An ECA-based Semantic Architecture for IoT Building Automation Systems

Paolo Lillo, Luca Mainetti, Vincenzo Mighali, Luigi Patrono, and Piercosimo Rametta

Original scientific paper

Abstract - **The Internet of Things (IoT), with its plethora of smart objects and technologies, allows to realize smart environments in several scenarios. However, the existing solutions are strictly intended for specific applications and their customization is often limited to what developers have considered at the design and implementation time. So, the integration of new functionalities requires significant changes by developers, while common users cannot make personalizations by themselves. For these reasons, this work deals with the definition of a novel rule-based semantic architecture for the easy implementation of building automation applications in an IoT context. Applications are structured as an Event-Condition-Action (ECA) rule and the layered architecture separates high-level semantic reasoning aspects from low-level execution details. The proposed architecture is also compared with main state-of-the-art solutions and a standard-based implementation framework is suggested. The last aspect is treated by referring to standardized guidelines and widely-accepted platforms, in order to make the proposal more attractive and robust.**

Index terms - **IoT, semantic technologies, ECA rule, ontologies, building automation, oneM2M**

I. Introduction

The current trend in the development of building automation applications in an Internet of Things (IoT) context is to provide users with simple Integrated Development Environments (IDE), giving them low-level control of sensor and actuator devices located in the real environment [1, 2]. Indeed, since the low-level issues around smart objects, first of all the power consumption problem, could be solved by exploiting several solutions already in the literature [3, 4, 5], the high-level accessibility of users to the smart environment is still an open issue. Currently, by leveraging simplified IDEs, users can express their preferences without worrying about hardware and programming languages details. Unfortunately, these preferences are limited to those implemented by system developers, so common users have to give up the opportunity to define new system behaviors by themselves. Since the IoT world evolves rapidly thanks to the development of new devices, an ideal IoT-oriented system should provide end-users with the possibility to define new software behaviors in declarative way. Indeed, during the interaction with the surrounding environment, user is only interested in the final high-level effect of his/her actions. These actions, related to certain physical or abstract properties, should expose only the final goal they are able to achieve and should be in charge of the interaction with physical devices. To reach this result, the system should follow some design principles. First of all, the applications to be created should have a well-defined and modular structure. In particular, the rule-based Event-Condition-Action (ECA) pattern could be a proper candidate. The structure of an ECA rule is: on EVENT if CONDITIONs then ACTIONs. It means that one or more actions can be performed on a system when certain events occur under certain conditions. Often, both conditions and events of an ECA rule could refer to the users' position in the environment, i.e. they could be based on location-aware concept [6, 7]. In addition, in order to make the rule processing and execution more autonomous and efficient, the system should be based on semantic technologies, so that all information is in a machine-understandable format. By modeling all concepts in the domain of interest with a semantic language, i.e. by defining one or more ontologies, a semantic reasoner can infer new information based on that already contained in the system. In this perspective, it is worth noting that, in order to run their rules, traditional Rule-based Engines (e.g. Drools Expert) need a Working Memory containing the Facts [8], i.e. items of information involved in the reasoning process during rule execution. For this reason, a key step is to map information contained in the ontology with Facts in a suitable format, and to re-map the generated information to ontology after rule execution. Avoiding this two-way mapping between Working Memory and Ontologies formats has been one of the main goals of the proposed rule-based semantic architecture. Another issue in this context is the implementation of custom SWRL Built-ins, which varies depending on the chosen semantic reasoner, such as Pellet [9].

This work is a natural extension of the work presented in [10], whereof it recalls the logical architecture and proposes an implementation framework, based on the oneM2M specifications (explained in section V.B), aimed to foster autonomous communications in a M2M context. In more details, the main goal of this work is to follow standardized guidelines in the development of the proposed architecture, in order to make it more attractive and meaningful. Indeed, the majority of IoT stakeholders, such as enterprises, is still reluctant to invest directly on IoT solutions and most of them seek to secure their entrance with standardization. So, a new solution based on IoT technologies should be oriented as much as possible on standardized specifications and technologies

Manuscript received October 30, 2015. Revised February 28, 2016.

P. Lillo, L. Mainetti, V. Mighali, L. Patrono and P. Rametta are with the Department of Innovation Engineering, University of Salento, Lecce 73100, Italy (E-mails: {paolo.lillo, luca.mainetti, vincenzo.mighali, luigi.patrono, piercosimo.rametta}@unisalento.it).

(both hardware and software). In this perspective the oneM2M specifications represents one of the best choices to achieve this objective.

To briefly recall what presented in [10], it is worth noting that in the proposed architecture the semantic reasoning layer is clearly separated from the action execution layer, which deals with low-level implementation details. This is achieved through some key design choices. First, in the domain ontologies, simple concepts as *System*, *High-level States*, and *High-level Actions*, allow to describe the environment and its changes in human-understandable way. Second, applications are actually *ECA Rules* and are defined by selecting the *High-Level State* of the *System* that acts as event trigger, the *High-Level States* of related *Systems* that must be satisfied when the event occurs and *High-Level Actions* to be taken on the target *Systems*. These actions are simply individuals of the domain ontologies and do not implement any actual (low-level) operation. Third, a simple semantic reasoner has seamless access to all information needed to perform *ECA Rules* evaluation, which will only affect the updating of certain individual properties in the domain ontologies. Finally, appropriate ontology listeners initiate the execution of low-level operations, which are carried out outside the semantic framework through an Action Engine. Low-level operations are defined by domain experts and exploit the devices' low-level APIs for implementing all the complexity of the physical interaction. The benefits of this solution are twofold. On the one hand, end-users are not required to know any programming language or control algorithm for defining applications, since visual wizards guide them during application creation. On the other hand, action implementations can be very effective and efficient, since they are defined by domain experts and are not tied to rule engines capabilities. Moreover, device heterogeneity is abstracted by introducing an architecture layer that hides all low-level details

The rest of the paper is organized as follows. Section II reports the state-of-the-art related to rule-based semantic architectures for building automation applications. A deep description of the domain model is presented in Section III, while the detailed description of the proposed architecture is given in Section IV. In Section V, the proposed architecture is firstly compared with similar solutions in the state-of-the-art, in order to evaluate, from a qualitative point of view, its effectiveness and potential benefits; then, a feasible implementation based on the oneM2M specifications and the OSGi framework is proposed. Finally, conclusions are drawn in Section VI.

