Security Policy Enforcement for Networked Smart Objects

Sabrina Sicari*‡, Alessandra Rizzardi*, Daniele Miorandi§, Cinzia Cappiello†, Alberto Coen-Porisini*
*Dipartimento di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria,
via Mazzini 5 - 21100 Varese (Italy)
§U-Hopper srl, via A. da Trento 8/2, 38122 Trento, Italy
†Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy
‡Corresponding author
Email: {sabrina.sicari; alessandra.rizzardi; alberto.coenporisini}@uninsubria.it,
daniele.miorandi@u-hopper.com, cinzia.cappiello@polimi.it

***Abstract*—In the Internet of Things (IoT) heterogeneous technologies concur to the provisioning of customized services able to bridge the gap between the physical and digital realms. Security, privacy and data quality are acknowledged to represent key issues to be tackled in order to foster the large-scale adoption of IoT systems and technologies. One instrumental aspect concerns the ability of the system to preserve security in the presence of external attacks. In such a scenario, the integration of a flexible IoT middleware, able to handle a large number of data streams and of interconnected devices, with a flexible policy enforcement framework is needed and presented in this paper. The proposed solution aims to ease the management of interactions across different realms and policy conflicts. Its effectiveness is validated by means of a lightweight and cross-domain prototypical implementation.**

I. Introduction

The Internet of Things (IoT) [1] represents a vision of future technological ubiquity, where the ability of devices to connect to a global infrastructure enables to bridge the gap between the physical and digital realms. The diffusion of the IoT paradigm would allow the implementation and the diffusion of innovative and customized services in several applications fields. From a technological point of view, the term *'things'* is used to denote various physical everyday objects that embed electronics (e.g., wireless sensor nodes, actuators, RFIDs, and so on) to make them *smart* and suitable to be part of a global networked infrastructure. From a logical point of view, an IoT system can be characterised as a collection of smart devices which interact on a collaborative basis to fulfill a common goal, acquiring data from and acting upon the environment they are in.

In such a context, security & privacy represent critical requirements, which can hinder the large scale adoption and diffusion of IoT applications [1] [2] [3] [4] [5] [6]. Traditional security countermeasures and privacy solutions cannot be directly applied to IoT scenarios due to various reasons, including, but not limited to, energy and computing constraints, scalability etc. Moreover, adaptation and self-healing play a key role in IoT infrastructures, which must be able to face sudden and unexpected changes in the operational environment. Accordingly, privacy and security issues should be treated with a high degree of flexibility [7] [8]. Together with the conventional security solutions, there is also the need to provide built-in security in the devices themselves (i.e., embedded) in order to pursue dynamic prevention, detection, diagnosis, isolation and countermeasures against successful breaches [9].

Security and privacy are two pillars for ensuring the effectiveness of IoT services, the third one being data quality. IoT services should provide correct, complete and updated information: in some scenarios indeed errors or missing values might have critical impact on actions or decisions [10]. Keeping in mind the crucial role of the satisfaction of these security, privacy and data quality requirements, it is important to remark that in IoT context the number of violation attempts is high [2]. In other words, in order to deal with the huge amount of critical situations typical of the sharing approach of IoT paradigm, it is fundamental to adopt well-defined enforcement mechanisms able to successfully tackle them. Furthermore, IoT deployments are characterized by a high degree of heterogeneity in terms of architectures and technologies, so that a suitable security framework should be highly flexible in order to adapt to various deployment features.

In order to address such emerging issues, in this work we propose to integrate an existing flexible and distributed IoT middleware, called NetwOrked Smart objects (NOS) [11], with a policy enforcement framework. More in detail, the extended middleware has to provide a policy enforcement system able to manage the resources in a secure way and to handle attacks and violation attempts. NOS is represented, in a previous work, as a security-and quality-aware system architecture [12], and is based upon the concept of a computationally powerful smart nodes' layer acting as a distributed database able to manage IoT-generated data. The basic idea underpinning NOS is of bringing processing, security and data

qualification closer to the actual data sources. To ease the development of applications and the management of such a system, in [11], the NOS middleware has been designed and prototyped. It includes provisioning for users and applications to dynamically specify the levels of security and data quality suitable for their own purpose.

However, the original NOS architecture does not define supporting mechanisms for: (i) controlling the access of both users and data sources; (ii) the data provision to users. An enforcement system would allow to overcome such limitations. As regards the enforcement mechanisms, few efforts are currently made by the scientific community [2] [13]. To the best of the authors' knowledge, no specific enforcement solution for IoT is currently available, although it is essential to ensure a safe deployment of IoT paradigm. To address such shortcoming, in this paper we propose a policy enforcement system specifically tailored to IoT, able to manage the interactions among the involved entities under well-defined policies. The proposed solution is able to guarantee data quality, security and privacy also in the presence of policy violation attempts.

The paper is organized as follows. Section II reviews the relevant state of the art. Section III presents the NOS architecture, with a specific focus on data management aspects. Section IV describes the proposed enforcement framework. Section V and VI present the prototypical implementation of the NOS policy enforcement framework and its validation, in order to demonstrate the feasibility of the proposed approach in a real IoT context. Section VII concludes the paper and provides some hints for future works.

II. Related Works

The most crucial challenge in building an IoT system lies in the lack of common, standardised and interoperable software frameworks. In order to fill this gap, the scientific community has started several interesting research initiatives. For example, in recent years, the availability of web service solutions has provided a common frame for building systems able to leverage the services of another one according to the principles of Service Oriented Architectures (SOA). Service-oriented Communications (SOC) technologies emerged as a way to manage web services by creating a virtual network and adapting applications to the specific needs of users rather than forcing users to adapt to the available functionality of applications [14] [15]. Although the decision of adopting SOA architecture in IoT is shared by the majority of scientific community, at the moment the state of the art in this area is mostly limited to research and innovation activities [16] [17] with limited commercial uptake.

Furthermore, due to the very large number of heterogeneous technologies normally co-existing within IoT deployments, several middleware layers are employed to enforce the integration and the security of devices and data within the same information network. Within such middlewares, data must be exchanged respecting strict protection constraints. Moreover, in middleware design and development, different communication protocols shall be supported: while many smart devices can natively support IPv6 communications [18] [19], existing deployments might not support the IP protocol within the local area scope, thus requiring ad hoc gateways and supporting middlewares [20]. Recent works on IoT middlewares are: VIRTUS [21], which relies on the open eXtensible Messaging and Presence Protocol (XMPP) to provide secure event-driven communications; Otsopack [22] and Naming, Addressing and Profile Server (NAPS) [23] are data-centric frameworks based on HTTP and REpresentational State Transfer interfaces. We differentiate from them since: (i) [21] focuses only on the application of an authentication system and on securing the communication channel by means of encryption mechanisms; (ii) [22] and [23] address, respectively, ambient intelligence in constrained environments and resources naming management, without dealing with security issues.

Many relevant activities have taken place within the framework of EU R&D actions. The FP7 COMPOSE (Collaborative Open Market to Place Objects at your Service) project [24] aims to design and develop an open marketplace, in which data from Internet-connected objects can be easily published, shared and integrated into services and applications. The basic concept underpinning such an approach is to treat smart objects as services, which can be managed using standard service-oriented computing approaches and can be dynamically composed to provide value-added applications to end users.

The iCORE project (iCORE) [25] aims to empower IoT with cognitive technologies and is focused on the concept of virtual objects (VOs). VOs are semantically enriched virtual representations of the capabilities/resources provided by real world objects. Through the inception of VOs it becomes possible to easily re-use Internet-connected objects through different applications/services, also supporting their mash-up into composite services. VOs provide a unified representation for smart objects, hiding from the application/service developers low-level details as well as from underlying technological heterogeneity. They also provide a standardised way to access objects' capabilities and resources. One key element in the iCORE project is the use of advanced cognitive techniques for managing and composing VOs in order to improve IoT applications and better match user/stakeholder requirements. The considered application scenarios include ambient assisted living, smart office, transportation and supply chain management.

A dynamic architecture for services orchestration and adaptation has been proposed in IoT.EST (Internet of Things Environment for Service Creation and Testing) [26]. The project defines a dynamic service creation environment that gathers and exploits data from sensors and actuators that use different communication

technologies and formats. Such an architecture deals with different issues such as composition of business services based on re-usable IoT service components, automated configuration and testing of services for "things", abstraction of the heterogeneity of underlying technologies to ensure interoperability.

Focusing on semantic web services, the Ebbits project [27] designed a SOA platform based on open protocols and middleware, effectively transforming every subsystem or device into a web service with semantic resolution capability. The goal is to allow businesses to semantically integrate the IoT into mainstream enterprise systems and support interoperable end-to-end business applications.

Finally, security, privacy and trust issues are addressed by the uTRUSTit [28] and the Butler [29] projects. The former one is a project integrating the user directly in the trust chain, guaranteeing transparency in the underlying security and reliability properties of the IoT. If successful, uTRUSTit aims to enable system manufacturers and system integrators to express the underlying security concepts to users in a comprehensible way, allowing them to make valid judgments on the trustworthiness of such systems. Butler aims to allow users to manage their distributed profile allowing data duplication and identities control over distributed applications. The final purpose is to implement a framework able to integrate user dynamic data (i.e., location, behaviour) in privacy and security protocols.

