Web-enabled processing of smart things resources for WoT applications

Stefano Turchi
dept. of Information Engineering
University of Florence
Florence, Italy 50139
Email: stefano.turchi@unifi.it

Federica Paganelli
National Interuniversity Consortium
for Telecommunications
University of Florence
Florence, Italy 50139
Email: federica.paganelli@unifi.it

Dino Giuli
dept. of Information Engineering
University of Florence
Florence, Italy 50139
Email: dino.giuli@unifi.it

Abstract—**The Web of Things is an active research field which aims at promoting Web standards and technologies adoption for handling smart things digital representations. In this context, many studies acknowledge that REST paradigm plays a decisive role, and this has prompted the emergence of systems for Web representation and management of real-world objects. However, objects exposure is only a first step. In fact, it is very likely that a client that exploits these representations to retrieve data from objects, also needs to process them. Typically, this processing is performed by an application which, in addition, has the burden of results exposure. In a data reuse perspective, these results are valuable: will they be re-exposed? If so, will they be exposed in a way enabling easy reuse? In this paper we present a graph model for RESTful publishing of Web resources in a WoT scenario and its implementation through the InterDataNet middleware. This graph model supports the definition of scriptable vertexes for processing other resources information. In this way, the paradigm is subverted: there is no need to set up an application to process smart objects information since the processing is externalized in a vertex of the resource model. Consequently, the RESTful exposure of results is utterly borne by the middleware. We also introduce an extension of the model to support event-driven capabilities.**

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

I. Introduction

Thanks to the dramatic reduction of technology costs, common use devices are getting increasingly smarter and more connected. This trend motivates the Internet of Things (IoT) concept that is a transformation of the Internet from a network of computers to a network of heterogeneous devices [1]. Similarly to what happened to computer networks, an application oriented drift is taking hold: the research community is starting to think about how existing technologies and standards including HTTP, URIs, etc. could be used to expose these smart objects on the Web, paving the way towards the Web of Things (WoT) vision. The WoT is deemed to simplify the access to smart things and foster the conception of novel, value added applications relying on the combination of conventional Web resources with those representing objects from the physical world (smartphones, appliances, street lighting, etc.) [2]. This has the big advantage of allowing the integration of smart things with the impressive amount of information resources and services already available on the Web while exploiting established technologies and best practices. In this context, the Representational State Transfer (REST) architectural style [3] is considered a reference paradigm for bringing sensors, and more generally smart things, into the Web [4]–[6]. Indeed, REST style defines a set of principles for designing distributed hypermedia applications by fulfilling scalability, simplicity and loosely-coupling requirements.

Unfortunately, although REST is considered much easier to use and suited for Web mashups than other paradigms (e.g., WS-*) [6], [7], most services claimed to be RESTful are not designed so diligently and neglect to abide by its principles [8]. Indeed, we face a systematic and widespread misunderstanding of REST basics [9] which complicates the road towards the effective realization of the WoT. Nevertheless, RESTful Web exposure of physical objects is just a first step. Objects are exposed to make their representations accessible to clients which, in all probability, are willing to do something with them. For instance, one could be interested in estimating the temperature in a location from some sensors displaced in an area, or rating different city zones depending on parking availability. Both use cases have as a prerequisite the exposure of smart objects, but the next step is data processing. Typically, this is performed by an *ad hoc* application, but this solution has some drawbacks. First, implementing a dedicated application is costly and second, if computation results can be published, it is highly desirable they are exposed following the aforementioned REST principles. And this delicate, yet non domain-specific operation is responsibility of the application.

In this work we present an approach for enabling the processing of Web resources and subsequent results exposure by a scriptable vertex, called Activity Node, defined for a RESTful, graph-based, information model. We also present an extension of the same information model for enabling event-driven capabilities. The application context is the WoT, but the model is general and Web resources are not limited to smart objects representations. This graph-based information model is part of InterDataNet [10]–[12], a framework for handling graphs of individually addressable information units with navigation, query, and composition support.

II. Related Work

In this section we briefly introduce the principles of REST architectural style and we analyze state of the art solutions about middleware for the Web of Things.

