WoT Store: Managing Resources and Applications on the Web of Things

WoT Store: Managing Resources and Applications on the Web of Things

Luca Sciullo[a], Lorenzo Gigli[a], Angelo Trotta[a], Marco Di Felice[a,1]

[a]*Department of Computer Science and Engineering, University of Bologna, Italy*

Abstract

The chaotic growth of the Internet of Things (IoT) determined a fragmented landscape with a huge number of devices, technologies and platforms available on the market, and the consequential issues of interoperability on many system deployments. The Web of Things (WoT) architecture recently proposed by the W3C consortium constitutes a novel solution to enable interoperability across IoT platforms and application domains. At the same time, in order to see an effective improvement, a wide adoption of the W3C WoT solutions from the academic and industrial communities is required; this translates into the need of well-defined and complete support tools easing the deployment of W3C WoT applications. In this paper, we meet such requirement by proposing the WoT Store, a novel platform for managing and easing the deployment of Things and applications on the W3C WoT. The WoT Store allows the dynamic discovery of the resources available in the environment, i.e. the Things, and to interact with each of them through a dashboard, by visualizing their properties, executing commands or observing the notifications produced. In addition, similar to popular app stores, the WoT Store allows the search and execution of third-party WoT applications that interact with the available Things again in a seamless way. We validate the operations of our framework with two evaluation studies. First, through a small-case testbed, we demonstrate the Thing discovery and the possibility to run WoT applications that orchestrate the operations of multiple, heterogeneous Wireless Sensor Networks (WSNs). Second, through a mixed real/simulated large-scale crowdsensing scenario, we demonstrate the scalability of the platform, and the possibility to aggregate and visualize the data-streams produced by the WoT applications with minimal efforts for the users.

[1]Corresponding author: marco.difelice3@unibo.it

Keywords: Internet of Things (IoT), Web of Things (WoT), Interoperability, Software platform, Resource management, Performance Evaluation.

1. Introduction

1.1. Context

Since the beginning, the Internet of Things (IoT) has been presented as a novel networking paradigm consisting of a huge base of connected devices that are able to produce and exchange data, and to enable new services thanks to the seamless interaction among physical and virtual components [1][2]. At the same time, the presence on the market of heterogeneous software platforms, network protocols and Machine-to-Machine (M2M) technologies [3], as well as the creation of proprietary silos often leaded by big vendors, have partially changed the vision of the IoT as a global interconnected and self-organizing system. Indeed, the lack of interoperability among heterogeneous platforms and devices has been indicated as one of the main issues of the IoT [4], since it might introduce additional costs for the system implementation, and additional complexity for the re-use of existing solutions on different contexts. From another perspective, the interoperability can be considered a remunerative research challenge [5]: a recent report from McKinsey quantifies in 40% the additional IoT value that can be unlocked when achieving full interoperability among heterogeneous IoT systems [6]. Solutions to support interoperability on IoT scenarios have been largely investigated by the academic research as well as by European projects: we cite, among others, the projects Arrowhead [7], BIG IoT [8] and Wise-IoT [9] that proposed reference platforms to connect and deploy cross-domain IoT applications. Interoperability can be supported at four different levels (device, network, semantic and cross-domain), as extensively surveyed in [4]. Device and network solutions (e.g. the 6LoWPAN stack [10]) face the existing fragmentation by defining common addressing, routing and data-exchange rules. Vice versa, semantic solutions focus on the definition of a common data model used by the IoT interacting components; to this aim, several IoT-related ontologies have been proposed [11]. Cross-domain solutions (e.g. [12]) represent the most general way of supporting interoperability on the IoT: rather than focusing on protocols and data, they aim to define common interfaces that should be implemented by the IoT components, in order to be discoverable and queryable. A key contribution in this field is provided by the W3C Web of Things (WoT) research group, which is converging on the definition of a reference architecture to enable interoperability across IoT platforms and application domains [13]. In the W3C perspective, a Thing can indicate both a virtual or

a physical device and each Thing is associated to a Thing Descriptor (TD), i.e. a sequence of meta-data modeling the interaction patterns (properties, actions, and events) as well as the security and access protocol information. Moreover, the TDs are encoded in JSON-LD [14], i.e. knowledge about Things can be represented in a machine-understandable way. Several architectural patterns (e.g. Thing-to-Thing, Thing-to-Cloud, Thing-to-gateway etc) are defined in the W3C draft [13], so that most of existing IoT scenarios can be easily adapted to a WoT deployment, hence addressing the interoperability issues in a straightforward way.

1.2. Research questions

The success of the W3C WoT initiative strongly depends on its wide acceptance from the academic and industrial communities, as well as from the end-users. At present, the existing WoT implementation frameworks (e.g. [15] [16]) provide several low-level functionalities for the Thing modeling and creation; however, their usage requires a solid knowledge of the WoT standard and coding skill, hence they are not easily accessible from the non technical personnel. The literature on WoT is quite scarce, and mainly limited to proof-of-concept applications [17][18][19][20]. Hence, we register the need of service tools (the so-called Software Ecosystem (SECO) [21]) that can facilitate the adoption of the W3C WoT technology on existing and novel IoT scenarios. In addition, IoT/WoT deployments are often characterized by dynamicity, e.g. the need of adding/removing new devices, of re-defining the devices' behaviour (e.g. software updates), of tuning system parameters, just to cite few examples. A straightforward research challenge is how to support the IoT deployment reconfiguration seamlessly, i.e. avoiding the manual intervention on the field. In this paper, we address both the research questions (*RQ*s) mentioned so far, i.e.: (*RQ1*) how to ease the discovery and the management of WoT resources (e.g. Things), in both private and public environments? (*RQ2*) how to support the dynamic reconfiguration of the WoT scenario, e.g. the deployment of new WoT resources or the interconnection among the existing ones, while minimizing the need of manual configuration (for system administrators) and coding (for programmers)?

1.3. Contributions

Our solution to the *RQ*s above is constituted by WoT Store, a novel platform for managing and deploying resources on the W3C WoT. The WoT Store is not an implementation of the W3C WoT (like [15]), rather a service platform that can be installed on top of it, adding novel functionalities for the end users.

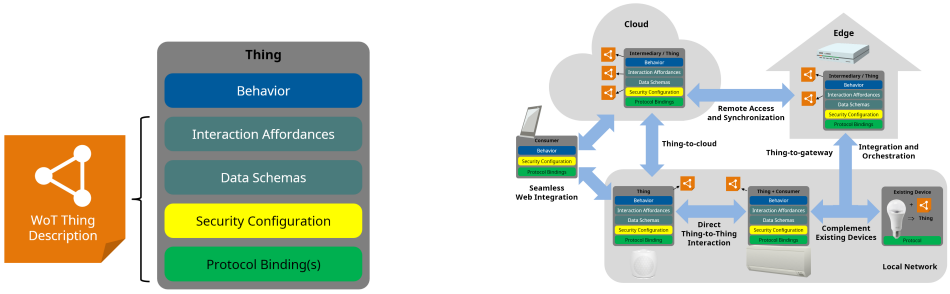
Indeed, the WoT Store allows the dynamic discovery and managing of the active Things available on a public or private deployment; through the Web dashboard, the users can search and list the Things in the scenario, monitor their properties and events, and execute their commands, without any change to the IoT layer. In addition, thanks again to the fact that the Thing interfaces are well defined and non-ambiguous, the WoT Store allows the management of applications that can make use of the available resources: in this sense, our platform recalls the operations of popular software repositories used for the mobile applications. We describe here the complete WoT Store platform, by extending the preliminary works [22][23] in terms of components and evaluation results. More in details:

- We present the WoT Store main functionalities, architecture, implementation and use-cases. The framework is micro-services oriented, with three main modules available, i.e: the Things Manager, the Application Manager and the Data Manager. We illustrate the operative flow from the Thing discovery and management, to the installation and execution of the applications till the aggregation and visualization of the data produced.

- We validate the operations of the WoT Store in a real-world testbed composed of three Wireless Sensor Networks (WSNs) mapped on different wireless access technologies (Wi-Fi, BLE and Zigbee). Specifically, the analysis provides evidence of the dynamic discovery of Things/sensors and highlights the possibility to deploy WoT applications orchestrating the sensing operations, regardless of the M2M technology used by each sensor.

- We test the scalability of the WoT Store on a mixed real/simulated large-scale IoT application, i.e. a pedestrian crowdsensing system; each Thing is associated to a simulated mobile smartphone, providing the sensing values and the positions over time on a urban scenario (the downtown area of Bologna). We demonstrate the capability of the Data Manager to aggregate and visualize the data-streams originated by the WoT applications in two formats (time-series and geographic data).

The paper is structured as follows. Section 2 discusses the recent works on the WoT and provides a brief review of the W3C WoT standard. Section 3 introduces the WoT Store framework; the main components and the implementation are described in Section 4 and 5, respectively. Section 6 presents the results of the WoT Store on two evaluation studies (testbed and simulation). Conclusions and future works are discussed in Section 7.



(a) (b)

Figure 1: The abstract architecture of a Web Thing is depicted in Figure 1(a); examples of WoT deployments and pattern of interaction among Things are shown in Figure 1(b).

2. Related Work

The first works on the WoT are prior to the W3C initiative, and focused on how to apply Web paradigms and protocols (e.g. REST, HTTP) on IoT scenarios (e.g. [24][25]). We do not consider non W3C-compliant approaches in the paper, since they are not aligned with the scope of this work; interested readers can refer to [26] for a complete survey on generic non-W3C WoT frameworks.

