A Web of Things framework for RESTful applications and its experimentation in a Smart City

Federica Paganelli, *Member, IEEE,* Stefano Turchi, and Dino Giuli, *Senior Member, IEEE*

***Abstract*—The Web of Things is as an active research field which aims at promoting the easy access and handling of smart things' digital representations through the adoption of Web standards and technologies. While huge research and development efforts have been spent on lower-level networks and software technologies, it has been recognized that little experience exists instead in modeling and building applications for the Web of Things. Although several works have proposed REST-inspired approaches for the Web of Things, a main limitation is that poor support is provided to web developers for speeding up the development of Web of Things applications, while taking full advantage of REST benefits. In this work we propose a framework which supports developers in modeling smart things as web resources, exposing them through RESTful APIs, and developing applications on top of them. The framework consists of a Web Resource information model, a middleware and tools for developing and publishing smart things' digital representations on the Web. We discuss main framework implementation choices and its compliance with REST guidelines. Finally, we report on our test activities carried out within the SmartSantander European Project to evaluate the use and proficiency of our framework in a smart city scenario.**

***Keywords*—web of things, internet of things, smart city, web, Representational State Transfer, web services, sensors, smart things.**

I. Introduction

THE Web of Things is an active research area that focuses on the specific challenge of making smart things accessible and manageable through open Web standards. This vision is correlated with the broader Internet of Things (IoT) research area aiming at enabling communication with and among smart objects, leveraging Internet standards and technologies [1].

As argued by the European Expert Group on Services in the Future Internet [2], huge research and development efforts have focused on lower-level networks and software technologies, while "there is currently little experience with building applications for this new, emerging ecosystem of IP-enabled devices and objects". As the Web facilitated both application development and use for the traditional Internet, it is now expected to "unleash the potential of the Internet of Things by making it accessible and programmable by developers who are not necessarily experts in ubiquitous computing" [2].

The Web of Things is expected to ease the access to smart things' capabilities and promote novel value added services based on the combination of traditional web resources with those representing entities from the physical world (smart things, sensors, appliances, etc.) [3]. This has the big advantage of allowing the integration of smart things with the impressive amount of information resources and services already on the Web, as well as exploiting available technologies and best practices for web resources publishing and management.

On this perspective, the Representational State Transfer (REST) architectural style, developed as an abstract model of the Web architecture [4] is considered a reference paradigm for bringing sensors, and more generally smart things, into the Web [5]–[7]. Indeed, REST defines a set of principles for designing distributed hypermedia application fulfilling scalability, simplicity and loosely coupling requirements.

Several research works have proposed approaches for implementing the Web of Things through REST principles, as discussed in the survey by Zeng et al. [8], [9]. However, the path to the effective realization of the Web of Things is made difficult by the widespread misunderstanding of REST basics [10] and, consequently, the lack of software development frameworks that comprehensively support them [11], [12].

As a consequence, a main limitation of related work is that poor support is provided to web developers for speeding up the development of web applications for accessing, modifying, and composing information resources in the Web of Things domain [2] while taking full advantage of REST benefits.

In this work we propose an approach towards the Web of Things based on a graph representation of web resources, which can be accessed and modified at the desired and meaningful level of granularity through a REST-compliant uniform interface. According to our approach, a smart thing can be modeled as a graph of individually addressable web resources whose edges represent simple structural aggregation (i.e. containment) and reference relations. Graph-based representation of smart things can be interlinked with other types of information nodes to build a growing graph of globally-addressable information resources that can be navigated, queried, and composed through a uniform REST interface.

The proposed framework consists of: i) a Web Resource information model for representing smart things as web-accessible resources, ii) a middleware for handling and exposing these resources through a uniform API, and iii) a set of tools for easing the creation and handling of Web Resources as well as their web-oriented programming and publishing.

By leveraging presented Web Resource model, middleware

F. Paganelli is with the National Interuniversity Consortium for Telecommunications, at the Research Unit of the University of Florence, 50139, Italy (phone: +39 055 4796382; fax: +39 055 4796427; e-mail: federica.paganelli@unifi.it).

S. Turchi and D. Giuli are with the Department of Information Engineering, University of Florence, 50139, Italy (e-mail: stefano.turchi, dino.giuli@unifi.it).

Manuscript received February 24, 2014. This work has been partly funded by the European Commission through the SmartSantander FP7-ICT project.

and tools users can easily create and modify digital representations of smart things, interconnect them with simple links and publish them on the Web as RESTful resources.

Finally, we report on the use of the framework in a Smart City scenario within the SmartSantander European Project. The experiment consisted in the development of a web application for accessing sensors deployed in the city of Santander, Spain, as well as creating new virtual sensors. We also evaluated end-users acceptance of provided tools.

The remainder of this paper is organized as follows. In Section II we present related work and motivate our contribution. Section III introduces the main principles of our approach and describes our framework. In Section IV we describe the experiment carried out within the SmartSantander European Project to evaluate the proposed approach. Section V concludes the paper with insights on future work.

II. Related Work

In this section we briefly introduce the main principles of the REST architectural style. Then, to motivate our work, we analyze state of the art solutions for middleware and application frameworks for the Web of Things.

A. REpresentational State Transfer architectural style

REST (REpresentational State Transfer) was proposed by Fielding [13] in his doctoral dissertation as an architectural style for building large-scale distributed hypermedia systems. By the REST vision, data sets and objects handled by client-server application logic are modeled as resources. REST key principles are fivefold [13]:

1) *URIs as resource identifiers*. Resources are exposed by servers through URIs. Since URIs belong to a global addressing space, resources identified with URIs have a global scope;
2) *Uniform interface*. The interaction with the resource is fully expressed with four primitives, i.e., create, read, update and delete; These operations can be mapped onto HTTP methods: GET reads the resource state; PUT updates the resource state; DELETE deletes a resource; POST extends a resource by creating a child resource;
3) *Self-descriptive messages*. Each message contains the information required for its management;
4) *Stateless interactions*. Each request from client to server must contain the information required to fully understand it, independently of any previous request;
5) *Hypermedia As the Engine Of Application State (HATEOAS)*. A hypermedia system is characterized by participants transferring resource representations that contain links; the client can progress to the next step in the interaction by choosing one of these links [14].

Historically, two styles opposed in the Web services field: SOA and ROA (Service Oriented Architecture and Resource Oriented Architecture, respectively), i.e. WS-* and REST. Pautasso et al. [15] compare RESTful and WSDL/SOAP based Web services, using architectural principles and decisions. The authors conclude that RESTful services are more suited for tactical, ad hoc integration over the Web (*à la Mashup*). Adamczyk et al. [12] evaluated some key questions regarding the real and perceived distinctions between the two styles, and carried out a thorough analysis of Web services exposed by four repositories: 1) xmethods.net; 2) webservicex.net; 3) webservicelist.com; 4) programmableweb.com reporting a very fast-pace growth of REST popularity. Moreover, in a study led in 2012 by Guinard et al. [7] involving 69 novice developers in implementing IoT client applications exploiting WS-* and REST approaches, results reported that REST is statistically significantly easier and faster to learn than WS-*. In addition, a considerable number of volunteers agreed that REST was more suited for Web applications requiring to integrate Web content: "[for] Web Mashups, REST services compose easily".

Although RESTful APIs are deemed to be relatively simple, developing RESTful systems is difficult and existing frameworks fail to support developers in conforming to REST constraints [11], [12]. The analysis provided by Zuzak and Shreier [11] highlights some weaknesses of existing general-purpose REST frameworks, such as the lack of structural and behavioral model for resource types, poor support of flexible mapping between resource representations and stored data entities, lack of support for scripting functions for read-write access to the application state at client-side. Adamczyc et al. [12] argue that current frameworks do not support well the "hypermedia constraint" or Hypermedia as the Engine Of Application State (HATEOAS) principle, and most of them does not fully support HTTP caching.

