A Cloud Architecture for Managing IoT-aware Applications According to Knowledge Processing Rules

Luca Mainetti, Luigi Manco, Luigi Patrono, and Roberto Vergallo

***Abstract*— The Web of Things paradigm has represented a shift in the conjunction of the Internet of Things (IoT) with people, as it allows treating a smart object as a Web resource. While in a first phase the challenge was the physically management of smart objects, the current demand is to help users in profitably introducing IoT in their own daily life.**

The paper presents a software architecture for IoT systems able to manage the behaviour of involved IoT entities basing on knowledge processing tools. The main goal is informing the user of the occurrence of events of interest semantically determined starting from actual state of the environment. The architecture exploits the potentialities of the Web of Topics (WoX) approach, a conceptual model that simplifies the designing of IoT applications. Leveraging the WoX approach, the architecture introduces an innovative way to mine knowledge from IoT devices aside from any technological background, so that facing the intrinsic heterogeneity affecting IoT entities. The discussed architecture is composed by different modules integrated into an Enterprise Service Bus (ESB), strongly decoupled and provided with RESTful-compliant web interfaces to communicate each other and with the external environment, according to a SOA structure. The paper shows how the system is able to receive data coming from sensors and to semantically interpret them by means of a series of business rules that act as knowledge processor.

***Index Terms*—Cloud, Internet of Things, Architecture, RESTful, SOA, validation**

I. INTRODUCTION

INTERNET of Things represents a new era in Computer Science. It has led to a pervasive computing and, so, to a new way to think at computational systems. Nowadays, informatics shifts from personal computers to smart objects and it becomes more and more ubiquitous, with the aims of simplifying and automatising our daily life.

However, at the application layer, smart devices still form multiple and incompatible islands and, for this reason, developing applications using them is a challenging task that requires deep knowledge of each hardware platform, operating system and programming language. To overcome this problem, the research in the field of the Internet of Things (IoT) is aimed to interconnect uniquely identifiable embedded devices within the existing Internet infrastructure, in order to allow the development of services and applications independently of both specific embedded technology and programming language. In simple terms, the main goal of the IoT is to enable things to be connected anytime, anywhere, with anything and anyone. According to the IoT vision, the environment should be populated by several smart things that, through wireless and wired connections and unique addressing schemes, are able to interact and cooperate with each other to create new services and achieve common goals. So, IoT applications can be applied to a number of different scenarios and devices, resulting in integration and interoperability issues due to the heterogeneity of the involved technologies.

In this regard, a new vision inspired from the IoT and better focused on a simpler and global integration of heterogeneous smart objects is represented by the Web of Things (WoT), which sees the Web as the best candidate for a universal integration platform [1]. According to it, everyday devices are interconnected through their full integration into the Web. In this way, unlike in existing IoT systems, the smart devices are able to communicate with each other by using Web standards and technologies, which can be reused and adapted to build new applications and services. Well-accepted and understood standards and blueprints (such as URI, HTTP, REST, etc.) are used to access the functionality of the smart objects. In other words, the data produced by smart devices can be directly accessible as normal Web resources, so as providing the so-called physical mash-up [2][3][4].

The current work aims to extend the Web of Things approach: an integrated cloud platform, a middleware implementing the communication between IoT devices and end-user applications, able to apply logical reasoning on IoT row data in order to retrieve complex information relating to the environment in which IoT devices are immersed: the core of the platform is able to elaborate environment state basing on sensors data and to undertake semantically defined actions. For example, the platform could advice the user that some relative is experimenting a sudden, or it could control rolling shutter in a smart home mixing information about wind and temperature.

The proposed platform grounds on an Enterpise Servise Bus (ESB) architecture. It is composed of two main components: (i) a knowledge processor able to semantically extract business

Manuscript received October 30, 2015; revised February 27, 2016.

L. Mainetti, L. Manco, L. Patrono and R. Vergallo are with the Department of Innovation Engineering, University of Salento, Lecce, Italy (E-mails:{ luca.mainetti, luigi.manco, luigi.patrono, roberto.vergallo}@unisalento.it).

rules from sensors data and (ii) a module for interacting efficiently with IoT devices and user applications. The latter is an implementation of a novel model-driven designing approach for IoT applications: Web of Topics (WoX) model [5]. In WoX, the key concept is the Topic, that is a feature in a specific location. WoX features are not technological but informal argument of discussion in the Topic, such as temperature, crowd, a formula, i.e. any human-definable, measurable, perceivable and controllable entity of the environment. Information producers and consumers talk to each other via the Topic of interest. Behind the scene, each end-user application and IoT node declares a set of roles (i.e., sensor capability, sensor need, actuator capability, actuator need) for the topics of interest. For its intrinsic simplicity, the WoX model opens also the doors to physical IoT nodes not only providing but also requesting services to (heterogeneous) peers. The implementation of WoX model consists of an architecture based on the Fosstrak EPCglobal standard [6]. WoX permits to overcome the intrinsic drawback of the Web of Things approach: it provides a way to design every IoT entity in a conceptual model, abstracting it from whatever technological detail. So, it permits to tackle the heterogeneity affecting IoT devices and to design and create IoT applications in a simplest way. By means of such model-driven method, the IoT insiders can focus only on the functional aspects of IoT devices, without taking care of any technological aspects.