2.1. W3C WoT Standard

The W3C WoT group started its activities on 2015 with the goal of defining a reference set of standards enabling interoperability among different IoT systems [13]. The core of the proposal is the definition of the Web Thing and of the Thing Descriptor (TD). Generally speaking, a WoT Thing indicates any entity that can be semantically represented. Using the W3C words: "*A Thing is an abstraction of a physical or a virtual entity whose metadata and interfaces are described by a WoT Thing Description*" [13]. A Web Thing has four architectural aspects of interest: the Interaction Affordances, the Security Configuration, the Protocol Bindings, and its Behaviour, as depicted in Figure 1(a). The first three aspects are made explicit by the TD, i.e. a sequence of standardized, machine-understandable meta-data that allows consumers to discover and interpret the capabilities of a Thing. More in details:

- The Interaction Affordances provide an abstract model of how Consumers can interact with the Thing, in terms of properties (i.e. the state variables of the Thing), actions (i.e. commands that can be invoked on the Thing) and events (i.e. notifications sent by the Thing).

- The Protocol Bindings define the mapping between the abstract Interaction Affordances and the network mechanisms (e.g. the protocols) that can be used to work with the Thing.
- The Security Configuration defines the mechanisms to control the accesses to the Interaction Affordances.

The TD can be encoded by using the JSON-LD language [14]. Finally the Behaviour constitutes the implementation of the Thing, including the interaction Affordances, e.g. the code of its actions. All the blocks above are implemented within a software runtime named *Servient*, which can indifferently act as a Server or as a Client. In the first case, the Servient is said to host and *expose* Things, i.e. it creates a run-time object serving the requests towards the hosted Things, like accessing the exposed properties, actions and events. In the second case, the Servient is said to *consume* Things, i.e. it processes the Thing Description (TD), generates a run-time representation called Consumed Thing, and makes it available to those applications that are interacting with the remote Thing. Figure 1(b) depicts the abstract W3C architecture for the WoT, including the blocks inside each Thing and examples of interaction among them; interested readers can refer to the latests draft of the standard [13] for the complete list of interaction patterns. The Node-WoT [15] framework is the reference implementation (in JavaScript) of the WoT architecture certified by the W3C. However, consequently to the increase in popularity, additional implementations for different programming languages are expected to be released. The WoTPy is an example for the Python language [16].

2.2. W3C WoT Research papers

Due to the recent appearance of the W3C WoT standard, few real-world applications and tools have been proposed so far in the literature. A demo showing the possibility to query a W3C WoT sensor device from a mobile phone is sketched in [17]. In [18], an interesting application of the W3C WoT architecture to the automotive industry is described; more specifically, the authors illustrate how to describe the car data with a semantic ontology, and how to make them available to external applications through the W3C WoT interaction patterns. Security risks and vulnerabilities presented by the WoT metadata are discussed in [19]. Versioning mechanisms for the TD are proposed in [20]; here, the authors discuss the relevance of life cycle mechanisms on Industry 4.0 scenarios, where there might be a constant change in the data structures exposed by each Thing. In [27] an ontology-driven discovery for the W3C WoT architecture is proposed, but few

implementation details are provided. The WoTify [28] is a W3C WoT platform that allows users to search for WoT projects, download or contribute to shared ones. As stated in the Introduction, the WoT Store is not an implementation of the W3C WoT standard, rather a support tool hence it must be considered complementary and not alternative to the frameworks in [15][16]. Moreover, differently from the cited works [17][18][20], it does not focus on a specific WoT application; instead, it provide generic, Thing-related functionalities that can be useful on different use-cases, as discussed in Section 4.4. The most similar work is [28], since -at the best of our knowledge- it is the only other study proposing a W3C WoT service tool: however, the goals are different from our framework, since [28] focuses on the discovery of WoT projects from developers' perspective (e.g. for code sharing and maintenance), while our proposal addresses the discovery of Things from users' perspective (for resource control and management).

.

3. WoT Store: Overview and Functionalities

The WoT Store framework is designed to be micro-services oriented, with the possibility to easily load/unload new modules based on the specific user requirements. We can group the WoT Store services into three main areas, i.e.:

1. *Things Management*: the WoT Store allows to discover the available Things in the environment, and/or to select a subset of them according to user-defined, semantic criteria (e.g. the location). Through the GUI, the user can interact with each of them according to its TD, i.e. display properties, execute actions or observe the notifications of events.
2. *Application Management*: beyond monitoring the existing resources, the WoT Store allows to perform changes to the actual WoT scenario, by instantiating new Things or executing applications involving the interaction among the available Things. Here, the WoT Store acts like an application repository: via semantic queries, the users can search for software matching specific criteria (e.g. the compatibility with the actual devices). In addition, the application can be executed on any of the Servient registered to the WoT Store, again minimizing the manual configuration efforts. We further distinguish between Thing Applications (TAs) and Mash-up Applications (MAs), as better detailed in the next Section.
3. *Data Management*: the WoT Store allows to process and aggregate the data produced by a WoT application, by providing proper facilities in order

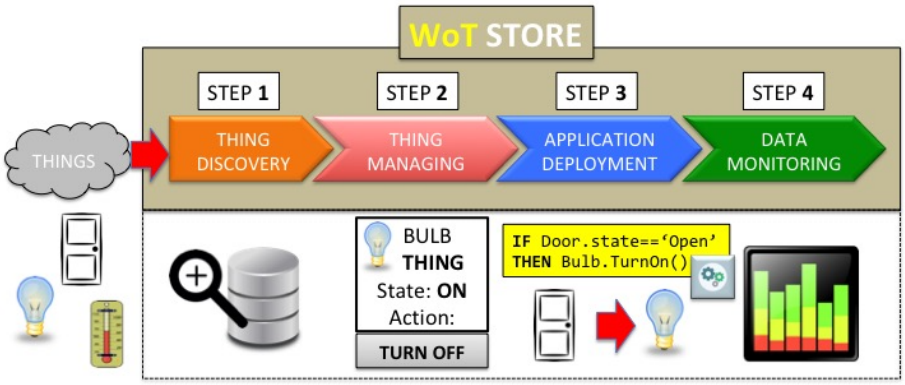


Figure 2: Main functionalities and sequence of operations with the WoT Store.

to gather the *data-streams*, aggregate them and visualize the results on the Web dashboard.

Figure 2 shows how the services can be used in pipeline on a typical use-case. Let us consider a generic IoT environments with heterogeneous devices in terms of communication protocols, data formats and implementations. No assumption is made regarding them; they could be native W3C WoT compliant devices or they could have been mapped into the W3C WoT ecosystem by means of any of the architectural patterns shown in Figure 1(b). In any case, we assume that the corresponding Web Things have been deployed on some Servient. First of all, through the Discovery Service, the Things are registered to the WoT Store tool; they are now searchable and displayable from the Web dashboard. For instance, let us assume the presence of a Thing connected to a Smart Bulb device; as soon as the Thing is connected to our platform, the user can perform the action `turnOn` or `turnOff` directly from the GUI. As next step, the user might be interested in downloading applications from the Store through which it might implement coordinated or autonomic behaviours involving multiple Things at the same time. For instance, assuming that the Web Things associated to a Smart Bulb and a Smart Lock are both available and active, the user might run an application that issues the action `turnOn` each time the Lock is generating an `open` event, in an automatic way. Behind the actuation, some applications might also produce streams of data that can be relevant for the context. In the previous example, the user might monitor the sequence of decisions performed by the application (i.e. the `turnOn` or `turnOff` actions) over a temporal window. This is possible by connecting the application to the proper data facilities of the WoT Store, hence closing the

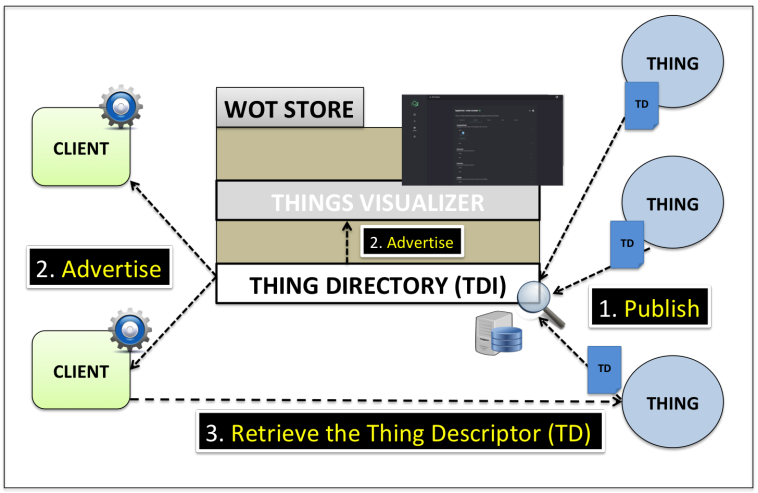


Figure 3: The operations of the Things Discovery Service (TDS).

pipeline. In next Section we provide an in-depth description of the three main modules of the WoT Store. The implementation details and the technologies used are sketched in Section 5.

4. Service Components

We detail here the main modules of the WoT Store i.e. the Thing Manager, the Application Manager and the Data Manager.

4.1. Things Manager

The Things Manager module allows the users to interact with the Things already available in the WoT environment. We further distinguish between two sub-modules. i.e.: the Thing Discovery Service (TDS), that is in charge of registering the active Things on our tool, and the Thing Visualizer Service (TVS), that is in charge of displaying the registered Things on the GUI of the WoT Store.