B. Middleware and frameworks for the Web of Things

Zeng et al. provide a survey on research works in the WoT domain [8], where they highlight two main analysis criteria: the approach for integrating the physical things to the Web (i.e., direct and indirect integration) and the provision of composable services for enabling physical-virtual mashups.

Guinard et al. [16] provide a pioneering work contribution by specifying and implementing a set of REST services that expose functionalities of sensor nodes as web resources and link them together hierarchically. In a later work, this approach is applied in the AutoWoT project [17], a toolkit that facilitates the rapid integration of smart devices into the Web by offering a general way of modeling web resources and a mechanism for building web server components that expose the functionality of these devices. If the user modifies the Web resource at a later time, AutoWoT can semi-automatically regenerate the software for the Web exposure of the device.

Also other works adopted the REST principles for implementing WoT-oriented systems. WebPlug [18] is a framework for the Web of Things that allows users to manage web resources representing sensors' data and compose personal services based on physical objects. The representation of a web resource is enhanced using MetaURLs, i.e. textual labels that can be appended to the URL of the base resource to access some related information (e.g., collection of historic values).

Christophe et al. [19] present a prototype framework allowing the creation and handling of virtual objects. To validate their framework, they also introduce a web application that

allows end-users to access the virtual objects and compose them according to basic event-based rules.

Uberdust [20] is a service-oriented platform that provides storage, sharing and discovery of real-time and historical data from smart objects and sensor networks through the Web. These data can be accessed using REST and WebSocket APIs.

Significant efforts are ongoing to develop a Semantic Sensor Web [21] and integrate it with the growing body of knowledge available as Linked Open Data [22].

SemSense [23] is a system that collects data from physical sensors and publishes these data enriched with semantic annotations on the Web. SPITFIRE [24] is a service infrastructure that offers semi-automatic generation of semantic sensor descriptions and efficient search for sensors and things based on their current states. Applications can issue search requests to SPITFIRE and then directly invoke retrieved services.

C. Motivation of our work

While most works focusing on semantic descriptions of smart things [21], [23], [24] aim at providing efficient search and discovery capabilities, the scope of our work is to provide users with tools for accessing and programming the Web of Things, in compliance with REST guidelines.

Towards this objective, some previously mentioned works [17]–[20] provide tools for speeding up the exposure of REST APIs for accessing smart things capabilities.

The original contribution of our work with respect to the ones mentioned above [18]–[20] is twofold: 1) the proposal of an information model representing web resources and their mutual relations; 2) the provision of tools supporting users in composing, publishing and sharing the web representation of smart devices, that can be accessed through common web browsers. In the context of this study, we consider two different target users: i) developers of the WoT (from now on, *developers*) and ii) tech-savvy users with rudiments of web technologies (from now on, *users*). In subsection III-C we will discuss tools created for both categories. Finally, the term *end-user* will refer to people without particular domain competences, e.g. ordinary "Web surfers".

Our work shows more similarities with AutoWoT [17] since both adopt a Web Resource model and provide services supporting the exposure of smart devices as web resources. However, a significant difference exists in the modeling approach. The AutoWoT resource model relies solely on a hierarchical relation among resources, consequently a physical thing is represented as a hierarchical collection of web resources. We argue that a mere hierarchical relation is not fully compliant with the REST hypermedia constraint (i.e. reference links among resources drive the application state evolution and shall be advertised by the server at each interaction step). Since, as argued by Vinoski [9], "relationships between resources and how the server makes those relationships available to applications are at least as important to REST developers as resource naming", our approach relies on a graph-based model that includes two main types of relations among web resources (i.e., aggregation, and reference). Purposely, these relations recall how, in fact, web resources are organized and can be navigated in traditional web sites. In addition, IDN is natively capable of handling sets of resources, and this has the following implications: i) through the Web Resource REST resource exposed by IDN API is possible to perform aggregated operations on graphs. This means that it is possible to retrieve, create or update more vertexes at the same time. ii) The Web Resource model describes sets of Nodes (at the limit, made of one element only) and is particularly suited for hosting properties which are strongly affected by information aggregation practices, such as privacy. Moreover, unlike AutoWOT, the IDN model offers a scriptable Node, called Activity Node (see section III-A), which is able to perform processing operations on other Nodes' contents and output the computation result. This scenario seems to be very promising for devices virtualization. Leveraging this model, we propose a set of capabilities supporting developers and users in representing smart things as Web Resources and composing them with existing web resources, to build novel applications.

III. Web of Things RESTful Framework

Our work aims at leveraging Web principles for making a Web of Things easy to access and program. Our framework provides tools for easing both exposure and handling of smart devices (and related information) as web resources. These web resources can be composed with "traditional" ones to facilitate the creation of web applications and services. This objective requires a dedicated adaptation layer mediating the interaction with sensor and actuator implementation technologies that have not been designed with the Web exposure requirement in mind.

It is widely accepted that REST is the architectural style for the Web, and our approach for web resources exposure is based on its principles. Since the mere adoption of mainstream REST frameworks do not guarantee the compliance with all REST guidelines [11], [12], our work aims at helping developers in taking such constraints into account. More specifically, our framework provides support to:

- *Resource types definition and design*. We propose a graph-based information model for representing web resources, called Web Resource model.
- *General-purpose software for exposing and handling operations on web resources*. Our system includes a set of middleware services for exposing web resources through a REST uniform interface.
- *Mapping between web resource representation and legacy data-sources*. The middleware is based on a layered architecture that distinguishes the web exposure layer (dealing with web resource model and representation, content negotiation, RESTful uniform interface exposure) and the layer dealing with storage of data entities and/or the retrieval from external sources.
- *Programming and Web Publishing tools*. These consist of Java and JavaScript libraries for easing the development of web applications for the Web of Things, a web application allowing users to graphically access, modify and create new web resources according to our model, and a plugin for their rendering and visualization in ordinary web browsers.

As detailed in section III-A2, our information model represents web resources as nodes (containing data and metadata) interconnected through links, to create a browsable graph of information. Our model has two types of structural links: aggregation (i.e. containment) and reference relations. Leveraging the modeling capabilities and programming tools made available by our framework, a smart thing is modeled as a graph of individually addressable web resources handled through a REST interface, at the desired level of granularity. This graph-based representation allows to easily represent smart things as sets of nodes that can be interlinked and connected with other resources (e.g. information resources of the "traditional" Web of Documents) to build a growing graph of globally-addressable information that can be navigated, queried, reused and composed through a uniform REST interface. Hereinafter, we will consider sensors as smart things reference examples.

A. Web Resource Model

In the following, few definitions are given before introducing the Web Resource concept.

Definition 1. A **Node** is a tuple $N = (C, P)$, where C is the set of content elements (i.e., data) and P is the set of properties (i.e., metadata) that characterize the content. A nice URI [13] associated to every node is used as a unique identifier.

Definition 2. An **Aggregation link** is a directed link between two Nodes that represents a container-content relation. The conveyed meaning is: the originating node aggregates, therefore contains, the destination node.

Aggregation is a transitive relation: given two nodes n_0 and n_1 connected via an Aggregation link departing from n_0 and pointing to n_1 ($n_0 \rightarrow n_1$), we are implicitly stating that n_0 contains n_1. As a consequence, given a third node n_2 so that $n_1 \rightarrow n_2$ we can say that n_0 contains n_2.

Definition 3. A **Reference link** is a directed link between two Nodes that represents a pointer towards a referred resource (i.e. the destination node). No further meanings are associated with Reference links. To better understand the Reference link role, it could be somehow compared with the HTML `href` attribute.

1) Node Data Model: The Node data model is made of four elements, as shown in Fig. 1: ApplicationData, ApplicationMetadata, StructuralMetadata and ManagementMetadata.