A. REpresentational State Transfer

REST was proposed by R. Fielding [13] as an architectural style for building large-scale distributed hypermedia systems. By the REST vision, objects handled by client-server application logic are modeled as resources. Key principles are:

1) *URIs as resource identifiers*. URIS are used by servers to expose resources. 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 verbs as follows: 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 must contain the sufficient information for fully understanding it, regardless of any previous request;
5) *Hypermedia As the Engine Of Application State (HATEOAS)*. In a hypermedia system participants transfer resource representations containing links which are used by clients to progress the interaction. [14].

Several studies indicate REST as an appropriate architectural pattern for Web of Things applications [4], [6], [15]. Guinard et al. [6] present a study based on both qualitative feedback and quantitative results from 69 developers who were asked to write IoT applications adopting REST and WS-*. Authors report that "participants almost unanimously found RESTful Web services easier to learn, more intuitive and more suitable for programming IoT applications than WS-*. The main advantages of REST are intuitiveness, flexibility, and the fact that it is more lightweight". In addition, a considerable number of volunteers agreed that REST was more suited for Web applications requiring to integrate Web contents.

B. Middleware for the Web of Things

The Web of Things is an active research area that, within the broader scope of the Internet of Things (IoT), focuses on the specific challenge of making smart things accessible and interoperable through open Web standards.

Guinard et al. provide a pioneering work contribution [16] by defining a set of RESTful services that expose sensor nodes as Web resources and link them together, hierarchically. In a later work, this approach is used in the AutoWoT project [17], a toolkit for the Web integration of smart devices that leverages i) a hierarchical resource model and ii) a tool for building Web servers that expose these devices. Upon Web resource modification, AutoWoT can semi-automatically regenerate the server logic. Our work shows some similarities with AutoWoT, such as the adoption of a general model and the Web exposure of smart devices. However, significant differences exist. First, our model defines two types of structural relations among Web resources (i.e., aggregation and reference), while the AutoWoT model relies only on a hierarchical one. 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). Second, the middleware can handle every type of resource that is compliant with our modeling primitives. In addition, unlike AutoWoT, our model offers both a scriptable Node for exposing processing results based on other Nodes contents, and a Node for managing event-driven activities (see section III).

Significant efforts are ongoing to develop a Semantic Sensor Web [18] and integrate it with Linked Open Data. SemSense [19] is a system that collects data from physical sensors and publishes them on the Web using semantic annotations. SPITFIRE [20] is an infrastructure that offers semi-automatic generation of semantic sensor descriptions as well as efficient search based on current sensors states. As discussed in section III-B, our model defines how information can be accessed and navigated, and does not aim at providing unambiguous specifications of concepts. However, we are aware that the use of semantic-based technologies can enable reasoning, efficient search, discovery and dynamic composition capabilities. To this purpose, future work will be devoted to build adapter components capable of interacting with existing semantic sensor Web implementations [20].

Finally, it is worth to mention the CityScripts experiment [21] carried out within the SmartSantander EU Project [22]. CityScripts is a Web application that offers the access to a personal workspace where users can compose public data from city sensors with public online data sources and personal data from social networks. Through the application palette, a user can create simple scripts defining basic workflows to connect available resources (for instance, an output of the sensor is passed as input to a social network service).

III. A Resource Model for the Web of Things

In this section we briefly present the InterDatNet information model and middleware (more details are available in previous works [10]–[12]) and discuss design and implementation of both resource-processing and event-driven capabilities.

A. The InterDataNet Middleware

InterDataNet Middleware (IDN) is a system conceived for RESTful Web exposure of information resources represented as graphs of individually addressable information units. More graphs can be browsed, queried and interlinked to create a growing structure of related information. IDN core modules are Virtual Resource, Information History and Storage Interface.

