S			(**)	(**)	(**)		(*)	(**)	
SA	*		*	(**)	(**)	(**)	(**)	(**)	
iot-lite	*		*	(*)				(*)	
spitfire				(*)	(*)	**		(*)	
ssn				**	**		*	**	
oneM2M			**					*	
IoT-O	**	**	(**)	(**)	(**)	(**)	(**)	(**)	*

3.2 Good Practices for Ontology Design

Functional Requirements: These requirements capture ontology design guidelines and general semantic web good practices in a domain-agnostic fashion.

Reusability: One of the most important aspects of an ontology in such a broad domain as IoT is reusability: if an ontology is ad-hoc to a project, the work done in its definition will not benefit further projects. It is a critical issue that can be solved by different, non-mutually exclusive approaches:

– **Modularization:** as stated in [7], designing ontologies in separated modules makes them easier to reuse and/or extend. IoT applications are related to many various domains, and it is difficult to capture all these application domains in the same ontology. Modular ontologies can be combined together according to specific needs, which is a more scalable approach.

[17] https://sites.google.com/site/smartappliancesproject/ontologies/ediana-ontology.

- **Ontology Design Patterns:** were introduced in [8]. Designing ontologies that respect Ontology Design Pattern (ODP) increases reusability and their potential for alignment, as shown in [9]. ODPs capture modelling efforts: using them is a way to capitalize on previous work, and to take advantage of the maturity of the semantic web compared to the IoT.
- **Reuse of Existing Sources:** avoids redefinition, and prevents from having to align a posteriori the redefined concepts to the existing sources for interoperability. It is a key requirement for interoperability, which is a real issue in heterogeneous systems.
- **Alignment to Upper Ontologies:** Upper-level ontologies define very abstract concepts in a horizontal manner. They articulate very diverse domain-specific ontologies, which is crucial for broad domains like IoT.
- **Compliance with the LOV Requirements:** The LOV[18] is an online vocabulary register that increases visibility of vocabularies, and favours reuse by ensuring the respect of good practices listed in [2].

Level of formalism: To use the full advantages of the semantic description of devices and data, the description should enable reasoning and inference. This choice is motivated by the possibilities it opens:

- Applied to data, it is a way to bring context-awareness, as presented in [10]
- Applied to devices, it enables Thing discovery or self-configuration [11]
- Applied to services it enables automatic composition as in [12]

However, for concrete applications, the model should also by decidable, and in reasonable time, which de facto excludes an OWL-full model: OWL-DL is therefore the best choice. All surveyed ontologies are expressed in OWL-DL.

Table 2. Reusability of IoT ontologies

	Structured by ODP	Modular	Reuses external ontologies	Aligned with upper ontologies	One the LOV	Available online
iot-ontology			*	**	N	Y
saref		**	*		Y	Y
OWL-IoT-S	(*)	*	**	*	N	Y
SA	(*)	*	**	**	N	N
iot-lite					N	Y
spitfire			*	**	Y	N
ssn	**	**	*	**	Y	Y
oneM2M					N	Y
IoT-O	(**)	**	**	**	Y	Y

Assessment of Existing IoT Ontologies: Table 2 shows that the semantic web best practices for reusability are not always followed: some ontologies are not available online, and the majority is not compliant with the requirements

[18] http://lov.okfn.org.

of the LOV. External ontologies are generally not reused, with the exception of SSN. OWL-S, a service ontology is reused in only one case. The other surveyed ontologies propose redefinitions of the service concept. For example, SAREF redefines the concepts present in multiple ontologies, and proposes alignments in an external, textual document. Design patterns have only been used in ontologies importing SSN. Upper ontologies used are DUL[19] (especially used by SSN) and SWEET[20] (for SA). The limited reuse of ontologies shows a lack of federating ontologies, apart from SSN. SSN being a modular ontology compliant with the semantic web good practices, it is possible to say that these guidelines favour reuse. Section 3.3 focuses on such good practices.