The ApplicationData is the section entitled to host contents, i.e. the information to be consumed or produced by applications. This section includes Content, ContentSchema, ContentEncoding, and ContentType sub-sections which wrap the content itself, the location hosting the related content schema (if available), the encoding algorithm (e.g., base64), and the content type expressed in MIME type notation [25], respectively. These subsections are crucial for supporting content reuse. Indeed, our paradigm is conceived to encourage information reuse and when a client (e.g., an application) retrieves a Web Resource it should be able to handle its contents by choosing proper decoding and parsing algorithms.

The information contained in the ApplicationMetadata section can be used for tracing applications' modifications. When a client modifies a Web Resource, the Information History module (see subsection III-B) runs a versioning procedure to persist that Web Resource latest version. These metadata provide information about the author of the modification such as the related time-stamp, the identifier of the application responsible for the creation and the author of the resource.

The StructuralMetadata section contains references to Link elements originating from the Node and, because of this, it takes part in the graph's structure. For every link type, additional information is provided such as a local name, description metadata, allowed operations to support the HATEOAS paradigm and, of course, the referred URI.

The ManagementMetadata section may contain properties that further characterize the Node in terms of licensing, privacy, provenance, time and versioning. More specifically, the LicensePolicy element may be used to declare the licensing policy in force for the Node. The PrivacyPolicy element basically states whether the Node contains identifying and sensitive information. The ProvenanceManagement element contains information concerning the enforced provenance strategy [26], [27]. The TimePolicy element specifies whether the node should be destroyed or invalidated after (or before) a certain date. Finally, VersioningPolicy specifies which versioning policy should be applied to the Node.

The Activity section is dedicated to the specification of a script for dynamically generating contents. A scripted Node is called Activity Node and presents special outgoing edges (Active Links) defining dependencies pointing to Nodes containing inputs required for the script computation. When a request is issued for an Activity Node, its dependencies are resolved before executing the script. The output of the script constitutes the content information for the current Node to be included in the ApplicationData section. In authors' experience, Activity Nodes are very handy to represent Web Resources whose content depends on contents of other Web Resources. A significant example is the Virtual Sensor, which is a non-physical sensor whose output is a function of other feeding sensors' outputs. This is particularly useful to increase the coverage of existing sensors or to define derived measurements, such as the Heat Index [28] which combines temperature and humidity.

2) Web Resource Exposure: The **Web Resource** is the REST resource exposed by the IDN and is defined as a directed graph $R(N, L_{Aggr})$ where N is the set of Nodes (with $|N| \geq 1$) and L_{Aggr} is the set of Aggregation Links. A Web Resource has a single Root Node n_R, i.e. a Node with no incoming Aggregation Links. A Web Resource representation can contain a different number of Nodes, determined by dedicated URI parameters. Nodes are served by resolution levels, taking as reference the geodesic distance [29] from n_R.

A URI for a Web Resource has the following syntax:

http://auth/node_name/$p{inner_path}/-
$c{content_path}/$v{version}/$r{res_depth}?keys=vals.

The `auth` fragment defines the authority as defined in the URI specifications [30], `node_name` is a path

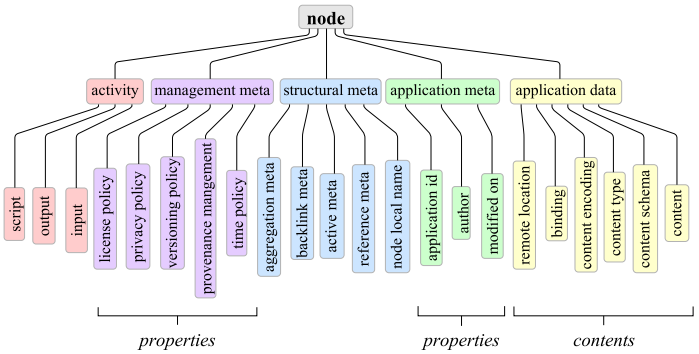

Fig. 1. The first level elements hierarchy of the Node data model. Vertical texts specify whether these elements are used to implement contents or properties.

TABLE I. WEB RESOURCE URI SYNTAX

Fragment	Regex	Semantics
node_name	[^\][/\w]*[^\]	The name of the Node.
inner_path	[^\][/\w]*[^\], iff the character sequence identifies a valid node's section.	The name of the inner section of the Node.
content_path	a valid (reduced) XPath expression	The specification to reach parts of structured contents.
version	[^.][.\d]*[^.]	The version of the Node.
res_depth	[\d]	The number of edges to explore starting form the Root Node, determining the retrieved Nodes.

determining the name of the current Node, `inner_path` is an optional filter selector specifying which part of the Node is required (e.g. the ApplicationData or StructuralMetadata explained in subsection III-A1), `content_path` is an optional XPath [31] expression targeting specific elements in structured contents, `version` is an optional parameter specifying which version of the Node is requested and finally, `res_depth` is an optional parameter specifying the resolution depth as the number of hops from the current Node. The `content_path` parameter requires an additional explanation: a Node can contain either structured (e.g. XML or JSON) or unstructured (e.g. plain text) data. In the first case, is possible to exploit the underlying structure to offer additional filtering capabilities based on content elements (for example, if the Node contains a book store, is possible to filter authors, provided they are represented in the book store structure). Since it is not possible to make any assumption on the adopted content format, `content_path` must use a generic syntax. To this end, we used a reduced set of the XPath syntax excluding XML peculiarities (e.g. attributes and namespaces). Tab. I summarizes the fragments' meaning.

A Web Resource is exposed to external clients through a uniform CRUD interface, made of a fixed set of operations: create, read, update, delete. These operations can be mapped onto HTTP methods: GET reads the resource state; PUT updates the resource state if the resource already exists and creates it otherwise; DELETE deletes a resource. In order to keep interaction with resources flexible, easy and comprehensive, much effort has been put in making it fully RESTful. Specifically, three aspects have to be highlighted: first, content negotiation is available for Nodes. The current implementation allows a client to request XML, JSON and HTML representations. As explained in subsection III-C4, HTML representations are produced by a JQuery plugin which requests (via AJAX) and processes a JSON representation, generating a human oriented version of it. Such strategy is motivated by the fact that a set of Nodes could have a meaningful visual arrangement that disregards the strictness of the graph structure. This can be seen as an application of the code on demand style [13]. Second, in order to avoid conflicts while updating resources ETags [32] are used to represent their status. Third, as stated above, HATEOAS [13] is supported by including in the Node representation links to the next related resources (whether Reference or Aggregation Links) as well as instructions on how to interact with them.

3) Examples: By the Web Resource model, a smart thing can be represented as a graph of structured information pieces.

Fig. 2 shows a simplified example of a sensor modeled as a graph of Nodes (depicted as circles). These Nodes contain the granular information (black pins) and are connected by arrows representing relations. The `sensor/{id}` labeled vertex carries the content describing sensor's characteristics, i.e. descriptive information (producer, serial number), capabilities such as type of sensor (e.g., temperature sensor) and dynamic information such as battery load percentage. This vertex has two children (`/location` and `/data_production`). The location Node is used to identify the sensor's geographical position in terms of longitude and latitude, while the sub-tree having `data_production` as root represents the sensor's observations. In particular, the `data_production` vertex contains information on last measurement date time, and its direct children identified with `physical_quantity` (i.e., temperature, light, noise, and CO index), contain actual measurement values. Please note that common sensors and Virtual Sensors are represented by the same model, with a minimal modification. In the case of a Virtual Sensor, the `physical_quantity` Node is an Activity Node and edges to `physical_quantity` Nodes belonging to other sensors are Active Links.

Tab. II provides an example of the uniform interface constraint applied to the a sensor Node. The API semantics applies identically to every Node belonging to the sensor or virtual sensor Web Resource graph.