Leveraging the Topic concept, the presented study provides an innovative way to mine complex information from IoT devices aside from any technological background and to organize them in cloud. It is able to orchestrate data coming from heterogeneous IoT devices, to infer logical reasoning on them and to interact with users in respect of their needs.

The next Section examines the state of art in terms of architectures and platforms for managing IoT applications, paying specific attention to those supported by knowledge processing tools. Section III formally illustrates the WoX model. Section IV gives a detailed picture of the whole architecture, while in Section V the platform is validated by a practical running case. The last two Sections provide some considerations on the conducted study and the conclusions and future work.

II. Related Work

In the literature, the interest in design and development of WoT software architectures, using different approaches, is very deep. Several Web platforms are emerging with the aim to abstract the heterogeneity of the physical embedded devices in order to facilitate their integration and interoperability.

Most of the existing architectures belong to two main categories:

- Architectures based on international standards. They enjoy several benefits arising from compliance with international standards, but are very close to the physical layer. Therefore, using these architectures requires expert knowledge of physical technologies and significant familiarity with programming languages. Furthermore, they are generally focused on a specific use cases and are not horizontal enough to support the integration of heterogeneous technologies.
- Horizontal architectures, which are explicitly designed to integrate heterogeneous protocols and standards. Semantic architectures, for example, belong to this category. Generally, these architectures are not compliant with international standards, but have the considerable advantage of being closer to the developers, which are not required to know about the involved physical technologies nor specific programming languages.

Regarding the first category, in [7] the authors propose an IoT framework, based on the EPCglobal [8] architecture, which is able to integrate the transducer capability of IEEE 1451 standards [9]. As the original EPCglobal only supports C-1 Gen-2 RFID tag identification, the authors propose to extend the framework to support more readers, tags, and transducers in versatile IoT applications. EPCglobal Application Level Events (ALE) middleware is provided with transducer capability of IEEE 1451.

Regarding the second category, the Semantic Web of Things (SWoT) is an emerging vision in Information and Communication Technology (ICT), joining together the Semantic Web and the Internet of Things. Its goal is to associate semantically rich and easily accessible information to real-world objects, locations and events, by means of inexpensive, disposable and unobtrusive micro-devices, such as Radio Frequency IDentification (RFID) tags and wireless sensors [6].

A widely used tool for the realization of semantic architectures is Smart-M3 [7]. It is a content-based, semantic subscribe-notify, and open-source middleware able to provide a Semantic Web information-sharing infrastructure among software entities and devices. The main goals of Smart-M3 concern sharing interoperable information in smart environment applications and making information in the physical world available for smart services.

The authors of [10] describe their own vision a middleware for the Internet of Things, with the aims of creating a new generation middleware platform which will allow creation of self-managed complex systems, in particular industrial ones, consisting of distributed, heterogeneous, shared and reusable components of different nature. Grounding on semantic and Multi-Agent System technologies and methodologies, they analyse and design such middleware and demonstrate how it is possible to enable various components to automatically discover each other and to configure a system with complex functionality based on the atomic functionalities of the components.

Another interesting horizontal approach is presented in [11], where the authors propose a software architecture to easily mash-up CoAP resources. The architecture is able to discover the available devices and to virtualize them outside the physical network. These virtualizations are then exposed to the upper layers by a RESTful interface, so that the physical devices interact only with their own virtualization. The architecture is designed to establishing a bidirectional communication channel, allowing not only to monitor but also to control the devices. The achieved platform, also, provides simplified tools allowing the development of mash-up applications to different-skilled users.

Relating to Enterprise Service Bus architectures for IoT, the study in [12] proposes an architecture for an effective

integration of the Internet of Things in enterprise services: the architecture exposes real-world devices with embedded software to standard IT systems by making them accessible in a service-oriented way.

The author of [13] show a new IoT sensing service system based on EDSOA (Event Driven SOA) architecture to support real-time, event-driven, and active service execution. The study also provides a new IoT browser that uses augmented reality technology to display IoT resource, realising the superposition presentation of the physical world and abstract information.

Despite these interesting approaches, in literature there is a lack of integrated platforms that can provide a cloud access to IoT resources along with a semantic management of them. There is the need of IoT architecture that can manage smart devices with the support of knowledge processing tools and, in the same time, that allow end-users to interact with them as Web services.