Besides security and privacy levels, which means ability to guarantee confidentiality, integrity and anonymity requirements, in order to allow a real diffusion of IoT paradigm also data quality has to be addressed. As regards data quality, several scientific publications recognize its pivotal role in the IoT research landscape. In [30], authors claim the need of controlling data sources to ensure their validity, information accuracy and credibility. Data accuracy is also covered in [31], where the authors observe that the presence of many data sources raises the need to understand the quality of data. In particular, they state that the data quality dimensions to consider are accuracy, timeliness and the trustworthiness of the data providers. Anomaly detection techniques are widely employed to remove noise and inaccurate data in order to improve data quality. The huge number of data sources is considered a positive aspect for data fusion mechanisms and for the provisioning of advanced services. Besides temporal aspects (i.e., currency) and data validity, a related work adds another important dimension such as availability [32], with focus on pervasive environments. Authors defined new metrics for the cited quality dimensions in the IoT environment and evaluate the quality of the real-world data available on an open IoT platform. They show that data quality problems are frequent and they should be addressed or, at least, users should be aware of the poor quality of the data sources being used.

The definition of security and data quality policies may be not sufficient for satisfying the requirements of an IoT system, because violation attempts should also be considered. This requires the inclusion of policy enforcement mechanisms, which define how the system shall reach in such cases. More in detail, policies are operating rules which need to be enforced for the purpose of maintaining data order, security, and consistency. The policy enforcement assures that the security tasks can only be fulfilled if they are in accordance with the underlying security policies, consulting the policy decision component and deciding whether to allow an entity to perform an operation on a system resource. This aspect is poorly covered in existing literature, which mostly focuses on how to manage policy enforcement.

[33] presents a simulation environment for various policy languages, such as WS-Policy (Web Services-Policy) and XACML (eXtensible Access Control Markup Language), used in different systems. Low-level enforcement mechanisms may indeed vary from system to system. Thus, it is difficult to enforce a policy across domain boundaries or over multiple domains. Before applying policies across domain boundaries, it is desirable to know which policies can be supported by other domains, which are partially supported, and which are not supported. For example, in a healthcare environment, the cooperation and communication among pharmacies, hospitals and medical schools are essential. They have their own policy enforcement mechanisms to protect their own proprietary data and patients records. The problem is that there are lots of collaborations and communications among these actors, therefore a cross-domain policy enforcement becomes an essential component. However, in most cases, these domains use different policy languages. When a new interaction or communication is required between two separate domains, we do not know how many rules from one domain can be enforced by current enforcement mechanisms. So in most cases, the technical departments from these two domains have to work together to evaluate whether or not it is possible to make their systems interoperating. The same problem also exists in social networking environment (e.g., Facebook, Linkedin). Most existing social networking sites have privacy configurations based on their own enforcement mechanisms. When two social networking sites or two healthcare domains need to communicate or collaborate with each other, they have to rebuild or reconfigure their systems to make sure these activities are consistent with their own and their partners policies. In [33] a simulation environment is proposed, using semantic model mapping and translation for policy enforcement across domain boundaries by means of the Web Ontology Language (OWL), which can be used to model both policy languages and enforcement mechanisms. Therefore, a configurable middle-level component is presented for the mapping process among such different domains.

In [34] the languages regarding the definition of obligations and policies are classified into two categories. On the one hand, there are policy enforcement

languages, which generally simplify the specification and interpretation of policies; however, they lack the formal semantics needed to allow the verification of the policies themselves by means of formal proofs. On the other hand, there are policy analysis languages, which allow the formal policies analysis and the expression of a large variety of obligations. In [34], the authors introduce a policy language which aims at combining the advantages of both approaches. Formalizing policy enforcement has several advantages: it reduces the gap between the specified policies and their deployment, thus it ensures that the policies are correctly applied in the system. To formalize policy enforcement, the target system should be modeled and then the effects of the application of the policies should be described. More in details, policies are enforced using reference monitors, and a set of active rules specifies that a set of actions should be executed after the detection of some events, if some conditions are met. However, such a language does not provide the operational semantics needed to dynamically enforce and manage obligations in a policy managed system.

In [35] a novel access control framework, named Policy Machine (PM), is proposed. It is composed by the following basic entities: authorized users, objects, system operations, and processes. Users may be either human beings or system users; objects specify system entities which are controlled under one or more policies (e.g., records, files, e-mails); operations identify the actions that can be performed on such resources (e.g., read, write, delete); finally, users submit access requests through processes. Policies are grouped in classes according to their attributes and, therefore, an object may be protected under more than one policy class, and, similarly, a user may belong to more than one policy class. In such a way, PM is a general purpose protection machine, since it is able to configure many types of access control policies, and it is independent from the different operating systems and applications; users need to login only to PM in order to interact with the secure framework. [35] demonstrates the PM ability to express and enforce the policy objectives of RBAC [36], Chinese Wall [37], MAC and DAC models [38]. Moreover, PM is able to face many Trojan horse attacks, to which DAC and RBAC are vulnerable.

[39] and [40] introduce a semantic web framework and a meta-control model to orchestrate policy reasoning with the identification and access of information sources. In open domains indeed enforcing context-sensitive policies requires the ability to opportunistically interleave policy reasoning with the dynamic identification, selection, and access of relevant sources of contextual information. Each entity (i.e., user, sensor, application or organization) relies on one or more agents responsible for enforcing relevant policies in response to incoming requests. The framework is applicable to a number of domains where policy reasoning requires the automatic discovery and access of external sources.

[41] introduces a formal and modular framework allowing to enforce a security policy on a given concurrent system. In fact, one of the important goals of the software development process is to prove that the system always meets its requirements. To deal with this problem, two different approaches are proposed. The former is a conservative enforcement: the program should be terminated as soon as it violates the security policy even if the current run could be partially completed. The latter is a liberal enforcement: the execution of the process is not aborted if it could be partially satisfied. With this approach, more properties are enforced than with the conservative one, but the program may terminate without fully satisfying the security policy. Therefore, the conservative enforcement will generate false negatives, while the liberal enforcement will generate false positives and no one of them reach the desired result. In [41] an extended version of Algebra for Communicating Process (ACP) [42], designed for specifying concurrent systems behaviour, and Basic Process Algebra (BPA) language for the specification of security policies are used. To achieve the goal, ACP is enhanced with an enforcement operator, whose actions run in parallel with the system, in order to monitor the requests and the satisfaction of the related policies.

[43] provides an overview of network security, security policies, policy enforcement and firewall policy management systems. As far as policy enforcement is concerned, it proposes to use security services such as authentication, encryption, antivirus softwares and firewalls in order to protect the data confidentiality, integrity, and availability. In contrast, the authors of [44] present a framework able to prove whether the code implementing access control respects access control policy specifications.

Expressing security policies to govern distributed systems is a complex and error-prone task. Because of their complexity and of the different degrees of trust among locations in which code is deployed and executed, it is challenging to make these systems secure. Moreover, policies are hard to understand, often expressed with unfriendly syntax, making it difficult for security administrators and for business analysts to create intelligible specifications. In [45] a Hierarchical Policy Language for Distributed Systems (HiPoLDS) is introduced; it has been designed to enable the specification of security policies in distributed systems in a concise, readable and extensible way. HiPoLDS design focuses on decentralized execution environments under the control of multiple stakeholders. It represents policy enforcement through the use of distributed reference monitors, which control the flow of information among services and are in charge of putting into action the directives output by the decision engines. For example, an enforcement engine should be able to add or remove security metadata such as signatures or message authentication codes, encrypt confidential information, or decrypt it when it is the case. [45] does not specify how the distributed system behaves and manages policy

reconfiguration (e.g., if a reboot is required).

The authors of [46] state that the application logic, embodied in the system components, should be separated from the related policies. Therefore, they propose an infrastructure which can enable policy, representing high-level (i.e., user) or systems entities, able to drive the system functionality in a distributed environment. To this end, a middleware, able to support a secure and dynamic reconfiguration and to provide a policy enforcement mechanism across system components, is introduced. However, neither a case study nor a working implementation is presented.

Summarizing, there are no available solutions able to handle both security & privacy and data quality requirements in IoT environments at the same time. In fact, [33] and [46] mainly address cross-domain policy issues, [34] and [45] focus on a policy language definition, [35], as well as [39] and [40], enforces only access control policies, [41] and [43] are about a formal proof of the correct behavior of a system with not well defined security rules. A first attempt to consider both the issues is presented in [10], [12] and [11]. In the first, a general UML conceptual model for IoT architecture is defined, while in the second one a high level design of such an architecture is detailed. In the third, a real implementation of the architecture is presented, which, in this work, is integrated with an enforcement engine able to manage the defined policies and the interactions among the involved entities. Note that the identification of the enforcement solutions suitable for the specific IoT context is fundamental, finding a suitable tradeoff between the guarantee of security, privacy and data quality issues and the computing efforts. The enforcement framework proposed in this paper aims to fill this gap.

III. Architecture and Prototype

In a generic IoT system we can identify two main entities: (i) the nodes, heterogeneous devices (e.g., RFID, NFC, sensors etc.) which generate data (ii) the users, who interact with the IoT system through services making use of IoT-generated data, typically accessing them by means of a mobile device (e.g., smartphone, tablet) connected to the Internet (through, e.g., WiFi, 3G, Bluetooth). The NOS layer has been introduced in [11] in order to process such a huge amount of data closer to the sources and to better serve the user in terms of quality and security. NOSs are networked smart nodes without strict constraints in terms of energy and computational capabilities. They have self-organizing features and can be deployed where and when needed, in a distributed manner; through their interface with enterprise platforms and IoT enabling technologies, they can be used to extend existing software platforms, making them able to interact with the physical world following well-defined templates and rules. In general, a NOS would act as a gateway with built-in processing capabilities, able to manage a number of IoT data sources. Multiple NOSs may co-exist, each of them serving a subset of the IoT devices present in the environment.