Virtual Resource exposes REST uniform APIs (IDN-APIs) used by applications to access and manage smart things Web representations called Web Resources. IDN-APIs is the only part of IDN that is visible to applications. As shown in Fig. 1, different Virtual Resource instances dialog as peers to realize a distributed graph of interconnected Nodes. Information History provides optional versioning capabilities. This is useful for tracking the history of a resource, supporting cooperation by providing branching and merging, and enforcing provenance by allowing thorough inspection of previous states. Storage Interface provides data and metadata persistence capabilities. IDN-specific Metadata (that determine Nodes relations and properties) are always persisted on the middleware side, while data can be native or external. Native data are the ones

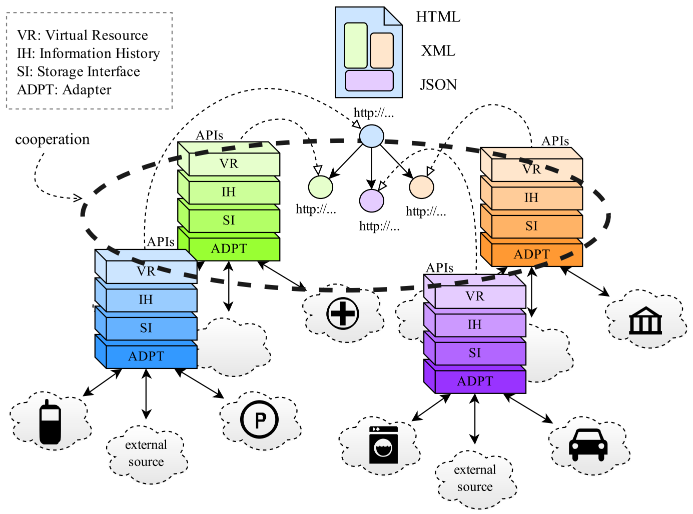


Fig. 1. The overall view of the InterDataNet architecture.

created under the middleware domain and are persisted in own repositories. External data are hosted outside the IDN boundaries instead, and require the Adapter component to be handled as native resources by the middleware.

Adapters are components that support the interworking of IDN with external data-sources (e.g. smart devices). Thus, an adapter contains custom intermediation logic for exposing data and services provided by legacy systems and devices according to our graph-based model. In this way, properties enabled by the IDN resource model can be applied to information originating from outer sources. Adapter is made of four components: 1) a Notification Manager, which enables a push/pull notification service and manages the Adapter subscription to a data-source; 2) a Transformer, 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; 4) a Translator, which translates requests coming from the IDN-APIs in a format supported by the data-source.

B. The InterDataNet Information Model

The InterDataNet Information Model (IDN-IM) is the set of rules that drives the representation of entities in IDN. Web Resource is the first class entity and is defined as a graph of uniquely URI-addressed information pieces, called Nodes. Nodes have a content and a set of properties such as privacy, licensing, provenance, etc. From Nodes may depart structural (Aggregation and Reference) and non-structural (Active and Notification) edges. Within a graph, every subset of vertexes connected via Aggregation edges is a Web Resource. Different Web Resources can be combined to form a richer Web Resource. More precise definitions are given in the following.

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] is used as a Node unique identifier.

Definition 2. An **Aggregation link** $l_{Aggr} \in L$, L being the set of edges, is a directed edge between two Nodes that

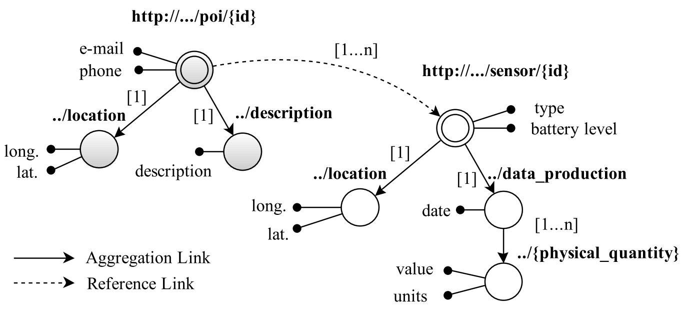


Fig. 2. Modeling of a PoI Web Resource referring to a number of different sensors. Nodes are represented as circles while black pins represent content data.

represents a transitive container-content relation. The conveyed meaning is: the originating Node aggregates, therefore contains, the destination Node. The Aggregation links are used for *intra*-Web Resource relation.