II. RELATED WORKS

One of the first attempts to apply ECA rule pattern for building automation applications is [11]. In this paper, authors propose a rule-based framework for the management of heterogeneous systems in a smart home environment. The proposed framework is based on an ECA rule mechanism with SOAP technology that provides interoperability among sensors and actuators systems. Main components of the framework consist of a home application server, a database module, and a service level application module. ECA rule mechanism is embedded in a database as data in the form of table, with a proper retrieval sequence. A more recent study on ECA pattern applied to building automation is reported in [12]. This paper suggests that an ECA-based framework is suitable both for describing the desired system operations in a user-centric smart environment, and for linking to an event-based network made up of constrained sensors and actuators. A service-oriented architecture provides a suitable mechanism for separating out the event signaling functions of sensors from the details of the underlying hardware technologies. In order to address device heterogeneity and foster machine-to-machine capabilities among systems, a key step in this context is to base systems on semantic technologies. The first studies in this direction were focused on Wireless Sensor Networks (WSN). In [13], authors propose a sensor information processing architecture, called SWASN (Semantic Web Architecture for Sensor Networks). The layered SWASN architecture deals with: (i) data gathering from heterogeneous sensor nodes and dispatching through sensor gateways; (ii) semantic annotation of collected data according to sensor ontologies; (iii) semantic data processing through rule engine to perform inference over semantically annotated sensor data; (iv) Web interface for sensor interaction. The same effort was done with constrained networks made up also of actuators. In [14] a framework is presented for actuator discovery and invocation in home care systems. By making use of an ontology to model services and operations of actuators, policy actions are made protocol-independent. This allows actions of a care policy to be specified abstractly using human-understandable concepts, hiding the low-level networking details from ordinary users. At run-time, the semantic discovery module searches for concrete actuator instances, so concrete action execution can take place. On a larger scale, [15] proposes a layered software architecture for delivering personalized services to residents in Self-Care Homes (SeCH). The architectural core layers grasp and understand the semantic of various situations that can be encountered in SeCH, through a variety of smart objects, which co-exist in pervasive environments. The decision making on appropriate actions is based on reasoning created by SWRL-enabled OWL ontologies, to ensure that residents are delivered suitable and personalized healthcare services in any situation. Concerning the application of semantic technologies within the building automation field, a good reference is [16], which presents a building automation system adopting SOA paradigm with devices implemented by Device Profile for Web Service (DPWS). Context information is collected, processed, and sent to a composition engine to coordinate appropriate devices/services based on context, composition plan, and predefined policy rules. A six-phased composition process carries out the task. In addition, in order to support the composition process, a building ontology is provided as a schema for representing semantic data, while a composition plan description language (CPDL) is used to describe context-based composite services. Focusing more on the visual definition of policy rules by end-users and their interaction with system elements, in [17] authors propose a visual user-centric service environment that helps users to create context aware services. It consists of a service mash-up process and a web-based service composition user interface. Through this environment and a step-by-step wizard, users define the service environment, context and behavior of actuators, indicating only actuator's function and service condition. In addition, an ontology based sensor data processing is proposed that removes heterogeneity of sensor data and processes the object data to the context information. A functional comparison between these works and the proposed solution is carried out in Section V.A.

III. DOMAIN MODEL

A. *High-level Context Modeling*

As stated in the Introduction Section, the main goal of the proposed architecture is to simplify the definition and implementation of building automation applications by common end-users. It can be achieved, firstly, by giving the applications a simple and well-defined structure, like the ECA rule pattern. When an event occurs, i.e. a fact with a particular meaning in the context of interest happens, if some conditions are verified, then one or more actions can be performed in response to the detected event.

Moreover, this model is much simpler and more effective the more event, condition and action components are abstract and close to the user. In smart home environments, people are generally interested in monitoring and controlling particular *properties* of some *entities* in a given *location*. Entities and related properties can be material or immaterial, such as temperature of the air, moisture of the soil, presence of a person, occurrence of rain, and so on. These are correlated with a location, which typically has (but is not limited to) an indoor scope. According to these considerations, we have defined in [10] the concept of (i) *System*, as the triple *Property-Entity-Location*, (ii) *Sub-System*, i.e. allowable System (e.g. temperature-air-room), and (iii) *Sub-Systems instances*, i.e. combination of concrete elements (e.g. temp-air-kitchen). Then, to each *System* (and each *Sub-System* for extension) it is associated a set of *High-Level States*, which can be changed by performing some *High-Level Actions*. Therefore, when state change occurs, user can select the desired *High-Level Action* and the desired *High-Level State*.

With this model in mind, the concept of *System High-Level State* change can be used as a trigger for the *EVENT* part of the ECA Rule and *System High-Level Action* as the corresponding *ACTION* part. *CONDITION* item can be the *High-Level* (current) *State* of any related *System*. For example, the rule expressed in human language “When the air temperature in the living room becomes hot, if Frank is in the room, then decrease the temperature to a normal value” can be mapped to the following ECA Rule (pseudo-notation):

```
ON temperature-air-living_room(hot)
IF presence-Frank-living_room(present)
THEN temperature-air-living_room.decrease(normal)
```

High-Level Actions can be classified according to the duration of their intervention. For example, increasing the luminosity of a room is an immediate action and its effect is stable in time; increasing room temperature, instead, requires some time before the desired result is got, hence actuators must be on for a certain period of time. When the controlled temperature exceeds the target threshold, all actuators must be switched off, otherwise it would raise indefinitely. The first example is a typical case of an “On/Off” control action, while the second is a kind of “Single Threshold” control action and both are included in the context model (with other types of action). In the last case, it is framework responsibility to create an ECA Rule that de-activates all actuators involved in an High-Level Action. Such a rule is named *De-Activate Rule* and the desired state of original ACTION becomes its trigger state, while its ACTION is the logical negation of the original action (i.e. *stopIncrease*, *stopDecrease*, etc.). For this purpose, a *HLService* entity must be modeled that keeps track of actuators activation within High-Level Actions.

B. *Low-level Action Modeling*

High-Level Actions labels must be translated into runnable applications in order to be executed. These applications are intended for hiding from end-user all complexity related to low-level details of action implementation. Instead, all these details are addressed by domain experts, who are in charge of creating action implementations exploiting context-aware information of the *High-Level Context Model*. Although it is not easy to define a common application structure, some common phases can be identified. For example, in a *DecreaseTemp* Action of a temperature-air-room *Sub-System*, the Action code, firstly, must retrieve the numerical value of the current High-Level State; then it must retrieve numerical value of desired High-Level State threshold, so the magnitude of the intervention can be calculated. As a mere example, if the difference between current and desired values of temperature is little, then it can be decided to open the window in the room; if the difference is a little bit greater, then the fan can be activated, or, if the difference is substantial, the air conditioner must be switched on. According to this business logic, a resource discovery phase is started, in order to get the URIs of devices that can execute the required service, and then proper commands can be issued to physical devices. Resource discovery and interaction are done by exploiting APIs provided by the framework (as explained in Section IV). Action codes are stored in a central repository, so they can be called by the proper module anytime are needed.