A high-level architecture of NOS is presented in Figure 1 [12]. In the remainder of this section we present the various components in detail.

NOSs aim to handle in near real-time the large amount of data coming from heterogeneous IoT devices. Following a bottom-up analysis of the architecture, we start from the southbound NOS interface, which are used by NOS to collect data from IoT devices. NOSs are able to deal with both registered as well as non-registered sources. NOSs provide a specific REST endpoint for handling source registration. The information related to the registered sources is put into the storage unit named *Sources*. Registered sources may specify an encryption scheme for their interactions with NOSs. For each incoming data unit, the NOS extracts the following fields: (i) the data source, which describes the kind of node (e.g., sensor node, actuator, RFID); (ii) the communication mode, that is, the way in which the data is collected (e.g., discrete or streaming communication); (iii) the data model in use, which represents the type (e.g., number, text) and the format of the received data; (iv) the data itself; (v) the timestamp registering when the data arrived to NOS. HTTP is assumed to be used for communication among the NOS and the data sources. Since the received data are of different types and formats, NOSs initially put them in the *Raw Data* collection. Data in such collection is periodically processed, in a batch way, according to the two-phases scheme shown in Figure 1. Data goes through the *Data Normalization* and *Analyzers* phases in order to obtain a uniform representation of data, including, as specified in the following, metadata useful for optimsing and customising the service provision.

First, the data stored in *Raw Data* is put in the format specified in Figure 2 by the *Data Normalization* module and stored in the *Normalized Data* unit. This represents a sort of pre-processing phase in which the unnecessary information is removed from the data (where the 'unnecessary' depends on the specific application domain) and a uniform representation thereof is built; at this stage, security and quality metadata fields are still empty. Then, a second module, consisting of a set of *Analyzers*, periodically extract such data from the *Normalized Data* storage unit and elaborates it in terms of security and data quality properties). Such an analysis implies that the data are annotated with a set of metadata (i.e., a score for each security and quality level). A sample semantic description of the data content is shown in Figure 2. The data thus processed is ready to be used for providing services to the interested users. Therefore, in order to achieve such a goal, the NOSs layer can be connected to IP-based networks (i.e., Internet, intranet).

The assessment of security and quality levels are based on a set of rules stored in a proper format in another NOS storage unit, named *Config*. It is worth remarking that such rules are not the subject of this work, since the focus of this paper is on the enforce-

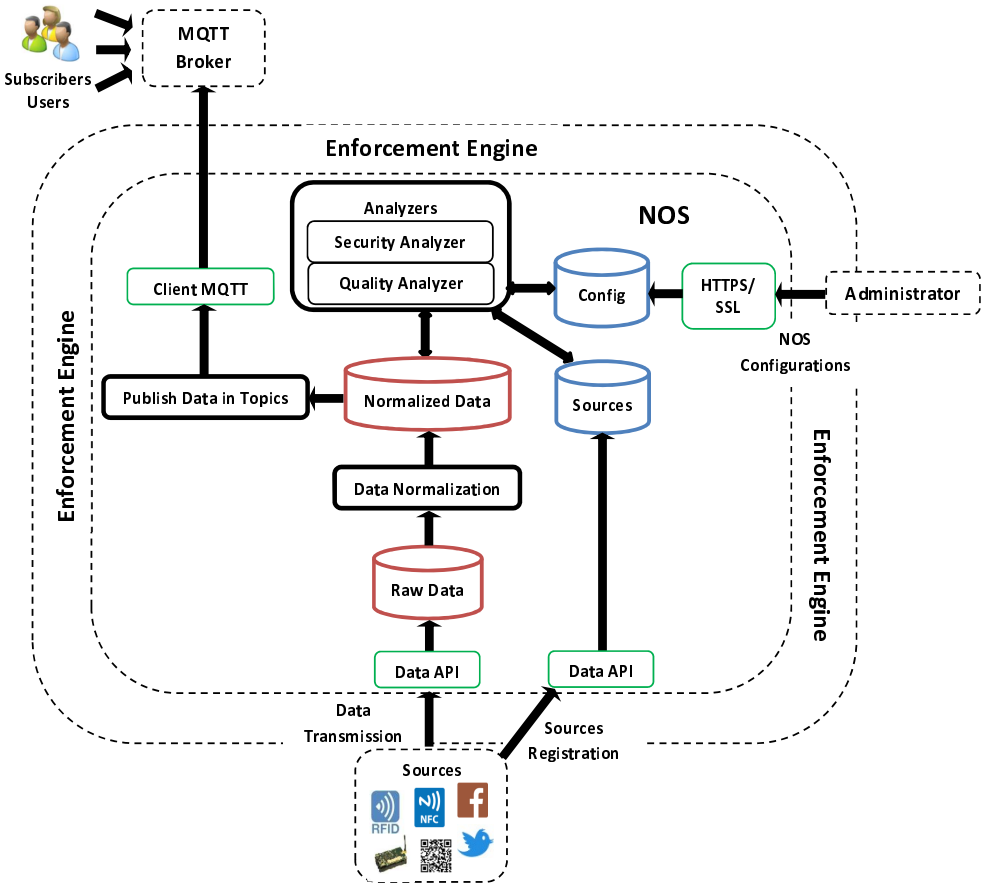

Fig. 1: System Architecture

ment mechanism. *Config* contains all the configuration information required for the correct management of the IoT system (including, e.g., how to calculate quality properties, which attacks or security countermeasures to consider etc.). Rules in the *Config* store can be dynamically configured at runtime by system administrators connecting remotely to the NOS over a secure connection (e.g., HTTPS, SSL) without the need to re-start the NOS services. The usage of a secure communication protocol is required in this case, as the policy adopted by the NOS for processing the IoT data has to be protected against external attacks. In this article we do not cover monitoring and event reporting (e.g., registered sources, source behavior, NOS performances, service utilization, occurred violations) aspects, yet it is understood that they should be included as in any operational system. Given the distributed nature of NOS a viable solution is represented by the popular ELK (Elastic, Logstash, Kibana) stack [1].

Analyzers query the *Config* store to retrieve a list of the operations they are intended to carry out. However, we remark that the assignment of security and quality scores provides a handle for letting the users filter directly by themselves the data processed by NOS according to their personal preferences. In fact, the choice to provide a score for each security and data quality dimension makes our approach extremely flexible and able to adapt to very different application scenario requirements. For example, there exist application scenarios (e.g., factory floor automation) in which only data with a high level of integrity and confidentiality shall be used, but there is no interest to satisfy privacy requirement. Another application domain may aim to provide a service characterized by error-free data and high confidentiality scores; therefore the data to be selected are those provided by sources able to satisfy these requirements. Such a feature makes our solution suitable for adoption in different contexts. In many situations indeed no description neither about the sources nor the acquired data may be available a priori; at the moment this requires labour-intensive search and selection of which data sources to use. Our approach automates such a task, leaving to the system administrator to define scoring policies and to the service provider to specify the requirements on the data to be used. The ability to support an automatic reasoning about data quality and security is what makes

[1] https://www.elastic.co/webinars/introduction-elk-stack

our approach able to deal properly with the scale and heterogeneity of IoT contexts.

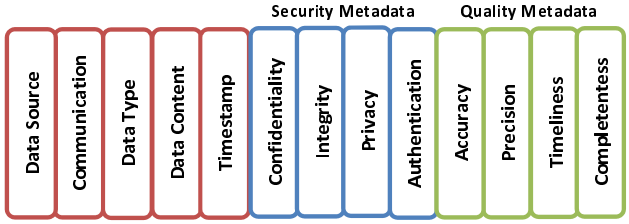

Fig. 2: NOS data format

NOS' northbound interface is based on the Message Queue Telemetry Transport (MQTT) protocol. It consists of a publish/subscribe mechanism, which aims to make the processed data available for interested applications and/or users. In fact, the system allows both the registration of users and of external applications, which authenticate to NOS and then make requests to the services made available by the NOS itself. In case of application registration, multiple users may register to such an application, instead of registering to NOS.

MQTT is a lightweight publish/subscribe connectivity protocol [47] specifically addressed for resource constrained devices. In IoT context, it is widely used to enable communications among devices using a publish/subscribe messaging approach. An MQTT client, as that contained in NOS, exchanges messages using an MQTT broker by means of publications and subscriptions to a topic. Such a mechanism is adopted to support interactions among services and IoT devices. In particular, each NOS has a module in charge of assigning the corresponding topic to data and, then, to send them to a MQTT client (module *Publish Data in Topics* in Figure 1). The assignment of the topics depends on the application domain and is out of the scope of this work, but it may need the definition of an ontology able to represent the semantics of the managed resources. In general, topics are multi-level structures separated by a forward slash similar to a directory structure. An example of a topic for publishing temperature information of a sensor with identifier *sensorId* could be *sensor/temperature/sensorId*. Note that subscribers may register for specific topics at runtime and NOS provides a mechanism for dynamic subscription e unsubscription to the topics. The MQTT client (*Client MQTT* in Figure 1) publishes messages to a MQTT broker.