Definition 3. A **Reference link** $l_{Ref} \in L$ is a directed edge between two Nodes that represents a pointer towards a referred resource. To better understand the Reference link role, it could be somehow compared with the HTML `href` attribute. The Reference links are used for *inter*-Web Resource relation.

Definition 4. A **Web Resource** is a digraph $R(N, L_{Aggr})$ 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 source vertex $n_R \in N$ called Root Node, i.e. a vertex for which $\deg^{-}_{Aggr}(n_R) = 0$.

A Web Resource is exposed by a uniform interface supporting CRUD operations mapped onto HTTP verbs: GET reads the Web Resource state; PUT updates the state of an existing Web Resource or creates it; DELETE deletes a Web Resource. In our model, content negotiation is available for Nodes (the implementation currently supports XML, JSON and HTML), ETags [23] are used to prevent conflicts during updates, and HATEOAS [13] is supported by including in the representation links towards the next related resources, provided with instructions for the following interaction. Fig. 2 shows the modeling of a Point of Interest (PoI) Web Resource which refers to a number of different sensors.

C. Enabling in-resource smart things processing

As described so far, the IDN-IM serves the purpose of *representing* smart objects as graph of individually addressable Web resources. For this to be functional, representations need to be used by clients and often this means being subjected to a *processing*. Generally, this is performed by a purposely implemented application that i) identifies the needed WoT resources, ii) extracts the proper information from their representations, iii) performs a processing with it, and possibly iv) re-exposes the computation results. This approach has two main drawbacks: first, implementing and maintaining a dedicated Web application is costly; second, although it is convenient that results are made available according to a RESTful (and hopefully shared) model, this is very unlikely to happen. In fact, custom applications tend to use *ad hoc* strategies and formats, not bothering with broader issues such as data reuse. To overcome these problems, we propose a different strategy. We introduce a special type of vertex, called Activity Node, which is provided with a script. When the Node is served by the middleware, the script is executed behind the scenes and the result is included as a Node content. This procedure is completely transparent to clients which see only a Node representing the result of the desired computation. Being a Node, an Activity Node is accessed with the same uniform REST interface and it can be requested, interlinked and shared as a Web Resource as usual. Activity Nodes use Active links to determine computation dependencies. We introduce Activity Nodes by extending the Definition 1.

Definition 1.1. A **Node** is a tuple $N = (C', P)$ where P has the same meaning given in Definition 1. and $C' = C + s(i_0, \ldots, i_j)$ is a set of contents that partially depends by a script s having $j+1$ inputs.

From now on, for the sake of simplicity, we call Activity Nodes those Nodes for which $s(i_0, \ldots, i_j) \neq 0$.

Definition 5. An **Active link** $l_{Act} \in L$ defines a direction in a processing flow. An Active link always starts from an Activity Node and points to a Node (whether it is an Activity Node or not), defining the dependencies to fulfill *before* executing the script s.

To clarify, a typical application of the Activity Node is presented. A major has invested in smart technologies by installing different sensors in a city. Let's assume that data from these sensors are exposed as Web Resources. In order to monitor the citizens' quality of life (QoL), the major hires a team of researchers who find a strong correlation of wellness with pollution, noise and traffic. To define the intervention strategy, the administration needs to monitor the QoL level in different areas. By leveraging IDN, they decide to create several QoL location-dependent sensors by using Activity Nodes whose dependencies are near pollution, noise, and traffic sensor Nodes. The script that compute the QoL value could be a mean $w(p, n, t, d)$, of pollution (p), noise (n) and traffic (t) measurements, weighted with the distance (d) of the QoL sensor from other sensors. When the QoL sensor resource is requested, the corresponding Activity Node is inspected, pollution, noise and traffic Nodes dependencies are resolved, measurements are extracted and entered as inputs of the script. The computation executes and the output $o = w(p, n, t, d)$ is used to fill the content section of the same Node. Finally, the QoL sensor Node is returned with the computed value in place as a Web Resource (see Fig. 3).