C. *Semantic Model and Domain Ontologies*

The *High-Level Context Model* must be implemented in order to share the knowledge contained in it among all framework components. Moreover, an execution environment for the modeled ECA Rules is needed. Both requirements can be fulfilled by means of semantic technologies. All the previously mentioned concepts can be seen as classes and individuals of an ontology [17]. Ontologies are used to formally name and define the types, properties, and interrelationships of the entities of a particular domain of discourse, in a machine-readable and understandable format. In this way, with the help of a semantic reasoner, it is possible to infer new information from data contained in the starting ontology. Furthermore, ECA Rules can be directly mapped to simple semantic rules written with a semantic rule language (like SRWL), so the semantic reasoner itself can execute them. By doing so, during rule execution all the information contained in this *Semantic Model* can be exploited and rule effects can be stored back in it to add new knowledge. Semantic rules are also used during the creation of the individual: when user inserts the *Property-Entity-Location* triple during the ECA Rule definition, this information is used to create an individual of the class *System*, which then is classified to infer its right *Sub-System* class; once the *Sub-System* is identified, available *High-Level States* and *High-Level Actions* are linked to the individual by running the so-called *Structural Rules* (Fig. 1). These rules represent structural knowledge for the *Semantic Model* and are created at design time by domain modelers.

For the sake of space and readability, the structure of Domain Ontologies is not deeply explained in this document, but this will be the focus of upcoming manuscripts. However, the core blocks are illustrated in Fig. 2, which shows some excerpts of the Domain Ontologies modeling the concept of *System*, *Sub-System* and their *Property-Entity-Location* components, *High-Level States*, *High-Level Actions*, *Resources* and their *Functionalities*. Whereas the picture does not show the *User* and *Time* entities, which also have an important role in the model. Fig. 3 shows an example of *Sub-System* individual (pseudo-notation), while Fig. 4 depicts ECA Rule components (pseudo-notation) mapped to a semantic rule (SWRL language). The *hasCurrentState* and *isFired* properties of Sub-System and Action individuals, respectively, play an important role in that they trigger rule evaluation and action execution (as explained in next Section).

IV. PROPOSED ARCHITECTURE

The structure of the proposed Rule-based Semantic Architecture is depicted in Fig. 5. Its main characteristic is the decoupling of the semantic layer from the actual Action execution layer. Once modeled all the entities and relationships of the context domain through ontologies, including physical resources functionalities and states, as well as ECA Rule structure and components, the execution of an ECA Rule at the semantic level implies only the change of the *isFired* property of the ACTION individual in the related ontology. A proper ontology listener observes this property and, on value change, triggers the actual action execution by calling the appropriate module and passing the action individual to it. Moreover, ECA Rule execution layer is independent from the semantic technologies chosen for its implementation, so different reasoners, rule languages and semantic framework can be used to create a working instance of the architecture. The main building blocks of each layer in the architecture are illustrated in the following sub-sections.

A. *GUI Layer*

This three-component layer represents the interface between end-user and framework. The *Rule Editor* allows user to visually create applications by selecting which *Systems* belong to *EVENT*, *CONDITION*, and *ACTION* part of the *ECA Rule*. This can be done by choosing, within a wizard, the *Property-Entity-Location* individuals triple to identify the *Sub-System* of interest, and then by choosing the related *High-Level State* or *High-Level Action* as needed. The *Dashboard Manager*,

```
Rule1: TempAirRoom(?s) -> hasAvailableAction(?s, decTempAirRoom),
hasAvailableAction(?s, incTempAirRoom), hasAvailableState(?s,
high), hasAvailableState(?s, low), hasAvailableState(?s, normal)
```

Fig. 1. Example of a Structural Rule

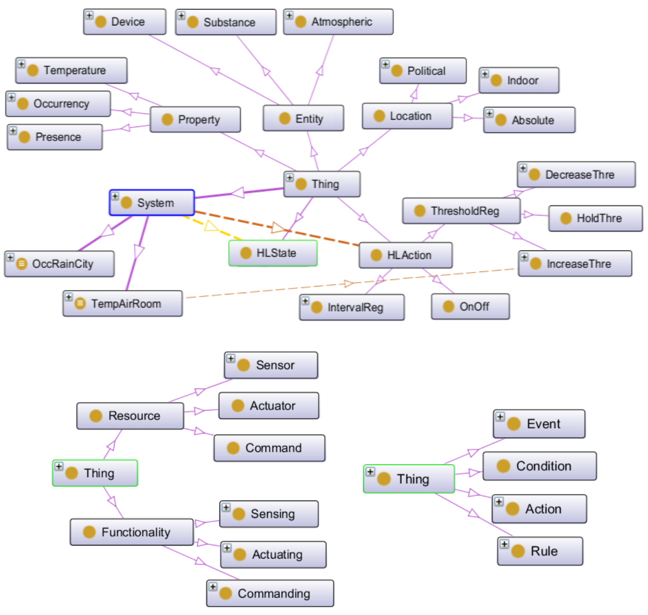

Fig. 2. Excerpts of the Domain Ontologies

instead, allows user to create simple dashboards for monitoring and controlling *System* related information. This can be high-level information, like High-Level State of the System, as well as low-level information, like sensor readings or actuator states related to the System. Finally, the *Framework Manager* allows user to configure various aspects of the Semantic Framework, like the environment layout, the type and position of devices deployed in it, and the numeric range of the High-level State labels of each System.

B. *Semantic Execution Layer*

This is the core layer of the whole architecture, since it deals with the management of the Semantic Model and semantic rule execution. Its first task, accomplished by the RULE MAPPER Module, is to take the ECA Rule components, visually defined by the user, and create the proper individuals of the Domain Ontologies. At the same time, a valid executable rule is created, using the semantic rule language set at configuration time (e.g. SWRL). Finally, the rule is stored in the Rule Repository. Any interaction with ontologies and repositories is managed by the KNOWLEDGE BASE UPDATER (KBU), which provides an interface for creating, updating and removal of individuals in the knowledge base. Then the OBSERVER MANAGER (OM) Module implements the Observer pattern and takes care of adding the RULE individual as an observer of the *hasCurrentState* property of the *Sub-System* individual indicated in the EVENT part, so that any state change will trigger semantic rule evaluation. Also the *isFired* property of

```
Sub-System: TempAirRoom (TAR)
sys1->      sysHasProperty: temperature
            sysHasEntity: air
            sysHasLocation: living_room
            hasAvailableState: TAR_low, TAR_normal, TAR_high
            hasCurrentState: TAR_high
            hasAvailableAction: TAR_increase, TAR_decrease,
            TAR_stopIncr, TAR_stopDec
```