B. Middleware

The framework includes a middleware, called InterDataNet (IDN), that offers capabilities for handling web resources compliant with our model (i.e. Web Resources, see III-A2), exposed through a RESTful uniform interface.

The core of the IDN middleware is the result of previous works [33], [34] and it was applied to the development of RESTful web services in different domains such as healthcare [35] and management of RFID-related events [34].

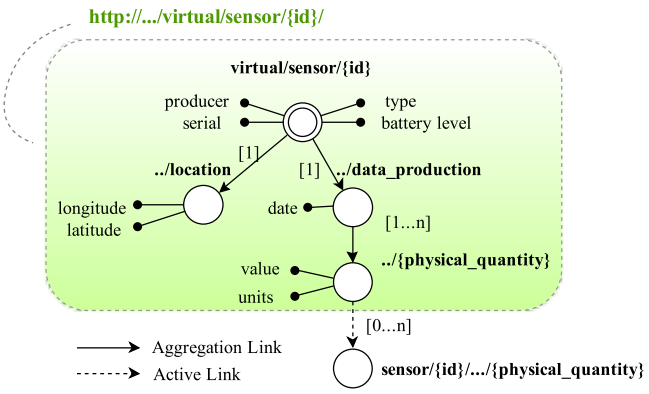

Fig. 2. A Web Resource representing a generic sensor (Virtual Sensors have Active links to other sensors while normal sensors have not). Since the Web Resource is defined as a graph of Nodes, operations on sets of Nodes are enabled. In this regard, the boxed sub-graph shows the Web Resource retrieved when a user requests a complete sensor resolution: not only the sensor Node, but also all its contents (i.e. all the aggregated Nodes) are returned. Please note that a complete resolution is triggered by issuing a GET request on the sensor's URI followed the ending "/" character.

TABLE II. INTERDATANET REST APIS FOR A SENSOR WEB RESOURCE

HTTP Verb	URI	Semantics
GET	.../sensor/{id}	To retrieve the single node identified by the URI.
GET	.../sensor/{id}/	To retrieve all the nodes aggregated by the node identified by the URI.
GET	.../sensor/{id}$tn	To retrieve all the aggregated nodes being distant *n* hops from the node identified by the URI.
PUT	.../sensor/{id}	To create the node identified by the URI, or modify it (if it already exists).
DELETE	.../sensor/{id}	To delete the node identified by the URI.

For the purpose of this work, the middleware core has been extended in order to: 1) cope with external information sources (e.g. sensors, gateways, etc.) interoperability requirements, and 2) improving both developer and user support in smart things' digital representation design and handling, through programming APIs or web browsers.

The IDN implementation adopts a modular pattern enforcing the separation of concern principle. Core modules are Virtual Resource, Information History and Storage Interface.

Virtual Resource is in charge of managing and exposing Web Resources. In fact, it exposes REST uniform APIs for accessing and managing smart things' web representations. These REST services can be accessed by end-users through web browsers (enhanced with appropriate REST-enabling plugins) or by other web applications. As shown in Fig. 3, different Virtual Resource instances dialog as peers to realize a distributed graph of interconnected Nodes. In this way, it is possible to reach Web Resources spanning different domains from a single access point.

Information History provides versioning capabilities for Nodes. This is useful for tracking the history of a resource,

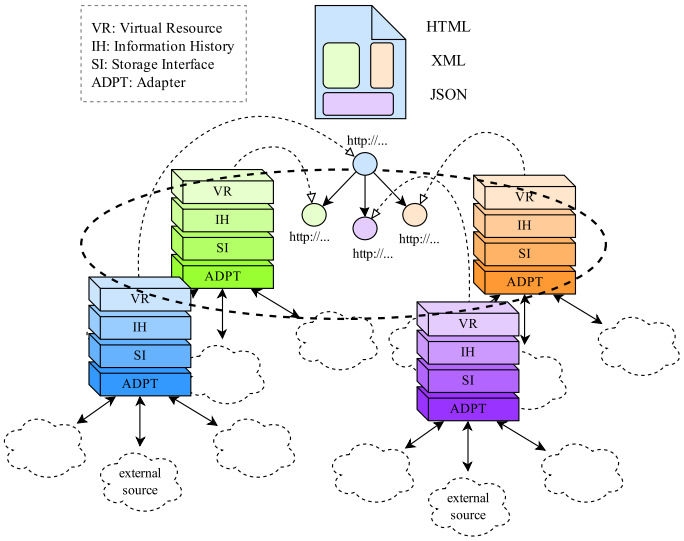

Fig. 3. The overall view of the InterDataNet architecture.

supporting cooperation by allowing branching and merging, and enforcing provenance by allowing thorough inspection of previous states. This service is not mandatory and can be disabled according to specific application/publisher requirements.

Storage Interface provides data and metadata persistence capabilities. IDN-specific metadata (e.g. the ones described in section III-A) are always persisted in the Storage Interface domain, while data can be either native or external. Native data are created directly in IDN and are managed by the Storage Interface persistence service. Conversely, external data are owned by other sources and are never replicated on the IDN side by default. In order to cope with external data, IDN stores their references (more precisely, IDN stores the references of interchange resources exposed by the Adapter which, in turn, keeps the mapping towards the external data-source). External data require the Adapter mediation to be managed as native resources. Storing references is a good strategy to avoid bottlenecks while dealing with frequently changing data (e.g. sensors' outputs). IDN applies this strategy also to treat historical data, whether they are provided by an external data-source or a custom *ad hoc* service.

Virtual Resource, Information History (which is in an ongoing stage of implementation) and Storage Interface have been developed as Java 7 Web Applications running on Apache Tomcat Application Server. They have been implemented leveraging the Spring framework [36] and Hibernate [37], JAXB [38] and JSP [39] technologies.

The core middleware includes also a Search Service, based on Apache Solr [40], offering search capabilities for retrieving indexed Web Resources. The Search Service exposes a REST-based search interface for submitting queries based on keywords, time and/or location parameters. Semantic search and discovery is currently under development.

The Adapter is the component that support the interworking of the IDN middleware with external data-sources. Thus, an adapter component contains *ad-hoc* intermediation logic for

exposing data and services provided by legacy systems and devices according to our graph-based Web Resource model. In this way, properties enabled by the IDN resource model can be applied to information originating from outer sources.

The Adapter is also designed with a modular approach, and includes four components: 1) a Notification Manager which manages the Adapter subscription to the data-source, enabling a push/pull notification service; 2) a Transformer module which refines data served by the outer source (e.g., de-multiplexing the information to achieve a more granular representation); 3) a Web Resource Manager which assembles the outer information in a graph structure compliant with the Web Resource model; 4) a Translator which translates requests coming from the IDN Service Architecture interface in a communication format supported by the data-source (e.g., a PUT request to IDN Service Architecture could map to a POST request to the data-source).

Basically, the Adapter is thought to be connected to services exposed on the Web such as WS-*, REST, RDF, and so on. However, there are no theoretical problems to connect it with different data-sources. Indeed, the Adapter is an intrinsic customizable component and different configurations can be supported by different Adapter instances. The Adapter can be semi-automatically configured in terms of communication interface towards the data-source (e.g. by exploiting interface descriptions such as WSDL) and Web Resource modeling of external data (e.g. by exploiting relational database schemas to build the graph model).

C. Programming and Web Publishing tools

To fully exploit the features provided by IDN, we also developed a set of tools for easing the use of the Web Resource model and REST APIs as well as the development of web applications on top of them. To this end, we provided four tools: i) a Java library for server-side programming, called IDN Java Library; ii) IDN.js, a client-side JavaScript programming library; iii) a web application providing a graphical interface for designing and managing Web Resources, called IDN-Studio; and iv) IDN-Viewer, a highly customizable jQuery plugin that automatically renders HTML views from Web Resources returned by the InterDataNet middleware.