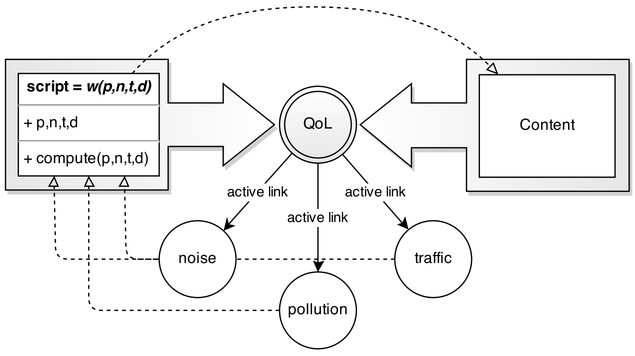


Fig. 3. The Quality of Life sensor implemented as an Activity Node.

1) Activity Node requirements and constraints: The first problem to face while dealing with processing is inputs and outputs definition. If they are not clearly defined, it will be hard to provide the script with compatible data and properly interpret its output. This is the reason why an Activity Node supports strong typed input and output definitions borrowed from XML Schema built-in data types [24], which is a the well-accepted and mature standard. The externalization of the script can be also envisioned to put it in a global space, fostering information reuse. In this way, many different Activity Nodes can import the same algorithm and apply own inputs. This concept motivates the creation of a trusted source providing stable and optimized code. Future works in this direction are planned.

The content of a Node can be any kind of information object with any granularity level, according to publisher's prescriptions. When the Node referred by an Active link has some unstructured data, the script is forced to process them as a whole. Conversely, when they are structured, the script can extract the inputs leveraging this underlying organization. To this end, it is crucial to have a system for addressing both Nodes and information grains contained in the data structure and URIs and XPath technologies are naturally fitted for the purpose. Since contents representation format is not constrained to XML, a strategy for addressing *generic* structured data is needed. To this end, we adopt a subset of XPath deprived of XML-specific elements such as namespaces or attributes. Nevertheless, if the data schema is unknown, it won't be possible to define the expression to target the information grain(s). This consideration calls for the definition of a Node content schema. It is not mandatory for the publisher to declare such schema, but it is strongly advised since it is an enabling factor for cooperation and reuse promotion.

Even though more Activity Nodes can be chained to achieve complex computation flows, the processing of scripts is kept completely isolated and independent. However, the Activity Nodes chaining introduces the problem of the infinite resolution. In other words, if there is any cycle in the dependency relations, the retrieval procedure will never end. This problem demands a cycle detection algorithm whose requirements can be stated as follows: i) if a dependency cycle exists, it will be detected, ii) if a dependency cycle is detected, the system is put in a safe state, and iii) if more dependency cycles exist, the detection of the first cycle should put the system in a safe state (optimization requirement). Basically, the rationale of the cycle detection algorithm consists in marking retrieval requests with Node-specific tokens. The algorithm is executed by a Virtual Resource instance that receives a request for a Node. As this happens, the instance retrieves the token list from the request and performs a lookup for the Node token. If it is found, a cycle is detected. Fig. 4 shows the algorithm pseudo-code.

More in detail, dependencies propagate through Active links, so the guard at line 3 stops the detection if they are missing. At this point, a token related to the requested Node is deterministically generated (e.g. performing an MD5 of the Node URI) and checked against the *tokenList* (line 5). If the token is found, the cycle is detected. Otherwise, no cycle is detected and the token is added to the list before issuing a new request (line 8).

```
 1: $node \leftarrow$ the current node
 2: $tokenList \leftarrow$ the list of tokens passed as an argument
 3: **if** $node$ is an Activity Node **and** has Active links **then**
 4:     $token \leftarrow$ a deterministically generated value
 5:     **if** $tokenList$ contains $token$ **then**
 6:         **return true**
 7:     **end if**
 8:     add $token$ to $tokenList$ and issue a new request for the
        succeeding Active link, passing $tokenList$
 9: **end if**
10: **return false**
```

Fig. 4. The pseudo-code of the cycle detection algorithm used to prevent the infinite resolution problem.