4.1.1. Things Discovery Service (TDS)

The overall discovery procedure is depicted in Figure 3. The procedure is initiated by the Web Things when they register themselves to the WoT Store and more specifically to the Thing DIrectory (TDI) module, assuming that the URI of this latter is known. In the registration phase, each Thing provides its Thing Descriptor (TD). The TDI works as a broker and as a repository of TDs; in the

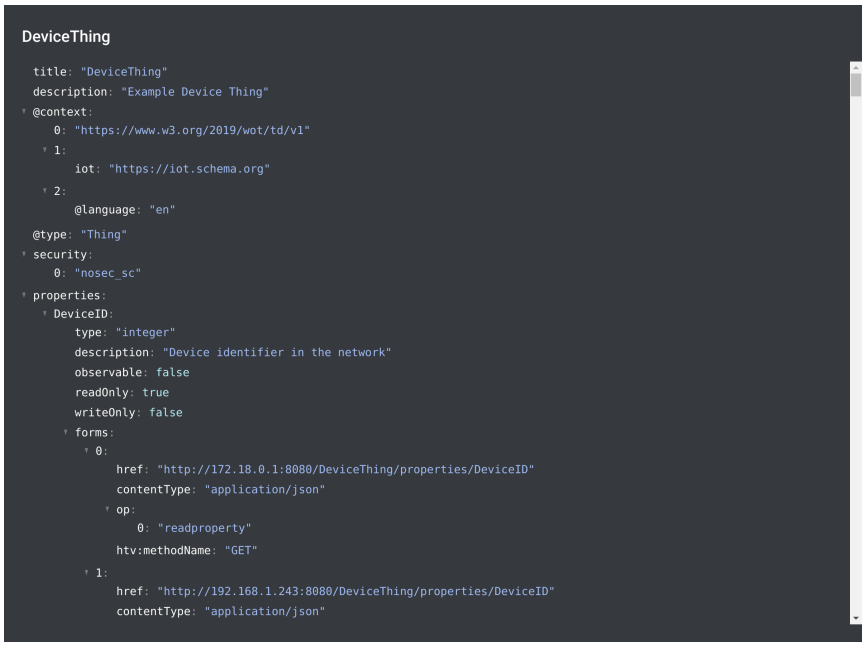


Figure 4: A portion of the TD of the Device Thing of Table 3. The Thing is associated to a wireless sensor producing temperature values.

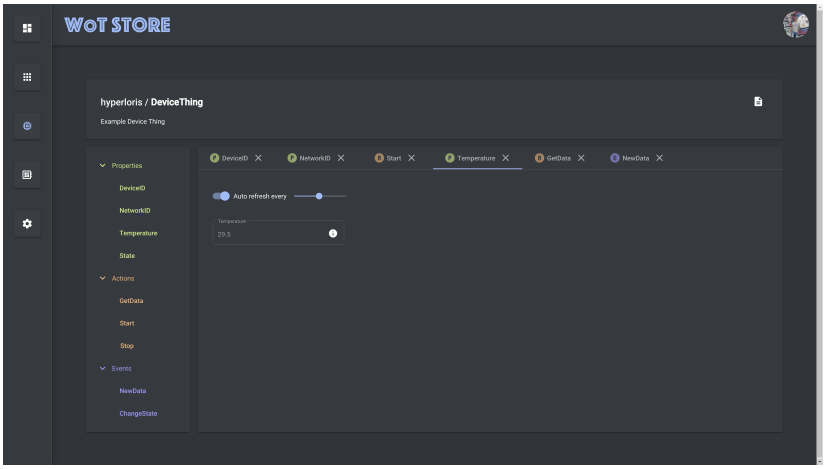


Figure 5: The rendering of the actions of the TD of Figure 4 within the WoT Store.

first mode, it notifies the presence of a new Thing to all the clients, including the TVS described below. Each client will then retrieve the TDs directly from the Things, in order to consume the most updated version. In addition, the TDI stores a copy of the TD of all the registered Things; this is required since the TDI can support search and filter operations, issued by the user through the Web dashboard. We consider two usage modes of the TDI, according to its visibility level: i.e. public TDI or private TDI. In the first case, all the Things registered to the TDI are searchable from the clients: this might be the case for instance of a smart city willing to share its IoT resources with all its citizens. Vice versa, in the private case, the access to the Things is restricted, and proper authorization mechanisms are employed by the WoT Store: this is the case of smart home or industrial IoT deployments with severe security concerns. The visibility flag must be set during the TDI configuration process, together with other meta-data such as the authentication mechanisms (e.g. header-based authorization, token-based authorization like OAuth 2.0[2]) required by clients to access the TDI.

4.1.2. Things Visualizer Service

The TVS is a Web dashboard and the main GUI of the WoT Store. It allows to visualize the list of available Things registered to the TDI (by subscribing to it). Moreover it supports search operations, where a subset of Things is selected according to user-defined conditions; search operations are enabled by a Web form with a list of predefined fields that can be filled through the GUI, and involve a subset of the meta-data contained in the TDs. Finally, the TVS allows the user to interact with each Thing available in the TDI or contained in the result of a search operation; this is performed by parsing the corresponding TD and creating an ad-hoc Web GUI, through which it is possible to monitor the state properties, click and execute actions (passing the needed parameters if requested), or receive notifications of the events occurred. Figure 4 shows a small portion of the TD of a Device Thing measuring temperature values and used in the Pervasive Sensing testbed of Section 6.1; the full interaction model is reported in Table 3. The corresponding GUI rendered within the WoT Store with the list of available actions and properties is depicted in Figure 5.

4.2. Applications Manager

The Application Manager supports the dynamic search, download and execution of third-party WoT applications, involving the interactions with the available

[2]https://oauth.net/2/

resources, or the creation/update of new resources. We assume that the current applications are coded in Javascript (JS), since this is the language of the WoT implementation made available by the WoT W3C community [15], although this choice does not impact the general functionalities of the WoT Store. Conceptually we distinguish between two classes of WoT applications supported by the WoT Store, i.e.: Things Applications (TAs), and Mash-up Applications (MAs). The TAs correspond to the source-code of a Thing, hence to a static object that can be activated when executing it. Through the TAs, it is possible to instantiate a new Thing in the WoT Store, or to update the behaviour of current Things, as better described in the following. Vice versa, the MAs implement automatic policies that involve the interactions of multiple Things (active and registered on the TDI); the result of a MA can be an actuation or a data-stream that can be processed through the Data Manager described in Section 4.3. More in details, the Application Manager provides three main functionalities:

- *App Storing*. The source code of the WoT applications is stored on a database. Moreover, each application (MA or TA) is associated to a semantic descriptor, including a list of meta-data, like its category, description and the resources required (e.g. the type of Things used). For instance, Table 1 contains the RDF description of a MA that queries all Things of type "Temperature" registered to the TDI and computes the average of the sensed data. In the current implementation, we describe each WoT application through a list of RDF fields; clearly, more formal descriptions of the MA and TA can be considered, by means of dedicated WoT ontologies. We discuss the issue among the future works (Section 7).

- *App Searching*. Through the Web dashboard and the compilation of specific fields, the user can build SPARQL queries[3] in order to filter the WoT applications matching specific criteria, defined again through the meta-data. The results are then displayed on the WoT Store GUI.

- *App Executing*. After having selected the application meeting his/her requirement, the user can download and execute it. In this case, the proper run-time environment (i.e. the Servient where to deploy the application) must be selected among the ones registered to the WoT Store. We introduce additional features for the execution of the TAs, that can occur in *Nor-*

[3]The SPARQL code in all the search operations must not be inserted manually, rather, it is generated automatically based on the search option fields filled by the user on the Web GUI.

subject	predicate	object
<WoTStore://temperatureMonitor>	*schema:applicationCategory*	Domotics
<WoTStore://temperatureMonitor>	*schema:downloadUrl*	coap://wotstore.cs.unibo.it:8081/market/actions/getApplication?application=temperatureMonitor
<WoTStore://temperatureMonitor>	*schema:downloadUrl*	http://wotstore.cs.unibo.it:8080/market/actions/getApplication?application=temperatureMonitor
<WoTStore://temperatureMonitor>	*wotstore:involve*	sosa:Sensor
<WoTStore://temperatureMonitor>	*rdf:type*	schema:SoftwareApplication
<WoTStore://temperatureMonitor>	*dcterm:description*	temperatureMonitor is an application that takes the temperature from several sensors and returns the average
<WoTStore://temperatureMonitor>	*rdfs:label*	temperatureMonitor

Table 1: Example of RDF Description of a MA application available in the WoT STORE.

mal or *Update* mode. The first case is equivalent of creating a new Thing, and registering it to the TDI. In the second case (*Update*), we provide the possibility to replace a list of active Things with new ones implementing the behaviour described by the TA downloaded by the WoT Store. Hence, a SPARQL search query is issued on the TDI in order to select the Things to unregister; then, a new set of Things is deployed with the updated source-code provided by the TA. In Section 4.4 we discuss the usefulness of such feature in industrial IoT use-cases.