3.3 Reused Ontologies for IoT-O

Identification of existing ontologies is included in the NeOn process. Some concepts, which are part of the conceptual requirements are defined by existing ontologies that are imported in IoT-O to avoid redefinition. SSN is a widely used W3C recommended ontology for sensors and observations. To define the notion of service, IoT-O imports Minimal Service Model (MSM), a lightweight service ontology which is generic enough to represent both REST and WSDL services (contrary to OWL-S[21]). The notion of energy consumption dedicated to the IoT is specified in PowerOnt, an ontology referenced by SAREF. The concepts of lifecycle are described using Lifecycle[22], a lightweight vocabulary defining state machines. We extended Lifecycle in the IoT-lifecycle[23] ontology with classes and properties specific to the IoT. Finally, to maximize extensibility and reusability, IoT-O imports DUL[24], a top-level ontology, and aligns all its concepts and imported modules with it.

Focus on SAN: However, no ontology describes the concept of actuator the way SSN describes the concept of sensor. This is why we propose the Semantic Actuator Network (SAN)[25] ontology. Actuators are devices that transform an input signal into a physical output, making them the exact opposite of sensors. SAN is built around Action-Actuator-Effect (AAE)[26], a design pattern we propose, inspired from the Stimulus Sensor Observation (SSO) design pattern described in [13]. Fig. 2 shows a representation of both the AAE and the SSO design patterns. SSN models the state of the world through stimuli converted by sensors into abstract observations, making the system able to be aware of the evolution of its environment. SAN is complementary: it models the transformation of abstract actuations by actuators into real-world effects, leading to the

[19] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl.
[20] http://sweet.jpl.nasa.gov/.
[21] https://www.w3.org/Submission/OWL-S/, more dedicated to WSDL-based services.
[22] http://vocab.org/lifecycle/schema.
[23] https://www.irit.fr/recherches/MELODI/ontologies/IoT-Lifecycle.
[24] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl.
[25] https://www.irit.fr/recherches/MELODI/ontologies/SAN.
[26] http://ontologydesignpatterns.org/wiki/Submissions:Actuation-Actuator-Effect.

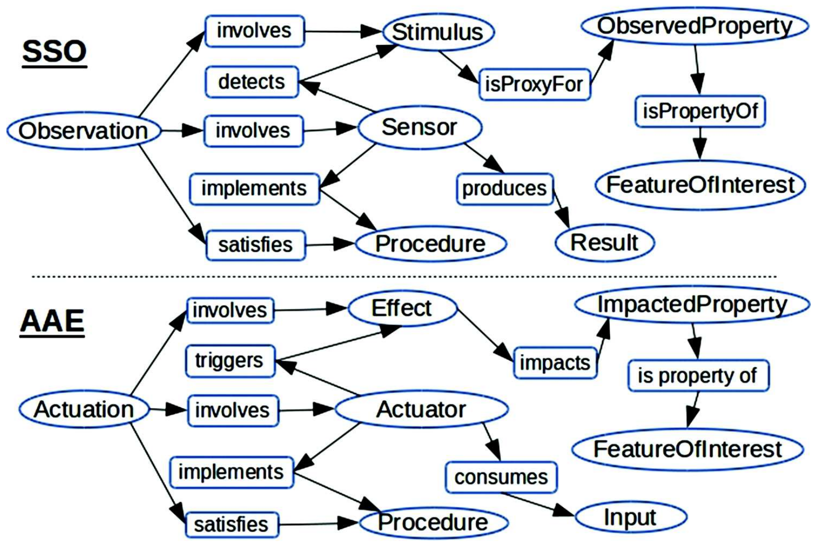

Fig. 2. The SSO and the AAE design patterns, structruring respectively SSN and SAN

representation of the evolution the system brings into its environment. Further details of the main classes of SAN are provided in Sect. 4.4.

3.4 IoT-O, a Modular Core-Domain IoT Ontology

IoT-O, the core-ontology we propose is composed of several modules. IoT-O's architecture is summarized in Fig. 3. The names of the newly created resources are in red and highlighted, the names of the reengineered resources are underlined, and the arrows show dependencies. Solid arrows represent imports, and dashed arrows the reuse of concepts without import.

