Managing Complexity of Data Models and Performance in Broker-Based Internet/Web of Things Architectures

Pierfrancesco Bellini, Luciano Alessandro Ipsaro Palesi, Alberto Giovannoni, Paolo Nesi
University of Florence, Distributed Systems and Internet Technology, DISIT Lab
https://www.disit.org , Https://www.snap4city.org, corresponding: paolo.nesi@unifi.it

Abstract **The Internet of Things (IoT) is becoming pervasive and with each new installation of IoT platforms new and legacy brokers have to be exploited. New internal brokers are those under the control of the platform, while legacy external brokers are those in place managed by third parties. The solution proposed addressed problems of (a) interoperability to reduce set up time to cope with unknown data structures (devices, entities) distributed via brokers; (b) performance by dimensioning both front-end and back-end processes to reach high rates in a broker-based platform, while preserving full capability features of the data warehouse. Interoperability aspects have been addressed by introducing our concepts and a reasoner into an IoT Directory tool to manage Internal and External brokers, automate device discovery and registration from both standard and customized data models. Despite the managed complexity, a broker-based solution turned out to provide high performance. To this end, a specific assessment and architecture tuning have been performed and reported in the paper to give evidence and validation. The proposed integrated IoT Directory has been developed in the context of the Herit-Data Project, and it is currently used in the whole Snap4City network of 18 tenants and billions of data. Snap4City is an open-source IoT platform for Smart Cities and Industry 4.0, which is an official FIWARE platform and solution, EOSC service and libs of Node-RED.**

Keywords— IoT (Internet of Things), Automated IoT Device Registration, Internal and External IoT Brokers, Smart Data model, Snap4City, FIWARE.

I. INTRODUCTION

The Internet of Things (IoT) defined a paradigm for the computation and communication among things, which is becoming every day more and more pervasive and adopted in many different domains [1], [2]. It is partially due to the worldwide intense deployment campaign about Low-Power Wide Area Network technologies [3], as well as to many approaches and protocols for communications among devices (e.g., Message Queue Telemetry Transport or MQTT, Next Generation Service Interfaces or NGSI, Advanced Message Queuing Protocol or AMQP, Constrained Application Protocol or COAP). The approach is also covering the cloud and fog infrastructures [4], [5]. Thus, IoT network infrastructures are becoming every day more complex to be managed due to the networks' structure and existence of several protocols, formats, and concepts [6], [1]. Hence, the complexity is growing in terms of data management, not only for huge amount of data but also for interoperability and abstraction levels needed to data managing. Relevant aspects to be considered are security and privacy [7] on specific data models and entities. To increase the complexity, there is a range of different owners and managers who may control different parts of such IoT networks [8].

In this context, the concept of **Gateway** is relevant for any segments of IoT networks connecting, as well as for the Web of Things, WoT [9]. Gateways may be integrated with one or more **Brokers** to send/receive data to/from Devices/entities. The Gateways and Brokers are typically compliant with a single protocol and may be managed by third parties with respect to data management platform. In some cases, they are provided as public services for several interested customers in the same area (for example: Proximus for LoraWAN services [10]). A Gateway may abstract from the IoT Broker level managing multiple brokers for multiple organizations/tenants (which can be regarded as customers of Gateway services to manage several **Devices**), via some API and/or Web user interface. Typical IoT Brokers can only manage one organization, and thus are single tenant, meaning they broker messages using topic/entity concepts (which can be regarded as the key for subscription on that specific device/entity) without any internal partition of services, but as a sort of family of devices and subscriptions. Some IoT Brokers can be multi-tenant, such as the FIWARE **Orion Broker** [11], [12], which provides support for partitioning the served devices/entities/topics in groups, and each of them may have a dedicated service/path for a specific scope (or a specific customer). Furthermore, devices of different tenants could exist physically in different places (even having identical identifiers, IDs), and the subscription to the broker's tenant may imply receiving all messages/services in the partition in push. That is feasible only if the subscriber knows the service/path identifier and, in the event of access control, the subscriber has the grant to access the broker's services. Different IoT Devices connected to the same broker adopt the same protocol, may use different data structures/models and have the same semantic information.

According to the above description, some problems have to be managed as described in this paper. Therefore, before describing them in detail, a short overview of the main needs is provided.