4.3. Data Manager

This module contains functionalities for the processing and visualization of the data produced by the running WoT applications. The block components of the Data Manager are the *data-streams*; each data-stream can be configured in order to be attached to a MA, and to gather data from it, through a set of APIs made available by the WoT Store. Each data-stream consists of two sub-components: a *data aggregator*, that filters/aggregates the output of the MA, and a *data plotter*, that creates the proper Web dashboard of the processed data. Clearly, the stages above are strictly dependent on the data-format, on the MA in use and on the user needs; it is nearly impossible to cover all possible requirements. For this reason, we provide basic data-stream templates that must be extended/customized by users/developers. Moreover, as proof of concepts, we implemented two specific data-flows: one for temporal data-series (composed by a time-stamp and a numeric field), the other one supporting geo-data (in GEOJSON) and generating the corresponding heatmap. Further details regarding the two data-streams are provided in Section 6.

4.4. Use cases

The WoT Store is application-agnostic, hence it can be used on several IoT scenarios where there is need of managing and integrating heterogeneous resources. In the following, we discuss two main use-cases, by highlighting how specific components of the framework can be leveraged.

1. *Industrial automation.* Industry 4.0 environments are often characterized by the large-scale installation of sensors, and by the need to process vast amount of sensor data in order to build the digital twin of a physical component [29]. We can assume that each sensor is represented by a Thing and runs a default factory application that publishes itself on the WoT Store. Through the TVS, it is possible to display and monitor the behaviour of the sensors available in the environment. In addition, let us consider a practical case where a bug is discovered on the current TA, and hence a patch must be applied on all the Things. Without the WoT Store, this would require a manual re-configuration of all the devices, with a clear impact in terms of time and cost. Through the *Update* mechanism, the users can issue a semantic query in order to select a set of Things all satisfying the same query conditions (e.g. "update all sensor devices of type temperature available in room 262"), and then install the new TA replacing the previous one.
2. *Home automation.* The home automation constitutes one of the main market of the IoT in terms of revenues and applications; however, it is also characterized by the proliferation of devices mapped on different hardware/software technologies and using specific APIs to be queried. Current platform enabling the execution of mash-up services (e.g. the IFTTT platform[4]) are limited in terms of IoT devices supported, since specific connectors to be used. The WoT Store allows users to search and deploy MAs that implement autonomic behaviors without requiring any connector from the devices, except to be provided with a TD.

5. Implementation

The WoT Store is composed of four internal components, reported in Figure 6: the Market Service (MS) and the Thing DIrectory (TDI) on the server side, the Market Interface (MI) on the client side, and the Runner (RNN) on the physical device hosting the W3C WoT Servient. The WoT Store implementation involved the usage of several software libraries: we briefly discuss here the main solutions adopted, while Table 2 provides a mapping of the service components of Section 4.1 with the enabling technologies.
The Market Service (MS) has been implemented as a `Node.js`[5] v10.x applica-

[4]https://ifttt.com
[5]https://nodejs.org

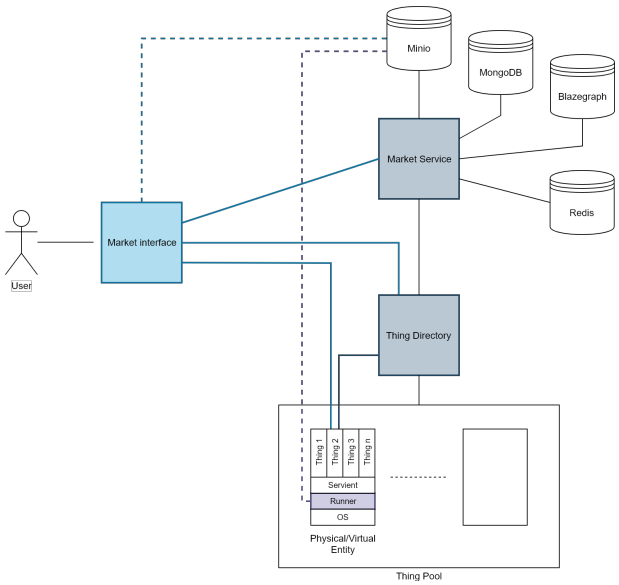


Figure 6: The WoT Store internals.

tion using the `LoopBack`[6] v3 framework and the Socket.IO[7] library. The MS exposes the REST APIs for all the Things-related and application-related operations and a WebSocket endpoint for real-time notifications. In addition, it stores the platform information through four database technologies: (*i*) `Minio`, an object storage server containing the WoT applications (MAs and TAs) source code, (*ii*) `MongoDB`, the popular NOSQL database used to store the user data, (*iii*) `Blazegraph`, the triplestore used to save the application metadata and the TDs and (*iv*) `Redis`, a high performance in-memory database, used for the real-time processing of the Things notifications.We developed a complete LoopBack connector for `Blazegraph` implementing all the necessary methods to initialize the connection, the data migrations and CRUD operations. In addition, a second component is in charge of converting the JSON-LD to N-Quads when pushing the data to the triplestore, and from JSON to JSON-LD when they are pulled out.
The Runner (RNN) is a piece of software developed to facilitate the installation of the WoT Store, and to automatize the execution of the WoT applications on the devices. The RNN is written in JavaScript and exploits the `ShellJS` library.

[6]https://loopback.io
[7]https://socket.io

Area	Service	Tecnologies / Libraries
Thing Manager	TDS	`Node.js`, `Blazegraph`, `SPARQL`.
Thing Manager	TVS	`Angular`, `ngx-mqtt`, `rxjs-websockets`, `socket.io-client`.
Application Manager	App Storing	`Minio`, `MongoDB`, `Blazegraph`.
Application Manager	App Searching	`SPARQL`.
Application Manager	App Executing	`Docker`, `shelljs`.
Data Manager	Data Aggregator	`bull`.
Data Manager	Data Plotter	`ngx-echarts`.

Table 2: List of technologies used for the implementation of the WoT Store service components of Section 4.1.

When installed on a machine (which could be a physical device, like a Raspberry Pi, or a Virtual Machine), the RNN registers the machine to the MS. Then, through the RNN, the user can install the WoT Servient on its device, by choosing the version compliant with the current hardware and software (operating system) configuration.[8] The RNN allows issuing commands from the WoT Store directly on the device, like for instance the execution of a MA or a TA selected from the repository. To this aim, it supports multiple run-time environments through the executors, i.e. the `Shell` and `Docker`[9] in the current implementation, while the support for `Kubernetes`[10] will be considered as future work. Finally, the Market Interface (MI) is an `Angular`[11] v6 web application composed of several modules. Among these, we find the Thing Visualizer Module, which implements the TVS introduced in the previous Section, i.e. it renders a Thing starting from its TD. Properties and events are updated in real-time thanks to libraries such as `ngx-mqtt` and `rxjs-websockets`; for each action, a specific form is created with the necessary constraints for the data input.

6. WoT Store Experimental Validation

The operations of the WoT Store have been validated through two evaluation studies: (*i*) a small case testbed of a pervasive sensing scenario (Section 6.1), and (*ii*) a large-scale simulation combining real Things and virtual devices in a urban crowdsensing scenario (Section 6.2). The studies addressed different goals and evaluated different components of the WoT Store framework. In the testbed, we

[8] At present, we rely on the JS Servient made available in [15]; however, we imagine the case where multiple Servient implementation will be available for a specific device.

[9] https://docker.com

[10] https://kubernetes.io

[11] https://angular.io

focus on the Thing Manager and Application Manager modules; more specifically, we demonstrate the possibility to orchestrate the sensing operations of multiple, heterogeneous wireless sensors through the MAs, and we provide evidence of the Thing Discovery Service. The crowdsensing study aims to verify the scalability of the WoT architecture and of the WoT Store under increasing number of Things to manage; moreover, it demonstrates the capabilities of the Data Manager to aggregate and visualize both time-series and geographic data-streams produced by MAs orchestrating the sensing operations of simulated mobile devices.

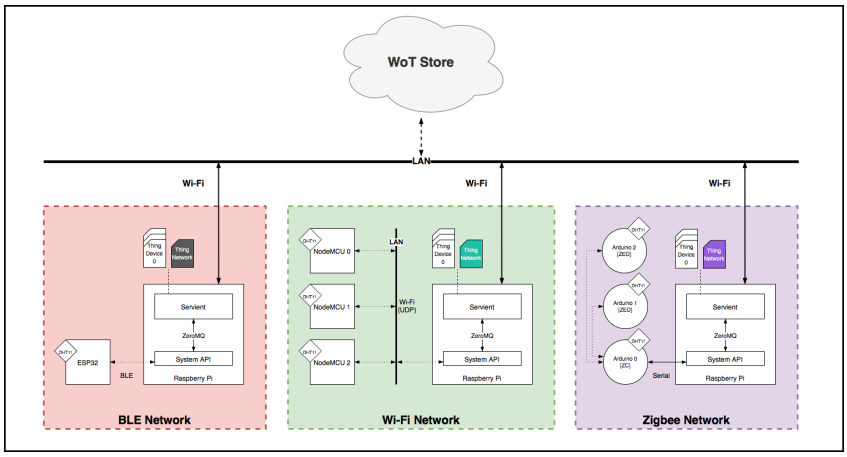


Figure 7: The IoT/WoT monitoring system deployed in this study.