The Modules of IoT-O:

- The **Sensing module** describes the input data. Its main classes come from SSN: *ssn:Sensor* and *ssn:Observation*. *ssn:Device* and its characteristics (*ssn:-OperatingRange*, *ssn:Deployment...*) provide a generic device description.
- The **Acting module** describes how the system can interact with the physical world. Its main classes come from SAN: *san:Actuator* and *san:Actuation*. It also reuses SSN classes that are not specific to sensing, such as *ssn:Device*.
- The **Lifecycle module** models state machines to specify system life cycles and device usage. Its main classes are *lifecycle:State* and *lifecycle:Transition*.
- The **Service module** represents web service interfaces. Its main classes come from MSM: *msm:Service* and *msm:Operation*. Services produce and consume *msm:Messages*, and RESTful services can be described with hRest.
- **Energy module:** IoT-O's energy module is defined by PowerOnt. It provides the *poweront:PowerConsumption* class, and a set of properties to express power consumption profiles for appliances.

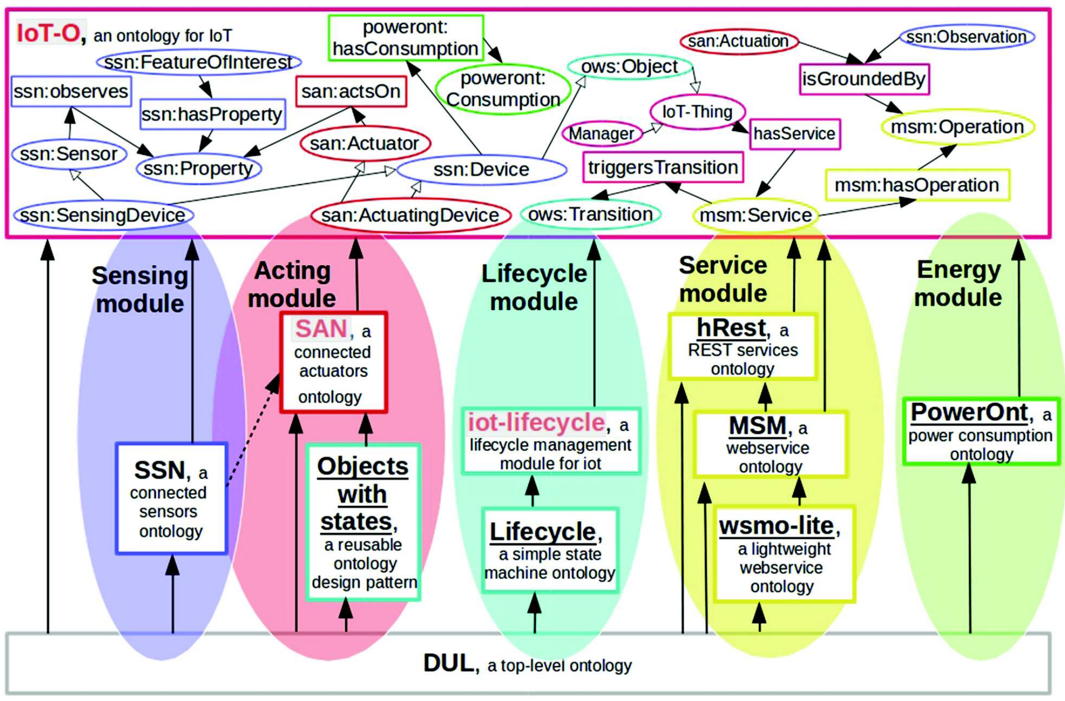
IoT-O, an ontology for IoT
poweront: hasConsumption
san:Actuation
ssn:Observation
ssn:FeatureOfInterest
ssn:observes
ssn:hasProperty
san:actsOn
poweront: Consumption
ows:Object
isGroundedBy
IoT-Thing
Manager
hasService
msm:Operation
ssn:Sensor
ssn:Property
san:Actuator
ssn:Device
triggersTransition
msm:hasOperation
ssn:SensingDevice
san:ActuatingDevice
ows:Transition
msm:Service
Sensing module
Acting module
Lifecycle module
Service module
Energy module
SAN, a connected actuators ontology
hRest, a REST services ontology
iot-lifecycle, a lifecycle management module for iot
MSM, a webservice ontology
PowerOnt, a power consumption ontology
SSN, a connected sensors ontology
Objects with states, a reusable ontology design pattern
Lifecycle, a simple state machine ontology
wsmo-lite, a lightweight webservice ontology
DUL, a top-level ontology