While our Java and JavaScript libraries are thought for developers and provide APIs for easing the interaction with the middleware as well as Web Resource instances handling, IDN-Studio and IDN-Viewer are conceived to lower the effort required by users for creating, modifying and publishing new resources to be shared with other users and end-users.

1) IDN Java Library: The IDN-Java Library provides features for easing the server-side programming of web applications that handle Web Resources exposed by the InterDataNet middleware. Its main components are the following:

- the `it.unifi.idn.webresource` package providing a java-based internal representation of the Web Resource model;
- the `it.unifi.idn.IdnElement` class that exposes methods for manipulating the Node;

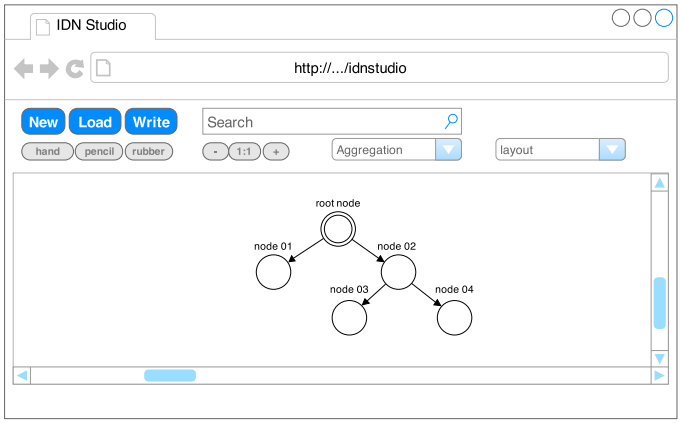

Fig. 4. Mockup of the IDN-Studio application.

- an InterDataNet client implementation in charge of interacting with the InterDataNet middleware for performing CRUD operations on Web Resources.

2) IDN JavaScript Library: This library provides a set of functions for client-side programming easing the interaction with the middleware and the handling of the returned Web Resource representations.

The `it.unifi.IDNjs.Manager` class defined in this library handles the RESTful invocations to the InterDataNet middleware to retrieve, delete and write single Node- or multiple Nodes- Web Resources, while the `it.unifi.IDNjs.Node` class represents the Node and exposes methods for internally managing the content, links and metadata. The offered APIs are not significantly different than the ones discussed for the Java Library, and therefore won't be detailed here. The library has been implemented by leveraging JQuery [41] as a helper library, JSON-js [42] for (un)marshalling document resources and an AJAX [43] client.

3) IDN-Studio: IDN-Studio is a JavaScript web application for the visual management of Web Resources, providing the user with a full control over the graph structure and data. IDN-Studio interacts with the middleware through the REST APIs exposed by the Virtual Resource and allows users to create new Web Resources, and delete or modify existing Web Resources by adding or removing Nodes, contents and properties. A search form is also available for querying the IDN Search Service in order to retrieve indexed Nodes.

Fig. 4 shows a mockup of the IDN-Studio GUI that emphasizes the main operations available (e.g. a search form, buttons for reading, writing and loading Web Resources and editing tools displaced on the dashboard). A running instance is available at http://idn.dinfo.unifi.it:20282/idnstudio.

4) IDN-Viewer: While IDN-Studio provides a human oriented representation of Web Resources conceived for easing their management, IDN-Viewer addresses the problem of their rendering on a web browser. Its main task consists in parsing the resources returned by the middleware and, depending on

their structure, generating rich HTML web pages.

IDN-Viewer is able to choose the proper visualization for different types of content by inspecting the ContentType section of the ApplicationData and selecting the appropriate widget. For example, structured contents such as JSON or XML are rendered as tables, images (jpeg, png, etc.) are rendered as clickable thumbnails, plain text is rendered as is.

More specifically, for each resource to be displayed, the plugin takes its URI as a key parameter and performs an AJAX request to retrieve it from the middleware. Then, the returned representation is parsed and proper transformation rules are applied. Moving from the Root Node, an algorithm is run to analyze the graph of interlinked Nodes. Resources referred by Reference links are organized as menus. The algorithm interprets Aggregation relations differently, depending on the geodesic distance d_j, i.e. the length of a shortest path evaluated on Aggregation edges between the Root Node n_R and the Node n_j. We define a configurable threshold D. Nodes for which $d_j < D$, are organized in container elements (e.g., `<div>` HTML elements) and shown on the canvas. If there exists at least one node n_j such that $d_j \geq D$, a "show more" button is included in the layout. As the button is pressed a (configurable) amount of remaining Nodes are loaded and displayed on the canvas. This strategy helps in keeping the Web Resource's HTML representation clean, supporting human-friendly inspection by incrementally filling the canvas space.

IDN-Viewer has been implemented as a plugin for the popular JQuery [41] JavaScript framework. It is highly customizable and configurable and is not coupled with any CSS framework.

Fig. 5 shows the rationale behind the HTML rendering of a Web Resource representing the "Universidad de Cantabria" Point of Interest (PoI)(Fig. 5.a). Nodes constituting the Web Resource are inspected to generate a widget that better presents their contents (Fig. 5.b). Finally, elements and widgets are arranged on the canvas to produce meaningful representation of the original Web Resource (Fig. 5.c).

IV. Evaluation

In this section we describe the experiment carried out in the context of the SmartSantander EU Project. This experiment, previously introduced in [44], aimed at evaluating how our proposed approach supports developers and users in taking advantage of Web of Things resources, and more specifically, in designing and developing web applications that mediate the end-user interaction with these resources.

The main objective of the SmartSantander project [45], [46] was to create a primary European test facility for the research and experimentation of architectures, services and applications for the IoT [46]. It consists of a real-world smart city deployment in Santander (Spain) that offers a large variety of sensors (e.g., temperature, CO, noise, light intensity and parking occupancy). Thanks to the SmartSantander facility, our experiment exploited real-world sensor data.

The experiment description is structured as follows. First, we introduce *MySmartCity*, a web application we developed exploiting our Java and JavaScript libraries. It served as a demonstration of a rich web application that can be implemented by developers on top of our middleware. Second, we show how IDN-Studio and IDN-Viewer enable a user to create, handle, compose and extend Web Resources that represent Santander's sensors facility. Then, we describe the testing activities aimed at evaluating user acceptance of implemented features. Finally, we report on further testing activities for evaluating the performance of IDN.

As an experiment prerequisite, we developed an Adapter for interacting with the push/pull HTTP-based APIs exposed by the Santander facility. The Adapter can execute an initialization routine that i) queries Santander APIs for sensor-related information and parses it, ii) for each sensor, generates a graph of Nodes (i.e. a Web Resource), and iii) commits it the middleware. These Nodes are stored in the system as metadata specifying their characteristics and relations along with a reference to the actual real-time measurement provided by Santander APIs. Once the initialization phase is over, sensors' data are exposed as a set of URI-addressable Nodes, accessible through REST APIs (called InterDataNet Santander APIs). In the normal operating phase, the Adapter retrieves up-to-date measurement values from the Santander facility when required by the Virtual Resource/Storage layer (upon a request). The Adapter also handles notifications sent by the Santander facility as some events of interest occur (e.g., a new measurement value, a new registered sensor).

A. MySmartCity application

MySmartCity is a web application that allows end-users to browse sensors, create new Virtual Sensors (such as an Heat Index sensor close the end-user's home) and store them in the system for later access. End-users can also create Web Resources representing Points of Interest (such as the one introduced in subsection III-C4), and enrich them with descriptive information and references to sensors in geographical proximity. Eventually, they can also share them with other end-users. A PoI can be public (a tourism location, a public transport station) or have meaning in a user's personal scope (e.g., user's home). Both Virtual Sensors and PoIs can be made public by the administrator, upon a request of the owner.