The proposed architecture tries to satisfy both the requirements.

III. The Web of Topics Model

In this section we summarize briefly the Web of Topics (WoX) model concepts. The reader could deepen the WoX treatise in [5].

WoX refers to both IoT hardware nodes and IoT applications generically as IoT entities. In WoX, the 'sensor' and 'actuator' concepts do not exist. The main concept is the Topic. A WoX Topic is about a quantity of interest – called feature – in a certain location. More rigorously, a feature is any characteristic or entity of the environment that can be perceivable, definable, measurable and/or controllable. Some bare examples of feature are: temperature, humidity, presence. A crowd of people can be a feature too, as well as an alarm. Also a mathematical function – e.g. sum, min, max – can be a feature. The set of WoX features is the following:

$$F = \{ f_i \} \tag{1}$$

The location is expressed hierarchically following the URN (Uniform Resource Name) scheme, e.g. "urn:italy:salento:highschool:firstfloor:phylab:desk1" or "urn:usa:california:la:westwood:overlandavenue:2801". The set of WoX locations si the following:

$$L = \{ l_j \} \tag{2}$$

where L also includes two special locations:

1. LOC_ANY, i.e. a wildcard for any location;
2. LOC_SELF, i.e. a pointer to the current location.

Hence WoX define T as the set of couples feature-in-location:

$$T = \left\{ t_{i,j} = \left(f_i, l_j \right) \in F \times L \right\} \tag{3}$$

In WoX, the topic is the key between who asks for services and who provides services. Separating the twos, WoX reaches the maximum abstraction possible. In fact, IoT designers can focus on concepts (features) rather than on hardware.

An IoT entity will introduce itself to a topic in different ways, depending on its nature. In WoX, such nature is analyzed in two dimensions:

1. Collaborative dimension. It includes two aspects: (i) the capability to perform a service within the topic, and (ii) the need for other entities who can perform a service.
2. Technological dimension. It includes the legacy (i) sensor and (ii) actuator distinction, as well as a generic (iii) function service.

Hence WoX defines the following two dimensions sets:

$$D_C = \{ capability, need \} \tag{4}$$

$$D_{TECH} = \{ sensor, actuator, function \} \tag{5}$$

Then, WoX defines the set of WoX roles R included in the following Cartesian product:

$$R = D_C \times D_{TECH} = \{ SC, AC, FC, SN, AN, FN \} \tag{6}$$

The items in the curly brackets respectively state for: sensor capability, actuator capability, function capability, sensor need, actuator need, function need. The generic IoT entity *z* can be modeled as a set of couples role-topic belonging to the following Cartesian product:

$$E_z \subset \left\{ \left(r_k, t_{i,j} \right) \in R \times T \right\} \tag{7}$$

By this way, WoX is able to model any device or app in the IoT, even the most complex. An example of IoT node implementing the WoX model is a personal enhanced RFID tag which need to know the temperature in the current room, is capable to perform a comparison between float numbers and can activate a LED alarm. In the next section we will cover the evolution of the WoX model towards and ESB approach that will include the use of knowledge processors.

IV. The Enterprise Architecture

This section introduces the enterprise architecture in which the WoX approach can be enhanced by the use of knowledge processing tools. The Section is divided in three parts, since the architecture can be analysed from just as many perspectives. The first subsection describes the WoX implementation and its integration on the top of Fosstrak middleware. The second subsection shows how the architecture logically works, while in the last subsection it is shown the technical deployment of the architecture on an enterprise system.

A. The WoX Technical Architecture

The WoX model requires a robust ICT architecture capable of facing the extremely high numbers of IoT entities and Topics, the intense exchange of messages and the heterogeneity of the IoT technologies. WoX architecture is built as instance of the publish-subscribe (pub/sub) software design pattern. Pub/sub is an enterprise integration pattern where senders of messages, called publishers, do not know a

priori what are the specific receivers of messages, called subscribers. Instead, published messages are characterized into classes, without knowledge of subscribers' identity. Similarly, subscribers express interest in one or more classes, and only receive messages that are of interest, without knowledge of publishers' identity.

The modelling of the pub/sub architecture starts from the Topic class. In Fig. 1 the UML dependence diagram is shown. The Observer software design pattern implements the main WoX idea. The Topic class has two member variables, to hold respectively the topic's actual value and preferred value: the actual value contains the most recent value for the feature in the location; the preferred value is used to send and receive requests about the desired topic value.

Every time an IoT node shows up, it handshakes with the WoX architecture, i.e. it declares its roles according to the formula (7). The corresponding subscriber classes are instanced and get attached to the Topic instance. When the Topic's actual value gets updated, SN subscribers (i.e. information consumers) get notified. Vice versa, when the Topic preferred value is modified, AC subscribers are notified so they can absolve their task. If the Topic's feature is a function, the topic's preferred value is ignored and the actual value is used both to pass function parameters as well as to obtain the function result. In the first case, FC subscribers get notified; in the second case, FN subscribers get notified.