In this context, an **IoT/WoT Platform** should abstract and manage all the entities/devices in the IoT Network, allowing to exploit them regardless of their position (connection with gateway/brokers), owner, protocol, format, etc. **Platforms** need to manage multiple IoT Networks and Brokers: some Brokers can be managed by third parties, e.g., the **External Brokers;** while the ones directly managed by the platform are called **Internal Brokers.** In realistic scenarios, third-party brokers are not setup and managed in terms of

Device/Entity registration, subscription, data storage, search, etc., by the platform. As to **External Brokers**, entities/devices are registered on the broker, without providing notification to the connected platforms. On the contrary, a Platform should be able to recognize and manage device messages exchanged with any kinds of broker, any kinds of Device structure (which can be called the **Device Model**, for example the FIWARE Smart Data Models, SDM [12]) in order to register, process and store messages. On the contrary, whenever a message arrives from an unknown device (which can partially provide pieces of information into its body, typically not the metadata, since most devices minimize data transmission), the Platform is not able to register the device, nor to correct the message link to former devices. On such grounds, a Platform cannot be totally agnostic about its data structure/model, neither can ignore the identifier (topic) of the Devices of its external/internal broker. In addition, most Platforms provide support for data storage, thus a data model should be known to perform in deep indexing and to manage data messages as time series. Any effective exploitation and connection of External Brokers is strongly relevant, when a Platform must be connected to another one for receiving new data in real time in push from brokers, and also when it comes to data migration. For example, when a legacy platform has to provide data to an upper-level data aggregation Platform. Both are interoperability aspects.

A. Related Work

Most IoT Platforms have some capabilities for interoperability and integration with legacy solutions. In most cases they do not provide support for integrating External Brokers and tend to push their customers to set up end-to-end solutions with the default internal brokers. For example, AWS IoT by Amazon (AWS) [13], and Siemens MindSphere [14] make the use of their internal brokers' structure transparent. Solutions like MS Azure IoT (MS Azure) [15], and IBM Watson [16] are more flexible in accepting multiple protocols and providing more info on brokers. MS Azure does not provide support to cluster devices/objects, in other words, they support only one organization per broker. Nevertheless, almost every platform allows interoperability by connecting to other IoT Brokers and networks by means of REST Calls API, where the platform calls any external broker APIs working in pull. This implies to brake event driven (push) chain of the IoT message exchange paradigm (publish subscribe).

Organizations such as OMA Spec Works (Open Mobile Alliance), OASC (Open and Agile Smart City) have proposed standardization of communication protocols in favour of interoperability among vendors [17], [18]. The Sensor Web Enablement (SWE), of the Open Geospatial Consortium (OGC) has provided specs for: Observation & Measurement (O&M), Sensor Model Language (SensorML) and Sensor Observation Service (SOS) [19]. The O&M and SensorML provide standard models for measures and sensors respectively. The Semantic Sensor Network (SSN) ontology/vocabulary of W3C provides a standard for modelling sensor devices in an ontology [20]. IERC (Internet of Things European Research Cluster) is working on both pre-standardisation activities of the EC and standardisation roadmap [21]. IoT interoperability is regarded as a very complex aspect to be fully addressed [22], [23]. In [22], Blackstock et al., proposed a solution based on the concept of IoT Hub HyperCAT addressing the problem of device model by creating a general catalogue and metadata for describing the IoT, thus recognizing data coming from the IoT Network. The hub was connected to CKAN to exploit the harvesting capabilities of the CKAN plugin. It was based on WoTKit [24] (Web of Things, WoT), assuming the possibility of accessing to the data models. In [23], the authors proposed a gateway solution as interoperability layer to map different protocols XMPP (Extensible Messaging and Presence Protocol), COAP and MQTT. Desai et al., in [23], implemented OGC schemas before annotating the sensor data with SSN, permitting descriptions/specifications for services by using a semantic SOS. The sensor data obtained from multiple channels are annotated with standard ontologies enabling service level interoperability, by classifying them in the ontology according to semantic similarity.

Chun et al., in [25], proposed an IoT directory with semantic support for discovery and integration with IoT devices without addressing the problem of interoperability. They propose a semantic IoT model for metadata based on static and dynamic properties. Static properties are the metadata that do not change over time, while dynamics are the ones changing, with the device communicating their values over time by sending messages to the broker.