In our prototypical implementation, which is openly accessible at https://bitbucket.org/alessandrarizzardi/nos.git we have used the Mosquitto [48] open-source MQTT broker, *Node.JS* platform [49] and *MongoDB* [50] for the data management part. In our implementation, NOS modules interact among themselves through *RESTful* interfaces. This allows us to add new modules or modify the existing ones at runtime, since they are able to work in parallel in a non-blocking manner. The non-relational nature of *MongoDB* allows the data model to evolve dynamically over time.

IV. Policy Enforcement

In order to effectively manage the available resources and to handle possible violation attempts, NOSs have to be provided with a set of well-defined policies, specifying the behaviour and the actions to be taken in a given situation. Accordingly, a fundamental role is played by the enforcement framework integrated in the NOS system, as it guarantees that the policies specified are correctly applied. In the NOS case, policies refer in particular to controlling access to IoT data and managing communications. This comes from the requirement to protect both data resources and user sensitive information. Technically, the main challenge to be faced is how to integrate an enforcement mechanism in the existing NOS architecture, without affecting the existing functionality. In our approach, as shown in Figure 1, the enforcement functionality is embedded in a wrapper layer, able to control the NOS operations without requiring major system-level modifications. In the remainder of this section we analyse the functionality in detail.

A. Enforcement Framework

The enforcement framework is in charge of handling access control and service provisioning under well-defined security and quality requirements. The framework is defined hereby to represent a redefinition of access control and data exchange in terms of a common set of functions and roles suitable for IoT applications. Functions and roles are dynamically configurable in order to provide the required level of flexibility to cover different application scenarios.

Conventional access control enforcement frameworks include a Policy Enforcement Point (PEP), a Policy Decision Point (PDP), and a Policy Administration Point (PAP) [51]. PEP is in charge of intercepting any requests of access to resources from users, and of making a decision request to PDP in order to obtain the access decision (i.e., approve or reject). Whenever a user or an application requests access to a data, this is routed through a PEP and transferred to a PDP for evaluation and authorization decision. PDP evaluates the access requests against the authorization policies in order to decide whether the request shall be accepted. To this end, the PDP refers to and queries a policies store. When the PDP completes the evaluation, it returns a response to the PEP. Based on such a decision, PEP either permits or denies access to the user/resource. The authorization policies are finally administered through a "centralized" PAP. The functions just described are usually performed by an application software. In our case, where communication is based on the MQTT protocol, all requests are handled via the MQTT broker (as also shown in Figure 1). The architecture underlying the framework may comprise one ore more NOS and a

huge amount of nodes, which act as data sources, and users, which act as data consumers (either directly or mediated by applications/services). Each NOS includes a PEP, a PDP and a PAP, while each user has an application representing an interface for the user personal device and the NOS. As far as nodes are concerned, a separate discussion has to be made, since the system has to be able to deal both with registered and non-registered nodes (i.e., data sources), while users may be directly registered to NOS or to another application, which is further registered to NOS. In the latter case, the application itself manages all the interactions with NOS and establishes the levels of security and quality for the data to be provided to the interested users. While, in the former case, a user, besides logs on the application running on his/her device using the provided GUI, opens a session, during which he/she can request for the services provided by NOS on the basis of the accessible resources. All components interact with the underlying PEP.

The structure of the presented enforcement framework is sketched in Figure 3. Although the figure shows only one NOS, the framework may be executed in a distributed manner on multiple NOS, whereby each NOS runs its own framework and a single application/service may interact with a plurality of NOSs. Therefore, the distribution of policies, their update and synchronization have to be considered (let us recall that this means, roughly speaking, to synchronize the content of the various instances of the *Config* store on multiple NOSs). We assume that, in the case of multiple NOSs interacting with each other, all NOSs within the same administrative domain share the same security policies and each of them has its own policy enforcement component.

Our approach follows the Attribute Based Access Control (ABAC) model [52]. In such a mechanism, both the subject who want to access or to provide the resources, and the objects (i.e., data), which represent the resources themselves, are described by means of specific attributes, which are used for the policies definition. Attributes can be based on the metadata fields natively supported in our data representation and control rules can be defined according to the specific needs of the application domain. As widely acknowledged in the relevant literature, ABAC presents better scalability and flexibility than Role Based Access Control (RBAC) [53].

To ease interoperability and to actually enable the implementation of a policy enforcement system, a policy representation language has to be chosen. Given the large number of IoT domain applications, such a language has to be flexible enough to represent the analyzed contexts both in a general-purpose and in a customizable way. The policy language proposed in this paper is specifically tailored to the management of enforcement, and is written in JSON syntax, being therefore suitable for the integration in the database management system used in our implementation (i.e., *MongoDB*). It allows to express the whole set of policies for each involved entity (i.e., nodes and users). Each of them has specific attributes, as described in the following section. According to the defined attributes, each entity can be allowed to perform different actions. The system allows the runtime change of policies, which can be dynamically loaded in the system through the aforementioned PAP.

B. Enforcement Engine

Security among the involved components (i.e., NOS, users, nodes) is guaranteed through the adoption of suitable encryption mechanisms. Another challenge is represented by the identification of a minimal set of primitives able to specify and enforce a large variety of attribute-based security and data quality policies. To this end, NOS are provided with an *Enforcement Engine* component, which is in charge of managing such policies for all involved entities. The *Enforcement Engine* implements to the PDP and PAP functions shown in Figure 3. An important feature of the proposed policy framework is that the *Enforcement Engine* also supports the loading of new policies at runtime, without disrupting service operations. Such a feature increases the flexibility of the framework and makes it particular suitable for IoT applications, which require a high degree of availability.

The advantage of the adopted policy-based control is that the controlling unit of the system (i.e., *Enforcement Engine*) is kept decoupled from other management components (i.e., *Data Normalization* and *Analyzers* phases). As a consequence, the system administrator can manage and change the system behaviour without modifying the software or the user/node interfaces. Furthermore, the entire system is controlled by policies which specify the rules interpreted and enforced by *Enforcement Engine*. Hence, if the conditions change or new services or applications are added, only the corresponding policy rules have to be adapted. Within NOS all the security related tasks are executed seamlessly so that services are not required to have explicit knowledge of security policies.

V. Implementation of the enforcement framework

In this section we present a prototypical implementation of the policy enforcement framework defined in the previous section. The key component in the NOS architecture is for us the *Enforcement Engine*, which is in charge of ensuring that the system satisfies the security and quality requirements of authorized users/nodes. Policies are applied to two types of entities: data producers (in our context: IoT devices or nodes, as we term them) and data consumers (users directly or applications). Six key actions for which policies are specified have been identified and formally described: node access control, node data transmission, node data processing, user/application access control,

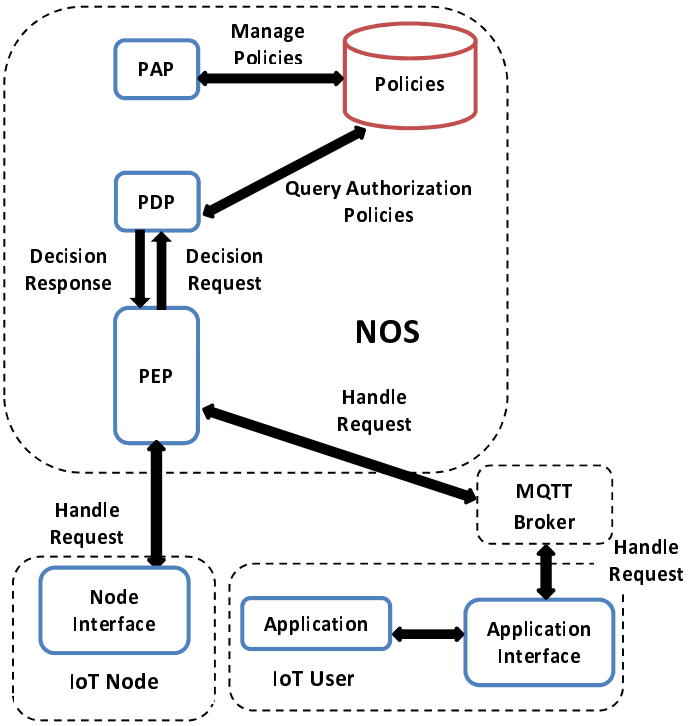

Fig. 3: Enforcement System

user/application service request, and service provision. In line with the ABAC approach used, policies are specified as a set of key-value pairs, each pair representing an attribute of the corresponding policy.

In our framework, a policy is composed of three main building blocks. The first one (*input*) defines the values that the NOS expects to receive in input from the requesting entity (nodes or users/applications) and that are used for evaluating the policy for a specific action. The second one (*security*) defines the functions to be executed on the provided inputs to assess the policy. Each function returns a value; such values are composed by means of the logic specified in the third block (*response*) to define whether the request shall be accepted or not.

In our implementation, the policies are represented in JSON format and are stored in the *Config* storage unit. The character @ is used to indicate the value taken by the corresponding field. In the following, we present some sample policy specifications.

