Autonomy through knowledge: how IoT-O supports the management of a connected apartment

Nicolas Seydoux, Khalil Drira, Nathalie Jane Hernandez, Thierry Monteil

Autonomy through knowledge: how IoT-O supports the management of a connected apartment

Nicolas Seydoux[1,2,3], Khalil Drira[2,3], Nathalie Hernandez[1], Thierry Monteil[2,3]

[1] IRIT Maison de la Recherche, Univ. Toulouse Jean Jaurès,
5 allées Antonio Machado, F-31000 Toulouse
{nseydoux,hernande}@irit.fr
[2] CNRS, LAAS, 7 avenue du Colonel Roche,
F-31400 Toulouse, France
{nseydoux,khalil,monteil}@laas.fr
[3] Univ de Toulouse, INSA, LAAS, F-31400, Toulouse, France

Abstract. The IoT is a domain in exponential growth: both the number of connected devices and the quantity of data they produce are increasing. The heterogeneity of technologies involved, and the diversity of domains impacted raise interoperability concerns. The semantic web principles and technologies help tackling these interoperability issues, and ontologies like SSN have been used in several IoT projects. However, many existing IoT ontologies fail to comply with the good practices of the semantic web. After detailing such good practices, this paper proposes IoT-O, a modular core-domain IoT ontology. IoT-O is then showcased in a home automation use case: it is used to semantically describe the devices of the system, and to guide the decisions of an autonomic agent.

1 Achieving semantic interoperability in the IoT

Connected devices, forming a so-called Internet of Things (IoT), are becoming part of our everyday lives: it is estimated that up to 50 billion of them will be exchanging data by the next five to ten years [1]. Such an increase in the number of devices, combined to the heterogeneity of applications domains (agriculture, domotics, smart cities, e-health...) and technologies, raises interoperability issues in the IoT. These issues can be divided in two main classes: syntactic and semantic, brought respectively by the diversity of domains and data models [2]. Two systems are semantically interoperable when they attribute the same meaning to the data they exchange, and it is this type of interoperability that is addressed in this paper. Initiatives such as [3] show the importance of semantic interoperability in the development of the IoT.

Semantic interoperability requires the use of shared, unambiguous, machine-understandable vocabularies that allow to transform raw data issued by sensors to be transformed into self-sufficient, interoperable knowledge. Such vocabularies can be defined using semantic web principles and technologies, and if they are expressed with specific formalisms are called ontologies. Among semantic web principles are **good practices** that should be followed in order to create semantic models that can be **reused** in various use cases, that can be **extended** according to needs discovered a posteriori, that can be **maintained** over time, and that are **compatible with existing ontologies** at the time of design. Being essential to semantic interoperability, many ontologies have been built within IoT projects, and are not always compliant with the aforementioned guidelines. This is why we propose IoT-O[4], an IoT core-domain modular ontology engineered for reusability and extensibility. IoT-O is available on the LOV[5], and based on the initial contribution of [4].

In the remainder of this paper, section 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. Section 4 details how IoT-O is instantiated in the use case, and finally section 5 concludes this paper and provides some insight about future works.

2 Motivating use case

The automation of the home, or domotics, is a domain of the IoT with direct impact on citizens. At LAAS-CNRS, the ADREAM project[6] aims at conducting research thanks to an instrumented, energy-positive building. This building 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 diverse vendors. Deployed devices include sensors (temperature, luminosity, humidity, pressure), actuators (fan, space heater, diverse lamps), which communicate using different technologies (phidget, ethernet, zigbee) with gateways connected to a central server (see fig. 1).

Fig. 1. The connected appartment inside ADREAM

[4] http://www.irit.fr/recherches/MELODI/ontologies/IoT-O
[5] http://lov.okfn.org/dataset/lov/vocabs/ioto
[6] http://www.laas.fr/public/en/adream

Our use case is defined as follows: the user should be able to define simple high-level policies to manage its environment ("the temperature in the living room should stay between 19^{o}C and 25^{o}C"), without having to select specific sensors or actuators to perform the task. He should also be able to extend the capabilities of the apartment by adding devices without restarting the system.