MySmartCity application logic runs both client side and server side. Apart form application specific details, MySmartCity contacts the IDN middleware to retrieve, update and manage Web Resources whether they are Sensors, Virtual Sensors or Points of Interest. To this end, the tools described in subsection III-C proved to be quite handy. Indeed, the Java and JavaScript libraries have been used for programming the server and client respectively, while IDN-Viewer has been used to display a human oriented representation of Points of Interest, enriching significantly the user experience. In this context, IDN-Studio played also a very important role in redesigning and adjusting Web Resources, created through the application GUI. In fact, to modify the model used for a certain information object (e.g. a PoI) a user can load it in IDN-Studio and visually edit it. With few clicks, modifications are committed to the architecture and the object is redesigned. Fig 6 provides a detailed outline of how the application interacts with the middleware. Programming and publishing tools are also represented.

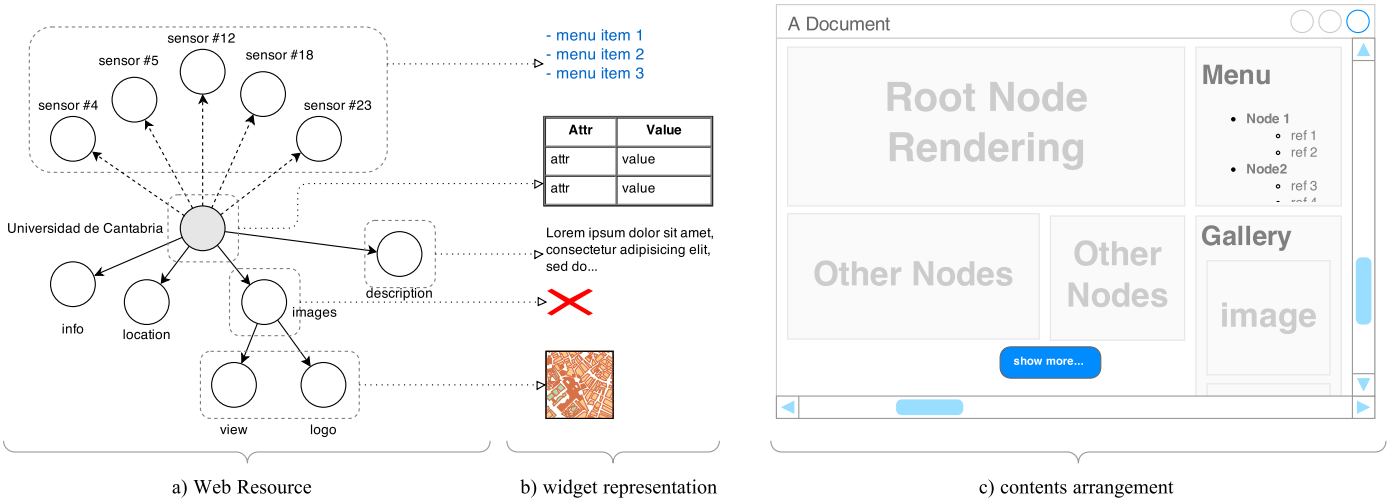

Fig. 5. Mapping rules in IDN-Viewer. a) A Web Resource representing the "Universidad de Cantabria" Point of Interest, including descriptions, location related data and near sensors. b) According to information formats, data are rendered as different types of widgets. c) Contents are finally arranged in the canvas to meaningfully fit the available space.

B. User-centric creation of Web of Things applications

MySmartCity application allows end-users to create and share new web contents, in a smart city scenario. However, the type of information they can add is predetermined by the developer who designed the Virtual Sensor and PoI information models and the related client- and server-side logic.

Hereinafter, we describe how IDN-Studio and IDN-Viewer tools enable a user-centric creation of a personal Web of Things application, by allowing end-users to freely modify and handle their custom Web Resources (i.e., Virtual Sensors, PoIs and also new resources defined by the user), publish them on the Web and access them through an ordinary web browser.

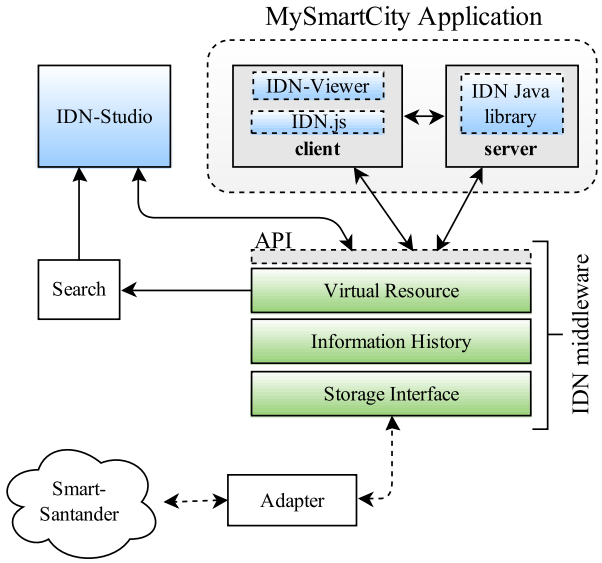

Fig. 6. The interaction of MySmartCity application with the middleware. All the framework components are visible: the IDN middleware, Java and Javascript libraries, IDN-Viewer and IDN-Studio. At the bottom is shown the Adapter component interfacing with the SmartSantander facility.

Through IDN-Studio, a user can access the model of a given Web Resource and browse the graph of Nodes by exploring Aggregation and Reference links. Leveraging the features provided by the tool, the user may modify the Web Resource's representation, for example by editing the ApplicationData, creating new links or deleting existing ones. The process of user-centric creation of Web Resources for a personal Web of Things is completed when the "write" button in the dashboard is clicked. This action triggers a sequence of REST operations for committing the changes to the middleware (e.g. PUT for modifications to an existing resource).

For instance, a user might be interested in creating a PoI that aggregates some noise and parking sensors close to his home. He might also create another PoI at a different location (for instance his relatives' home with surveillance and health sensors), and create a third Node that aggregates these two PoIs. The URI of this new Node can be seen as the root page of a personal web application created to easily access and monitor personal information. Indeed, the newly created and modified Web Resources are now accessible from standard web browsers and can be navigated as plain web pages that have been automatically generated by the IDN-Viewer plugin.

C. Evaluation with users

We conducted testing activities focused on evaluating user acceptance of implemented features. More specifically, our objective was threefold: i) evaluate the perceived ease of use and effectiveness of IDN-Studio and IDN-Viewer tools; ii) analyze potential disadvantages and qualify them (e. g., major usability problems and technical bugs); iii) gather subjective user satisfaction evaluation and possible suggestions for future

TABLE III. EVALUATION RESULTS FOR IDN-STUDIO AND IDN-VIEWER

Question	Definitely Yes	Yes	Neutral	No	Definitely No
Was IDN-Studio intuitive and easy to use?	20%	20%	60%	0%	0%
Was it easy to modify a PoI through IDN-Studio?	0%	40%	60%	0%	0%
Did you succeed in designing the target PoI?	0%	60%	40%	0%	0%
Were you satisfied by the graphical rendering of your personalized PoI on the web browser?	20%	60%	20%	0%	0%
Was the content of the PoI organized as you expected?	0%	80%	20%	0%	0%

improvements. Please note that Java and JavaScript libraries haven't been tested in this session, so we are interested in the *user* profile (see section II-C).

The evaluation session has been organized as a half-day workshop and was conducted with 20 users. Users were selected among our colleagues in the Department of Information Engineering at the University of Florence. The majority of them (80%) work in the field of Radar, Remote Sensing and Signal Processing fields and, to our purposes, can be considered average users, while a minority of them work in the fields of middleware and application protocols (10%) and are also expert web developers. None of them work in the field of IoT or WoT nor have previously used the tools under evaluation. All users are aged within 30-44 years.