In our system three types of entities are present: users, applications and nodes. Users and applications consume data and they must be registered. Nodes generate data; as explained in detail above the system accepts also data generated by non-registered (anonynmous) nodes. The user/application registration phase takes place through an exchange of credentials between the user/application and the NOS. In order to perform registration operations, a user/application must be authenticated with admin privileges. From the operational perspective, we do expect that entities consuming data (i.e., users/applications) will, in most cases, be registered by a system administration. When registering, an identifier is assigned by the system to each registered user/application, along with a function, conceived as a set of attributes used for filtering the access to resources (an example is presented in Section VI). Note that such attributes and the related access permissions are established by a system administrator, decoupled from NOS.

On the other hand, as already introduced in Section III, nodes can optionally self-register with the NOS. Note that the NOS also assigns an identifier to the registered nodes, for which in the registration phase is specified the signature key used for encrypting the data they send. Such credentials are eventually exchanged between the node and NOS through the proper *Sources Registration* interface.

Listing 1 describes a sample version of the *NodeAccessControl* policy, which covers nodes wanting to send data to the NOS. This policy is invoked by the NOS before inserting the data into the *Raw Data* storage unit. The *Enforcement Engine* verifies whether the node is transmitting, along with the data, the node identifier and a signature key (specified as *node inputs* in Listing 1). The verification process is split in two branches. If the source is a registered one, the NOS receives in input the node identifier and the signature key and can perform a *registrationCheck* (i.e., checking whether the identifier is known and valid) and the *signaturekeyCheck*

(i.e, checking that the identifier and key are compliant for the requesting node). Conversely, if the source is unknown, the key is marked as *undefined* by the function *signaturekeyUndefinedMark* for the corresponding node. In case of a non-registered node, the NOS keeps track of the source by assigning to it a pseudo-random identifier at the first communication exchange; such an identifier will be used in following interactions. This approach allows a NOS to verify whether the node is a new or a known one just by looking up its identifier. If *registrationCheck* and/or *signaturekeyCheck* reveal that the credentials are not compliant with the requesting source, then the enforcement engine prevents such a node from interacting with NOS.

Once these checks have been passed and the node is allowed to send data to the NOS, a pseudo-random session identifier is created and assigned by the *sessionAssignment* function. Data quality and security are assessed on a per-session basis.

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "@NodeID",
      "signaturekey": "@Key"
    }
  },
  "security": [ {
    "verificationRegistration": [{
      "registrationCheck": "@NodeID",
      "signaturekeyCheck": "@NodeID,@Key"
    }],
    "verificationUnknownSource": [{
      "signaturekeyUndefinedMark": "undefined",
      "identifierCheck": "@NodeID"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "@NodeID, timestamp"
    },
    {
      "signaturekeyCheck": false,
      "accessDenied": "@NodeID"
    }],
    "verificationUnknownSource": [{
      "sessionAssignment": "@NodeID, timestamp"
    }]
  }]
}}
```

Listing 1: Node access control sample policy

Once the node has completed the access control phase, then it can send data to NOS. Listing 2 highlights the requested inputs for the corresponding policy, named *NodeDataTransmission*, which are: the session identifier, previously assigned by the NOS to the node during the access control phase; the node identifier; the data itself and the data type. Note that at this stage no security operations have been performed yet: if all the requested inputs are present (i.e., *requiredInformation* action is *true*), the data are stored in *Raw Data* storage unit (i.e., *storeData* action is activated for the actual node). Data gets discarded only if the transmitting node fails to provide the required information to the NOS (i.e., *discardData* action is undertaken).

```
{ "NodeDataTransmission": {
  "policy": "NodeDataTransmission",
  "input": {
    "message": {
      "session": "@Session",
      "identifier": "@NodeID",
      "data": "@d",
      "datatype": "@dt"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "storeData": "@NodeID,@d,@dt"
    },
    {
      "requiredInformation": false,
      "discardData": "@NodeID,@d,@dt"
    }]
}}
```

Listing 2: Node data transmission sample policy

As detailed in Section III, in the NOS there are processing modules that periodically fetch data from *Raw Data* or *Normalized Data* storage units and process them. The policy invoked in this step is called *NodeDataProcessing* and, as shown in Listing 3, receives in input the same values of the *NodeDataTransmission* policy, but the action of data evaluation is enforced, before sending processed data to the publish/subscribe system. For registered sources, the NOS performs *decryptionData* and *decryptionDatatype* operations, thus decrypting the data and the corresponding data type using the key of the actual node. Hence, for both registered and non-registered sources, *scoreAssessment* is executed and a score for each security and quality property is assigned to the data.

```
{ "NodeDataProcessing": {
  "policy": "NodeDataProcessing",
  "input": {
    "message": {
    "session": "@Session",
    "identifier": "@NodeID",
    "data": "@d",
    "datatype": "@dt"
    }
  },
  "response": [{
    "evaluationRegisteredSource": [{
      "decryptionData": "@d,@NodeID",
      "decryptionDatatype": "@dt,@NodeID",
      "scoreAssessment": "@d,@NodeID"
    }],
    "evaluationNonRegisteredSource": [{
      "scoreAssessment": "@d,@NodeID"
    }]
  }]
}}
```

Listing 3: Node data processing sample policy

Listing 4 refers to the access request from a user/application who wants to receive data from a NOS;

such a request is sent to the MQTT broker, which performs an access request to the *Enforcement Engine*. The corresponding policy, named *UserAccessControl*, is invoked before sending any data to the requesting entity. It verifies whether the user/application is registered, using for such a purpose the following parameters: user name, user/application identifier, signature key and a function, previously specified during the registration phase. The policy verification process includes two cases. If the user/application is registered, the NOS receives in input the user/application identifier, the function and the signature key and can perform *registrationCheck* (i.e., the identifier is known and valid for the specified function) and *signaturekeyCheck* (i.e, identifier, function and key are compliant for the requesting user/application). If the user/application is unknown, then the key is marked as *undefined* by the action *signaturekeyUndefinedMark* and the user/application is not authorized by the enforcement engine to access the system. If a user/application tries to register with wrong credentials, for example with a function different from the one declared during the registration phase or with a different key, then the enforcement engine generates a negative response, alerts the user/application (see Figure 6 in Section VI) and does not allow any interactions with IoT system. Otherwise, in the case that the checks have been passed, the user/application is allowed to interact with NOS and a pseudo-random session identifier is assigned by the *sessionAssignment* function.

```
{ "UserAccessControl": {
  "policy": "UserAccessControl",
  "input": {
    "user": {
      "username": "@Username",
      "identifier": "@UserID",
      "signaturekey": "@Key",
      "function": "@Function"
    }
  },
  "security": [{
    "verificationRegistration": [{
      "registrationCheck": "@UserID,@Function",
      "signaturekeyCheck": "@UserID,@Key,
    @Function",
    }],
    "verificationUnknownUser": [{
      "signaturekeyUndefinedMark": "undefined"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "@UserID,@Function,
    timestamp"
    },
    {
      "signaturekeyCheck": false,
      "accessDenied": "@UserID,@Function"
    }],
    "verificationUnknownUser": [{
      "accessDenied": "@UserID,@Function"
    }]
  }]
}}
```

Listing 4: User access control sample policy

Once the user/application has completed the access control phase and is authenticated, then it can receive data from NOS by activating the corresponding subscription to the MQTT broker. Listing 5 highlights the requested inputs for the corresponding policy, named *ServiceRequest*, which are: (i) a session identifier, given by NOS to the user/application after the access control phase (computed randomly at each access, as for the nodes); (ii) the username and the identifier; (iii) the function; (iv) the requested service, along with the user preferences in terms of security and quality.

For the service invocation, as already discussed in Section III, we treat a resource as an object identified by a hierarchical name (e.g., a URI). A service is conceived as a software able to fulfill a specific task making use of the available data. There is no direct interaction among users/applications and NOS resources, but a well-defined programming interface is needed through a software application. Resources can be accessed by users/applications only once they are published as object instances. Note that at this stage no security operations are performed: the request is elaborated by the system if all the inputs are valid (i.e., *requiredInformation* action is *true*); if not, the request is discarded by the enforcement engine (i.e., *requiredInformation* action is *false*). The security and quality preferences are not mandatory: if they are omitted, the enforcement engine does not discard the request, but sets the corresponding constraints to the lowest admissible values.

```
{ "ServiceRequest": {
  "policy": "ServiceRequest",
  "input": {
    "message": {
      "session": "@Session",
      "username": "@Username",
      "identifier": "@UserID",
      "function": "@Function",
      "service": "@Service",
      "securityPreferences": "@confidentiality,
     @integrity, @privacy, @authentication",
      "qualityPreferences": "@accuracy,
    @precision, @timeliness, @completeness"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "processRequest": "@Session,@Username,
    @UserID,@Function,@Service,
    @confidentiality, @integrity, @privacy,
    @authentcation,@accuracy, @precision,
    @timeliness, @completeness"
    },
    {
      "requiredInformation": false,
      "discardRequest": "@Session,@Username,
    @UserID,@Function,@Service"
    }]
  }]
}}
```

Listing 5: User service request sample policy

Finally, the *ServiceProvision* policy is activated after a data request, in order to verify the matching between the request itself and the requesting user/application, in terms of identifier and function, by performing *serviceAccessVerification* action (Listing 6). Such a policy receives in input the same values of the *ServiceRequest* policy. Note that the parameters describing the requested data are sent encrypted by the requesting user/application. Therefore, the NOS has to decrypt it (i.e., *decryptionRequest* action). From the identifier, the NOS derives the signature key of the authenticated user/application and uses it to decrypt the message. After the verification step, the *retrieveResults* action is performed. It retrieves the data corresponding to the requested service, for which the user/application is allowed to access. Before sending them back, the retrieved data are filtered on the basis of the security and quality constraints. In case there is no matching among the described parameters (i.e., the user with the specified identifier and function is not allowed to access the requested service), the enforcement engine blocks the data provision process and sends an error message to the requesting entity.