To fulfill these requirements, both syntactic and semantic interoperability among devices are required. Syntactic interoperability is ensured using OM2M[7], an open-source horizontal integration platform implementing the oneM2M standard. On top of OM2M, another platform, SemIoTics, is in charge of ensuring semantic interoperability and of implementing the policies defined by the user. SemIoTics is guided by a knowledge base containing information about the devices, described with our ontology, IoT-O. This use case is applied to home automation and is described in a dedicated knowledge base extending IoT-O, ADREAM-Model[8], but the genericity of IoT-O makes it relevant to any domain impacted by the IoT.

3 IoT-O, an IoT ontology designed in a requirement-driven process

To ensure the respect of good practices, IoT-O design is compliant with the NeOn methodology, presented in [5]. Its first step is to define ontology requirements. We split these in two categories: **conceptual**, regarding the concepts that should be present in the ontology (detailed in section 3.1), and **functional**, regarding the ontology structure and design principles (detailed in section 3.2).

These requirements are used to analyze existing IoT ontologies: Semantic Sensor Network (SSN)[9], Smart Appliance REFerence (SAREF)[10], iot-ontology [11], IoT-lite [12], Spitfire [13], IoT-S[14], SA[15] and the oneM2M base ontology[16].

Had one of these ontologies matched all the requirements, we would not have proposed a new ontology. Even more, ontologies compliant with parts of the requirements and covering concepts useful to IoT-O are reused to limit redefinition, as it is recommanded in NeOn. Details about such ontologies are given in section 3.3.

Studied ontologies are the ones for which we have found information on the web. Further details are available on the Linked Open Vocabularies for the IoT (LOV4IoT)[17], a recent initiative that lists IoT ontologies, even if they are

[7] om2m.org
[8] http://www.irit.fr/recherches/MELODI/ontologies/Adream-Model
[9] http://purl.oclc.org/NET/ssnx/ssn
[10] http://sites.google.com/site/smartappliancesproject/ontologies
[11] http://ai-group.ds.unipi.gr/kotis/ontologies/IoT-ontology
[12] http://iot.ee.surrey.ac.uk/fiware/ontologies/iot-lite
[13] http://sensormeasurement.appspot.com/ont/sensor/spitfire.owl
[14] http://personal.ee.surrey.ac.uk/Personal/P.Barnaghi/ontology/OWL-IoT-S.owl
[15] http://sensormeasurement.appspot.com/ont/sensor/hachem_onto.owl
[16] http://www.onem2m.org/ontology/Base_Ontology/
[17] http://www.sensormeasurement.appspot.com/?p=ontologies

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.

3.1 Identifying IoT core concepts

Conceptual requirements The conceptual requirements aim at capturing knowledge that should be present in an IoT ontology. They are deduced from a bottom-up analysis of the IoT domain. Even if a typical IoT application is presented in section 2, IoT-O is not be limited to this use case. To be reusable in a wider scope, an IoT ontology should necessarily contain some identified concepts tightly associated to the IoT, independently of the applicative context. This approach makes the ontology horizontal and core-domain to the IoT, and suitable for applications in different domains of the IoT that can extend it accordingly. 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.
- **"Actuator"** are the devices that enable the system to act on the physical world, and **"action"** represents what they can perform.
- **"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 (few specializations, 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[18], 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. section 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 shown in section 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 Designing ontologies according to good practices

Functional requirements The functional requirements aim at capturing good practices for ontology design and general semantic web guidelines.

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 [6], 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.
- **Ontology Design Patterns**: were introduced in [7]. Designing ontologies that respect Ontology Design Pattern (ODP) increases reusability and their potential for alignment, as shown in [8]. 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.