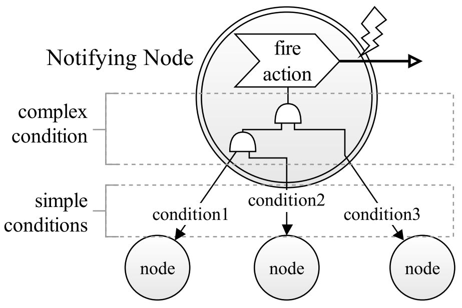


Fig. 5. Modeling of Notifying Node. All the edges directed from the Root Node to other Nodes are Notification links. The AND gates just represent some configurable logical connectives.

In addition, the implementation of the algorithm must comply with the *stateless interaction* principle (see subsection II-A). This principle is critical to make the algorithm lightweight and resource-saving. Indeed, if the server stores the state of every request, as they increase, the system will incur in a resource shortage and ultimately in a crash. In our solution, the transfer of the token list is implemented via a dedicated HTTP header (in compliance with the HTTP standards) and token generation is executed on the fly by the Virtual Resource instance authoritative for the requested Node. Thus, the information critical for the algorithm execution is not gathered server-side.

D. Notification and actuators

The Activity Node is a pull-based strategy for defining virtual resources. However, in order to address real-world requirements, push support is mandatory. Practically, it is desirable to have a model able to support event-driven scenarios such as switching on the air conditioner when the temperature in a room exceeds a threshold. To this end, we extend Definition 1.1 and we introduce the Notification link.

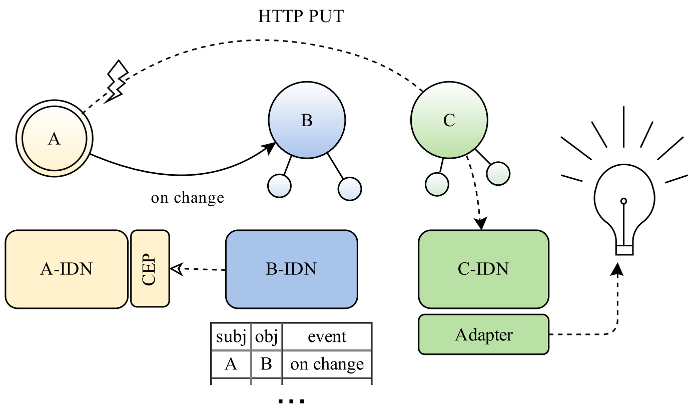
Definition 1.2. A **Node** is a tuple $N = (C', P, t)$, where C' and P are the same as defined in Definition 1.1, while t is a trigger function $t(d_0, \dots, d_k)$ that executes when $k+1$ conditions are satisfied.

Definition 6. A **Notification link** $l_{Not} \in L$ is a directed edge that defines a condition concerning the pointed Node to be satisfied for the trigger function to be executed.

From now on, for the sake of simplicity, we call Notifying Nodes those Nodes for which $t(d_0, \dots, d_k) \neq 0$.

Fig. 5 shows the modeling of a Notifying Node: along the edges basic events are defined such as *on deletion* or *on change*, that can be further composed with logical connectives to form a complex condition. When the condition is met, a script is executed and an action is fired. Such action can be directed to a different Web Resource (changing its state), to the Node itself (changing own state) or to an external resource.

Notification is implemented using the publish/subscribe pattern [25]. When a Notifying Node is created or updated, it is inspected by the authoritative middleware instance to retrieve: i) the URIs of the Nodes referred by Notification links and ii) both simple and complex conditions. For each URI, the middleware registers itself as a subscriber for the event specified by the simple condition to the IDN instance responsible for that Node. Fig. 6 clarifies the concept: `A` is a Node under the authority of a middleware instance (`A-IDN`) and has a Notification link to Node `B` which belongs to a different one (`B-IDN`). During the creation/update phase of `A`, `A-IDN` inspects it to find Notification links. When the link to `B` is found, `A-IDN` subscribes `A` to `B-IDN`, for a modification event occurred on `B`. If the fire condition is complex (like the one depicted in Fig 5), a Complex Event Processing [26] (CEP) service is used. When `B` changes, `B-IDN` sends a notification to `A-IDN` that processes the event with a CEP configured with the contingent complex condition. If the condition is met, `A-IDN` retrieves `A` and executes the contained notification script. In this case, the script causes the update of the external Web Resource `C`, performed through an HTTP PUT request. As this happens, the state modification of `C` is communicated to an Adapter which, in turn, interprets it and performs the actions required to turn the light on.



subj	obj	event
A	B	on change

...