```
{ "ServiceProvision": {
  "policy": "ServiceProvision",
  "input": {
    "message": {
      "session": "@Session",
      "username": "@Username",
      "identifier": "@UserID",
      "function": "@Function",
      "service": "@Service",
      "securityPreferences": "@confidentiality,
      @integrity, @privacy, @authentication",
      "qualityPreferences": "@accuracy,
      @precision, @timeliness, @completeness"
    }
  },
  "security": [{
      "decryptionRequest": "@Service,@UserID",
      "serviceAccessVerification": "@Service,
      @UserID,@Function"
  }],
  "response": [ {
      "serviceAccessVerification": true,
      "retrieveResults": "@Service,@UserID,
      @Function,@confidentiality, @integrity,
      @privacy, @authentcation,@accuracy,
      @precision, @timeliness, @completeness"
    },
    {
      "serviceAccessVerification": false,
      "accessDenied": "@Session,@Username,
      @UserID,@Function,@Service"
    }]
}}
```

Listing 6: Service provision sample policy

VI. Validation and Evaluation

In order to verify the effectiveness of the proposed solution, we developed a simple use case based on the usage of open IoT data feeds. In particular, we relayed on six sensors measuring weather-relevant parameters and co-located within the meteorological station in the small town of Campodenno (Trentino, Italy) and can be accessed through the Trentino Open Data portal[2]. The measurements cover temperature, humidity, wind, energy consumption and air quality parameters.

TABLE I: Source parameters

Parameters	*Source 1*	*Source 2*	*Source 3*	*Source 4*	*Source 5*	*Source 6*
Authentication	1	1	0	1	0	0
Security schema score	10	6	0	2	0	0
Privacy schema score	10	6	0	2	0	0
Timeliness	9	8	6	3	9	7
Completeness	9	10	8	6	7	10
Accuracy	9	7	5	6	7	10
Precision	9	8	4	5	8	9

In the experimental setup, the prototypical NOS implementation is deployed on a Raspberry Pi. A laptop is used to emulate the behaviour of a set of nodes, basically reading data from the aforementioned feeds and sending them to the NOS as if they came from six different nodes. Laptop and Raspberry communicate via a WiFi network. The laptop runs also a simple visualization service, which fetches, according to user-defined constraints, data from the NOS and displays them. User can express constraints in terms of the required security and quality levels, including aspects such as confidentiality, integrity, privacy, authentication, completeness, timeliness, and accuracy, as shown in the dashboard in Figure 4. As an example, we assigned to the six different data sources considered the security and quality scores reported in Table I.

Fig. 4: User Dashboard

A first analysis carried out concerns the storage capacity required by the system for carrying out its operations. In this respect, it is worth recalling that NOSs do not support persistent storage of IoT-generated data. Rather, data are temporarily cached on the NOS while being processed before being submitted to the MQTT broker. NOS has therefore to provide only a temporary storage. When data are further pushed to or pulled from

[2]http://dati.trentino.it/dataset/raw-data-in-near-realtime-stazione-cmd001

the server which handles the topics notification to subscribers, data can be safely flushed from the NOS. In our prototypical implementation, we used the in-memory capability of *MongoDB* for *Raw Data* and *Normalized Data* collections, while *Config* and *Sources* databases are persistently stored on the local hard disk. Since NOS runs on a Raspberry Pi, the maximum storage capacity for IoT data with the actual technology corresponds to 1 gigabyte (i.e., the RAM provided by Raspberry Pi 2 and 3). In our implementation, we include a routine in charge of removing from *Raw Data* the data already normalized and from *Normalized Data* the data already published. We measured the memory occupancy of NOS during operations, which resulted in an average slightly less than 7 megabyte. Such a value is only indicative, as the memory occupancy depends on a number of factors, notably: (i) the frequency of data fetching from sources (in our example 10 packets per second); (ii) the frequency of execution of the routines for removing data from non-persistent collections (in our example every 5 minutes); (iii) the number of sources.

A further evaluation is performed in order to estimate the latency introduced by NOS and the policy enforcement framework. Figure 5 shows the results obtained from one run of one NOS prototype with the six data sources just described over a period of one hour for two different reading data rates (i.e., how often the NOS queries the data sources to fetch data), 10 and 20 packets per second, respectively. The graph shows that the mean delay is almost constant over time. Furthermore, the introduced latency does not exceed 6.5 ms in our test case, a promising result in terms of the ability of the solution to deal with near real-time analysis of IoT data.

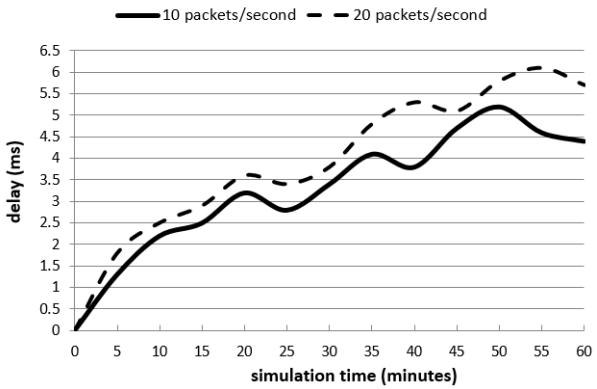

Fig. 5: Latency introduced by the NOS and the policy enforcement framework.

As regards the enforcement actions undertaken by NOS, we have analyzed: (i) the node access control for a registered source; (ii) the node access control for a non-registered source; (iii) the data transmission for a registered source; (iv) the data processing for a registered source; (v) the user access control along with a violation attempt; (vi) a user service request along with a violation attempt; (vii) a user service provision.

Before sending their data to NOS, the registered sources perform the access control operation. For example in Listing 7 is represented the policy activated for *Source 1*: the request is valid if the signature key is compliant with the one owned by NOS. We suppose that the session identifier randomly generated by the node identifier and the actual timestamp is *2016-03-24 09:15:00* (as shown in Listings 8 and 9).

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "1",
      "signaturekey": "*****"
    }
  },
  "security": [ {
    "verificationRegistration": [{
      "registrationCheck": "1",
      "signaturekeyCheck": "1,*****"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "1, 2016-03-24 09
    :15:00"
    }]
  }]
}}
```

Listing 7: Node access control - registered source

In Listing 8 the same action is presented for the non-registered *Source 3*; in this case the enforced action is the tracking of the node.

```
{ "NodeAccessControl": {
  "policy": "NodeAccessControl",
  "input": {
    "node": {
      "identifier": "3"
    }
  },
  "security": [ {
    "verificationUnknownSource": [{
      "signaturekeyUndefinedMark": "undefined",
      "identifierCheck": "3"
    }]
  }],
  "response": [ {
    "verificationUnknownSource": [{
      "sessionAssignment": "1, 2016-03-24 09
    :15:00"
    }]
  }]
}}
```

Listing 8: Node access control - non-registered source

Now we suppose that the registered *Source 1* transmits data to NOS (Listing 9). Function *encr* specifies that the parameters of the message are encrypted. The only difference between this node and a non-registered node is that, in the latter case, the parameters of the message would not be encrypted. The enforcement engine verifies whether the four requested values (i.e.,

session, identifier, data, datatype) are present in the message received by NOS. If this is not the case, the NOS discards the message and, possibly, prevents other communications with the same source. Note that a message sent from a node may include also other data (which depend on the specific device) besides the ones requested by the policy (e.g., a timestamp, a location): the policy specifies conditions on the mandatory ones only.

```
{ "NodeDataTransmission": {
  "policy": "NodeDataTransmission",
  "input": {
    "message": {
      "session": "12345",
      "identifier": "1",
      "data": "encr(10.7)",
      "datatype": "encr(double wind speed)"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "storeData": "1,encr(10.7),encr(double
    wind speed)"
    }
        }]
  }]
}}
```

Listing 9: Node data transmission

After validating such data, NOS can process them. Listing 10 shows the corresponding processing policy.

```
{ "NodeDataProcessing": {
  "policy": "NodeDataProcessing",
  "input": {
    "message": {
    "session": "12345",
    "identifier": "1",
    "data": "encr(10.7)",
    "datatype": "encr(double wind speed)"
    }
  },
  "response": [{
    "evaluationRegisteredSource": [{
      "decryptionData": "encr(10.7), 1",
      "decryptionDatatype": "encr(km/h wind
    speed), 1",
      "scoreAssessment": "10.7, 1"
    }]
  }]
}}
```

Listing 10: Node data processing

Now we suppose that the user *Bob* had registered himself to the weather service with the username *Bob* and *473* as identifier, in order to be notified about the measurements acquired in the considered area for some monitoring actions he has to do for his employer. Therefore he registers himself with the role of *Monitor*. Firstly, he has to perform the access control: Listing 11 describes the activated policy. We suppose that the generated session identifier is *13240*. If the credentials are not valid, it could be a violation attempts, then the enforcement engine forces the NOS to prevent user interactions, as illustrated in Figure 6.

```
{ "UserAccessControl": {
  "policy": "UserAccessControl",
  "input": {
    "user": {
      "username": "Bob",
      "identifier": "473",
      "signaturekey": "*****",
      "function": "Monitor"
    }
  },
  "security": [{
    "verificationRegistration": [{
      "registrationCheck": "473,Monitor",
      "signaturekeyCheck": "473,*****,Monitor"
    }]
  }],
  "response": [ {
    "verificationRegistration": [{
      "registrationCheck": true,
      "signaturekeyCheck": true,
      "sessionAssignment": "473,Monitor,
    2016-03-24 09:20:00"
    }]
  }]
}}
```