Fig. 3. Overview of IoT-O's architecture (Color figure online)

The Core of IoT-O: IoT-O[27] is both the name of the ontology and of the top module. It gives a conceptualization of the IoT domain, independent of the application, providing classes and relationships to link the underlying modules. Since many concepts are already defined in the modules, IoT-O's core is limited: it defines 14 classes (out of 1126 including all modules), 18 object properties (out of 249) and 4 data properties (out of 78). IoT-O key class is iot-o:IoT_Thing, which can be either an *ssn:Device* or an *iot-o:SoftwareAgent*. The power consumption of *ssn:Devices* is associated to *lifecycle:State* and *poweront:PowerConsumption*. *iot-o:IoT_Thing* is a provider of *msm:Service*, and an *msm:Operation* can have an *iot-o:ImpactOnProperty* on an *ssn:Property*, linking abstract services to the physical world through devices.

As a core domain ontology, IoT-O is meant to be extended regarding specific applicative needs and real-life devices and services. This design, inspired by SSN, makes IoT-O independent of the application.

4 SemIoTics and the Robot: Using IoT-O for Semantic Interoperability

4.1 Implementation of the MAPE-K Loop by the Robot and SemIoTics

[14] describes the concept of autonomic computing, or the control of an entity by an agent thanks to high-level policies and introspective knowledge: the controlling agent and the controlled entity form an autonomic system. The MAPE-K

[27] http://www.irit.fr/recherches/MELODI/ontologies/IoT-O.owl.

loop is a classic control structure in autonomic computing (see Fig. 4), separated in four steps: Monitoring, Analysis, Planning and Execution. The K stands for Knowledge, because the behaviour of the autonomic agent at each step of the loop is guided by a knowledge base, in the general meaning of the term (including but not restricted to the W3C's formalisms of knowledge representation).

In this use case, SemIoTics is performing the Monitoring and the Execution steps when connected devices are involved, and the robot performs the Analysis and the Planning, as well as part of the Monitoring and Execution steps. The robot and SemIoTics have distinct knowledge bases, even if in Fig. 4, a unique knowledge base is represented as the two systems exchange knowledge freely through a rest interface. Consistence issues are not considered in this work, as only one smart agent interacts with the system.

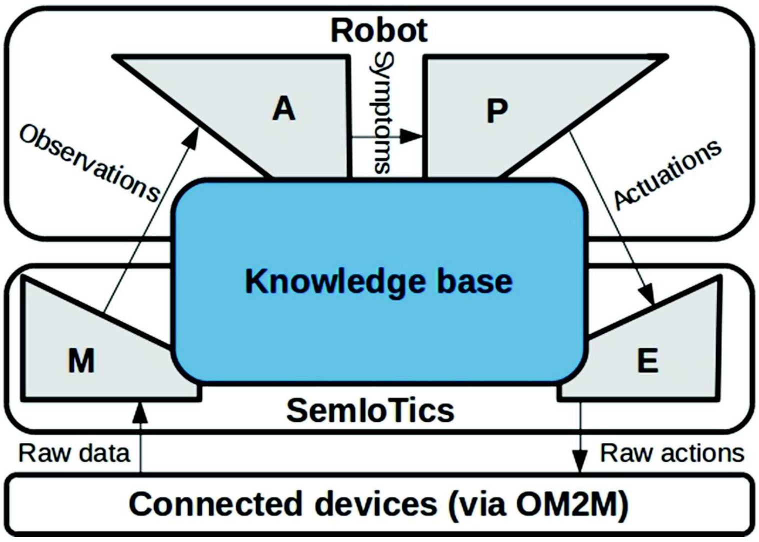

Fig. 4. A representation of the MAPE-K loop, split between the robot and SemIoTics

The process described in Fig. 4 structures the use case: data is first gathered by the sensors, and enriched by SemIoTics. The enriched observations are processed by the robot, which decides to perform actions represented as enriched actuations. These actuations are sent to SemIoTics, which translates them into raw commands for the actuators to perform. In complement to IoT-O, the dogont[28] ontology is used to describe the apartment and the location of devices inside it. Dogont is an ontology identified in the SAREF project, and it is imported by Poweront. We aligned it to IoT-O to integrate it to the use case.

