

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

Nicolas Seydoux^{1,2,3}, Khalil Drira^{2,3}, Nathalie Hernandez¹, 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 are semantic interoperability providers, 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 Semantic interoperability in the IoT

The IoT is gaining more and more traction: some projectionists predict up to 50 billion devices connected in the next five to ten years [5]. IoT applications are based on very heterogeneous devices and technologies and are deployed in domains as diverse as agriculture, domotics, 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 [6]. This paper focuses on semantic interoperability, the ability of systems to attribute the same meaning to the data they exchange.

Semantic interoperability requires the use of shared, unambiguous, machine-understandable vocabularies, which is why semantic web principles and technologies are seen as semantic interoperability providers. 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. These ontologies are not always

compliant with these guidelines. This is why we propose IoT-O⁴, an IoT core-domain modular ontology engineered for reusability and extensibility. IoT-O is also available on the LOV⁵.

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. Finally, section 4 details how IoT-O is instantiated in the use case.

2 Motivating use case

The miniaturization of devices has made it possible to disseminate multiple low-power devices in an everyday environment, such as the home. The automation of the home, or domotics, is a domain of the Internet of Things (IoT) with direct impact on citizens. At LAAS-CNRS, the ADREAM project⁶ 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, zig-bee) with gateways connected to a central server (see fig. 1).



Fig. 1. The connected apartment inside 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°C and 25°C"), without having to select specific

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>

sensors or actuators to perform the task. The user 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 OM2M⁷, 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-Model⁸, but the genericity of IoT-O makes it relevant to any domain impacted by the IoT.

3 IoT-O, not just another IoT ontology

The design of IoT-O is compliant with the NeOn methodology, presented in [3].

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 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-ontology¹¹, IoT-lite¹², Spitfire¹³, IoT-S¹⁴, SA¹⁵ and the oneM2M base ontology¹⁶. 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)¹⁷, 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 [6]. 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 satisfying parts of the requirements are analyzed and presented in section 3.3. The core-domain ontology we propose is finally described in section 3.4.

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>

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 section 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.
- **”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 2 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 upon, 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 eDI-ANA¹⁸, 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 is shown in section 3.2.

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

Fig. 2. Key concept coverage in IoT ontologies

3.2 Good practices for ontology design

Functional requirements

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 [1], 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 [4]. Designing ontologies that respect Ontology Design Pattern (ODP) increases reusability and their potential for alignment, as shown in [13]. 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.

¹⁸ <https://sites.google.com/site/smartappliancesproject/ontologies/ediana-ontology>

- **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 LOV¹⁹ is an online vocabulary register that increases visibility of vocabularies, and favours reuse by ensuring the respect of good practices listed in [6].

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 [8]
- Applied to devices, it enables Thing discovery or self-configuration [2]
- Applied to services it enables automatic composition as in [7]

However, for concrete applications, the model should also be 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.

	Structured by ODP	Modular	Reuse of external ontologies	Aligned with upper ontologies	On 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

Fig. 3. Reusability of IoT ontologies

Assessment of existing IoT ontologies Table 3 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 DUL²⁰ (especially used by SSN) and SWEET²¹ (for SA). The limited reuse of ontologies shows a lack of federating ontologies, apart from SSN. SSN being a modular ontology compliant with the

¹⁹ <http://lov.okfn.org>

²⁰ <http://www.ontologydesignpatterns.org/ont/dul/DUL.owl>

²¹ <http://sweet.jpl.nasa.gov/>

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 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 [9].

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-S²⁴). 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 Lifecycle²⁵, a lightweight vocabulary defining state machines. We extended Lifecycle in the IoT-lifecycle²⁶ ontology with classes and properties specific to the IoT. Finally, to maximize extensibility and reusability, IoT-O imports DUL²⁷, 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 4. 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.

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/>, which is more dedicated to WSDL-based 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>

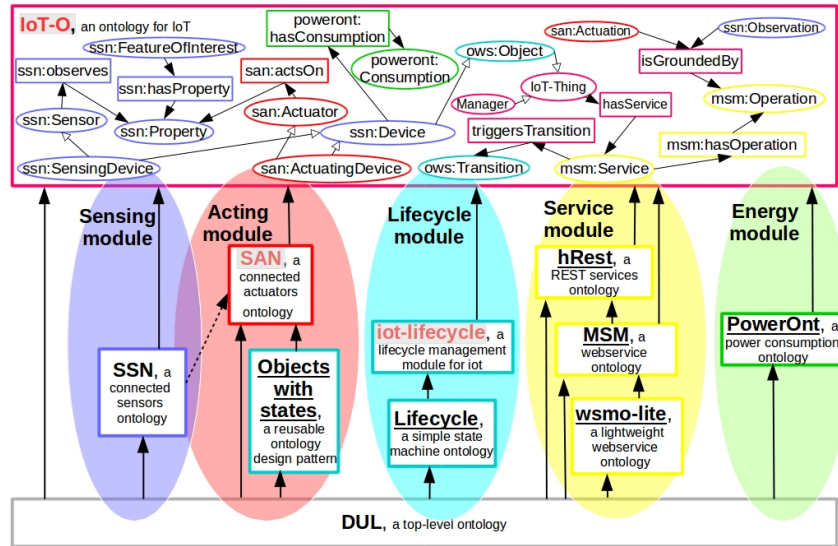


Fig. 4. Overview of IoT-O’s architecture

- 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-O²⁸ 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