6.1. Pervasive Sensing Testbed

We built the testbed represented in Figure 7 and preliminary presented in [23]: it consists of an indoor monitoring system composed of three layers: sensing, fog, and processing. The sensing layer is constituted by three Wireless Sensor Networks (WSNs), operating at different rooms of the same building: an IEEE 802.15.4 WSN network, a IEEE 802.11 Wi-Fi WSN network and a Bluetooth Low Energy (BLE) WSN. The 802.15.4 network includes four devices (*Arduino Xbee* boards), with one Coordinator and three Leaf nodes equipped with sensing units (*ThinkerKit* temperature sensor). The Wi-Fi network includes three devices (two *NodeMCU* and one *Arduino WiFly* board), all provided with a direct link toward the Access Point (AP) and with a *DHT11* temperature/humidity sensor. The BLE WSN consists of one *ESP32* board, provided with a *DHT11* sensor. The 802.15.4 coordinator, the BLE and the Wi-Fi devices are connected to a *Fog* node, via USB cable links (for the 802.15.4 Coordinator) or wireless links (for the BLE

Name	Type	Description
DeviceID	Property	Device identifier in the network.
NetworkID	Property	Network identifier the device belongs to.
Temperature	Property	Last temperature value.
State	Property	Current state of the device.
GetData	Action	Get the temperature data.
Start	Action	Start sending data at each time-slot.
Stop	Action	Stop sending data.
NewData	Event	This event is fired when a new sensor data is produced.
ChangeState	Event	This event is fired when the connection state changes.

Table 3: List of Properties, Actions, and Events of a Device Thing.

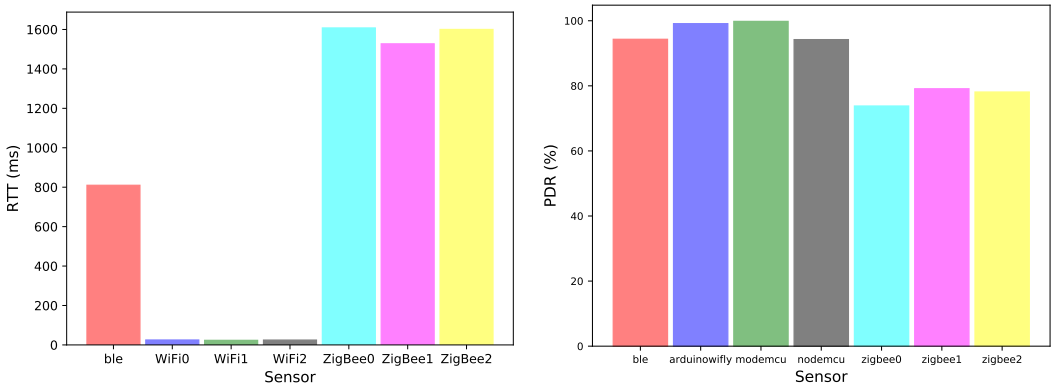
and the IEEE 802.11 devices). Finally, the processing layer is constituted by a Linux server, connected to the Fog nodes via Wi-Fi links. We deployed the W3C WoT architecture and the WoT Store as follows:

- Edge devices, i.e. the wireless sensors, implement low-level communication and sensing operations in the embedded firmware (written in C language). The implementation as well as the list of operations and the data format used by each device are technology dependent. This layer is part of the IoT, while it is not covered by the WoT architecture.

- Fog nodes run a W3C WoT Servient, by using the JavaScript (JS) framework available at [15]. Each Fog node exposes two types of Web Things, i.e.: multiple (*i*) *Thing Devices*, describing the properties, events and actions of physically managed edge devices, and one (*ii*) *Thing Network*, describing the overall performance of the virtual WSN composed by the list of connected Thing Devices. Table 3 displays some of the properties, actions, and events described in the Thing Description (TD) for a Device Thing.

- Finally, the Processing node hosts the WoT Store. This latter allows to manage the Web Thing Devices and Web Thing Networks through the Thing Manager presented in Section 4.1. Also, we implemented multiple MAs that are in charge of orchestrating the sensing operations, i.e. of selecting a subset of devices to query at each sensing slot, according to MA-specific policies. At each interval, the MA works by querying the TDI and gathering the list of Things Devices available on the WoT Store; hence the MAs are also able to adapt to dynamic conditions where the number of available Things is varying over time, as demonstrated by the analysis below.

We highlight that the WSNs are heterogeneous in terms of M2M technology and of network performance. To this purpose, Figure 8(a) depicts the average

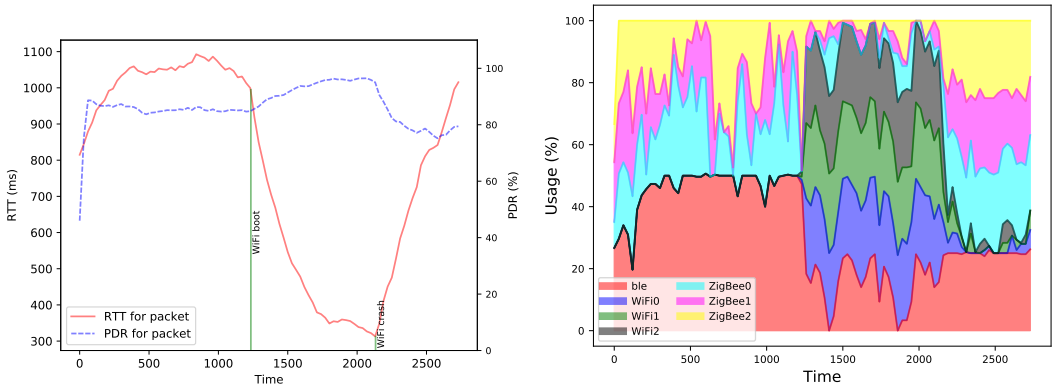


(a) (b)

Figure 8: The per-sensor RTT and PDR metrics are shown respectively in Figures 8(a) and 8(b).

per-sensor Round Trip Time (RTT), computed as the delay to issue the `getData` command from the MA and to receive the sensor data. As expected the Wi-Fi devices experience the lowest RTT values due to the higher channel bandwidth provided by the M2M technology. Figure 8(b) shows the per-sensor Packet Delivery Ratio (PDR), defined as the ratio of successful `getData` command issued by the MA. As expected, the Wi-Fi sensors are also the most reliable nodes. Based on these results, we implemented three MAs on the WoT Store, simply denoted as P_1, P_2, P_3. Each MA selects M different Things Device to query at each sensing slot, but according to different policies, i.e.: (*i*) the MA P_1 (RTT-aware) selects the M Things with the lowest mean RTT values; (*ii*) P_2 (PDR-aware) selects the M Things with the highest mean PDR values; (*iv*) P_3 (PDR-RTT aware) selects the M Things providing the best RTT-PDR trade-off. We assume that -at system startup- the MAs have no knowledge about the WSN performance (i.e. the results shown in Figures 8(a) and 8(b)), and hence they have to discover the optimal set of M Things maximizing the specific policy in use. This is implemented through basic online reinforcement learning mechanisms, provided by the Q-learning algorithm [30]: the MA computes a numeric reward each time the `getData` command is issued toward a sensor, related to the specific policy (e.g. the packet RTT in case of MA P_1). We skip the details of how the Q-learning algorithm has been applied in the testbed, since they can be found in [23], rather we focus on the operations of the Things and Applications Managers.

To this purpose, Figure 9(a) and 9(b) provide a validation of the Thing Discovery Service (TDS). We considered the following experiment: at system startup,



(a) (b)

Figure 9: The impact of the Thing Discovery Service on the PDR and RTT performance indexes on a scenario with varying number of available Things/devices is shown in Figure 9(a). The Thing/device utilization over time is depicted in Figure 9(b).

only the BLE and Zigbee Things are registered to the WoT Store. Hence, the MA relies exclusively on them for the sensing operations. At t=1200 seconds, the Wi-Fi Things are activated; they autonomously publish their TDs and hence become discoverable by the MA via the TDI. We highlight that the process above occurs in an autonomic way without any manual configuration. At t =2100 seconds, the Wi-Fi devices are physically detached from the environment, without notifying the WoT Store. Figure 9(a) shows the average PDR and RTT metrics over time as computed by the running MA (in this case, we used P_3). It is easy to notice the impact of the TDS since both the metrics improve from t >1200 seconds, as direct consequence of the fact that the Wi-Fi devices are used by the MA; we recall that - in accordance with the results of Figures 8(a) and 8(b), the Wi-Fi technology maximizes both the RTT and PDR performance. At the same time, we can notice that the RTT decrease and the PDR increase occur gradually and not instantaneously; this is due to the Q-learning convergence delay, since the usage of Wi-Fi is reinforced at each packet transmission, hence increasing the selection probability over time. Finally, both the performance indexes become worse when the Wi-Fi devices stop sending the data because of the hardware crash. Figure 9(b) supports the discussion by showing the sensor utilization over time. For t <1200 seconds, the MA queries the BLE and two zigbee Things, while from $1200 \leq t < 2100$, it mostly relies on the Wi-Fi Things; however, the Wi-Fi Things do not achieve the 100% of utilization due to random actions performed by the Q-learning for