Hao and Schulzrinne, in [26], proposed GOLDIE, a GlObaL Directory for the IoT meant for device indexing and not for interoperability. They have collected a list of requirements for an efficient updating working flow when there are constraints regarding visibility and geographic permissions.

Zyrianoff, Heideker et al., in [9], have analysed the differences between WoT and FIWARE, both are interoperability oriented. The main approach for WoT is based on direct usage of data coming from a set of data channels including IoT protocols. The integration with the IoT world has been proposed by providing a connection with Orion Broker via an adapter. The authors have observed that the insertion of the Broker was a limitation for interoperability, while a certain advantage is provided by the fact that it is an off-the-shelf solution. Its downside lies on the direct usage of WoT requiring a lot of coding applications from the ground up.

Jacoby & Usländer [27] pointed out how it is possible to integrate WoT concepts with IoT FIWARE to cope with Digital Twins. In this context, interoperability aspects can be better addressed by using NGSI-LD, which provides a linked data approach to the model metadata. To this end, a comparison of DTDL (Digital Twin Definition Language), NGSI-LD and WoT is presented. *Also in this case, interoperability at External Brokers level is not addressed.*

Conde et al., in [28], have explained how NGSI (Next Generation Service Interfaces) standard by ETSI (European Telecommunications Standardization Institute) and FIWARE [29] are creating support for the Digital Twin (DT) beginning with Orion Broker concept. In this case, interoperability is performed by writing a number of

adapters (for adapting LWM2M over CoaP, JSON, or UltraLight over HTTP/MQTT, OPC-UA, Sigfox, or LoRaWAN) via the so called IoT Agents which avoid coping with different standards into the core part of the platform.
Recently, the Data Space concept has been introduced and it can be regarded as a generalization of the IoT Device concept [30], [31]. In the approach, a clear distinction from IDS (International Data Space) Metadata and messages is made, thus giving support for IDS Metadata Brokers. The main aim, in this case, is to provide a semantic query support among thousands of different models by performing some mapping on XML and JSON schemas. Data Space approaches are pushing a change from solutions designed as data-warehouse and data-lake into broker-based solutions [32].

B. Paper Scope and organization

In most cases, the usage of broker-based architecture preserves a data driven approach, while most data warehouse and data lake solutions are massively based on the presence of ETL/ELT (Extract Transform/Load Load/Transform) tools, which are mainly pull-based rather than push-based (event driven). *The trend of increasing data models and their corresponding back-office interoperability is making the interoperability with External Brokers more relevant, and it may be supported by evidence that broker-based architecture can sustain relevant data ingestion and access rates in highly interoperable solutions.* A first step to solve the above-described problem is to rely on formal defined Data Model. This approach is followed by FIWARE with SDM, by Snap4City [33] with its IoT Device Model and by the strong push on Data Spaces mentioned above. On this fact, there is an architectural convergence to cope with WoT, IoT and DT approaches.

In this paper, Snap4City/Industry architecture is discussed to highlight how the above-mentioned problem has been solved, allowing the solution to group the aspects of WoT, IoT and DT, with a special case dealing with interoperability of brokers for data entity automated registration and ingestion. Such a solution has been developed and tested into Snap4City open source IoT platform [33] for Smart Cities and Industry 4.0, the official FIWARE platform, platform on EOSC, and a set of libraries on Node-RED [7], [34], and it is at present in operational use.

The proposed solution is based on: (a) leverage interoperability reducing set up time to efficiently detect and learn how to process unknown data structures (devices, entities) distributed via brokers; (b) provision of data driven high rates in a broker-based platform, thus preserving full capability features of the data warehouse. To this end, an extension of the Snap4City Directory concept and tool has been created. The Directory is the main drive for interoperability in an efficient manner, and a number of other platform components serving the Directory are involved in obtaining the required performance to satisfy point (b). The solution supports: (i) Internal and External brokers, (ii) automated registration of devices/entities managed into External Brokers' single- or multi-tenant services, (iii) automated registration by harvesting and reasoning of data models/entities compliant with standard models such as FIWARE SDM, and any custom Data Model in Snap4City IoT Device Model providing a formal semantic definition of device attributes, (iv) fast data ingestion for ingesting / migrating historical data from legacy platforms and services to a new established uplevel platform, (v) sustained data usage from query demand and for data driven show changes in real time. As to validation, the platform has been assessed in terms of performance of the: IoT/Entity Directory device recognition and registration, brokering data ingestion, and data access processes. The research presented in this paper has been developed in the context of the Herit-Data Interreg Project of EC to use big data to better manage touristic flows in natural and cultural heritage sites; results have been validated in the wide condition of the whole Snap4City network of 18 tenants, and billions of data.