4.2 Monitoring, Where Raw Sensor Data Become Meaningful Observations

The first step of the MAPE-K loop is the monitoring of the controlled system. In the apartment, sensors produce data reflecting their observations. This data

[28] http://elite.polito.it/ontologies/dogont/dogont.html.

is enriched to become a reusable piece of knowledge. Enrichment of sensor data is performed using the SSN ontology, which is in the Sensing module of IoT-O. Each *ssn:Sensor* has an *ssn:Observation* stream composed of *ssn:SensorOutput* whose value is described by *ssn:ObservationValue*. For provenance purposes, a *ssn:SensorOutput* can be linked to its original representation (before enrichment) with the *iot-o:hasRawRepresentation* data property. The sensor's characteristics (*ssn:MeasurementProperty*, the *ssn:Property* of the *ssn:FeatureOfInterest* it observes) are used to enrich the observation as well. IoT-O and SSN are generic ontologies, so they might need to be extended with application-specific modules to be fully functional. Such extension is proposed in the Adream-Robot module[29]. The *ssn:Observation* allow the representation of a characteristic of the environment at a given point in time. The temporality of the sensor measures (and of actuators actions) are represented by a *san:hasDateTime* relations with a http://w3c.org/2006/time#Instant, itself characterized by an *iot-o:hasTimestamp* data property. All the observations related to the same point in time are connected to the same individual, allowing the agent to have a timed representation of its environment and of its evolution.

In our use case, presence sensors and a temperature sensors produce raw observations in the form of XML documents standardized according to the oneM2M Content Instance resource type. The enrichment process requires an approach specific to the data, either by writing a dedicated enrichment script, or by using semantic annotations in the data as in [15], where raw data is stored in relational databases and the database schema is annotated for enrichment. SemioTics uses a dedicated enrichment script that could in the future be extended by producing annotated data.

The presence observation indicates the position of the person in the apartment, and the temperature observation measures the temperature at a given point in space and time, both in the form of *ssn:ObservationValue* instances. This enriched information is accessed by the robot through SemIoTics' REST interface, and it is used to update the robot's representation of the world. This representation of the world is stored in the robots knowledge base, and used as a context in the Analysis step.

4.3 Analysis: Aggregation of Observations in Abstract Symptoms

In the Analysis step, the robot processes his own representation of the world to determine high-level symptoms that need to be addressed by actions.

In the medication scenario, the robot compares the present time to the time when the medication is due to generate the symptom "Medication must be delivered" if necessary.

In the temperature control scenario, user preferences are represented using the concepts defined in yet another module: Autonomic[30]. *ssn:Property* of the environment controlled by the robot within explicit boundaries expressed in the

[29] https://www.irit.fr/recherches/MELODI/ontologies/Adream-Robot.
[30] http://www.irit.fr/recherches/MELODI/ontologies/Autonomic.

form of *autonomic*:*PropertyConstraints* are classified as *autonomic*:*Constrained-Property*. In our use case, the *ssn*:*Property* temperature of the *ssn*:*FeatureOf-Interest* living room air has two constraints, instances of *autonomic*:*Maximum-Value* (25 °C) and *autonomic*:*MinimumValue* (19 °C). The last *ssn:Observation-Value* of the *autonomic*:*ConstrainedProperty* is out of the bounds defined by the *autonomic*:*PropertyConstraint* (26 °C instead of 25), so the temperature is classified by the reasoner as an *autonomic*:*OutOfBoundsProperty* thanks to custom rules.

4.4 Planning, Where Symptoms Are Used to Create a Plan

In the planing phase, the autonomic agent uses the inferred symptoms and policies defined by the user or by the administrator beforehand to define a series of actions that have to be implemented on the system.