The pub/sub architectural pattern centralizes the core information separating who provides data from who consumes it. IoT-based apps perform a one-time subscription to the topic(s) of interests, without perceiving the hardware layer. Subscribing to topics is easy also for "stupid" IoT nodes (e.g. RFID tags). Moreover, the total number of exchanged messages is drastically reduced because of dropping all the point-to-point connections between IoT entities.

Fig. 2 shows the WoX technical architecture. A pub/sub architecture alone cannot satisfy the requirements of hardware abstraction, event filtering, and standard-compliant persistence of an IoT middleware. To this aim, the pub/sub architecture rests on the top of an EPCglobal middleware. Such design choice guarantees quality and performance to the whole architecture. As aforementioned, the WoX architecture grounds on the Fosstrak. It is made up of four main layers:

1. Environment Level: it comprises the physical layer as well as any virtual environment that can generate events. Social Networks chats can be source of events too.
2. ALE middleware: it is responsible of querying/piloting the Environment Level and packing event reports for the upper layers. It also normalizes the events and filters duplicated data. On the bottom side, a series of adaptor is present for each IoT technology, both physical (e.g. RFID, WSN, KNX, etc.) and virtual (CVEs, Facebook, etc.). On the top side, three SOAP (Simple Object Access Protocol) Web services are needed to send hardware configuration (when needed), receive event reports formatted with the ECSpec (Event Cycle Specification) schema, and to send data to the technologies using the respective technology's format.
3. WoX Capturing Application: it is the architectural level implementing the WoX model. It instances the topics, updates them, takes care of the topic map, and make such data available to the end-user apps by a set of REST interfaces.
4. End users applications use the WoX APIs to subscribe to topics and they can run on any kind of device. For testing purposes, in the current work, Java and Android APIs have been used, but APIs in other programming language will be developed in the next steps.

An additional element, the EPCIS, concludes the architecture. It aims to persist IoT events for asynchronous usage. Nevertheless in WoX the EPCIS role is not crucial.
The Fig. 2 evidences the two possible flows of information: (i) node-to-node, the curved arrow, i.e. any IoT node can request services to other nodes (even of different technology), and (ii) node-to-app, the straight arrow, i.e. any app can receive data from any node, and vice-versa. The cornerstone of each information exchange is the WoX Capturing application, where the WoX model is implemented and the topics are instanced

B. Enterprise System Logic Architecture

The whole enterprise architecture depends on the WoX model, as presented in the previous section. The platform receives the row data from sensors and it elaborates them in order to extract complex information. The workflow of data within the application is basically depicted in Fig. 3.

The logic architecture introduces two kinds of topics, both compliant with the definition of Topic, but referring to