The paper is organized as follows. Section II presents major requirements for platform interoperability and comparison with a number of other well-known platforms. Section III shows the overview of Snap4City architecture focusing on the management of internal/external brokers with the aim of harvesting external unknown brokers, device discovery, automating device registration, by performing the recognition and management of formal data models and their attributes. In Section IV, the automated harvester of data model is presented with its formal processing language. Section V provides details regarding the validation performed on the solution, meant to assess the maximum performance which can be obtained in harvesting/registration, data ingestion and data access in the platform and how these aspects are connected one each another. In Section VI, conclusions are drawn.

II. Requirements analysis

In this section, the focus is set on the main requirements a Broker based Platform for data and network management should satisfy. According to the above-presented related work, the broker-based approach for data gathering can be used for WoT, DT and Data Spaces. Requirements have been identified in the context of workshops, interviews, and analysis for the development and exploitation of the Snap4City/Industry platform covering smart city, Industry, Energy, and other domains. The following requirement analysis for a IoT Platform is presented in logical order. In **Table I**, the comparison of the main platforms is considered in terms of functional requirements. The comparison in terms of non-functional requirements as performance is almost impossible to carry out due to the impossibility of installing the solution on the same hardware. As far as we know, only Snap4City can be installed as a full platform in **single VM** (Virtual Machine) preserving scalability via a docker based approach and architecture. Large scale Snap4City installations are already in place, the largest one is [33], while a number of other installations are already set and some of them are listed on https://www.snap4city.org/661 .
On other aspects, surveys about IoT Platforms are provided in [35], [36]. Among the compared platforms, there is also Snap4City, which is presented in more details in this paper.

Table 1. Comparison of Platforms: Y indicates a satisfied requirement, N the Req. is not satisfied, and (y) partial coverage.

Req.	Snap4 City	Azure IoT	Aws IOT	IBM Watson	Mind sphere
R1	Y	N	(y)	(y)	(y)
R2	Y	N	(y)	N	(y)
R3	Y	N	N	(y)	N
R4	Y	Y	Y	Y	Y
R5	Y	Y	Y	Y	N
R6	Y	N	(y)	N	(y)
R7	Y	N	N	N	N
R8	Y	Y	(y)	N	N
R9	Y	N	N	N	N
R10	Y	(y)	(y)	(y)	(y)
R11	Y	(y)	Y	Y	Y

A **Platform** should provide support to:

R1. **Manage different kinds of Brokers, Devices and Edge Devices**. They could be based on different protocols, formats, and modalities to establish connections with the Platform. Almost all platforms support MQTT and HTTP, while Azure IoT supports MQTT and AMQP brokers. Most platforms provide specific components for different protocols, for instance: Amazon MQ supports Broker with AMQP, MQTT, OpenWire, and STOMP, protocols.

R2. **Connect External and Internal Brokers.** Internal Brokers should be deployed, registered and managed by the Platform, while External Brokers would be only registered to use them, since they are managed by third parties. Brokers could be multiservice, for example, the Orion Broker with NGSI V2/LD protocol. In the Platforms under comparison, brokers are product core of stakeholders' offers; this is reason whyR2 is partially satisfied, offering the possibility of adding some other brokers.

R3. **Register, manage and use messages conformant to any Data Model with any data type**. Providing, receiving, managing, storing, and retrieving messages for any Device (or Data Model) with its attributes and data types, and related access control. It is difficult to manage the huge variety of data kinds. For example, GPS coordinates can be defined by different approaches: a couple of variables, a GeoJSON, and a vector. Similar problems may occur with dates (since they can be formalized in several different formats) or with relationships among entities (e.g., URI, URL, URN, IDs). A Data Model should provide a formal model format for IoT Device messages with formalized variables/attributes with data types, units, etc. As to the Platforms under analysis, messages from IoT Devices are freely shaped, to assure data flexibility. For example, IBM Watson uses formats such as JSON or XML, without supporting FIWARE SDMs [37].