Fig. 3. Sub-System individual example

```
Rule                          Event
R1 ->      hasEvent: E1       E1 ->     hasESystem: sys1
           hasCondition: C1             hasETriggerState:
           hasAction: A1                             TAR_high

Condition                     Action
C1 ->      hasCSystem: sys5   A1 ->     hasASystem: sys1
           hasCTriggerState:            hasADesiredState:
                     PPL_yes                       TAR_normal
                                        hasADesiredHLAction:
                                                 TAR_Decrease
                                        isFired: false
SWRL Rule1
hasETriggerState(?E1,TAR_high), hasCTriggerState(?C1,PPL_yes) ->
isFired(?A1,true)
```

Fig. 4. ECA Rule individuals and related SWRL semantic rule

the ACTION individual is added to a list of observable objects, which will trigger an eventual action execution after rule execution. The RULE EXECUTOR (RE) Module, in collaboration with the SEMANTIC REASONER (SR) Module, deals with ECA Rule execution. The SR contains the whole domain model implemented in a semantic format (OWL/RDF), so all the knowledge can be used to make inference. The DE-ACTIVATE RULE COMPOSER (DRC) Module deals with the creation of the De-Activate Rules, as explained in Section III.A. All their components are created as individuals of the related ontologies and registered as proper observers. Finally, the QUERY MANAGER (QM) Module provides an interface for querying domain ontologies and repositories.

C. *Action Execution Layer*

This layer implements all the mechanisms needed for translating High-Level Actions into a series of commands destined to physical actuator devices. The ACTION EXECUTOR (AEx) Module, when called by the OM, retrieves from the Action Repository the business code associated with the Action and gives it in input to the ACTION ENGINE (AEn). This module executes each instruction and, when needed, it queries the knowledge base (through the QM) in order to discover which resources offer

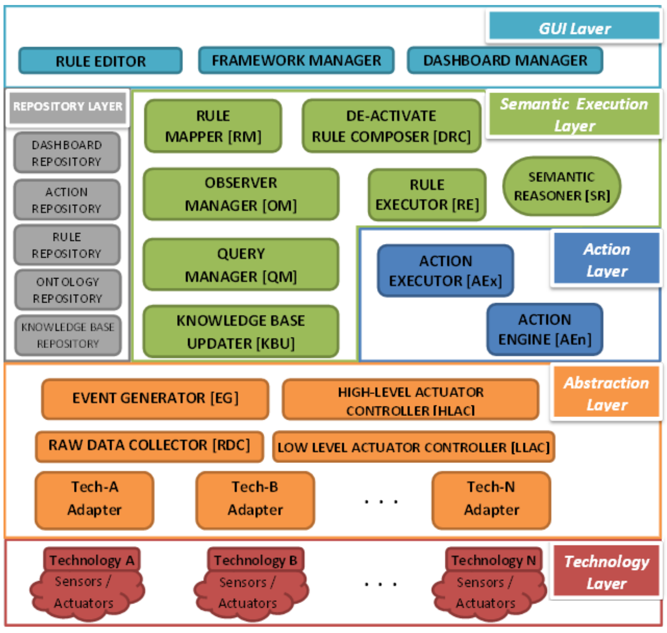

Fig. 5. Framework Architecture

the required functionalities. For each bounded device, it passes the device URI and the command string to the HLAC Module, contained in the Abstraction Layer.

D. *Repository Layer*

This layer contains several repositories for storing ontologies and their individuals (the so-called knowledge base), the Rules in semantic format (that must be included in the knowledge base only when they have to be executed), the Action business codes and the Dashboard templates.

E. *Abstraction Layer*

This layer acts as a middleware between the Technology Layer and the Semantic Layer, in that it semantically annotates data coming from physical devices and stores it in the knowledge base. The same applies in the opposite way, when high-level operations on actuators are translated into low-level commands according to the given technology. For each technology a TECH-X ADAPTER Module is provided, which is in charge of collecting data from physical devices, arrange them in an intermediate format and send them to the RAW DATA COLLECTOR (RDC) Module. The RDC Module, firstly semantically annotates data and stores them into the knowledge base via the KBU, then it passes these semantic data to the EVENT GENERATOR (EG) Module, which accomplishes the task of comparing numerical data with predefined thresholds in order to classify the *High-Level State*. If a state change is detected, an *Event* is triggered to the upper layer of the architecture. On the other side, messages coming from the Action Execution Layer, containing high-level commands and directed to physical actuators, are firstly handled by HIGH-LEVEL ACTUATOR CONTROLLER (HLAC) Module. It is in charge of reporting the intended operation in the knowledge base by creating a new individual of the class *HLService*, indicating which device has been activated, with which command and under which High-Level Action. After that, flow control is taken by the LOW LEVEL ACTUATOR CONTROLLER (LLAC) that, starting from device URI, identifies its native technology and prepares a proper message format according to it.

F. *Technology Layer*

This layer encompasses the main technologies and protocols used for managing devices in building automation applications. These can be roughly divided into three main categories: (i) wired and wireless standards for home automation networks, (ii) protocols for WSNs management, and (iii) indoor positioning systems. All these technologies are completely unaware of the upper layers, but, exploiting the proper TECH-X ADAPTER provided by the Abstraction Layer, they are simply seen as data sources and sinks, and they can seamlessly interoperate together.

G. *Runtime Lifecycle*

This section briefly explains how the whole architecture works. Firstly, user visually sets up the system through the Framework Manager, where s/he defines building layout, device deploying and threshold intervals. Then, end-user applications (i.e. ECA Rules) are defined through the Rule Editor, as explained in Section IV.A. In the background, a mapping algorithm creates executable semantic rules and the De-Activate Rule components (if needed). Then all individuals are stored in the related ontologies. This way the system is initialized, and can start its runtime life-cycle, as illustrated in Fig. 6. Raw data coming from heterogeneous devices, including sensor readings and actuator states, are collected by the related TECH-X ADAPTERs and sent to the RAW DATA COLLECTOR (a). Here, data are semantically annotated and stored into the knowledge base as current state value of the resource associated to the physical device. Since each resource is linked to a *Sub-System* by means of Property-Entity-Location triple, this information is used to retrieve the *High-Level State* threshold definitions, so the current reading can be classified (b). If this comparison results in a *High-Level State* change, the *Sub-System hasCurrentState* property is updated in the knowledge base. This is detected as an *Event* at the Semantic Execution Layer (c), so the OBSERVER MANAGER scans the list of Rules interested in this particular state change and, eventually, Rule individuals are sent to the RULE EXECUTOR. This module retrieves the Rule semantic code and gives it in input to the SEMANTIC REASONER. It evaluates the Rule according to information currently contained in the knowledge base. If all conditions in the rule antecedent[1] part are met, then its consequent part can be executed, so the *isFired* property of Action individual can be set to true. This means that current rule has been successfully evaluated and the related Action can be executed (d). After semantic rule evaluation, the rule code is removed from the knowledge base, otherwise it would be executed after every event detection. If any condition in the antecedent part is unmet, then the consequent part will not be executed and no action execution will be fired. When the *isFired* property of the Rule Action is updated, the OM selects the Action individual and passes it to the ACTION EXECUTOR. It retrieves the business code of the Action from the repository and starts its execution routine through the ACTION ENGINE (e). This module queries the knowledge base through the QUERY MANAGER to identify the URIs of devices implementing the required service (f). Once URIs are identified, they are sent to the HLAC, along with the command strings. This component reports the required operation in the knowledge base by creating an individual of the *HLService* class, then formats the message and sends it to