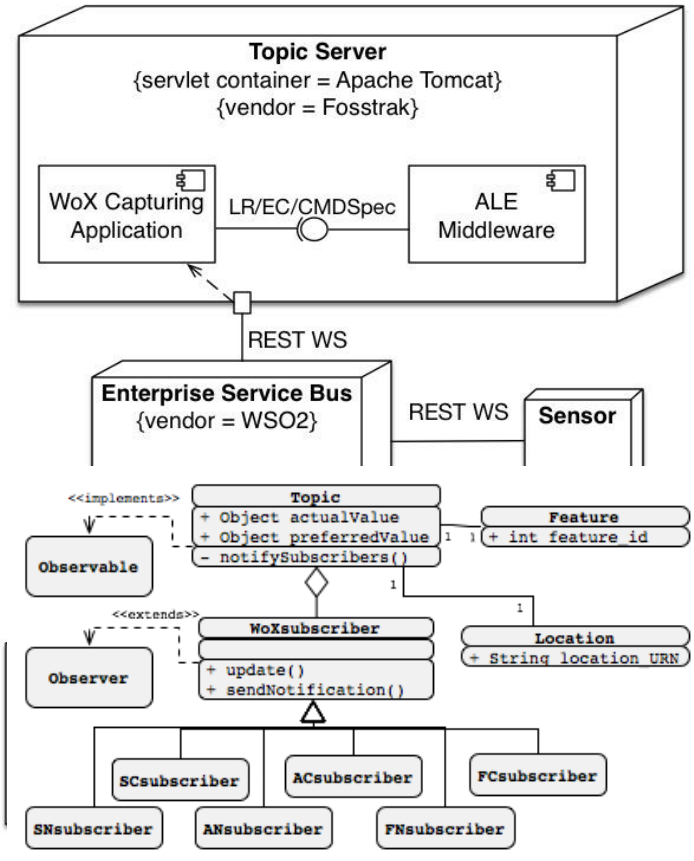

Fig. 1. The WoX architecture conceptual UML diagram

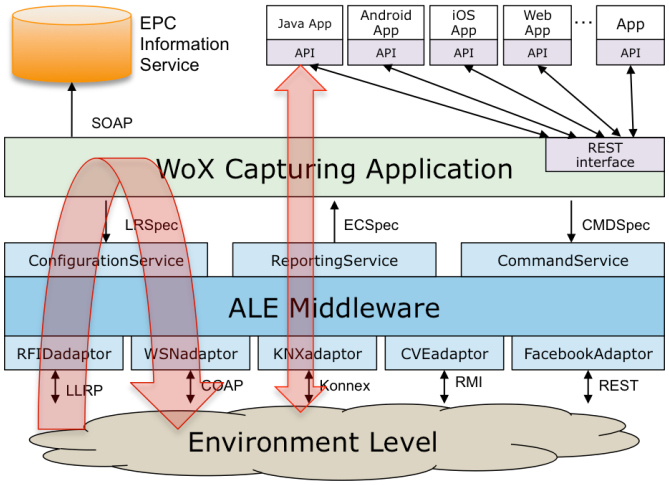

Fig. 2. The WoX technical architecture, based on the EPCglobal standard and its open source implementation Fosstrak. The arrows indicate the possible flows of information in WoX: node-to-node and node-to-app

different levels of abstraction:

1. Low-Level Topic: it is related to data belonging to observable/controllable IoT entities without any semantic connotation;
2. High-Level Topic: it represents a refined information semantically determined

The other components in the logic architecture are:

3. Enterprise Service Bus (ESB): it is the solution adopted to connect the various component of the application. It is compliant to Service-Oriented Applications (SOA) architectural pattern and it guarantees the interoperability among heterogeneous technologies.
4. Business Rules Web Service: it is a middleware with the specific function of knowledge processor. The module subscribes to the Low-Level Topic and applies the business rules on the data deriving from it, in order to draw events of interest and to update the corresponding High-Level Topic;
5. User Application: high abstraction level module appointed to subscribe to the High-Level Topic and to notify the user when an event of interest occurs.

The first kind of Topic is used to catch data provided by sensors, to control actuators and, generally, to process row data in an IoT network. The second one is used to treat elaborate information mined from row data according to the logic rules imposed by the knowledge processor.

Once a High-Level Topic changes its own internal state – that is, some event of interest has occurred – the User Application component gets notified and it can inform the end user. So, the User Application makes up the bridge between the end users and the Enterprise architecture.

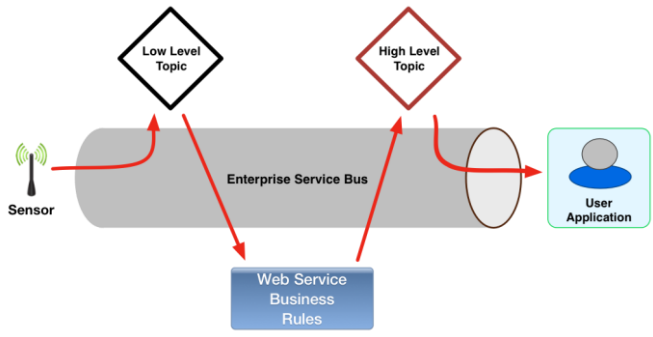

Fig. 3. Enterprise System Logic Workflow

C. *Enterprise System Implementation*

Referring to Fig. 4, most part of the architecture is implemented using WSO2 technologies. Particularly, WSO2 takes part in the constitution of the components: Enterprise Service Bus, Business Rules Server and Web Service Application Server.

As discussed in the Subsection IV.A, the module for managing topics is implemented on the top of Fosstrak EPCGlobal implementation and is deployed on a dedicated server.

The different modules in the architecture communicate by way of RESTful interfaces, in accordance with WSO2 directives.

The sensor component sends the data describing the state of the environment to the Enterprise Service Bus.

The ESB sends the data from sensors to the Fosstrak Capturing Application component, where a specific instance of Low-Level Topic encapsulates the data from the sensor.

The Application Server implemented on the WSO2 Web Service Application Server subscribes to all the topic instances in the Capturing Application, firing different actions depending on the type of topic.

In case of Low-Level Topic subscription, the Application Server receives notifications from it in conjunction with sensor updates and it forwards the information included in the topic to the Business Rule Server. In the latter component there is a Web service equipped with a Knowledge Processing Engine. It acts as Knowledge Processor and computes the received data: on the basis of the business logic rules, the modules decrees if an event of interest has occurred or not. If so, the Business Rule Server updates the High-Level Topic corresponding to such event. the business logic rules are implemented by means of Drools tools [14].

Since the Application Server can perform subscriptions also to High-Level Topic, an event of interest occurrence is seamless notified to it and, so, to the end users.