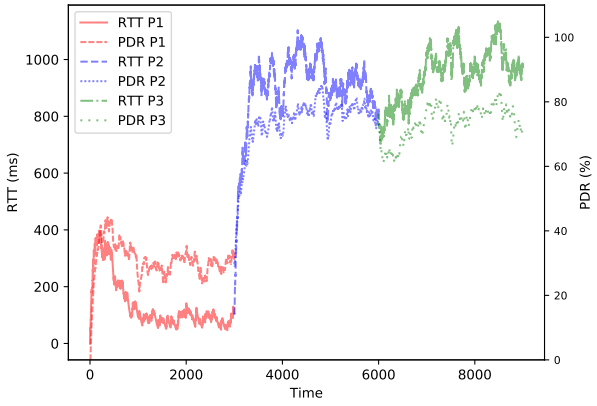


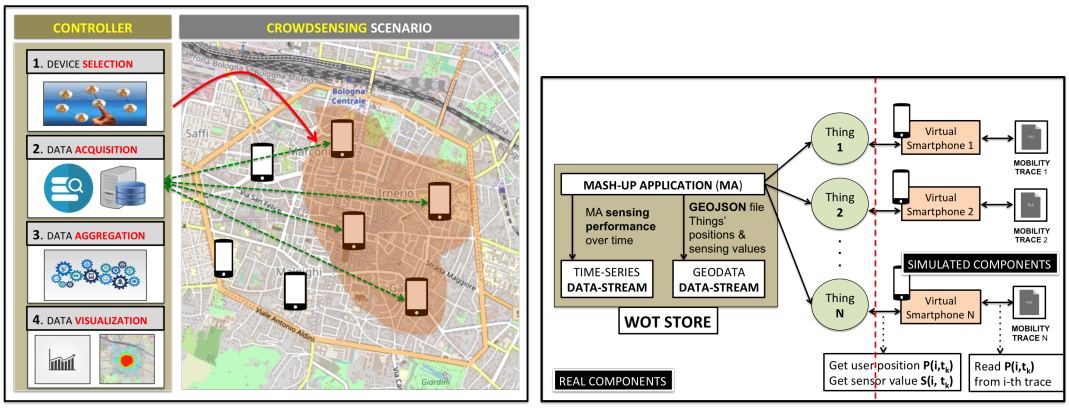
Figure 10: The RTT and PDR values when switching the MA in use over time.

the periodic exploration phase [30]. Thanks to it, the Q-learning mechanism is able to discover alternative sensor selections once the Wi-Fi devices become not available (t >2100 seconds).

Figure 10 shows the RTT and PDR metrics over time when dynamically switching from one MA to another. Moreover specifically, from t=0 to t=3000, policy P_1 is used (delay minimization), then P_2 from t=3001 to t=6000 (PDR maximization), finally we switch to P_3 (delay-PDR trade-off) from $t > 6000$. We remark that the application replacement is performed through the WoT Store GUI, and consists in selecting a new software, and the Servient where to execute it. No hardware or software re-configuration of the WSNs is required. We can notice the values of the metrics (RTT and PDR) vary over time in accordance with the MA that is in execution in that temporal instant.

6.2. Urban Crowdsensing Scenario

In the second study, we consider a urban crowdsensing application composed of multiple, heterogeneous mobile devices (e.g. smartphones). Like in most of existing crowdsensing systems [31], the mobile devices perform environmental sensing through their embedded sensors and transfer the data to a central processing unit; here, data are aggregated and analyzed. We assume that the central unit is also in charge of orchestrating the sensing operations, similarly to the testbed described in the previous Section. The overall architecture of the crowdsensing system is reported in Figure 11(a). A W3C Thing is associated to each mobile device: the list of actions and properties is provided in Table 4. The system administrators can download MAs implementing different sensing policies from the



(a) (b)

Figure 11: The crowdsensing system considered in this study is depicted in Figure 11(a). The abstraction of the WoT deployment with the WoT Store and the real/simulated entities is represented in Figure 11(b).

WoT Store; in addition, they can aggregate and visualize the data gathered from the mobile devices through the Data Manager. Differently from the testbed, the system APIs of the Web Things do not query a physical device rather a simulated entity -denoted as Virtual Smartphone (VS)- that provides the current position and the result of each sensing action. In the following, we detail how the mobility and the sensing phases have been modeled.

Mobility simulation. We consider a pedestrian mobility model on a realistic city map (in our case, the downtown area of Bologna), extracting the street information from the OpenStreetMap web service[12]. A random direction model is considered: each user moves toward a random point of the scenario on the shortest path (computed over the graph of streets), and stops there for a random interval before selecting a new destination. We assume that the sensing and mobility phases are not mutually dependent; hence, the mobility traces of the N users have been generated offline and saved on N different files. The position information of each Thing (i.e. the *Latitude* and *Longitude* properties at each time-slot t_k) is provided by the VS by reading the corresponding entry on the trace file owned by the Thing.

Event simulation. We model the sensing operations through a function that returns the sensing value at each location of the environment and at each time-slot. To this purpose, we consider a generic situation where an event occurs in the urban sce-

[12]https://www.openstreetmap.org

Name	Type	Description
PhoneID	Property	Smartphone unique identifier.
Latitude	Property	Current latitude coordinate.
Longitude	Property	Current longitude coordinate.
State	Property	Current state of the device (connected/disconnected).
GetSensingData	Action	Perform a sensor reading.
NewData	Event	This event is fired when a new sensor data is produced.

Table 4: List of Properties, Actions, and Events of a Virtual Smartphone Thing in the crowdsensing scenario.

nario -and more specifically in its central position- and the crowdsensing system is used to monitor the event and its evolution over time. Let $C = \langle c_{lat}, c_{long} \rangle$ denote the center of the scenario that coincides with the event origin. We abstract from the physical meaning of the event, of the sensing values and of the type of sensor in use, since they are not relevant for this study. Let $S(i, t_k)$ be the event sensing function that provides the intensity of the event as sensed by Thing i at time slot t_k (i.e., the values returned after invoking the *GetSensingData* action). The $S(i, t_k)$ function is modeled as follows:

$$S(i, t_k) = e^{\frac{d_i(t_k, C)}{\sigma}} \cdot I(t_k) + \chi \tag{1}$$

where $d_i(t_k, C)$ is the distance between the position of Thing i at time slot t_k (denoted as $P_i(t_k) = \langle lat_i(t_k), long_i(t_k) \rangle$) and the event origin C, σ is a normalization value, χ is a Gaussian noise with zero mean and variance equal to β (it models the sensing error of each device) and $I(t_k)$ is the function modeling the event intensity over time. Hence, on the spatial domain, the event intensity assumes the maximum value in C while it decreases proportionally with the distance from it. Let I_{max} and I_{min} be the maximum and minimum event intensity values. The $I(t_k)$ function defines an event with a time-varying intensity, i.e.: (*i*) it is equal to the minimal value (i.e. $I(t_k) = I_{min}$) till instant t_k=900 seconds; (*ii*) it increases linearly till instant t_k=1350 seconds, when the maximum value (I_{max}) is achieved; (*iii*) it remains equal to the maximum value (I_{max}) till instant t_k=2150 seconds; (*iv*) it decreases linearly till becoming equal again to the minimum value (I_{max}) at time instant t_k=2600 seconds. Let $I_{min} < \eta < I_{max}$ be a system threshold; a Thing/device[13] is said to detect the event at time t_k if $S(i, t_k) > \eta$.

[13]Things and devices are used indifferently in the following, since each Thing corresponds to one device; we used the word *device* when referring to the operations of the crowdsensing system, and *Thing* when referring to its implementation using the WoT architecture.

The WoT deployment of the crowdsensing system with real/simulated entities is shown in Figure 11(b). In order to save the battery of the mobile devices, the controller is querying only a subset of the available N devices at each sensing interval t_k. Let $\Psi(t_k)$ be such subset, and M be the number of devices queried, assumed constant over time. More formally, we have: $M = |\Psi(t_k)| = \lfloor N \cdot \gamma \rfloor$, with $0 < \gamma \leq 1$. Also, we denote with $\Omega(t_k) \subseteq \Psi(t_k)$ the list of devices detecting the event at time slot t_k, i.e $\Omega(t_k) = \{i \in \Psi(t_k) | S(i, t_k) > \eta\}$. We implemented two MAs in WoT Store with differentiated policies to compute the $\Psi(t_k)$ set:

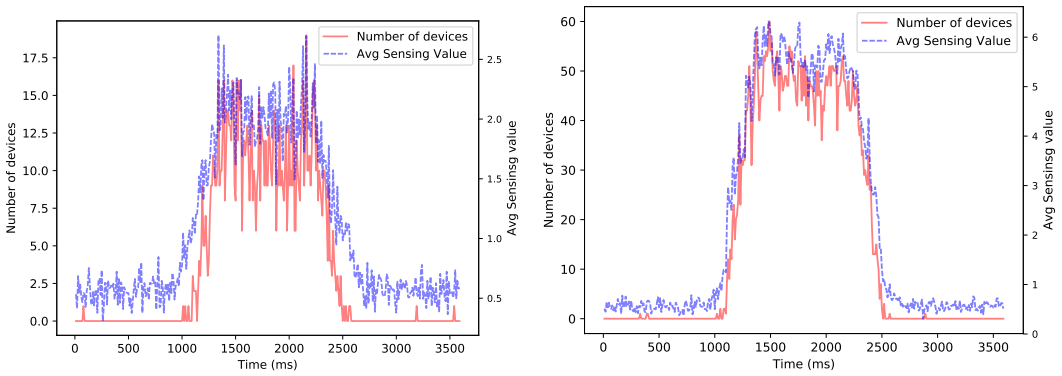
- *Random* MA. At each sensing interval t_k, the MA chooses randomly the subset of M sensors to query among the available N.
- *Adaptive* MA. Like the previous policy, the MA chooses M sensor to query at each sensing interval; it chooses them randomly if $|\Omega(t_k)| \leq \alpha$, the number of devices detecting the event is below a system threshold α. Vice versa, when $|\Omega(t_k)| > \alpha$, the MA attempts to estimate the area where the event is occurring and to concentrate the sensing operations over it. To this purpose, the position of the event $C^{est}(t_k) = \langle c^{est}_{lat}(t_k), c^{est}_{long}(t_k) \rangle$ at time t_k is estimated as the centroid of the position of the users detecting it, i.e.:

$$c^{est}_{lat}(t_k) = \frac{\sum_{i \in \Omega(t_k)} lat_i(t_k)}{|\Omega(t_k)|} \quad (2)$$

$$c^{est}_{long}(t_k) = \frac{\sum_{i \in \Omega(t_k)} long_i(t_k)}{|\Omega(t_k)|} \quad (3)$$

  Similarly, the radius of the event $R(t_k)$ is estimated as the maximum distance between $C^{est}(t_k)$ and all the devices that detected the event in $\Omega(t_k)$, i.e. $R(t_k) = max_{i \in \Omega(t_k)} (d_i(t_k, C^{est}(t_k))$. In order to build the list of devices to query at the next time-slot (i.e. $\Psi(t_{k+1})$), we consider only the devices at a distance lower than $R(t_k)$ from c^{est}_{long}. Let $\Gamma(t_k)$ denote such set. Then, we order $\Gamma(t_k)$ according to the distance values $d_i(t_k, C^{est}(t_k))$, and we select the top M elements. In case $|\Psi(t_{k+1})| < M$, the remaining $M - |\Psi(t_{k+1})|$ devices are randomly chosen as for the Random Policy.