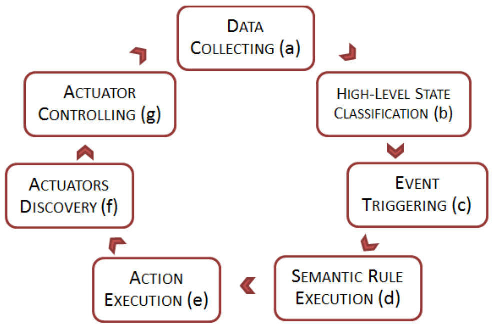

Fig. 6. Framework Runtime Lifecycle

[1] Semantic rules generally have the form: $A_1 \wedge A_2 \wedge \ldots \wedge A_n \rightarrow H$, where the left–hand side is also called *antecedent* or *body*, while the right-hand side is called *consequent* or *head*

the LLAC. This module, starting from device URI, identifies its native technology and accordingly formats the message destined to it (g). This pattern is repeated until all operations in the Action business code are executed. At this point, the whole Action is considered as fully executed, and all actuator state changes are reported back in the knowledge base as an update notification. When the desired *High-Level State* of each Sub-System involved in the Action part of the ECA Rule is reached, the related *De-Activate Rule* (if any) must be executed, in order to set Rule state as Stopped. In the meanwhile, the current ECA Rule must be considered in a Running state. The *De-Activate Rule* execution follows the same pattern of a common ECA Rule, except for the fact that it does not imply another De-Activate cycle. In any moment, all information related to each Sub-System can be monitored and controlled through a visual dashboard, which contains a set of widgets showing System High-Level State and associated sensor values, all Rules related to the System (both active and inactive), the Actions being executed and involved actuators state. Values displayed within these widgets are gathered from the knowledge base by the OM, which listens for resource state changes. Commands fired by GUI widgets get directly to the HLAC and follow the common execution path.

V. DISCUSSION

In this section, the proposed architecture is discussed by comparing it with similar solutions in the State-of-the-Art and by proposing a reference framework that can be used for its implementation.

A. *Architecture comparison*

The rule-based semantic architecture described in this paper presents the following core features:

- ECA Rules abstraction: explicit use of the ECA pattern as application template;
- High-Level abstraction: representation of context information and actions with a format close to user, hiding all low-level details;
- Semantic modeling: application of semantic technologies for context modeling;
- Device heterogeneity: abstraction of device functionalities, independently from their native technologies;
- Visual interface: provision of a graphic user interface for system setup and application creation and control;
- DSL compliance: possibility to define a domain specific language to facilitate action implementation.

The analysis of the architectures reported in Section II with respect to these functionalities is shown in Table I. ECA rules are treated only in [11], [12] and [14], while [15] uses normal semantic rules in SWRL language. In the proposed work, instead, ECA rules are the core of the entire architecture, because their well-defined and meaningful structure helps end-users in the application definition process. Moreover, also a context abstraction based on familiar concepts, can further facilitate users during applications creation. High-level abstraction is addressed by all the analyzed works, though at different levels. Although most solutions abstract raw sensor data to extract useful information for end-users, in the proposed architecture the abstraction is done at System level.

TABLE I
ARCHITECTURE FEATURES COMPARISON

	[11]	[12]	[13]	[14]	[15]	[16]	[17]	Proposed
ECA Rules	✓	✓	-	✓	-	-	-	✓
High-Level Abstraction	✓	✓	✓	✓	✓	✓	✓	✓
Semantic Modeling	-	-	✓	✓	✓	✓	✓	✓
Device Heterogeneity	✓	✓	✓	✓	✓	✓	✓	✓
Visual Interface	-	-	✓	✓	-	✓	✓	✓
DSL Compliance	-	-	-	-	-	✓	-	✓

The state of the considered System can be affected by a set of available actions that involves physical or virtual devices. This abstraction is used by the user to compose the elements of the application, by selecting the items of interest among ones proposed through a visual wizard. An interactive GUI can improve user experience, and has a central role in [13] and [17], whereas in [14] and [16] is a system functionality. From a technological point of view, semantic technologies are increasingly being used in such architectures, and they are applied in [13-17]. The main difference is that, in the proposed architecture, also ECA rules are executed by traditional semantic reasoner, exploiting the context modeling. Device heterogeneity is addressed by all works, though [13] only focuses on WSN devices and [16] only considers DPWS compliant devices. Finally, a domain specific language to compose applications is provided only in [16]. Taking into account this comparison, it is clear that, even if other scientific works already address some of the issues faced by the proposed system, none of them is able to provide a complete solution that efficiently solves all these issues.

B. Towards implementation

This paper focuses mainly on the context modeling and on the logical design of the proposed architecture, which is independent from any real technology; however, at the time of writing, its implementation is in an early stage. The aim is to prove the overall feasibility of the framework by mapping all of its logical blocks to the structure of the oneM2M project, described in the following paragraphs. Moreover, since Domain Ontologies are a central block of the architecture, this sub-section firstly proposes a scouting about already existing suitable ontologies that can be linked to the Semantic Model, in order to augment its expressivity and to foster knowledge sharing.