²⁸ <http://www.irit.fr/recherches/MELODI/ontologies/IoT-O.owl>

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 : using IoT-O in a smart building

4.1 SemioTics, an implementation of the MAPE-K loop

Autonomic computing is a programming paradigm proposed in [10] 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. 5), separated in four steps : Monitoring, Analysis, Planning and Execution, all exchanging Knowledge with the same knowledge base.

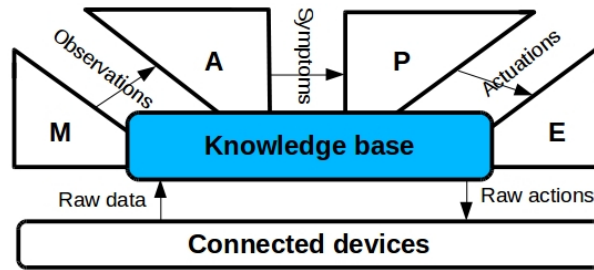


Fig. 5. A representation of the MAPE-K loop, adapted to our use case

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, from a temperature sensor measure to an actuator action.

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 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) can be 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-Model module²⁹.

In our use case, the temperature sensor produces 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 [12], where raw data is stored in relational databases and the database schema is annotated for enrichment. For the example's sake, the sensor observes a temperature of 26°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, where observations are aggregated in abstract symptoms

In the Analysis step, the enriched observations are compared to the needs expressed by the user (represented by rules). User preferences are represented using the concepts defined in yet another module: Autonomic³⁰. The user creates *autonomic:PropertyConstraints* (seamlessly through a graphical interface), transforming a *ssn:Property* into a *autonomic:ConstrainedProperty*. In our use case, the *ssn:Property* temperature of the *ssn:FeatureOfInterest* living room air has two constraints, instances of *autonomic:MaximumValue* (25°C) and *autonomic:MinimumValue* (19°C). The last *ssn:ObservationValue* 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. 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 *san:receivesActuation* an actuation that *iot-o:hasImpact* an *autonomic:ImpactOnProperty* that 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.

²⁹ <http://www.irit.fr/recherches/MELODI/ontologies/Adream-Model>

³⁰ <http://www.irit.fr/recherches/MELODI/ontologies/Autonomic>

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 Lifecycle³² vocabulary, extended by the IoT-Lifecycle³³ 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 be represented. 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, where the plan is converted into actions

With the monitoring step, the execution is the moment when the agent is in direct contact with the controlled system: it implements the different actions composing the plan. The agent can execute a *san:Action* if it *iot-o:isGroundedBy* an *msm:Operation*. During the execution, the agent transforms the *san:ActuationValue* in a representation which is suitable for the corresponding operation. 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 [11], 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 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.

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>

The system implements the MAPE-K loop, an element of autonomic computing, 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.

In this paper, enrichment and translation 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, laptops... 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. Aquin, M.: Modularizing ontologies. *Ontology engineering in a networked world* Springer B, 9–34 (2012)
2. Chatzigiannakis, I., Hasemann, H., Karnstedt, M., Kleine, O., Kröller, A., Leggieri, M., Pfisterer, D., Römer, K., Truong, C.: True Self-Configuration for the IoT. In: 3rd International Conference on the Internet of Things (IOT) (2012)
3. del Carmen Suarez de Figueroa Baonza, M.: NeOn methodology for building ontology networks : specification, sheduling and reuse. Ph.D. thesis (2010)
4. Gangemi, A.: *Ontology Design Patterns for Semantic Web Content*. *History* 3729(4), 262–276 (2005)
5. 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), 340–354 (2015)
6. Gyrard, A., Serrano, M., Atezing, 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). pp. 412–417. IEEE (2015)
7. 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)
8. Henson, C., Sheth, A., Thirunarayan, K.: Semantic perception: Converting sensory observations to abstractions. *IEEE Internet Computing* 16(2), 26–34 (2012)
9. 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)
10. Kephart, J., Chess, D.: The vision of autonomic computing. *Computer* 36(1), 41–50 (jan 2003)
11. Kopecký, J., Vitvar, T., Bournez, C., Farrell, J.: SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing* 11(6), 60–67 (2007)
12. Le-Phuoc, D., Quoc, H., Parreira, J.X., Hauswirth, M.: The linked sensor middle-ware—connecting the real world and the semantic web. In: *Semantic Web Challenge 2011*. pp. 1–8. No. April 2005 (2011)
13. Scharffe, F., Euzenat, J., Fensel, D.: Towards design patterns for ontology alignment. In: Proceedings of the 2008 ACM symposium on Applied computing - SAC '08. p. 2321. ACM Press, New York, New York, USA (mar 2008)