R4. **Verify if Data Messages are correct with respect to the defined data model.** The platform should be able to verify if the messages received from Devices are correct in terms of data model including verification of attribute conformance before accepting them. Please note: this requirement implies that every Device should be formally registered in the platform before accepting their data, and they have to pass a verification phase at run time.

R5. **Semantic Interoperability** is fundamental to achieve coherence among different Device data models (e.g., provided by different builders, addressing same concepts, and information on attributes). In most cases, semantic interoperability means a more relaxed constrain of classifying data model/devices according to their structure. A Platform should be able to recognize/classify/retrieve information/attributes and behave accordingly to the semantic data model including data types and units of measure. For example, an application should prevent from any misunderstanding of the unit of measure assigned to attributes of different messages/devices which have the same name and different units. For example, two builders of air conditioners could accept setpoints like temperature with different units: one in Celsius and the other one in Fahrenheit.

R6. **Support automatic cloud deployment of Internal Brokers**, on which the Platform performs the registration of IoT Devices. The result is a simplified experience for users to populate the network.

R7. **Register External Brokers.** The platform must support the registration of IoT External Brokers. Brokers can be single- or multi-tenant. This means that the Platform should be somehow able to **automatically register** the IoT Devices/Entities of the External Broker into the Platform. Therefore, the possibility of recovering the device data model managed by the External Broker is the first step to perform their registration. In the case of External Brokers, the endpoint URL and the service and/or service path specifications would be needed to subscribe. Not one of the considered commercial platforms provides a solution to register External Brokers and thus allowing automated registration of their devices and retrieval of their models. In this sense, they do not provide interoperability to integrate other IoT networks in place.

R8. **Discover Devices on Brokers.** The platform must be able to harvest Devices from their IoT Brokers' protocols. This is needed for any device automated registration and thus for semantic **indexing and retrieval**, on the basis of their nature, position, attributes (value types and units), etc. In other words, it should be possible to discover/search (subscribe, get, send data) to/from Devices regardless from their position/connection in the IoT Network.

R9. **Semantic identification and match.** The platform should automatically recognize the device model and semantic information of attributes. When the discovery is done, messages from devices should be automatically added, and the mismatch problems solved. In the case of mismatch, the platform must adapt the ingestion process. The above-mentioned Platforms do not provide this feature being oriented to manage what they know as in literature.

R10. **Easy management to list and test Brokers, and Devices** and query them for example via a graphic user interface. As to each IoT Device, it should be possible to perform testing activities.

R11. Manage Device Model and Device Data Type ownership and access grants. This allows the assignment/change of the ownership and the creation of access grants to entities (IoT Brokers, Devices, and Data Models). In delegation management, it has to be possible to grant, list and revoke grants. According to GDPR (European Union General Data Protection Regulation 2016/679) [38], any entity has to start as private of the owner. The delegation should be possible for organizations, groups of users, and single users and it can deal with different types. In particular, it can be about:

a. **Messages, data messages** - delegated users can read and write or only read messages of a certain IoT Device;
b. **Device Models, data models** – delegated users can modify or only read model structure;
c. **Devices, data entities** – delegated users can modify or just view device structure.

A. IoT Security Aspects

The main issue of IoT security is to provide a full stack security (end-to-end secure connection based on TLS, Transport Layer Security) ranging from IoT devices, IoT edge on premises, IoT applications on the cloud and on premises, data analytics, and dashboarding. This implies to have solutions for authentication and authorisation, secure communications, in human-to-machine and machine-to-machine communications, secure communications, detecting and monitoring intrusions, controlling vulnerability [39]. This also implies the adoption of penetration tests verifying the robustness of the solution with respect to a large number of potential vulnerability aspects [7]. On the other hand, traditional encryption mechanisms should be substituted with post-quantum cryptographic protocols which are efficient and resistant to attack by quantum computers [40], [41], [39]. The security has also to preserve privacy according to GDPR [38]. To this end the IoT platform have to pass vulnerability and penetration tests [7]. The main issues related to security on IoT are discussed in [7] for Snap4City platform in comparison with major solutions. For the functional topic discussed in the paper, the presence of authentication and authorisation mechanisms, and those related to the channel protection are not relevant since they are addressed at the moment in which the connection is established.