In the first part of the workshop, a member of the experiment team presented the SmartSantander project, briefly introduced the tools to be evaluated and explained the meaning of Virtual sensors and Point of Interest resources.

In the second part of the workshop, the users were asked to use the MySmartCity application and IDN-Studio to customize an existing PoI through IDN-Studio (e.g. by enhancing its representation with a picture and historic information) and evaluate its rendering performed by IDN-Viewer.

User feedbacks have been collected through interviews and a questionnaire at the end of the evaluation session. Tab III shows feedbacks provided on the use of IDN-Studio for modifying and enriching the description of a PoI. Users succeeded in completing the assigned tasks, and 40% considered the application intuitive and easy to use. The majority of users provided positive comments on the visualization of the changes they applied to the PoI on the web browser. Some suggestions were made to enhance the interface with help texts to guide a first-time user in the use of the tool.

Although these evaluation activities are preliminary, results are encouraging and provided useful suggestions for correcting our applications. Since assessing user acceptance is considered a key topic to be investigated towards the Future Internet [47], a more in-depth evaluation of factors relevant to user acceptance, as those analysed by Lizcano et al. [48], would be worthwhile in the near future involving a wider and more heterogeneous user group.

TABLE IV. PERFORMANCE RESULTS FOR THE INTERDATANET MIDDLEWARE

Request Type	Web Resource size	Response Message Body (KB)	Request-response time (ms)
Creation	1 Node	0.722	192
	10 Nodes	5.6	2253
	30 Nodes	15.1	5606
	50 Nodes	25.2	10748
Retrieval	1 Node	0.772	73
	10 Nodes	5.6	625
	30 Nodes	15.1	1704
	50 Nodes	25.2	2865

D. Performance Evaluation

We also carried out testing activities to analyze the performance of the InterDataNet middleware implementation.

The test environment was deployed on two virtual machines, both equipped with 1 GB RAM and running the Debian 6 distribution. These VMs were installed on a HP ProLiant ML350 server with a two Intel Xeon quad-core processors, 17 GB RAM and Debian 6 OS. One VM hosted the InterDataNet Middleware, while the second VM hosted the MySmartCity, and IDN-Studio applications.

The objective was to assess the time required by the following two main operations: i) creation of a Web Resource (HTTP PUT request), ii) retrieval of a Web Resource (HTTP GET request). Each operation was performed on resources with varying size (i.e. composed by 1, 10, 30 and 50 Nodes) whose application content was stored in the InterDataNet middleware (no requests to external information systems were required). We iterated each test 10 times and measured the elapsing time from request delivery to response reception at server side.

Tab. IV shows the truncated mean value of the obtained measurements. It is worth observing that the time required for creating or retrieving a Web Resource grows linearly with the number of aggregated nodes. This behavior was expected since sets of Nodes belonging to the same Web Resource are managed sequentially by the middleware components in the current prototype implementation. Such behavior can be improved by implementing parallel processing tasks.

V. CONCLUSIONS

In this work, we have proposed a framework for modeling and turning smart things into web-addressable resources that can be accessed and handled through a REST uniform interface. The framework is composed of a general-purpose model for web resources and a middleware that implements RESTful services for handling these resources through open Web standards.

The proposed model is based on two simple relations (Aggregation and Reference), chosen for their relevance in the Web domain. Through these relations, a graph of browsable information nodes can be built and extended via a REST uniform interface. The node can contain locally stored information, data from external sources or content dynamically generated upon script-based processing.

We showed how through our framework digital representations of sensors, Virtual Sensors and Points of Interest can be designed and published on the Web. We provided users with a set of tools for creating, modifying and composing these resources, to build a Web of interlinked "smart" objects. Our target users are developers, who are provided with Java and JavaScript libraries for easing server-side and client-side programming, and non-expert users who are provided with the IDN-Studio and IDN-Viewer plugin for creating, composing and publishing new Web Resources for their own application needs.

We also discussed the compliance of the current middleware implementation with REST constraints, especially with HATEOAS.

We presented the results of an experiment carried out within the SmartSantander FP7 EU project for validating the proposed Web of Things framework with qualitative and quantitative evaluations. Users who tested the IDN-Studio and IDN-Viewer tools provided good overall feedbacks. Based on these results, some conclusive reflections have been drawn. First, the Web Resource model has proved to be effective in modeling, handling and combining smart things digital representations. Nodes can be created from scratch with ease and immediately made available by the middleware through IDN-Studio or be generated from external sources (sensors or sensors' gateways) through an adapter component. Thanks to the provided programming and Web publishing tools, these Nodes can be managed and further refined, by drawing aggregation and reference links with other Nodes, thus updating the corresponding Web Resource representation served by the middleware.

In the future, we are planning to take into account users' suggestions to improve the ease of use and intuitiveness of IDN-Studio. It will be also worth extending the IDN-Viewer JQuery plugin for supporting additional rendering models. Subsequently, a more comprehensive and in-depth assessment will be performed with a wider user base.

Another direction for future work would comprise the extension of our middleware with adapter components capable of interacting with existing semantic sensor web implementations [21] and the extension of our model to support semantic annotations specifications, e.g., Semantic Annotations for WSDL and XML Schema (SAWSDL) [49] and Semantic Annotations for REST [50]). These extensions are required to enable lifting and lowering mechanisms, opening up our resources graph to advanced reasoning, discovery and search services.

To conclude, it is worth noticing that our services will be kept running for at least three years after the completion of the SmartSantander project. Thus, programmatic access to the REST APIs and web-based access to the applications for accessing the existing Santander Smart City resources and creating new Smart City applications will be made available upon registration.

ACKNOWLEDGMENT

The authors would like to thank Mr. Andrea Failli and Mr. Luca Capannesi for their technical support.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] F. E. Group, "Services in the future internet," http://cordis.europa.eu/fp7/ict/ssai/docs/softwareconsultationreportfeb2011.pdf, february 2011, online resource.

[3] E. Wilde, E. C. Kansa, and R. Yee, "Web services for recovery. gov," *School of Information*, 2009.

[4] R. T. Fielding and R. N. Taylor, "Principled design of the modern web architecture," *ACM Transactions on Internet Technology (TOIT)*, vol. 2, no. 2, pp. 115–150, 2002.

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

[6] C. Pautasso and E. Wilde, "Why is the web loosely coupled?: a multi-faceted metric for service design," in *Proc. of the 18th international conference on World wide web*. ACM, 2009, pp. 911–920.

[7] D. Guinard, I. Ion, and S. Mayer, "In search of an internet of things service architecture: Rest or ws-*? a developers perspective," in *Mobile and Ubiquitous Systems: Computing, Networking, and Services*. Springer, 2012, pp. 326–337.

[8] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey," *Journal of Communications*, vol. 6, no. 6, pp. 424–438, 2011.

[9] S. Vinoski, "Restful web services development checklist," *Internet Computing, IEEE*, vol. 12, no. 6, pp. 96–95, 2008.

[10] I. Zuzak, I. Budiselic, and G. Delac, "A finite-state machine approach for modeling and analyzing restful systems," *Journal of Web Engineering*, vol. 10, no. 4, pp. 353–390, 2011.

[11] I. Zuzak and S. Schreier, "Arrested development: Guidelines for designing rest frameworks," *Internet Computing, IEEE*, vol. 16, no. 4, pp. 26–35, 2012.

[12] P. Adamczyk, P. H. Smith, R. E. Johnson, and M. Hafiz, "Rest and web services: In theory and in practice," in *REST: from research to practice*. Springer, 2011, pp. 35–57.

[13] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, 2000.

[14] L. Richardson and S. Ruby, *RESTful web services*. O'Reilly, 2008.

[15] C. Pautasso, O. Zimmermann, and F. Leymann, "Restful web services vs. big'web services: making the right architectural decision," in *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008, pp. 805–814.