The Business Rules Server exposes the logic rules as Web services. Therefore, it is possible adding new rules or modifying existing ones by means of http requests, without the need of redeploy the application.

V. Proof of Concepts

In order to validate the architecture, a real scenario has been implemented in-vitro.

The main subject of the scenario is an elderly person, or a person with balance and/or ambulation difficulties. A generic user of the analysed platform is interested in remotely monitoring the person. Also, s/he has to be alerted by necessity.

For this purpose, the senior wears a wearable device equipped with an embedded accelerometer. The device forwards the accelerometer data to a Low-Level Topic. The Knowledge Processor will process the data in that topic and, in case of an incident related to a fall, it will update a High-Level Topic. End users interested in the status of the elderly will subscribe the topic and will be updated about any notifications

from the platform.

The use case requires the implementation of two Topics in the WoX framework:

- Shock: sudden change in the values of vertical acceleration measured by sensors.
- Fall: event of fall of the person to be monitored.

Topics' properties are detailed in TABLE I and TABLE II.

The step-by-step scenario is the following:

1. The on-board accelerometer on the wearable device detects acceleration data along three axes.
2. The data are sent to the Smart Gateway (by means of WiFi technologies).
3. The Smart Gateway forwards the gathered information to the ESB platform.
4. The ESB communicates the data to the WoX Capturing Application, which updates the Shock Topic in the corresponding location
5. The Application Server on the WSO2 Web Service Application Server component subscribes to the Shock Topic, receiving notifications from it in case of updates.
6. Each notification is forwarded to the Business Rule Server, where the Web service serves as Knowledge Processor. It triggers one of the following two rules:
    a. "Elder person has fallen": the Business Rules Server updates the *Fall* Topic in the Capturing Application.
    b. "Elder person is unhurt": do not perform any operation.
7. A summary response is sent back to the Application Server, which has initiated the processing request to Knowledge Processor.
8. A person interested to the health condition of the monitored subject has to subscribe the *Fall* Topic (by means of a dedicated user application).

The implemented Knowledge Processor is a Web service that exposes the business logic operation *SendFallNotification*: it triggers an event that updates the topics related to the fall of the monitored person.

Fig. 5 shows the Drools implementation of the business logic rules related to the *SendFallNotification* operation: each incoming request sent to the Knowledge Processor Web service is mapped into a *TopicNotification_Shock* object; then, on the basis of a condition placed on average values for the vertical accelerations received within the incoming requests, the Knowledge Processors triggers the two already discussed rules: "Elder person has fallen" and "Elder person is unhurt".

```
rule "Elder person has fallen" no-loop true
    when
       //conditions
       notification : TopicNotification_Shock()
       eval(notification.average() <= 0.5)
    then
       //actions
       Topic topic = new Topic(5, notification.getLocation());
       PublisherBL pub = new PublisherBL(true, topic);
       pub.publish();
       NotificationMessage msg = new NotificationMessage(notification, "FALL", "KO");
       insertLogical(msg);

end

rule "Elder person is unhurt" no-loop true
    when
       //conditions
       notification : TopicNotification_Shock()
       eval(notification.average() > 0.5)
    then
       //actions
       NotificationMessage msg = new NotificationMessage(notification, "FALL", "OK");
       insertLogical(msg);

end
```

Fig. 5. *SendFallNotification* implementation

Behind the Web Service, the KP consists in a semantic agent that uses an ontology to create relations about concepts. Such relationships allow excluding that the falling event is critical, for example when the WSN node is worn by an athlete performing his daily training session. In Fig. 6 the ontology used by the KP is shown. The ontology allows matching a 'Shock' event incoming from the PHY layer with a series of contextual information, like the user type, the timestamp, and the place of happening. Reasoning over the provided ontology allows to determine the criticism of the shock event.

The Fig. 7 collects some screenshots of the 'grandma fell down' notifications.

Considering that the system has to interact in real-time with users, system latency is a good parameter for quantitatively evaluating it. An Android mobile application connected to the Internet by LTE network was used as client for getting notified by the system in case of falling event. The average timespan between the falling event and the notification to the mobile application has been computed: on 5000 empirical samples, the average system latency value was 4980 milliseconds.