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

– **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[19] 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 [9]
– Applied to devices, it enables Thing discovery or self-configuration [10]
– Applied to services it enables automatic composition as in [11]
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 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[20] (especially used by SSN) and SWEET[21] (for SA). The limited reuse of ontologies shows a lack of federating

[19] http://lov.okfn.org
[20] http://www.ontologydesignpatterns.org/ont/dul/DUL.owl
[21] http://sweet.jpl.nasa.gov/

ontologies, apart from SSN. SSN being compliant with the semantic web good practices, it is possible to say that these guidelines favour reuse.

Both concept coverage and compliance with the functional requirements of existing ontologies fail to match the requirements we expect from an IoT ontology, which provides the motivation for our proposal of IoT-O.

3.3 Integrating existing ontologies in IoT-O

Identification of existing ontologies is part of 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. However, no ontology describes the concept of actuator the way SSN describes the concept of sensor. This is why we propose Semantic Actuator Network (SAN)[22], the Semantic Actuator Network 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)[23], a design pattern we propose, inspired from the Stimulus Sensor Observation (SSO) design pattern described in [12].

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[24]). 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[25], a lightweight vocabulary defining state machines. We extended Lifecycle in the IoT-lifecycle[26] ontology with classes and properties specific to the IoT. Finally, to maximize extensibility and reusability, IoT-O imports DUL[27], a top-level ontology, and aligns all its concepts and imported modules with it.

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 figure 2. The names of the resources created from scratch 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.

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

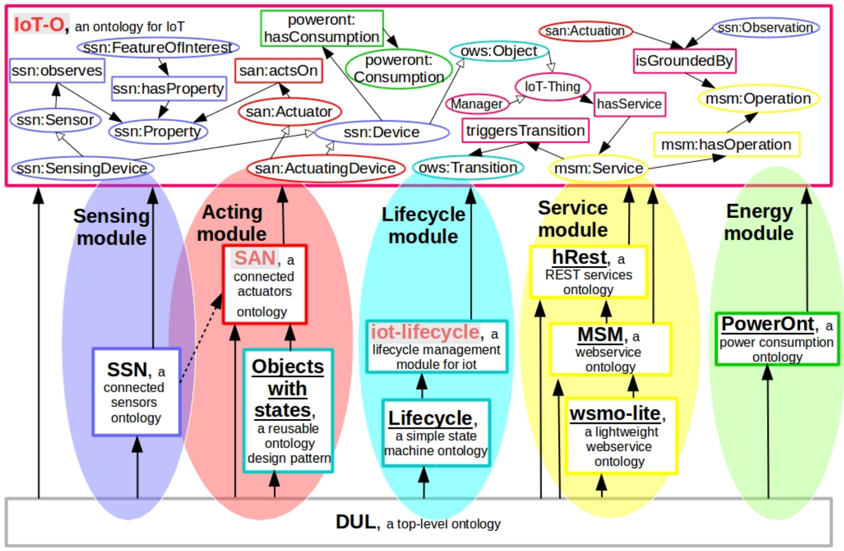

Fig. 2. Overview of IoT-O's architecture

- 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.

The core of IoT-O: IoT-O[28] 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.

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

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 Using IoT-O to manage a smart building

This section gives an overview of the technical choices to implement our use case, and a detailed analysis of the scenario execution that instantiates IoT-O, allowing to see the interest of our requirements.

4.1 Implementing the MAPE-K loop with SemIoTics

Autonomic computing is a programming paradigm proposed in [13] focused on allowing a system to control an entity thanks to high-level policies and introspective knowledge. A classic control structure in autonomic computing is the MAPE-K loop (see fig. 3), separated in four steps : Monitoring, Analysis, Planning and Execution, all exchanging Knowledge with the same knowledge base.