Listing 11: User access control

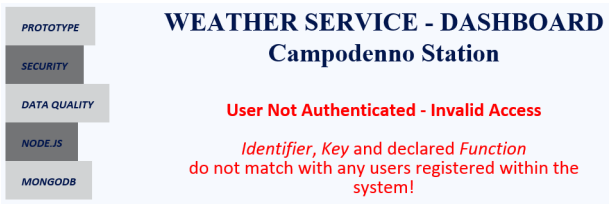

Fig. 6: Violation attempts of user access control

Otherwise, if the credentials are correctly verified, the user can make the desired service requests. Listing 12 shows an example of the corresponding policy invoked for requesting the service, named *Humidity and Wind Speed Real Time Measurements*, along with the security and quality constraints.

```
{ "ServiceRequest": {
  "policy": "ServiceRequest",
  "input": {
    "message": {
      "session": "13240",
      "username": "Bob",
      "identifier": "473",
      "function": "Monitor",
      "service": "Humidity and Wind Speed Real
    Time Measurements",
      "securityPreferences": "2, 0, 2, 1",
      "qualityPreferences": "4, 4, 4, 4"
    }
  },
  "response": [ {
    "verificationInput": [{
      "requiredInformation": true,
      "processRequest": "13240,Bob,473,Monitor,
    Humidity and Wind Speed Real Time
    Measurements,2, 0, 2, 1,4, 4, 4, 4"
    }
  }]
```

```
}}
```

Listing 12: User service request

At this point, such a request has to be analyzed in order to establish if the user is entitled to receive the data corresponding to the service (Listing 13 represents a user which has access to requested data).

```
{ "ServiceProvision": {
  "policy": "ServiceProvision",
  "input": {
    "message": {
      "session": "13240",
      "username": "Bob",
      "identifier": "473",
      "function": "Monitor",
      "service": "Humidity and Wind Speed Real Time Measurements",
      "securityPreferences": "2, 0, 2, 1",
      "qualityPreferences": "4, 4, 4, 4"
    }
  },
  "security": [{
      "decryptionRequest": "Humidity and Wind Speed Real Time Measurements,473",
      "serviceAccessVerification": "Humidity and Wind Speed Real Time Measurements,473, Monitor"
  }],
  "response": [ {
      "serviceAccessVerification": true,
      "retrieveResults": "Humidity and Wind Speed Real Time Measurements,473,Monitor, 2, 0, 2, 1, 4, 4, 4, 4"
    }]
}}
```

Listing 13: Service provision

There are two possible outcomes: (i) the user with *Monitor* function is allowed by the sensors data administrator to access the requested measurements for the monitoring scope; (ii) the user is not allowed to access these data for the declared scope. In the former case, the dashboard shown to the user is the one represented in Figure 7(d); from here, the user can also change dynamically the security and quality settings as can be seen in Figure 7. If no security or quality constraint is specified, all data is considered valid and the resulting graphs for wind speed (in Km/h) and humidity (in %) look as in Figure 7(a). In Figure 7(b) some security filters are applied; in particular, the system shows to the user only the data (i) provided by authenticated sources, (ii) for which the integrity is verified, (iii), for which the level of privacy and confidentiality is equal or higher than 6. Obviously in this case some data is dropped, as it fails to meet the criteria specified by the user in terms of security and quality of the data to use. The graph in Figure 7(c) is obtained without any constraint specified on security, but considering valid only data scoring at least 6 in completeness and timeliness, and 8 in accuracy and precision. In the latter case, it would be a violation attempt, then the response of the system is the one shown in Figure 8.

We remark that this represents only an example of a very simple NOS application in a context characterised by the analysis of real-time data. Other possible applications include energy management in a smart home/smart building scenario; monitoring of business processes and productive activities in real time; smart retail experiences services and, more in general, any application/service where decisions (either manual or automated) have to be taken based on IoT-generated data. This class of applications is expected to play a key role in the adoption of IoT technologies across a variety of vertical domains.

One aspect which deserves some further clarifications refers to the fact that in our example we considered one single NOS. While indeed we aim to deploy the presented middleware in a distributed environment, from an analysis of NOS functionality it is not difficult to see that no NOS-to-NOS coordination is strictly required. In fact, NOSs are able to: (i) independently handle the data sources, without the need to inform the other NOSs of their active and past interactions; (ii) be independently re-configured by IoT system administrators; (iii) independently assign topics and publish data on the basis of the defined rules; (iv) enforce the application of the policies defined for the IoT system. We can safely conclude therefore that considering a single NOS-scenario for validation purposes does not represent a limiting factor.

VII. Conclusions

Security and data quality issues represent potential show-stoppers for the market take-up of IoT-based products and services in various operational scenarios and vertical application domains. To tackle these issues, in this article we have introduced and discussed a flexible security and data quality enforcement framework, coherently integrated within a distributed IoT middleware platform.

The presented framework supports security and data quality enforcement policies, re-usable across different domains and able to detect violation attempts. The feasibility and performance of the proposed approach have been validated by means of a prototypical implementation and the development of a simple, yet real-world, use case.

In the next future we will focus on the deployment of the middleware and the framework in a large-scale pilot focussed on building automation, in order to test its robustness and scalability in operational environment.

References

[1] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Survey internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[2] S. Sicari, A. Rizzardi, L. A. Grieco, and A. Coen-Porisini, "Security, privacy and trust in internet of things: The road ahead," *Computer Networks*, vol. 76, pp. 146–164, 2015.

[3] R. H. Weber, "Internet of things - new security and privacy challenges," *Computer Law & Security Review*, vol. 26, no. 1, pp. 23–30, January 2010.

(a) No_Filters

(b) Security_Filters

(c) Quality_Filters

(d) All_Filters

Fig. 7: Results

Fig. 8: Violation attempt of user service request

[4] H. Feng and W. Fu, "Study of recent development about privacy and security of the internet of things," in *2010 International Conference on Web Information Systems and Mining (WISM)*, Sanya, October 2010, pp. 91–95.

[5] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Computer Networks*, vol. 57, no. 10, pp. 2266–2279, July 2013.

[6] J. Anderson and L. Rainie, "The internet of things will thrive by 2025, PewResearch Internet Project," *http://www.pewinternet.org/2014/05/14/internet-of-things/, May 2014*, May 2014.

[7] S. Bandyopadhyay, M. Sengupta, S. Maiti, and S. Dutta, "A survey of middleware for internet of things," in *Third International Conferences, WiMo 2011 and CoNeCo 2011*, Ankara, Turkey, June 2011, pp. 288–296.

[8] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the internet of things," in *2012 International Conference on Collaboration Technologies and Systems (CTS)*, Denver, CO, May 2012, pp. 21–26.

[9] S. Babar, A. Stango, N. Prasad, J. Sen, and R. Prasad, "Proposed embedded security framework for internet of things (iot)," in *2011 2nd International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace and Electronic Systems Technology, Wireless VITAE 2011*, Chennai, India, March 2011, pp. 1 – 5.

[10] S. Sicari, A. Rizzardi, C. Cappiello, and A. Coen-Porisini, "A NFP model for internet of things applications," in *Proc. of IEEE WiMob*, Larnaca, Cyprus, Oct 2014, pp. 164–171.

[11] A. Rizzardi, D. Miorandi, S. Sicari, C. Cappiello, and A. Coen-Porisini, "Networked smart objects: Moving data processing closer to the source," in *2nd EAI International Conference on IoT as a Service*, Oct 2015.

[12] S. Sicari, C. Cappiello, F. D. Pellegrini, D. Miorandi, and A. Coen-Porisini, "A security-and quality-aware system architecture for internet of things," *Information Systems Frontiers*, pp. 1–13, 2014.

[13] S. Sicari, S. Hailes, D. Turgut, S. Sharaffedine, and D. U., *Security, Privacy and Trust Management in the Internet of Things era- SePriT*, 11th ed. Special Issue of Ad Hoc networks, Elsevier, 2013.

[14] M. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann,

"Service-oriented computing: State of the art and research challenges," *Computer*, vol. 40, no. 11, pp. 38–45, Nov 2007.

[15] Q. Yu, X. Liu, A. Bouguettaya, and B. Medjahed, "Deploying and managing web services: Issues, solutions, and directions," *The VLDB Journal*, vol. 17, no. 3, pp. 537–572, May 2008.

[16] "Peertrack," http://cs.adelaide.edu.au/peertrack/.

[17] "Perci (pervasiveservice interaction)," http://www.hcilab.org/projects/perci/index.htm.

[18] M. Palattella, N. Accettura, X. Vilajosana, T. Watteyne, L. Grieco, G. Boggia, and M. Dohler, "Standardized protocol stack for the internet of (important) things," *Communications Surveys Tutorials, IEEE*, vol. 15, no. 3, pp. 1389–1406, Third 2013.