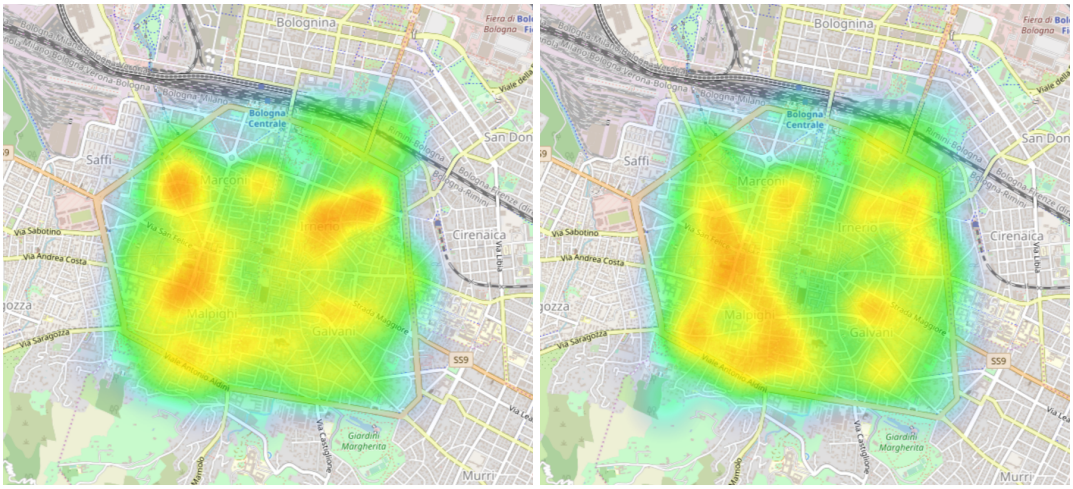
Clearly, much more complex MAs can be defined for the scenario in use. However, we remark that the goal of the study is not on the crowdsensing algorithms rather on the deployment and execution of MAs through the WoT Store. Unless stated otherwise, we used the following parameters in our tests: N=400, M=80, γ=0.2, η=5, α=2, β=2, σ=300.



(a) (b)

Figure 12: The average sensing value and the number of Things detecting the event for the *Random* MA are shown in Figure 12(a). The same metrics for the *Adaptive* MA are shown in Figure 12(b).

We implemented two data-streams in the Data Manager, one handling time-series data and the other handling geographic data, coded in GEOJSON. Each MA generates two time-series: (*i*) the average sensing intensity at each t_k, computed as the average value over the M queried devices, i.e. $\frac{\sum_{i \in \Psi(t_k)} S(i,t_k)}{M}$, and (*ii*) the number of devices detecting the event at each t_k, i.e. $|\Omega(t_k)|$. For readability reasons, we visualized both the time-sequences in the same plot. Figure 12(a) refers to the *Random* MA, while Figure 12(b) to the *Adaptive* MA. Both the plots show a similar trend, since the average sensing intensity follows the variations over time of the event intensity, provided by the $I(t_k)$ values; however, it is easy to notice that the absolute values are much greater for the *Adaptive* MA compared to the *Random* MA, since the sensing activities are focused an the area where the event is occurring. As a result, the number of devices detecting the event is also significantly higher for the *Adaptive* MA. In addition to the time-series, at pre-defined time-slots each MA generates a GEOJSON file, containing the position (i.e. $P_i(t_k)$) and the instantaneous sensing value (i.e. $S(i, t_k)$) of the Things queried (i.e. belonging to the set $\Psi(t_K)$). The Data Manager allows visualizing the GEOJSON file as heatmaps. Figure 13(a) and 13(b) show the heatmaps of the *Random* MA at t_k=100 and t_k=1000, i.e. before and during the occurrence of the event. We can notice that the sensing actions of the *Random* MA are equally distributed over the scenario in both cases. Figure 14(a) and 14(b) show the heatmaps of the *Adaptive* MA at the same time slots. Before the occurrence of the event (Figure 14(a)), the *Adaptive* MA behaves similarly to the *Random* MA since no device has detected



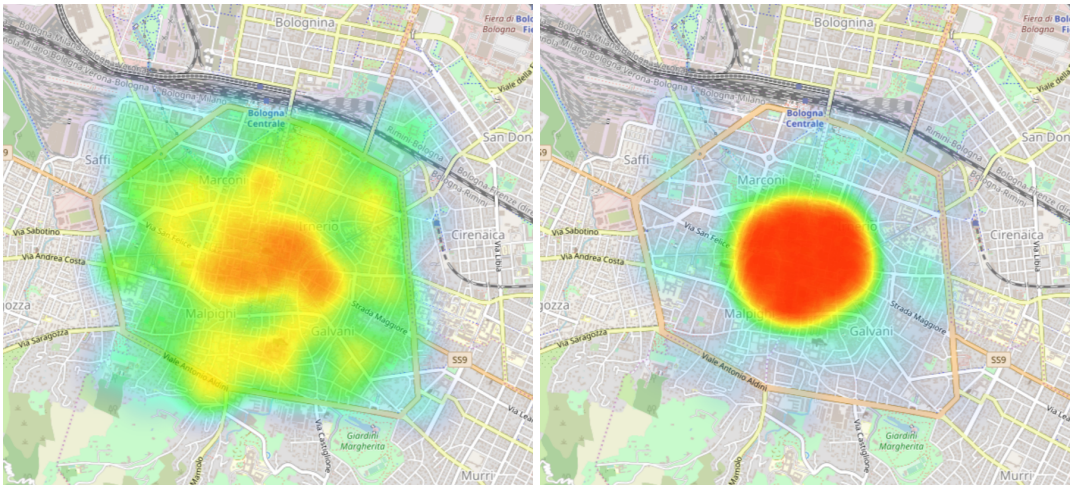
(a) (b)

Figure 13: The geodata stream visualization for the *Random* MA before the occurrence of the event (Figure 13(a)) and during the occurrence (Figure 13(b)).

the event, and hence the device selection is performed randomly. Vice versa, after the occurrence (Figure 14(b)), most of the M sensing actions are performed on the central area of the scenario, i.e. on the estimated event origin C^{est} that also coincides with the real event origin C. This result further justifies the higher performance of the *Adaptive* MA in terms of number of devices detecting the event compared to the *Random* MA (Figures 12(a) and 12(b)). In conclusion, the Data Manager can be useful to monitor the crowdsensing operations over both the time and space dimensions; moreover, since different MA can be installed from the WoT Store, the Data Manager allows to compare the performance of different sensing control policies in a straightforward way.

We conclude the analysis by investigating the scalability of our platform when increasing the number of deployed Things. Figure 15 shows the usage of resources (RAM utilization, CPU time) of the machine[14] hosting the Servient. It is easy to see that the resource utilization increases linearly with the number of Things to manage, and in any case no performance bottlenecks are introduced even for large-scale WoT deployments.

[14]Core i5 7600 Kaby 3,5GHz with 16GB RAM DDR4 and ArchLinux OS.



(a) (b)

Figure 14: The geodata stream visualization for the *Adaptive* MA before the occurrence of the event (Figure 14(a)) and during the occurrence (Figure 14(b)).

7. Conclusions

In this paper, we have addressed the interoperability issue on the Internet of Things (IoT), and more specifically the design and implementation of cross-domain solutions able to cope with the heterogeneity of devices and platforms. To this purpose, we have proposed the WoT Store, a novel resource management platform for IoT scenarios compliant with the recent W3C Web of Things (WoTs) architecture. The WoT Store allows to discover and manage all the Things available in the environments in a seamless way; moreover, it provides functionalities to deploy new resources in the current scenario, by acting as a WoT application marketplace. The framework operations have been validation through a twofold evaluation: a small-case testbed composed of heterogeneous wireless sensor devices, and a large-scale simulation related to an urban crowdsensing scenario. It is worth highlighting that the relevance of the contributions presented in this paper is strictly related to the W3C WoT standard, which is quite recent in its definition. At the same time, the availability of support tools constituting the WoT SECO can facilitate the adoption of the W3C standard from the academic and industrial communities and the definition of novel use-cases.