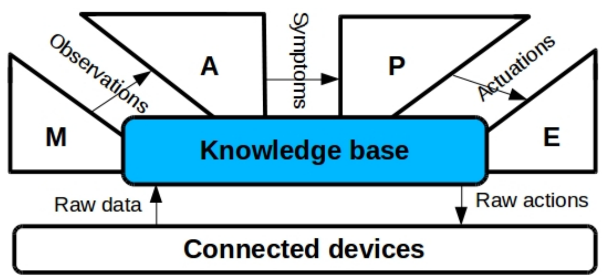

Fig. 3. The MAPE-K loop, adapted to our use case

SemIoTics implements the MAPE-K loop to control the connected devices in the apartment according to the policies fixed by the user. It is Java-based, and uses Apache Jena to manage the knowledge base and query it in SPARQL. The remainder of this section describes the usage of IoT-O at each step of the MAPE-K loop representing the introductory use case, from the temperature sensor measure to the actuator action.

4.2 Monitoring: Enrichment of sensor data into Observations

Monitoring is the collection of signals regarding the controlled entity, here the apartment. The connected sensors collect observations, and produce raw data. This data 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, as well as upper ontologies like QUDT (for the units), and possibly more sensor-specific domain ontologies, necessary due to the horizontal nature of IoT-O and SSN. An application-dedicated extension is for instance proposed in the Adream-Model module[29].

Each *ssn:Sensor* produces *ssn:Observation*, themselves composed of *ssn:-SensorOutput* whose value is described by *ssn:ObservationValue*. To maintain provenance knowledge, a *ssn:SensorOutput* can be linked to its raw representation with the *iot-o:hasRawRepresentation* data property. The sensor's characteristics (*ssn:MeasurementProperty*, the *ssn:Property* of the measured *ssn:Feature-OfInterest*) can lift the observation as well.

In the use case, the data initially observed by the sensor is standardized according to oneM2M, and then enriched. The enrichment process is ad-hoc, being performed by a dedicated enrichment script. For the example's sake, the sensor observes a temperature of 26^{o}C, converted into a *ssn:ObservationValue*. Once enriched, the observation is stored in the knowledge base to be used in the Analysis step.

4.3 Analysis: Abstraction of Observations in symptoms

Analysis is the identification of observed signals as hiegher-level symptoms. In the use case, enriched observations are processed and matched against rules representing user preferences. These preferences are described using the concepts defined in yet another module: Autonomic[30]. The user creates *autonomic:Property-Constraints* (seamlessly through a graphical interface), transforming a *ssn:-Property* into a *autonomic:ConstrainedProperty*. Specifically, the *ssn:Property* temperature of the *ssn:FeatureOfInterest* living room air has two constraints, instances of *autonomic:MaximumValue* (25^{o}C) and *autonomic:MinimumValue* (19^{o}C). The last *ssn:ObservationValue* of the *autonomic:ConstrainedProperty* is out of the bounds defined by the *autonomic:PropertyConstraint* (26^{o}C instead of 25), so the temperature is classified by the reasoner as an *autonomic:Out-OfBoundsProperty* thanks to custom rules expressed in the Jena rule engine language.

4.4 Planning: Deducing Actions from symptoms

Planing is the computation to an appropriate response to the monitored system's symptoms. To determine its actions, the autonomic agent relies on its knowledge base, which contains a priori defined policies. For semIoTics, actions to be implemented by the agent are described using SAN, the action and actuator ontology. The agent queries the knowledge base to look for *san:Actuator* instances that *san:actsOn* the *autonomic:OutOfBoundsProperty*, and which *san:-receivesActuation* an actuation that *iot-o:hasImpact* an *autonomic:ImpactOn-Property* that is coherent with the symptom. In the example, since it is too hot, the *adream-model:fan* can be used, but also potentially the *adream-model:space-Heater*, since its *adream-model:turnOff* operation has a *adream-model:Negative-Impact* on the temperature.