[19] I. Bagci, S. Raza, T. Chung, U. Roedig, and T. Voigt, "Combined secure storage and communication for the internet of things," in *2013 IEEE International Conference on Sensing, Communications and Networking, SECON 2013*, New Orleans, LA, United States, June 2013, pp. 523–631.

[20] D. Boswarthick, O. Elloumi, and O. Hersent, *M2M Communications: A Systems Approach*, 1st ed. Wiley Publishing, 2012.

[21] D. Conzon, T. Bolognesi, P. Brizzi, A. Lotito, R. Tomasi, and M. Spirito, "The virtus middleware: An xmpp based architecture for secure IoT communications," in *2012 21st International Conference on Computer Communications and Networks, ICCCN 2012*, Munich, Germany, July 2012, pp. 1–6.

[22] A. Gòmez-Goiri, P. Orduna, J. Diego, and D. L. de Ipina, "Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications," *Computers in Human Behavior*, vol. 30, pp. 460–467, January 2014.

[23] C. H. Liu, B. Yang, and T. Liu, "Efficient naming, addressing and profile services in internet-of-things sensory environments," *Ad Hoc Networks*, vol. 18, no. 0, pp. 85–101, 2013.

[24] "European FP7 IoT@Work project," http://iot-at-work.eu.

[25] "iCORE project," http://www.iot-icore.eu.

[26] "IOT-EST project," http://ict-iotest.eu/iotest/.

[27] "EBBITS project," http://www.ebbits-project.eu/.

[28] "Usable trust in the internet of things," http://www.utrustit.eu/.

[29] "BUTLER project," http://www.iot-butler.eu.

[30] B. Guo, D. Zhang, Z. Wang, Z. Yu, and X. Zhou, "Opportunistic iot: Exploring the harmonious interaction between human and the internet of things," *J. Netw. Comput. Appl.*, vol. 36, no. 6, pp. 1531–1539, Nov. 2013.

[31] A. Metzger, C.-H. Chi, Y. Engel, and A. Marconi, "Research challenges on online service quality prediction for proactive adaptation," in *Software Services and Systems Research - Results and Challenges (S-Cube), 2012 Workshop on European*, June 2012, pp. 51–57.

[32] F. Li, S. Nastic, and S. Dustdar, "Data quality observation in pervasive environments," in *Proceedings of the 2012 IEEE 15th International Conference on Computational Science and Engineering*. IEEE Computer Society, 2012, pp. 602–609.

[33] Z. Wu and L. Wang, "An innovative simulation environment for cross-domain policy enforcement," *Simulation Modelling Practice and Theory*, vol. 19, no. 7, pp. 1558–1583, August 2011.

[34] Y. Elrakaiby, F. Cuppens, and N.Cuppens-Boulahia, "Formal enforcement and management of obligation policies," *Data & Knowledge Engineering*, vol. 71, no. 1, pp. 127–147, January 2012.

[35] D. Ferraiolo and V. A. ans S. Gavrila, "The policy machine: A novel architecture and framework for access control policy specification and enforcement," *Journal of Systems Architecture*, vol. 57, no. 4, pp. 412–424, April 2011.

[36] R. Sandhu, E. Coyne, H. Feinstein, and C. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, February 1996.

[37] D. Brewer and M. Nash, "The chinese wall security policy," in *Proceedings., 1989 IEEE Symposium on Security and Privacy*, Oakland, CA, May 1989, pp. 206–214.

[38] M. Bishop, *Computer Security: Art and Science*. Addison Wesley, 2003.

[39] J. Rao, A. Sardinha, and N. Sadeh, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40 – 56, 2009.

[40] ——, "A meta-control architecture for orchestrating policy enforcement across heterogeneous information sources," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 7, no. 1, pp. 40–56, January 2009.

[41] M. Langar, M. Mejri, and K. Adi, "Formal enforcement of security policies on concurrent systems," *Journal of Symbolic Computation*, vol. 46, no. 9, pp. 997–1016, Semptember 2011.

[42] J. Baeten, "A brief history of process algebra," *Theoret. Comput. Sci.*, vol. 335, no. 2-3, pp. 131–146, December 2005.

[43] R. Macfarlane, W. Buchanan, E. Ekonomou, O. Uthmani, L. Fan, and O. Lo, "Formal security policy implementations in network firewalls," *Computers & Security*, vol. 31, no. 2, pp. 253–270, March 2012.

[44] J. A. Pavlich-Mariscal, S. A. Demurjian, and L. D. Michel, "A framework for security assurance of access control enforcement code," *Computers & Security*, vol. 29, no. 7, pp. 770 – 784, 2010.

[45] M. Dell'Amico, M. S. I. G. Serme, A. S. de Oliveira, and Y. Roudier, "Hipolds: A hierarchical security policy language for distributed systems," *Information Security Technical Report*, vol. 17, no. 3, pp. 81–92, February 2013.

[46] J. Singh, J. Bacon, and D. Eyers, "Policy enforcement within emerging distributed, event-based systems," in *DEBS 2014 - Proceedings of the 8th ACM International Conference on Distributed Event-Based Systems*, 2014, pp. 246–255.

[47] "Ibm and eurotech, "mqtt v3.1 protocol specification"," http://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html.

[48] "Mosquitto, "an open source mqtt v3.1/v3.1.1 broker"," http://mosquitto.org.

[49] "Node.JS," http://nodejs.org/.

[50] "MongoDB," http://www.mongodb.org/.

[51] N. Ulltveit-Moe and V. Oleshchuk, "Decision-cache based XACML authorisation and anonymisation for XML documents," *Computer Standards & Interfaces*, vol. 34, no. 6, pp. 527 – 534, 2012.

[52] V. Goyal, O. Pandey, A. Sahai, and B. Waters, "Attribute-based encryption for fine-grained access control of encrypted data," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, 2006, pp. 89–98.

[53] R. S. Sandhu, "Role-based access control," vol. 46, pp. 237–286, 1998.

Sabrina Sicari Sabrina Sicari is Assistant Professor at Universita' degli Studi dell'Insubria (Italy). She received her master degree in Electronical Engineering in 2002 and her Ph.D. in Computer and Telecommunications Engineering in 2006 from Universita' degli Studi di Catania (Italy). From September 2004 to March 2006 she has been a research scholar at Politecnico di Milano. Since May 2006 she works at Universita' degli Studi dell'Insubria in the software engineering group. Her research interests are on wireless sensor networks (WSN), risk assessment methodology and privacy models. She is a member of the Editorial Board of Computer Network (Elsevier). She is the general co-chair of S-Cube'09, a steering committee member of S-Cube'10, S-Cube'11, S-Cube'13 and S-Cube'14, guest editor for the ACM Monet Special Issue, named "Sensor, System and Software" and Ad Hoc Special Issue on Security, Privacy and Trust Management in Internet of Things era (SePriT), TPC member and reviewer for many journals and conferences.

Alessandra Rizzardi Alessandra Rizzardi received BS and MS degree in Computer Sciences with 110/110 cum laude at University of Insubria (Italy) in 2011 and 2013 respectively. Since 2011 for her MS thesis, she began working in the research group of Prof. Alberto Coen-Porisini and Dr. Sabrina Sicari. From November 2013 she is a Ph.D. student at the University of Insubria under the guidance of Dr. Sabrina Sicari. Her research activity is focused on issues related to wireless sensor networks and internet of things, in particular on security and privacy issues in IoT.

Daniele Miorandi Daniele Miorandi is Executive VP R&D at U-Hopper. He received a PhD in Communications Engineering from Univ. of Padova, Italy, in 2005. His current research interests include modelling and performance analysis of large-scale networked systems, ICT platforms for socio-technical systems and distributed optimisation for smart grids. Dr. Miorandi has co-authored more than 120 papers in internationally refereed journals and conferences. He serves on the Steering Committee of various international events, for some of which he was a co-founder (Autonomics and ValueTools). He also serves on the TPC of leading conferences in the networking and computing fields. He is a member of ACM, ISOC and ICST.

Cinzia Cappiello Cinzia Cappiello is Assistant Professor in computer engineering at the Politecnico di Milano (Italy) from which she holds a Ph.D. in Information Technology (2005). Her research interests regard data and information quality aspects in service-based and Web applications, Web services, sensor data management, and Green IT. On such topics, she published papers in international journals and conferences. Cinzia is Associate Editor of the ACM Journal of Data and Information Quality. She has been co-chair of the workshops "Quality in Databases" in conjuction with VLDB 2010, "Data and Information Quality" in conjuction with CAiSE 2005, "Quality in Web Engineering" in conjuction with ICWE 2010-2013, and of the tracks "Information Quality Management in Innovative IS" of MCIS 2012 and "Data and Information quality" of ECIS 2008.

Alberto Coen-Porisini Alberto Coen Porisini received his Dr. Eng. degree and Ph.D in Computer Engineering from Politecnico di Milano (Italy) in 1987 and 1992, respectively. He is Professor of Software Engineering at Universita' degli Studi dell'Insubria (Italy) since 2001, Dean of the the School of Science from 2006 and Dean of the Universita' degli Studi dell'Insubria since 2012. Prior to that he was Associated Professor at Universita' degli Studi di Lecce (1998-2001), Assistant Professor at Politecnico di Milano (1993-2001) and Visiting Researcher with the Computer Security Group at University of California, Santa Barbara (1992-1993). His main research interests are in the field of specification and design of real-time systems, privacy models and wireless sensor networks.