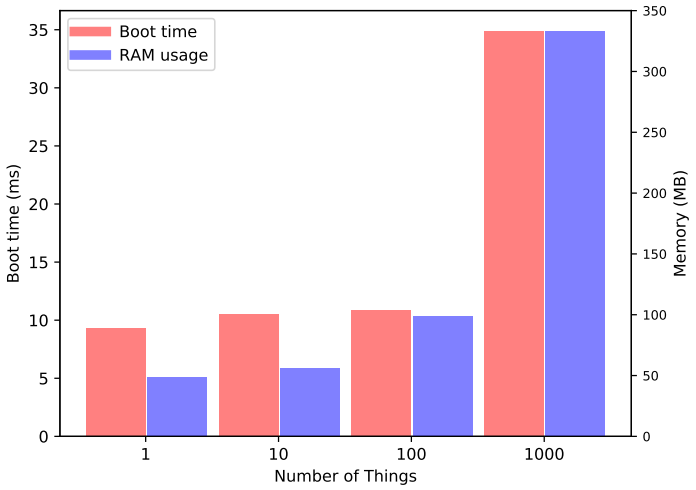


Figure 15: The resource (CPU,RAM) utilization of the WoT Store for increasing number of deployed Things in the crowdsensing scenario.

7.1. WoT Store Limitations and Future extensions

Far from being a reference solution covering all the requirements of WoT deployments, the WoT Store can still be considered -at the best of our knowledge- the first W3C-compliant WoT tool providing Things management in dynamic WoT scenarios. The micro-services oriented architecture facilitates further improvement, platform extension and customization. We discuss here the planned future works in four complementary research directions, i.e.: security support, services/functionalites and data support.

- *Security support*. Security aspect is a crucial issue for the IoT domain as well as for the WoT Store, given the possibility offered by our tool to discover, interact, and potentially update remote Web Things. Hence, proper authorization mechanisms must be designed to avoid data leaks or harmful operations on shared WoT resources. However, it is important to distinguish between two aspects, i.e. platform control access and Things control access. Regarding the first aspect, the WoT Store relies on state-of-art Web technologies to authenticate users when logging in onto the platform; also, the visibility of available Things is manually configurable during the TDI configuration process explained in Section 4.1.1. At present, the latter includes only two visibility levels (public or private): however, more fine-grained registration policy can be defined. Vice versa, the Thing access policy is

defined within its TD: the current W3C WoT standard already identifies several authorization patterns (e.g. token-based) for restricted access to the Things. We are going to extend the WoT Store by providing support to the mechanisms defined in [13] in order to keep the full standard compliance.

- *Functionalites*. The Data Manager as presented in the paper provides the template and few specific modules for data aggregation and visualization; however, commercial IoT applications ([2] [29]) often involve intensive data processing by means of Machine Learning (ML) techniques. We highlight that the focus of the WoT Store (and also of the Data Manager) is not on the IoT analytics, rather on the interoperability support; however, we might implement proper bindings in order to connect the Data Manager to existing IoT data processing platforms, hence enabling the processing of the data generated by a MA on external tools and the visualization of results within the WoT Store dashboard.

- *Data support*. The semantic description of Things and WoT applications (both MAs and TAs) are only sketched in the current proposal; a relevant contribution might be constituted by the definition of a reference WoT ontology (or extending available ones, e.g. the SWOT proposal [32]) in order to semantically characterize both Things and application meta-data within the same formalism, so that it will be possible to check the compatibility of MAs/TAs with deployed Things in a fine-grained way.

Finally, we are planning to further validate the operations of the WoT Store on large-scale IoT deployments, e.g. the MAC4PRO project [33] addressing structural monitoring in industrial environments.

References

[1] A. I. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys and Tutorials, 17(4), pp. 2347-2376, 2015*

[2] B. Di Martino, M. Rak, M. Ficco, A. Esposito, S. A. Maisto and S. Nacchia. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *Internet of Things, 1(1), pp. 99-112, 2018*

[3] F. Montori, L. Bedogni, M. Di Felice, L. Bononi. Machine-to-Machine Wireless Communication Technologies for the Internet of Things: Taxonomy, Comparison and Open Issues. *Pervasive and Mobile Computing, 50(1), pp. 56-81, 2018*

[4] M. Noura, M. Atiquzzaman, Martin Gaedke. Interoperability in Internet of Things: Taxonomies and Open Challenges. *Mobile Networks and Applications, 24(3), pp. 796-809, 2018*

[5] M. Donovan. Interoperability and the Internet of Things. *NDP Analytics*, Report, 2017.

[6] McKInsey Global Institute. The Internet of Things: Mapping the value beyond the hype. Executive Summary. 2015.

[7] Arrowhead - Ahead the Future. https://www.arrowhead.eu

[8] Big IoT - Bridging the Interoperability Gap of the Internet of Things. http://big-iot.eu

[9] Wise IoT - Worldwide Interoperability for Semantics IoT. http://big-iot.eu

[10] IPv6 over Low-Power Wireless Personal Area Network (6LoWPAN). https://tools.ietf.org/html/rfc8138

[11] A study of existing Ontologies in the IoT-domain. Garvita Bajaj, Rachit Agarwal, Pushpendra Singh, Nikolaos Georgantas, Valerie Issarny. Technical Report, HAL repository, https://hal.inria.fr/hal-01556256/document.

[12] L. Roffia, P. Azzoni, C. Cristiano, F. Viola, F. Antoniazzi, and T. Salmon Cinotti. Dynamic Linked Data: A SPARQL Event Processing Architecture *Future Internet*, 10(4), pp. 1-33, 2018.

[13] WoT Reference Architecture (W3C Candidate Recommendation 16 May 2019). http://www.w3.org/TR/wot-architecture/

[14] JSON-LD, JSON for Linking Data https://json-ld.org

[15] Eclipse Thingweb node-wot. Source repository: https://github.com/eclipse/thingweb.node-wot

[16] A. G. Mangasa, F. J. S. Alonso. WOTPY: A framework for web of things applications. *Computer Communications*, 147(1), pp. 235-251, 2019.

[17] Y. Ji, K. Ok and W. Suk Choi. Demo Abstract: Web of Things based IoT standard interworking test case. *Proc. of ACM BuildSys*, Shenzen, China, 2018.

[18] B. Klotz, S. K. Datta, D. Wilms, et al. A car as a semantic Web Thing: motivation and demonstration. *Proc. of IEEE GIoTS*, Bilbao, Spain, 2018.

[19] M. McCool and E. Reshetova. Distributed Security risks and opportunities in the W3C Web of Things. *Proc. of IEEE DISS*, San Diego, USA, 2018.

[20] M. Blank, S. Kaebisch, H. Lahbaiel, and H. Kosch Role models and lifecycles in IoT and their impact on the W3C WoT Thing Description. *Proc. of IEEE IoT*, Santa Barbara, USA, 2018.

[21] J.V. Joshua, D.O. Alao, S.O. Okolie, O. Awodele Software Ecosystem: Features, Benefits and Challenges. *International Journal of Advanced Computer Science and Applications(IJACSA)*, 4(8), 2013

[22] L. Sciullo, C. Aguzzi, M. Di Felice, T. S. Cinotti WoT Store: Enabling things and applications discovery for the W3C Web of Things. *Proc. of IEEE CCNC*, Las Vegas, USA, 2019.

[23] L. Sciullo, A. Trotta, L. Gigli, M. Di Felice. Deploying W3C Web of Things-based Interoperable Mash-up Applications for Industry 4.0: A Testbed. in *Proc. of IFIP WWIC*, Bologna, Italy, 2019.

[24] F. Paganelli, S. Turchi and D. Giuli. A Web of Things framework for RESTful applications and its experimentation in a smart city. *IEEE Systems*, 10(4), pp. 1412-1423, 2016.

[25] E. Mingozzi, G. Tanganelli and C. Vallati. CoAP proxy virtualization for the Web of Things. *Proc. of IEEE CloudCom*, Singapore, 2014.

[26] A. Kamilaris and M. I. Ali. Do "Web of Things platforms" truly follow the Web of Things?. *Proc. of IEEE WF-IoT*, Reston, USA, 2016.

[27] F. Serena, M. Poveda-Villalon and R. Garcia-Castro. Semantic discovery in the Web of Things. *Proc. of ICWE (Springer LNCS)*, Rome, Italy, 2017.

[28] E. Korkan, H. Hassine, V. Schlott, S. Kabisch and S. Steinhorst. WoTify: A platform to bring Web of Things to your devices. *W3C Workshop on the Web of Things*, Munich, Germany, 2019.

[29] E. Sisinni, A. Saifullah, S. Han, U. Jennehag and M. Gidlund. Industrial Internet of Things: Challenges, Opportunities, and Directions. *IEEE Transactions on Industrial Informatics*, 2018.

[30] A. Barto and R. S. Sutton Reinforcement Learning: An Introduction MIT Press, 1998

[31] A. Capponi, C. Fiandrino, B. Kantarci, L. Foschini, D. Kliazovich and P. Bouvry. A Survey on Mobile Crowdsensing Systems: Challenges, Solutions and Opportunities *IEEE Communications Surveys and Tutorials, 2019.*

[32] F. Antoniazzi, F. Viola. Semantic Web of Things Ontology (SWOT) https://fr4ncidir.github.io/SemanticWoT/

[33] MAC4PRO BRIC 2018 Project. https://site.unibo.it/mac4pro/it