B.1 Ontology enhancement

The Semantic Execution Layer focuses on semantic model management; in particular, it is in charge of receiving both ECA rules from upper layers and event notifications from lower layers, and to process them in order to achieve the desired behavior. To do so, it has to interact with the Domain Ontologies to (i) create the proper individuals, (ii) infer new information from the starting knowledge, and (iii) store back in it new knowledge arising from ECA rules execution. So, it is clear that the structure of the Domain Ontologies is critical for the effectiveness of the Semantic Execution Layer, thus the possibility to rely on standard ontologies is an important aspect of the proposal. Among all the entities of the architecture, the User, the Location, the Time and the Physical Devices are the most likely to have well-defined and shared ontologies, due to their common and widely-used concepts. The User has the capability to define the desired actions through ECA rules, the Location and Time are essential to identify the occurrence of events, which are the triggers of the ECA rules, whereas the Physical Devices are the interface between virtual and real world. Regarding the User, the FOAF ontology [18] seems to be a significant candidate. It is a machine-readable ontology describing persons, their activities and their relations to other people and objects. In more detail, it can be seen as a descriptive vocabulary expressed through the Resource Description Framework (RDF) and the Web Ontology Language (OWL). About the Time concept, several ontologies could be suitable for the proposed architecture. The OWL-Time [19] provides a vocabulary for expressing facts about topological relations among instants and intervals, together with information about durations, and about date/time information. Moreover, the Timeline Ontology [20] defines the TimeLine concept, representing a coherent backbone for addressing temporal information. Each temporal object (signal, video, performance, work, etc.) can be associated to such a timeline. As an alternative, the SWRLTemporalOntology [21] defines a temporal model that can be used to model complex interval-based temporal information in OWL ontologies. It also defines a library of SWRL built-ins to perform temporal operations on information described using this ontology. An introduction to this ontology and an explanation of how can be used to perform temporal reasoning is covered in [22]. The model defined by SWRLTemporalOntology is based on the valid-time temporal model [23], a commonly-used model to represent temporal information in many systems. The valid-time temporal model provides an approach for consistently representing temporal information. In this model, a piece of information, which is often referred to as a fact or a proposition, can be associated with instants or intervals denoting the time or times that the fact is held to be true. Such facts have a value and one or more valid times. In other words, every temporal fact holds information denoting the valid-time of the facts. Conceptually, this representation means that every temporal fact is held to be true or valid during the time or times associated with it. No conclusions can be made about the fact for time periods outside of its valid-time. Going ahead with the Location component, it is worth noting that it is not simple to find a universally adopted standard ontology. Even if there are several research works aimed at providing location-based solutions dedicated to smart environments, they often define their own location ontologies. Furthermore, these works usually focus on a wider location ontology concept, namely a context ontology, which is not limited to places' description but to define all aspects characterizing the surrounding environment [24]. Finally, regarding the networks of Physical Devices, a reference ontology could be the oneM2M Base Ontology draft [25]. It aims at providing syntactic and semantic interoperability to the oneM2M Project (described in more detail in the following paragraph) with external systems. In particular, external organizations and companies are expected to contribute their own ontologies that can be mapped to the oneM2M Base Ontology. These external ontologies might describe specific types of devices or, more generally, they might describe real-world “things” that should be represented in an oneM2M implementation. In this way, oneM2M data can be enhanced with information on the meaning/purpose of these data. Therefore, the oneM2M Base Ontology is the minimal ontology that is required such that other ontologies can be mapped into oneM2M. About Physical Devices, it is worth considering also the Semantic Sensor Network (SSN) Ontology [26]. This ontology is based around the concepts of systems, processes, and observations and it supports the description of the physical and processing structure of sensors. Sensors are not constrained to physical sensing devices, rather a sensor is anything that can estimate or calculate the value of a phenomenon; so a device or computational process or combination could play the role of a sensor. The representation of a sensor in the ontology links together what it measures (the domain phenomena), the physical sensor (the device) and its functions and processing (the models).

B.2 The oneM2M project

The goal of oneM2M is to develop technical specifications aimed to create a common M2M Service Layer, which can be readily embedded within various hardware and software, in order to connect heterogeneous devices with M2M application servers worldwide. The oneM2M system architecture provides both basic functionalities (e.g., registration and message handling) and various advanced functionalities (e.g., interconnections with other systems). To achieve this, oneM2M defines a common service layer providing M2M services, independently from the underlying networks. The latest oneM2M functional architecture [27] is shown in Fig. 7. An oneM2M system is composed of functional entities, called nodes, which can be application dedicated node (ADN), application service node (ASN), middle node (MN) and infrastructure node (IN). Nodes consist of at least one oneM2M Common Services Entity (CSE) or one oneM2M Application Entity (AE). A CSE is a logical entity that is instantiated in an M2M node and comprises a set of service functions called Common Services Functions (CSFs), which can be used by applications and other CSEs. An AE is a logical entity that provides application logic for end-to-end M2M solutions. oneM2M currently defines three reference points, namely Mca, Mcc, and Mcn. The Mca reference point enables AEs to exploit the services provided by the CSE,

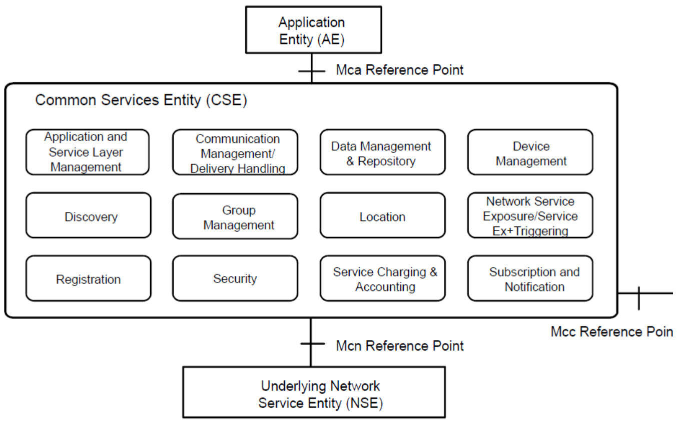
Application Entity (AE)
Mca Reference Point
Common Services Entity (CSE)
Application and Service Layer Management
Communication Management/ Delivery Handling
Data Management & Repository
Device Management
Discovery
Group Management
Location
Network Service Exposure/Service Ex+Triggering
Registration
Security
Service Charging & Accounting
Subscription and Notification
Mcc Reference Poin
Mcn Reference Point
Underlying Network Service Entity (NSE)

Fig. 7. oneM2M functional architecture

whereas the Mcc reference point enables inter-CSE communications. The Mcc’ reference is similar to Mcc, but provides an interface to another oneM2M system. The Mcn reference point is between a CSE and the service entities in the underlying networks. oneM2M has specified a set of core CSFs for its initial release. Some CSFs provide administrative functions for the service layer and other CSFs, like the Registration (REG) CSF that provides a means for an AE or a CSE to register to a CSE and be able to use the services provided by that CSE. An AE and a Service Layer Management (ASM) CSF provide functions to configure, troubleshoot, and upgrade CSEs and AEs. A Device Management (DMG) CSF manages device capabilities such as firmware updates. The Communication Management and Delivery Handling (CMDH) CSF, instead, is responsible for the delivery of service layer messages. The Network Service Exposure (NSSE) CSF acts as the anchor point between the service layer and services provided by different underlying networks. The Security (SEC) CSF enables secure connections and data privacy. Some CSFs provide value-added services to registered AEs and CSEs. For example, the Data Management and Repository (DMR) CSF is responsible for user data storage and processing, so users can also subscribe and get notifications of changes in the data. The Discovery (DIS) CSF provides a means to make the services and resources discoverable by other CSEs and AEs. A Subscription and Notification (SUB) CSF manages subscriptions to changes on the oneM2M platform. The Service Session Management (SSM) CSF supports end-to-end service layer sessions. A Group Management (GMG) CSF supports bulk operations and manages group membership. The Location (LOC) CSF allows M2M AEs to obtain geographic location information of an entity and receive location-based services. The Service Charging and Accounting (SCA) CSF provides mechanisms to support service-layer-based charging. In addition to CSFs, a CSE includes a service enabler to ensure the extensibility of services.