Fig. 6. A light bulb is turned on as a consequence of the modification of the `B` Node. `A` has a Notification link pointing to `B`, i.e. `A-IDN` is registered as a subscriber to `B-IDN`, for the modification of `B`. As this occurs, `A-IDN` receives a notification and executes the script contained in `A` which performs an update of `C`. `C` is the Web Representation of a light bulb and its state transition is mediated by the Adapter which sends the command to the physical object.

We make two considerations: first, Notification dependencies can not be circular to avoid infinite triggering loops. To this end, a cycle detection algorithm similar to the one described in section III-C1 can be used. Second, notification is available for all Nodes, including Activity Nodes, and this brings some complexity. Indeed, an Activity Node state depends on a computation based on inputs. Therefore, before its execution, it is not possible to know the Node state that may trigger some push activities. This means that when a Notification link is directed to an Activity Node, the middleware must subscribe to all the Activity Node's dependencies. When one of these changes, the Activity script must be executed to check whether the firing condition is met or not.

IV. Conclusion

A consistent number of researchers indicate REST as the style for managing WoT resources but, unfortunately, the thorough application of its principles is often disregarded by Web developers. Starting from this "lesson learned", we propose an approach for simplifying RESTful Web exposure of smart objects by designing a scriptable vertex for a graph-based Web Resource representation offered by the InterDataNet system. Indeed, given proper representations of smart objects, it is very likely that their contents will be taken and processed to get some valuable results. Typically, this is performed by an *ad hoc* application which is also responsible for the proper results re-exposure. However, applications are costly to design, implement and maintain and developers tend not to bother with broader scope problems such as data reuse, making the proper exposure of computation results a very unlikely event. To overcome these problems, we took the processing out of the application by designing a scriptable vertex which is automatically exposed by InterDataNet, according to REST principles. Moreover, to cope with real world scenarios, we introduced a methodology for extending the model with event-driven capabilities based on dedicated Notification edges.

Future works include the externalization of scripts used by Activity Nodes to promote high quality code reuse. Moreover, a deeper study of scalability issues caused by the combination of event-driven capabilities with scriptable Nodes and a more refined strategy for defining events. In addition, an integration of the Information Model with semantics is ongoing.

Acknowledgment

The authors would like to thank Mr. Ivan Zappia and Mr. Giacomo Niccoli for fruitful discussions on the Activity Node subject and Mr. Luca Capannesi for 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] E. Wilde, E. C. Kansa, and R. Yee, "Web services for recovery. gov," *School of Information*, 2009.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[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] V. Trifa, D. Guinard, and S. Mayer, "Leveraging the web for a distributed location-aware infrastructure for the real world," in *REST: From Research to Practice*. Springer, 2011, pp. 381–400.

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

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

[20] 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.

[21] A. Badii, D. Carboni, A. Pintus, A. Piras, A. Serra, M. Tiemann, and N. Viswanathan, "Cityscripts: Unifying web, iot and smart city services in a smart citizen workspace," *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, vol. 4, no. 3, pp. 58–78, 2013.

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

[23] 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

[24] P. V. Biron and A. Malhotra, "XML schema part 2: Datatypes second edition," W3C, W3C Recommendation, Oct. 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/.

[25] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM Computing Surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[26] D. C. Luckham and B. Frasca, "Complex event processing in distributed systems," *Computer Systems Laboratory Technical Report CSL-TR-98-754. Stanford University, Stanford*, vol. 28, 1998.