In the medication scenario, the robot uses its representation of its environment to locate the person, as it is kept updated in the monitoring phase thanks to the knowledge produced by the sensors and SemioTics. The robot will plan a trajectory to fetch the medication and to reach the person. In this case, the representation of the trajectory itself is ad-hoc to the robot, and isn't linked to IoT-O. The ontology is used to connect the robots internal representation of the world with the observations collected by the sensors and enriched by SemIoTics, providing semantic interoperability between the robot and SemIoTics. If the robot expresses its needs using the same ontology as SemIoTics, or if their ontologies are aligned, it can seamlessly use elements measured by the sensors to plan its trajectory.

In the temperature control scenario, the description of the actions is performed using SAN, the actuator ontology that also describes the actuators in the system. The agent, with successive queries to the knowledge base, will look for *san*:*Actuator* instances that *san*:*actsOn* the *autonomic*:*OutOfBoundsProperty*, and which *autonomic*:*ImpactOnProperty* is coherent with the symptom. In the example, since the temperature is too high, the *adream-model*:*fan* can be used, but also the *adream-model*:*spaceHeater*, since its *adream-model*:*turnOff* operation has a *adream-model*:*NegativeImpact* on the temperature. The orchestration of these actions (if need be) are determined using the Lifecycle module of IoT-O, which represents the devices as state machines by integrating the Objects with States (ows)[31] ontology design pattern. *ssn*:*Device* (superclass of both *ssn*:*SensingDevice* and *san*:*ActuatingDevice*) are objects that *ows*:*hasState exactly 1 ows*:*State*, because objects should only be in one state at a time. The *ows*:*State* is equivalent to the *lifecycle*:*State* (from the Lifecycle[32] vocabulary, extended by the IoT-Lifecycle[33] ontology), and *lifecycle*:*State* are connected by *lifecycle*:*Transition* instances. Thanks to this vision of state machines, stateful transitions (that are only available in certain states of the device) can

[31] http://delicias.dia.fi.upm.es/ontologies/ObjectWithStates.owl.
[32] http://vocab.org/lifecycle/schema.
[33] http://www.irit.fr/recherches/MELODI/ontologies/IoT-Lifecycle.

be represented. Only *msm:Operation* instances that *iot-o:isGroundedBy* a *san:-Actuation* that *iot-lifecycle:triggersTransition* a *lifecycle:Transition* that is a *life-cycle:possibleTransition* of the device current *lifecycle:State* can be called at a given time. For instance, the fan *adream-model:turnOff* operation will only be available if the space heater is on. In our example it is off, so the agent plans to turn on the fan and creates the corresponding *san:ActuationValue*. The selection of devices and their operations is driven by necessity (only the devices impacting the right property are selected), but it can also be driven by policies based on knowledge about the devices intrinsic characteristics expressed with *san:-ActuatingCapability*, composed of *san:ActuatingProperty* that create an actuator profile. It can be used to minimize energy consumption (combined with the Energy module), to optimize reaction time...

4.5 Execution, Where the Plan Is Converted into Actions

In the execution step, the robot implements the planned actions.

For the medication scenario, the robot fetches the medication and brings it directly to the person, it doesn't have to search for her in the apartment. The MAPE-K loop can be repeated while the robot is moving to update the trajectory if the person moves in the house.

For the temperature control scenario, the robot transmits the *san:Actuation-Value* that it wants the system to implement to SemIoTics via a REST interface. SemIoTics will handle the transformation of the knowledge into a representation that can be processed by the target device. This translation can be driven by the semantic description of *msm:Operations*, or dedicated annotations as in [16], where XML schemas are annotated for transformation from RDF to XML. SemIoTics uses the semantic description of operations to perform lowering, and perspectives for this technique are presented in Sect. 5. This translation enables the interaction with low-level, constrained devices that are not able to process complex knowledge representations.