B.3 Mapping with the oneM2M architecture

All the functionalities introduced in the proposed architecture (Section IV) can be mapped to the components of the oneM2M architecture. For the sake of clarity, let suppose a testing scenario in which a smart building is equipped with a wired building automation system (for example based on the KNX standard), a WSN for ambient data sensing and a BLE-based indoor positioning system that interacts with user’s smartphone to detect his/her position. All these systems communicate with a local smart gateway to make their data available to the global framework. The semantic framework runs in a proper container, which can be deployed on the smart gateway, on a dedicated local server, or in the cloud. Comparing this scenario with the proposed architecture in Fig. 5 and the functional architecture in Fig. 7, the functionalities of the local gateway and the TECH-X Adapter for each low-level technology can be mapped to the NSE; in fact, it provides services from the underlying networks that can be accessed through the NSSE CSF of the CSE. As usual, physical devices remains completely unaware of the upper layers, but they can be still managed through the DMG CSF. Since the DMR CSF offers data storage and mediation functions in order to store big amounts of data for analytics and semantic processing, the entire Repository Layer can be mapped to the DMR CSF, which can handle all needed repositories. It represents a low-level interface that can be used to implement advanced functionalities related to data. In the oneM2M architecture, the role of part of the Abstraction Layer and all of the original Semantic and Action Layers can be seen as a new CSF, called Semantic Rule and Action Engine (SRAE), which offers services aimed at the execution of semantic rules and low-level actions through the interpretation of the provided DSL. In particular, the SRAE interacts with the SUB CSF to carry out the concerns of the OBSERVER MANAGER and with the DMR to implement the QUERY MANAGER and the KNOWLEDGE BASE UPDATER. Moreover, the SRAE interacts with the DMR also to implement the functionalities of RAW DATA COLLECTOR, EVENT GENERATOR, HIGH-LEVEL and LOW-LEVEL ACTUATOR CONTROLLERS, which are specialized services built upon DMR services. By doing so, the SRAE can handle all the needed semantic data. The remaining modules of the semantic layer, instead, become inner blocks of the SRAE. The same applies to the ACTION EXECUTOR and the ACTION ENGINE modules, which become an autonomous block within the SRAE and interacts mainly with the DIS CSF for service discovery and the GMG CSF for group commands. Finally, all modules of the GUI Layer become AEs, in that they implement pure application business logic. Obviously, AEs register through the REG CSF in order to access all the services provided by the CSE, especially the ones of the ASM, CMDH and SEC CSFs.

B.4 OSGi-based implementation design

In order to build IoT-based applications, OSGi [28] provides an excellent programming model based on modern principles of modularity and service-oriented computing. In addition, the OSGi lifecycle operations provide a high level of dynamism about application deploying, managing and updating. OSGi builds an abstraction layer over plain-Java, defining a software infrastructure by means of which modern patterns like code-injection, aspects, discovery, messaging, event bindings are easily integrated. OSGi can be a good choice in order to implement an oneM2M specification, since specialized distributions, like Apache ServiceMix [29], provide support for messaging, routing, integration patterns, remote services, transaction management and, in general, all that needs to implement CSEs functionalities. At one time, more compact distributions of OSGi can be deployed on constrained devices, like the TECH-X Adapters running on NSEs in our scenario. In an OSGi context a TECH-X Adapter can be packed as "bundle" and easily deployed on a NSE making use of a remote deployment tool like Apache ACE [30] which is all about provisioning software artifacts to OSGi (and not only) target systems. A TECH-X Adapter, implementing a typical OSGi "extender pattern" [31], declaratively exposes meta-data to the discovery services and then makes the connected devices discoverable. An OSGi container permits bundles to be hot deployed/undeployed and updated within a running system without restarting the application or the JVM, making the system very reactive. Once started, the local NSSE CSF holds the connection with the adapter and then with devices behind the adapter that are registered in the oneM2M system as "resources". In an oneM2M context, our Semantic Rule and Action Engine (SRAE) CSF can discover and subscribe a device via NSSE CSF, then receiving notification about its state changes (sensor) or send actuation commands to it (actuator) via SUB CSF.

VI. Conclusion

In this work, the design of a rule-based semantic architecture has been proposed with the aim to help common end-users in defining their building automation applications. This architecture addresses the main issues involved in application management in an Internet of Things context, such as: (i) application and context modeling, by means of *ECA Rule*, *System*, *High-Level States* and *High-Level Actions* abstractions, (ii) application creation and control, through the introduction of a visual IDE, (iii) execution environment definition, by means of a Semantic Framework and an Action Engine, (iv) physical device management, by introducing a low-level layer that abstracts device heterogeneity. Moreover, a reference framework has been proposed in order to implement each layer of the architecture, on the basis of a set of novel standard specifications, namely the oneM2M project. These specifications, aiming to foster autonomous communications in a M2M context, can be easily implemented by means of the OSGi framework. Further developments are being carried out to improve overall architecture effectiveness, especially concerning the Semantic Model and Domain Ontologies. Finally, the last step will be to setup a real test bed and assess system performances.

References

[1] L. Mainetti, V. Mighali, S.L. Oliva, L. Patrono, P. Rametta: *A novel architecture enabling the visual implementation of web of Things applications*, The 21st Int. Conf. on Software, Telecommunications and Computer Networks, SoftCOM 2013, Split (Croatia), Sept. 18-20, 2013

[2] L. Mainetti, V. Mighali, L. Patrono, P. Rametta: *Discovery and Mash-up of Physical Resources through a Web of Things Architecture*, Journal of Communications Software and Systems, vol. 10, no. 2, pp.124-134, June 2014

[3] L. Anchora, A. Capone, V. Mighali, L. Patrono, F. Simone: *A novel MAC scheduler to minimize the energy consumption in a Wireless Sensor Network*, Ad Hoc Networks, vol. 16, pp. 88-104, 2014.

[4] L. Catarinucci, R. Colella, G. Del Fiore, L. Mainetti, V. Mighali, L. Patrono, M. L. Stefanizzi: *A cross-layer approach to minimize the energy consumption in wireless sensor networks*, International Journal of Distributed Sensor Networks, vol. 2014, ID 268284, 11 pages, 2014.

[5] L. Catarinucci, R. Colella, L. Tarricone: *Sensor data transmission through passive RFID tags to feed wireless sensor networks*, in 2010 IEEE MTT-S International Microwave Symposium, MTT 2010, Anaheim, CA, 2010, pp. 1772-1775.