III. Architecture Overview for Interoperability

As reported in **Figure 1**, the **Directory** interacts with **Internal Brokers** to perform registration of devices/data flow (represented as D1, D2… Dn, sensors, actuators and data flow channels). It performs the semantic registration of device (data entities) into the **Knowledge Base**, KB (which is a semantic database RDF store) where all the entities and their relationships are modelled. The interoperable composition of data entities is guaranteed by the adoption of Km4City Ontology [7], [42] that creates a uniform layer abstracting from physical details and mechanisms needed to access them through different Brokers and usage of several data models and their validation, as well as semantic interoperability and matching. In the event of data lack, the KB provides knowledge to complete information on devices with the semantic part, as we can see in the next section. In most Platforms, storage (including time series) is called Data Shadow and it allows to create some historical data of the Devices/Entities. In Snap4City, data storage feeding is performed by Apache Ni-Fi Cluster; IoT Directory automatically performs the subscription of Ni-Fi to the topics of Internal and External Brokers. Ni-Fi is a scalable low latency tool to handle a huge volume of data coming from several devices/brokers to save them into the storage, which in this case is an Open Search (i.e., AWS Elastic Search fork).

Noteworthy is the fact that in the architecture also actions on the fields can be performed by sending Messages to IoT Devices via Brokers. In Snap4City, these data flow/messages can be produced by processes such as: **IoT App** (node-RED), Dashboards, and data analytics processes (in Python, Rstudio, etc.), etc.; *they are not described in* ***Figure 1****, but will be introduced later in the paper*. Moreover, as far as the stored data consumption is concerned, both KB and OpenSearch are made available to API (Smart City API), so as to provide collected data to other applications such as: Dashboards, IoT Apps, Mobile Apps, Dashboard Synoptics, Data Analytics, external services, etc. (see **Figure 2**).

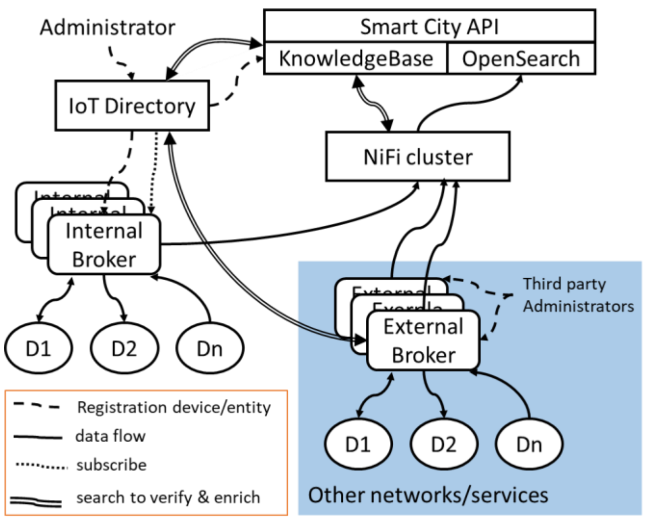

Figure 1 – Overview of the Architecture for data ingestion.

In order to enforce the above-described *interoperability requirements* into the Platform, we have designed and developed the IoT Directory concept and tool. It satisfies the above-described requirements with the (i) possibility of detecting and managing any Data Model, FIWARE SDMs, IoT Device Models, (ii) registering and accepting data from IoT External Brokers (Orion Broker), (iii) discovery IoT Devices from IoT External Brokers, (iv) managing multi-tenancy brokers and multiple brokers, (v) managing multiple organizations and tenants, (vi) providing GDPR compliance, (vii) providing a small footprint, permitting to start from a single VM all in, and to scale up by adding components, etc. (due to lack of space, in this paper we have to focus on interoperability and impact on performance).

As to **External Brokers**, device registration is performed on broker by third parties and thus a different approach has to be taken to perform a device registration on IoT Directory and KB, as described hereafter. With no device registration on KB, new messages might arrive on storage without any semantic control and cause difficulties for their data warehousing. Therefore, a set of tools is involved in the process of new data model identification, device registration and data ingestion; their interconnections have to be optimized to provide high performance, as described hereafter. In subSection III.A, we have focussed on the KB Role and highlighted how the Snap4City model matches the FIWARE SDMs. In subSection III.B, some details about data model formal definition in Snap4City are provided. SubSections III.C, D, E provide details about the broker registration and related processes.