[16] D. Guinard, V. Trifa, T. Pham, and O. Liechti, "Towards physical mashups in the web of things," in *Networked Sensing Systems (INSS), 2009 Sixth International Conference on*. IEEE, 2009, pp. 1–4.

[17] S. Mayer, D. Guinard, and V. Trifa, "Facilitating the integration and interaction of real-world services for the web of things," in *Urban Internet of Things (UrbanIOT 2010); Workshop at the Internet of Things 2010 Conference (IoT 2010), Tokyo, Japan*, 2010.

[18] B. Ostermaier, F. Schlup, and K. Romer, "Webplug: a framework for the web of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on*. IEEE, 2010, pp. 690–695.

[19] B. Christophe, M. Boussard, M. Lu, A. Pastor, and V. Toubiana, "The web of things vision: Things as a service and interaction patterns," *Bell Labs Technical Journal*, vol. 16, no. 1, pp. 55–61, 2011.

[20] A. Orestis, A. Dimitrios, and C. Ioannis, "Towards integrating iot devices with the web," in *Emerging Technologies & Factory Automation (ETFA), 2012 IEEE 17th Conference on*. IEEE, 2012, pp. 1–4.

[21] A. Sheth, C. Henson, and S. S. Sahoo, "Semantic sensor web," *Internet Computing, IEEE*, vol. 12, no. 4, pp. 78–83, 2008.

[22] C. Bizer, T. Heath, and T. Berners-Lee, "Linked data-the story so far," *International Journal on Semantic Web and Information Systems (IJSWIS)*, vol. 5, no. 3, pp. 1–22, 2009.

[23] A. Moraru, D. Mladenic, M. Vucnik, M. Porcius, C. Fortuna, and M. Mohorcic, "Exposing real world information for the web of things," in *Proc. of the 8th International Workshop on Information Integration on the Web: in conjunction with WWW 2011*. ACM, 2011, p. 6.

[24] D. Pfisterer, K. Romer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kroller, M. Pagel, M. Hauswirth *et al.*, "Spitfire: toward a semantic web of things," *Communications Magazine, IEEE*, vol. 49, no. 11, pp. 40–48, 2011.

[25] N. Freed and N. Borenstein, "Multipurpose internet mail extensions (mime) part one: Format of internet message bodies," 1996.

[26] G. T. Lakshmanan, F. Curbera, J. Freire, and A. Sheth, "Provenance in web applications," *IEEE Internet Computing*, vol. 15, no. 1, pp. 0017–21, 2011.

[27] Y. Theoharis, I. Fundulaki, G. Karvounarakis, and V. Christophides, "On provenance of queries on semantic web data," *Internet Computing, IEEE*, vol. 15, no. 1, pp. 31–39, 2011.

[28] L. P. Rothfusz and N. S. R. Headquarters, "The heat index equation (or, more than you ever wanted to know about heat index)," *Fort Worth, Texas: National Oceanic and Atmospheric Administration, National Weather Service, Office of Meteorology*, pp. 90–23, 1990.

[29] J. Bouttier, P. Di Francesco, and E. Guitter, "Geodesic distance in planar graphs," *Nuclear Physics B*, vol. 663, no. 3, pp. 535–567, 2003.

[30] T. Berners-Lee, R. Fielding, and L. Masinter, "Rfc 3986: Uniform resource identifier (uri): Generic syntax," *The Internet Society*, 2005.

[31] J. Clark, S. DeRose *et al.*, "Xml path language (xpath) version 1.0," 1999.

[32] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616 (Draft Standard), Internet Engineering Task Force, Jun. 1999, updated by RFCs 2817, 5785, 6266, 6585. [Online]. Available: http://www.ietf.org/rfc/rfc2616.txt

[33] M. C. Pettenati, L. Ciofi, F. Pirri, and D. Giuli, "Towards a restful architecture for managing a global distributed interlinked data-content-information space," in *The future internet*. Springer, 2011, pp. 81–90.

[34] F. Paganelli, S. Turchi, L. Bianchi, L. Ciofi, M. C. Pettenati, F. Pirri, and D. Giuli, "An information-centric and rest-based approach for epc information services." *Journal of Communications Software & Systems*, vol. 9, no. 1, 2013.

[35] L. Bianchi, F. Paganelli, M. Pettenati, S. Turchi, L. Ciofi, E. Iadanza, and D. Giuli, "Design of a restful web information system for drug prescription and administration," *Journal of Biomedical and Health Informatics*, 2013.

[36] R. Johnson, J. Hoeller, A. Arendsen, and R. Thomas, *Professional Java Development with the Spring Framework*. Wiley. com, 2009.

[37] C. Bauer and G. King, *Hibernate in action*. Manning Greenwich, 2005.

[38] E. Ort and B. Mehta, "Java architecture for xml binding (jaxb)," *Sun Developer Network*, 2003.

[39] H. Bergsten, *Java Server Pages*. O'reilly, 2003.

[40] D. Smiley and D. E. Pugh, *Apache Solr 3 Enterprise Search Serve*. Packt Publishing, 2011.

[41] J. Resig *et al.*, "jquery: The write less, do more, javascript library," *Disponivel em http://jquery. com/, Acesso em*, vol. 18, no. 04, p. 2009, 2009.

[42] D. Crockford, "Json in javascript," https://github.com/douglascrockford/JSON-js, nov 2010.

[43] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

[44] S. Turchi, L. Bianchi, F. Paganelli, F. Pirri, and D. Giuli, "Towards a web of sensors built with linked data and rest," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2013 IEEE 14th International Symposium and Workshops on a*. IEEE, 2013, pp. 1–6.

[45] J. M. Hernández-Muñoz and L. Muñoz, "The smartsantander project," in *The Future Internet*. Springer, 2013, pp. 361–362.

[46] L. Sanchez, L. Muñoz, J. A. Galache, P. Sotres, J. R. Santana, V. Gutierrez, R. Ramdhany, A. Gluhak, S. Krco, E. Theodoridis *et al.*, "Smartsantander: Iot experimentation over a smart city testbed," *Computer Networks*, 2013.

[47] G. Tselentis, *Towards the Future Internet: A European Research Perspective*. IOS press, 2009.

[48] D. Lizcano, F. Alonso, J. Soriano, and G. López, "End-user development success factors and their application to composite web development environments," in *ICONS 2011, The Sixth International Conference on Systems*, 2011, pp. 99–108.

[49] J. Kopecky, T. Vitvar, C. Bournez, and J. Farrell, "Sawsdl: Semantic annotations for wsdl and xml schema," *Internet Computing, IEEE*, vol. 11, no. 6, pp. 60–67, 2007.

[50] J. Lathem, K. Gomadam, and A. P. Sheth, "Sa-rest and (s) mashups: Adding semantics to restful services," in *Semantic Computing, 2007. ICSC 2007. International Conference on*. IEEE, 2007, pp. 469–476.

Federica Paganelli (M07) received a Ph.D. degree in Telematics and Information Society from the University of Florence, Italy, in 2004. She is a Senior Researcher at the National Interuniversity Consortium for Telecommunications (CNIT), Italy. Her research interests include context-aware systems, service-oriented computing and communication, and next generation networks.

Stefano Turchi graduated in Computer Engineering (M.S.) in 2010 at the University of Florence and received a Ph.D. degree in Informatics, Systems and Telecommunications in 2014 by the same university. Currently he is researching in the field of Distributed Architectures, Web Applications, Data Security and Data Privacy in distributed and granular data scenarios.

Dino Giuli (SM84) is Full Professor in Telecommunications at the Department of Electronics and Telecommunications of the University of Florence. Since 1996 he is Promoter and Scientific Coordinator of the Ph.D. program in Telematics and Information Society. His research activities are focused in the domain of Telematics and Environmental Monitoring Systems. He is AEI member and IEEE senior member.