[6] L. Mainetti, V. Mighali, L. Patrono: *An IoT-based User-centric Ecosystem for Heterogeneous Smart Home Environments*, in 2015 IEEE Int. Conf. on Communications, IEEE ICC 2015, London (UK), June 8-12, 2015

[7] L. Mainetti, V. Mighali, L. Patrono: *A Location-aware Architecture for Heterogeneous Building Automation Systems*, in 14th IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2015, Ottawa (Canada), May 11-15, 2015

[8] Drools Expert: https://docs.jboss.org/drools/release/5.5.0.Final/drools-expert-docs/html_single/

[9] Pellet Reasoner: https://github.com/complexible/pellet

[10] L. Mainetti, V. Mighali, L. Patrono, P. Rametta: *A novel Rule-based Semantic Architecture for IoT Building Automation Systems*, The 23rd Int. Conf. on Software, Telecommunications and Computer Networks, SoftCOM 2015, Split (Croatia), Sept. 16-18, 2015

[11] Chui Yew Leong, A.R. Ramli, T. Perumal: *A rule-based framework for heterogeneous subsystems management in smart home environment*, Consumer Electronics, IEEE Transactions on , vol.55, no.3, pp.1208,1213, August 2009.

[12] S.R. Bhandari, N.W. Bergmann: *An Internet-of-Things system architecture based on services and events*, Intelligent Sensors, Sensor Networks and Information Processing, 2013 IEEE 18th Int. Conf. on , vol., no., pp.339,344, 2-5 April 2013

[13] V. Huang, M.K. Javed: *Semantic Sensor Information Description and Processing*, Sensor Technologies and Applications, 2008. SENSORCOMM '08. 2nd Int. Conf. on , vol., no., pp.456,461, 25-31 Aug. 2008.

[14] Feng Wang, Kenneth J. Turner: *An Ontology-Based Actuator Discovery and Invocation Framework in Home Care Systems*, in Proc. 7th Int. Conf. on Smart Homes and Health Telematics, pp. 66-73, LNCS 5597, Springer, Berlin, June 2009

[15] R. Shojanoori, R. Juric, M. Lohi, G. Terstyanszky: *ASeCS: Assistive Self-Care Software Architectures for Delivering Service in Care Homes*, System Sciences (HICSS), 2014 47th Hawaii Int. Conf. on , vol., no., pp.2928,2937, 6-9 Jan. 2014

[16] S.N. Han, Gyu Myoung Lee, N. Crespi: *Semantic Context-Aware Service Composition for Building Automation System*, Industrial Informatics, IEEE Transactions on , vol.10, no.1, pp.752,761, Feb. 2014

[17] Hoan-Suk Choi, Jun-Young Lee, Na-Ri Yang, Woo-Seop Rhee: *User-centric service environment for context aware service mash-up*, Internet of Things (WF-IoT), 2014 IEEE World Forum on , vol., no., pp.388,393, 6-8 March 2014.

[18] Friend Of A Friend (FOAF) Vocabulary Specification: http://xmlns.com/foaf/spec/

[19] Time Ontology in OWL: http://www.w3.org/TR/owl-time/

[20] The Timeline Ontology: http://motools.sourceforge.net/timeline/timeline.html

[21] SWRL Temporal Ontology: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalOntology

[22] SWRL Temporal Built-ins: http://protege.cim3.net/cgi-bin/wiki.pl?SWRLTemporalBuiltIns

[23] Valid-Time Temporal Model: http://protege.cim3.net/cgi-bin/wiki.pl?ValidTimeTemporalModel

[24] I. Roussaki, M. Strimpakou, N. Kalatzis, M. Anagnostou, C. Pils: *Hybrid context modeling: a location-based scheme using ontologies*, in Pervasive Computing and Communications Workshops, 2006. PerCom Workshops 2006. Fourth Annual IEEE International Conference on , vol., no., pp.6 pp.-7, 13-17 March 2006

[25] oneM2M Base Ontology Draft: http://www.onem2m.org/technical/latest-drafts

[26] Semantic Sensor Network (SSN) Ontology: http://www.w3.org/2005/Incubator/ssn/ssnx/ssn

[27] oneM2M Specifications: http://www.onem2m.org/technical/published-documents.

[28] OSGi Alliance: http://www.osgi.org/Main/HomePage

[29] Apache ServiceMix: http://servicemix.apache.org/

[30] Apache ACE: https://ace.apache.org/

[31] OSGi Extender Pattern: http://blog.osgi.org/2007/02/osgi-extender-model.html

Paolo Lillo graduated in Electronic Engineering at University of Florence (Italy) in 1987. He is currently a PhD student in Complex Systems Engineering at University of Salento. His research topics are mainly focused on development of architectures based on OSGi, Internet of Things, design and development of model-based multi-platform UI by using Eclipse/Xtext/Xtend technology, developing software interfacing C/Java/Python layers and ARM boards via Bluetooth/BluetoothLE channels, development of multi-platform clients to control building automation systems compliant with KNX.

Luca Mainetti is an Associate Professor of Software Engineering and Computer Graphics at the University of Salento. His research interests include web design methodologies, notations and tools, services oriented architectures and IoT applications, and collaborative computer graphics. He is a scientific coordinator of the GSA Lab - Graphics and Software Architectures Lab and IDA Lab - IDentification Automation Lab at the Department of Innovation Engineering, University of Salento.

Vincenzo Mighali received the "Laurea" Degree in Computer Engineering with honors at the University of Salento, Lecce, Italy, in 2012. Since January 2009 he collaborates with IDA Lab — IDentification Automation Laboratory at the Department of Innovation Engineering, University of Salento. His activity is focused on the definition and implementation of new tracking system based on RFID technology and on the design and validation of innovative communication protocol aimed to reduce power consumption in Wireless Sensor Networks. He is also involved in the study of new solutions for building automation. He authored several papers on international journals and conferences.

Luigi Patrono received his MS in Computer Engineering from University of Lecce, Lecce, Italy, in 1999 and PhD in Innovative Materials and Technologies for Satellite Networks from ISUFI-University of Lecce, Lecce, Italy, in 2003. He is an Assistant Professor of Network Design at the University of Salento, Lecce, Italy. His research interests include RFID, EPCglobal, Internet of Things, Wireless Sensor Networks, and design and performance evaluation of protocols. He is Organizer Chair of the international Symposium on RFID Technologies and Internet of Things within the IEEE SoftCOM conference. He is author of about 100 scientific papers published on international journals and conferences

Piercosimo Rametta received the "Laurea" Degree in Computer Engineering with honors at the University of Salento, Lecce, Italy, in 2013. His thesis concerned the definition and implementation of a novel mash-up tool for Wireless Sensor Networks' configuration. Since November 2013 he collaborates with IDA Lab — IDentification Automation Laboratory at the Department of Innovation Engineering, University of Salento. His activity is focused on the definition and implementation of new mash-up tools for managing Internet of Things based smart environments by using semantic technologies