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

If multiple, sequential actions have to be performed, they are orchestrated using the Lifecycle module of IoT-O, which rrelies on the the Objects with States (ows)[31] ontology design pattern to represent devices as state machines. *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. This description of devices allows for the representation of transitions only available in certain states of the device. Only *msm*:*Operation* instances that *iot-o:isGroundedBy* a *san:-Actuation* that *iot-lifecycle*:*triggersTransition* a *lifecycle*:*Transition* that is a *lifecycle*:*possibleTransition* of the device current *lifecycle*:*State* can be called at a given time. For instance, the space heater *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 system, to minimize energy consumption, to optimize reaction time...

4.5 Execution: Transmitting the Actions to the actuators

Execution is the concrete implementation of the plan by the agent on the controlled system. An *san*:*Action* can be executed if it *iot-o*:*isGroundedBy* an *msm:Operation*. The agent converts the *san*:*ActuationValue* format that target devices can process, here REST commands. The translation of knowledge into a simpler data format (the opposite process of enrichment) can be driven by the semantic description of Operations, or dedicated annotations as in [14], where XML schemas are annotated for transformation from RDF to XML. This translation enables the agent to interact with low-level, constrained devices that are not able to process complex knowledge representations. The example cycle is complete: the agent calls the *adream-model*:*turnOn* operation, and the fan cools the apartment.

5 Conclusion and future works

This paper introduces both functional and conceptual requirements to build an IoT ontology usable in a wide scope of projects and compliant with the semantic web good practices. These requirements drove the development of the core contribution of the paper: IoT-O, a modular core-domain IoT ontology. IoT-O's modules are presented in details, to show their compliance with the requirements. IoT-O is then used by semIoTics, a system implementing the MAPE-K loop, an

[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

autonomic computing structure. SemIoTics is described in a home automation use case that instantiates knowledge described using IoT-O's modules.

In the IoT, data is produced raw by sensors, and needs to be enriched to become reusable knowledge. However, enriched data is heavier to exchange and process than raw data, making it unsuitable to be consumed by the more constrained devices (typically sensors and actuators). To allow these IoT nodes to be semantically interoperable with more powerful ones (e.g. gateways, servers, laptops), lowering techniques, transforming knowledge into raw data, are required. We are currently working on such an approach. Other perspectives of our work will be to define an intuitive way to help end users/administrators express constraints and policies to drive the system.

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 of Things Journal **2**(4) (2015) 340–354
2. Gyrard, A., Serrano, M., Atemezing, G.A.: Semantic web methodologies, best practices and ontology engineering applied to Internet of Things. In: 2015 IEEE 2nd World Forum on Internet of Things (WF-IoT), IEEE (2015) 412–417
3. Murdock, P.: White paper: Semantic interoperability for the web of things (2016)
4. Ben-Alaya, M., Medjiah, S., Monteil, T., Drira, K.: Toward semantic interoperability in oneM2M architecture. IEEE Communications Magazine **53**(12) (2015)
5. del Carmen Suarez de Figueroa Baonza, M.: NeOn methodology for building ontology networks : specification, sheduling and reuse. PhD thesis (2010)
6. Aquin, M.: Modularizing ontologies. Ontology engineering in a networked world **Springer B** (2012) 9–34
7. Gangemi, A.: Ontology Design Patterns for Semantic Web Content. History **3729**(4) (2005) 262–276
8. Scharffe, F., Euzenat, J., Fensel, D.: Towards design patterns for ontology alignment. In: Proceedings of the 2008 ACM symposium on Applied computing - SAC '08, New York, New York, USA, ACM Press (mar 2008) 2321
9. Henson, C., Sheth, A., Thirunarayan, K.: Semantic perception: Converting sensory observations to abstractions. IEEE Internet Computing **16**(2) (2012) 26–34
10. 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)
11. 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. (2012) 685–686
12. 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. (2010) 7–11
13. Kephart, J., Chess, D.: The vision of autonomic computing. Computer **36**(1) (jan 2003) 41–50
14. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: Semantic Annotations for WSDL and XML Schema. IEEE Internet Computing **11**(6) (2007) 60–67