VI. Discussion

To perform the discussion of the WoX model, the current

TABLE III
QUALITATIVE COMPARISON WITH EXISTING APPROACHES

Quality	Industry-based	Semantic-based	Web of Topics [a]
Human understandable		X	X
Native compliance with standards	X		X
Multiple scenarios implementable		X	X
Compliance with virtual environments		X	X

TABLE IV
TECHNICAL COMPARISON WITH EXISTING APPROACHES

Quality	Industry-based	Semantic-based	Web of Topics
Real time	X	X	X
Quick response time	X	X	X
Scalability	X		X
Integration	X		X
Heterogeneous System support		X	X
Design	X		X
Collision Avoid	X		
Usability		X	X
Simplicity		X	X
Fault Tolerant	X	X	X
Trust	X		
Security	X		
Suitability		X	X
Cooperation		X	X
Privacy	X		

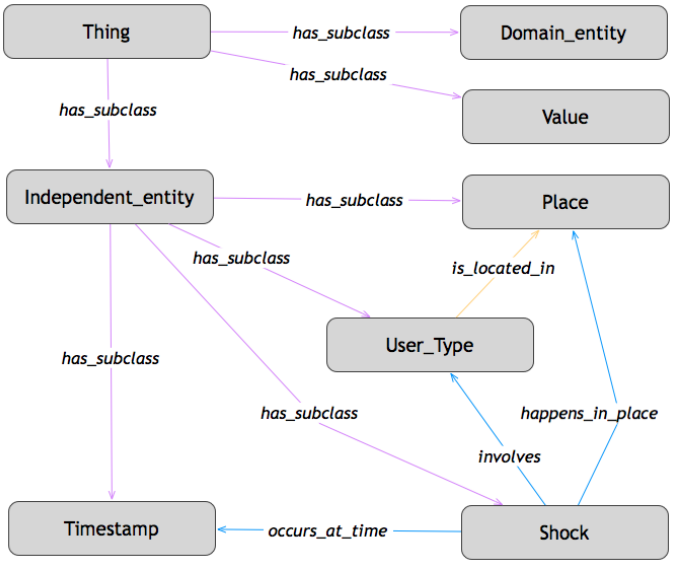

Fig. 6. Ontology model used in the KP in the proof of concepts

Section presents a practical comparison with the two main categories of approach discussed in the "Related Work" Section: architectures based on international standards, and horizontal (semantic) architectures. Table III shows a qualitative comparison with the two existing approaches. It is straightforward affirming that the WoX approach inherits the

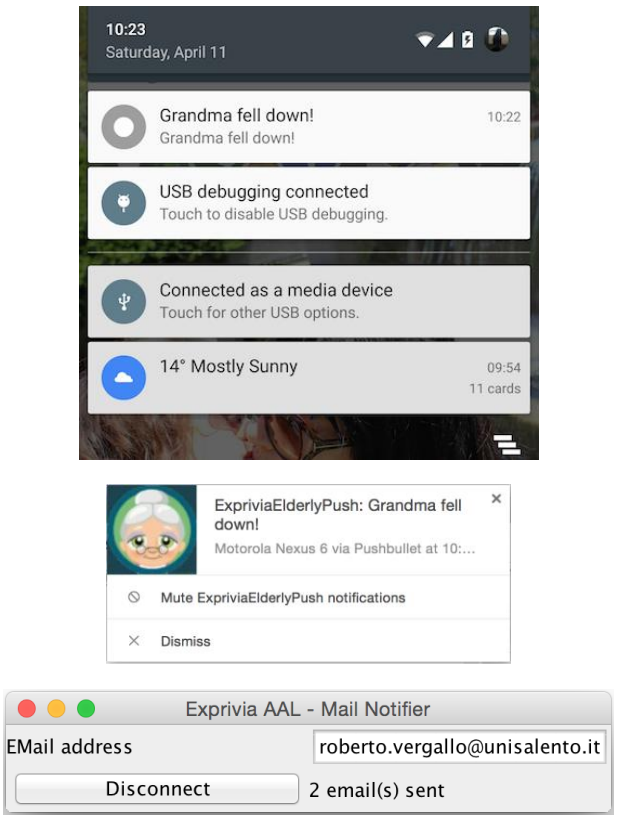

Fig. 7. Heterogeneous notifications about the falling down event

advantages of both the standard- and semantic-based ones. Table IV presents a technical comparison with the same approaches but from the ISO/IEC 9126 software quality standard viewpoint. Also in this case, WoX model appears as a merge of the characteristics of both the two different approaches. Thanks to the high levels of abstraction it implements, it gets close to the human way of thinking, so praising good level of usability, simplicity and design. At the same time, it leverages the EPCglobal standard, reaching the necessary features of robustness, security, reliability, and so on. As a con, WoX opens to threats in the means of information trust and security, as well as to privacy issues. This is due to the current lack of a mechanism able to certify the origin of the Topic information and protect itself from leaks. We are aware of these lacks and their resolution represents a future work. Another room for improvement will be the release of more programming languages with an enhanced set of exposed APIs.

VII. Conclusion

In this paper, an extension of the Web of Topics (WoX) approach was presented. WoX is a novel design model shortening the gap between the design and the solution domains in the IoT. In WoX, the generic IoT entity is seen as a set of couples Topic-Role. A WoX Role is expressed in terms of two dimensions: technological (sensor, actuator, computational node) and collaborative (service capability or need).