A. The Role of Knowledge Base

As above mentioned, the variety and variability of data increase with the arrival of several different data models and corresponding attributes coming from brokers in all architectures for DT, Data Spaces, WoT, and IoT (as described in the introduction). To guarantee interoperability of different applications and data streams coming from brokers (avoiding data pillars), it is fundamental to harmonize different data types. In other words, every time new data models are introduced into the environment, they have to be analysed and processed to become semantically interoperable with the rest of information and knowledge. This approach has to be adaptive and automated as to registering, accepting ensuring that the semantically searchable new data models can take into account relationships with other entities. The resulting benefit is an improved performance of data ingestion and insertion into the storage (thus avoiding a large number of manual operations carried out by many platforms as described in the introduction).

For these reasons, many data streams and applications have been analysed in order to create a semantic model general and flexible enough to cope with a large range of solutions and domains. The result of this process has been the new version of the Km4City ontology, modelling city entities and their relationships. Km4City presents 7 main areas of macro-classes, i.e.: administration (segmentation of geographic areas), city-structure (buildings, roads, etc.), points of interest (POI, information data), sensor/entities (data flow, IoT, sensors, actuators, time series), temporal (single instant, intervals, etc.), structures (hierarchical description of entities, physical and virtual for modelling building and organizations, industries, etc.). The KB also include a Dictionary to model relationships among data types and units as described hereafter. Km4City is based on a set of vocabularies: DCTERMS: for metadata Dublin Core Metadata Initiative; FOAF: friends of friends; Good Relation: entities relationships; iot-lite: IoT Vocabulary; OTN: Ontology of Transportation Networks; OWL-Time: time reasoning; SAREF Smart Appliances REFerence extension for building devices available at https://saref.etsi.org/saref4bldg/; Schema.org for people and organizations; SSN: Semantic Sensor Network Ontology (see https://www.w3.org/TR/vocab-ssn/; WGS84 Datum of Geo-Objects; GTFS, General Transit Feed Specification, and Transmodel, for public transport infrastructures: lines/rides time schedules, real-time records, paths, etc.;

The Km4City ontology keeps always updated its data model providing an API exposed for the IoT Directory, which is sending out any newly registered devices [34]. The KB storage is based on Virtuoso RDF store, and every information is coded in terms of triples. The resulting model can be queried in SPARQL and is accessible as Linked Open Data, for public information. To get a better understanding of the ontology structure, it is possible to explore the Linked Open Graph (LOG) associated with it. The Smart City API, SCAPI are implemented on top of the SPARQL semantic query interface, and they exploit the Open Search elastic storage as to time series. Additional storage can be provided as well for BIM (Building Information Modelling), 3D Models, and GIS (Geographic Information System) data such as maps, orthomaps, etc. (not described in **Figure 1**).

In this context, the KB role is to provide (i) information to IoT Directory when a new entity/device is detected so as to register it in a correct and harmonized manner with the rest of information, (ii) information to Ni-Fi when a new message arrives from some brokers with minimal information and needs to be semantically enriched with the correct information to be indexed into the Open Search storage, and (iii) support to Smart City API rest call to solve them by spatial, temporal and relational reasoning engine [43]. This approach is a data warehouse broker-based solution.

B. Snap4City models vs FIWARE models

FIWARE is a foundation which promotes open-source ORION Broker in standard NGSI. FIWARE also includes a series of Generic Enablers software modules that perform functions in various IoT-based applications. FIWARE provides mechanisms for modelling and managing data and introduces RESTful NGSI API to interact with Orion Broker. Main elements of NGSI data are context entities: representations of physical or logical objects. Snap4City is an official platform and solution of FIWARE being compliant with NGSI, Orion Broker, etc., and providing several additional open-source tools for setting up full platforms and solutions [33].