5 Conclusion and Future Works

This paper introduces IoT-O, a modular core-domain IoT ontology designed to be compliant with identified requirements. After a detailed presentation of its modules, an instantiation of IoT-O is presented in a home automation use case. IoT-O is used to bring semantic interoperability between SemioTics, a platform enabling semantic access to connected devices, and a robot. SemIoTics and the robot implements the MAPE-K loop, an autonomic computing pattern, and uses IoT-O at each step of the loop to describe knowledge about the connected devices and about the data they produce and consume. The ontology describes the evolving state of the robot's environment through sensor observations, and the capabilities the system offers to impact this environment through the devices.

In this paper, enrichment and lowering techniques (allowing the transformation back and forth from data to knowledge) have been overviewed. Such techniques are essential to include constrained devices into the IoT: enriched data is more reusable than raw data, but it is heavier to exchange and process, so transformation is required between the end devices (sensors, actuators) and the more powerful nodes of the IoT, e.g. gateways, servers and laptops. We are currently working on such an approach. Other perspectives of our work will be to manage data flows over time in order to learn from previous decisions and their consequences to produce explicit knowledge and enrich policies, and synchronization of the distributed knowledge bases of the smart agents: compared to the use case, multiple agents should be able to exchange knowledge about their environment and to maintain coherence between their representations of the world.

References

1. Ganz, F., Puschmann, D., Barnaghi, P., Carrez, F.: A practical evaluation of information processing and abstraction techniques for the internet of things. IEEE Internet Things J. **2**(4), 340–354 (2015)
2. Gyrard, A., Serrano, M., Atemezing, G.A.: Semantic web methodologies, best practices and ontology engineering applied to Internet of Things. In: IEEE 2nd World Forum on Internet of Things (WF-IoT), pp. 412–417. IEEE (2015)
3. Murdock, P.: White paper: semantic interoperability for the web of things (2016)
4. Alaya, M.B., Medjiah, S., Monteil, T., Drira, K.: Toward semantic interoperability in oneM2M architecture. IEEE Commun. Mag. **53**(12), 35–41 (2015)
5. Lemaignan, S.: Grounding the interaction: knowledge management for interactive robots. Ph.D. thesis (2012)
6. del Carmen Suarez de Figueroa Baonza, M.: NeOn methodology for building ontology networks: specification, sheduling and reuse. Ph.D. thesis (2010)
7. Aquin, M.: Modularizing ontologies. In: Suárez-Figueroa, M.C., Gómez-Pérez, A., Motta, E., Gangemi, A. (eds.) Ontology Engineering in a Networked World, pp. 213–233. Springer, Heidelberg (2012)
8. Gangemi, A.: Ontology design patterns for semantic web content. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 262–276. Springer, Heidelberg (2005)
9. Scharffe, F., Euzenat, J., Fensel, D.: Towards design patterns for ontology alignment. In: Proceedings of the 2008 ACM Symposium on Applied Computing - SAC 2008, p. 2321. ACM Press, New York, March 2008
10. Henson, C., Sheth, A., Thirunarayan, K.: Semantic perception: converting sensory observations to abstractions. IEEE Internet Comput. **16**(2), 26–34 (2012)
11. Chatzigiannakis, I., Hasemann, H., Karnstedt, M., Kleine, O., Kröller, A., Leggieri, M., Pfisterer, D., Römer, K., Truong, C.: True self-configuration for the loT. In: 3rd International Conference on the Internet of Things (IOT) (2012)
12. Han, S.N., Lee, G.M., Crespi, N.: Towards automated service composition using policy ontology in building automation system. In: 2012 IEEE Ninth International Conference on Services Computing, pp. 685–686 (2012)
13. Janowicz, K., Compton, M.: The stimulus-sensor-observation ontology design pattern and its integration into the semantic sensor network ontology. In: Proceedings of the 9th International Semantic Web Conference, 3rd International Workshop on Semantic Sensor Networks, pp. 7–11 (2010)

14. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1), 41–50 (2003)
15. Le-Phuoc, D., Quoc, H., Parreira, J.X., Hauswirth, M.: The linked sensor middleware-connecting the real world and the semantic web. In: Semantic Web Challenge 2011. Number April 2005, pp. 1–8 (2011)
16. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: semantic annotations for WSDL and XML schema. IEEE Internet Comput. **11**(6), 60–67 (2007)