The innovation presented in this paper regards the extension of the WoX approach with the definition of a Knowledge Processors (KP). The aim of the KP is to determine new knowledge using heterogeneous information sources. A KP subscribes to a set of Topic just like a regular end-user application. Nevertheless, it uses the information incoming from the WoX layer to determine new information, hence updating upper level Topics.

In order to implement the enterprise architecture, the WSO2 open source ESB implementation has been used, thereby providing a cloud infrastructure for IoT applications management. The knowledge processing tools leverage the WSO2 Business Rules server, which uses Apache Drools as a Business Rules Engine (BRE). Each KP is mapped onto a Drools rule. The KP subscribes to a set of Topics just like a regular end-user application. Unlike them, the KP can produce new knowledge and push it on upper-level WoX topics, hence making it available for the IoT ecosystem.

In order to validate out work, an in-vitro validation was performed. The KP configuration aimed to determine the falling down of an elderly person by mashing up heterogeneous information sources, including a WSN node placed on the person and a set of data: timestamp, age, location. By interpreting such data in an ontology, the KP can determine whether a falling event happened and it regards a person in need of special care. The information is then propagated via different kind of notifications using the regular WoX approach.

As a next step, we are planning to submit the validation experiment inside a public software development challenge, in order to reach a valid amount of empirical data.

VIII. References

[1] D. Uckelmann, M. Harrison, and F. Michahelles, Eds.,

Architecting the Internet of Things. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[2] D. Guinard, V. Trifa, and E. Wilde, "A resource oriented architecture for the Web of Things," in *2010 Internet of Things (IOT)*, 2010, pp. 1–8.

[3] L. Mainetti, V. Mighali, L. Patrono, and P. Rametta, "Discovery and Mash-up of Physical Resources through a Web of Things Architecture," *J. Commun. Softw. Syst.*, vol. Vol. 10, N, pp. 124–134, 2014.

[4] L. Mainetti, V. Mighali, L. Patrono, P. Rametta, and S. L. Oliva, "A novel architecture enabling the visual implementation of web of Things applications," in *2013 21st International Conference on Software, Telecommunications and Computer Networks - (SoftCOM 2013)*, 2013, pp. 1–7.

[5] L. Mainetti, L. Manco, L. Patrono, and R. Vergallo, "Web of Topics: An IoT-aware Model-driven Designing Approach," in *IEEE World Forum on Internet of Things, Milan, Italy, Dec. 14-16, 2015*, pp. 46-51.

[6] "Fosstrak." [Online]. Available: http://fosstrak.github.io/. [Accessed: 20-Oct-2015].

[7] C.-W. Tseng, Y.-S. Lin, W.-H. Lu, and C.-H. Huang, "Extending EPCglobal ALE middleware to integrate transducer capability of IEEE 1451 standards," in *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, 2014, pp. 289–294.

[8] "EPCglobal | GS1." [Online]. Available: http://www.gs1.org/epcglobal. [Accessed: 20-Oct-2015].

[9] E. Song and K. Lee, "Understanding IEEE 1451-Networked smart transducer interface standard - What is a smart transducer?," *IEEE Instrum. Meas. Mag.*, vol. 11, no. 2, pp. 11–17, Apr. 2008.

[10] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, and V. Y. Terziyan, "Smart Semantic Middleware for the Internet of Things," in *ICINCO 2008, Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics, Intelligent Control Systems and Optimization, Funchal, Madeira, Portugal, May 11-15, 2008*, 2008, vol. 8, pp. 169–178.

[11] L. Mainetti, V. Mighali, and L. Patrono, "A Software Architecture Enabling the Web of Things," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 445–454, Dec. 2015.

[12] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. de Souza, and V. Trifa, "SOA-Based Integration of the Internet of Things in Enterprise Services," in *2009 IEEE International Conference on Web Services*, 2009, pp. 968–975.

[13] L. Lan, F. Li, B. Wang, L. Zhang, and R. Shi, "An Event-Driven Service-Oriented Architecture for the Internet of Things," in *2014 Asia-Pacific Services Computing Conference*, 2014, pp. 68–73.

[14] "Drools - Business Rules Management System (Java™, Open Source)." [Online]. Available: http://www.drools.org/. [Accessed: 24-Oct-2015].

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.

Luigi Manco graduated in Computer Engineering in 2012 at University of Salento (Italy), after a 6-mounths internship at the Vicomtech-IK4 Spanish research centre. Currently, he is a PhD student and collaborates with DEIB at Polytechnic of Milano. His research topics are semantic-based Multi-Agent Systems for Smart Environments.

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 and four chapters of books with international diffusion.

Roberto Vergallo graduated cum laude in Computer Engineering at University of Salento (Italy) in October 2010. Currently he is a post-doc fellow at the same university. He also collaborates with the Smart Grid Energy Research Center at the University of California Los Angeles (UCLA).