According to NGSI V2 standard, each attribute has a name, a value and may provide its own metadata, and among the metadata, it may also define the unit of measure, the *unitCode*. In most SDMs, the *unitCode* is not defined leaving to data producers the choice to adopt some of its own. In the Snap4City model, each attribute has to be defined by the proprieties: Value Name, Value Type, Value Unit and Data Type. In this way, each device model attribute has a precise semantic formalization by name (e.g., V1), Value Type (e.g., Voltage), a specific unit of measurement by the Value Unit (e.g., V, mV, KV, vector of mV values) and the Data Type clarifies the type of the data (e.g., *integer, float, string, json*). Please note that, Type in NGSI and Data Type of Snap4City refer to a different meaning. In particular, the NGSI Type is more generic than the Snap4City Data Type. For example, if the value is a number the NGSI Type can be "*numeric*", while a similar Snap4City Data Type can be "*integer*", "*float*" or "*double*".

The FIWARE NGSI attribute definition is not specific enough to be processed by an inferential engine, due to its lack of semantic details; in fact, the NGSI *unitCode* is user defined and does not provide a unified semantic (does not belong to a common Dictionary), thus it may not be enough for the automated process. In the FIWARE usage of NGSI, the resolution is outsourced at application level. In Snap4City, the resolution is defined in a Dictionary (of KB) to conform any arrival message, to be faster and simpler in data ingestion and processing, while in FIWARE NGSI each message could change the *unitCode*. On such grounds, the Snap4City Platform needs to map the NGSI attributes to a more formal definition at least once, to automatize message ingestion and interoperability. In FIWARE, the same SDM, used in different brokers or applications could lead messages using different *unitCode*, and their definition is left out of the formal SDM formalization, leaving the issue to an application level.

C. Brokers' Registration

The first step to exploit the architecture as reported in **Figure 1** consists in the broker registration. As seen in the introduction, some platforms only provide support for a number of ready to use internal brokers. Snap4City allows the automated deployment of dedicated Orion brokers connecting them as Internal Brokers and it also supports the registration of External Brokers. In the event of External Orion Brokers, a given number of additional capabilities is possible. In the broker registration phase, several parameters are requested such as: endpoint, security, name, External/Internal, single/ multiple tenants, etc. Each broker is associated with a specific user or public, and each user belongs only to a single organization for security and privacy aspects [7]. External Brokers are managed by third parties including their accessibility and usage. Other differences between Internal and External Brokers consist in the management of IoT Devices as explained hereafter.
Once a broker is registered, the IoT Directory automatically performs the data platform subscription (Ni-Fi) to the new broker for all its devices/topics, so that each new message generated by the broker would be directly brokered to data storage. On the other hand, this may not be true for the External Brokers since the IoT Directory/KB does not know all the entities/topics if they are not provided in the External Broker registration phase.
The following subsections are focusing on these aspects.

D. Internal Broker and their Devices/Entities

As to **Internal Brokers,** the IoT Device/Data Entity registration/definition is performed on IoT Directory, or via its API. The registration may start with the exploitation of a Data Model (IoT Device Model), the provision of a device ID and the definition of GPS location, plus other details. In Snap4City, an IoT Device registration can be performed:

- *Manually:* register an IoT Device by using a graphic interface. Register a device based on a specific IoT Device Model or create a totally customised device and refer to a specific broker.
- *In Bulk:* upload a file (Excel Files/tables) with: (i) a list of Devices, defining the IoT Broker, Model and details, (ii) a list of data out of which the platform can derive IoT Device Model and Devices instances (so called Data Table Loader) [44].
- *Via IoT App:* Users can build an IoT App (which are Node-RED processes with Snap4City MicroServices [34]) to process incoming messages or files to transform them in several IoT Device registrations (by Model or Custom) and posting possible messages to those devices (see **Figure 2**). IoT Apps can perform massive registration of IoT Devices/Entities, data adaptation, transformation, load, production, redistribution, business logic of dashboards, etc.

Each registered Device/Entity is also registered on KB, with its information and metadata (static information). KB indexes devices and establishes every explicit relationship (declared in the data with other entities), as well as implicit relationships (with other entities located in the same area, place, city, region, road, GPS position, etc.). These relationships would be exploited by queries when each message arrives from a broker via Ni-Fi in the storage (see **Figure 2**). The correct and complete indexing is fundamental to enable the spatial, relation, and temporal search of IoT data via Smart City API by IoT App Node-RED microservices [34], Data Analytics and Dashboards [45].

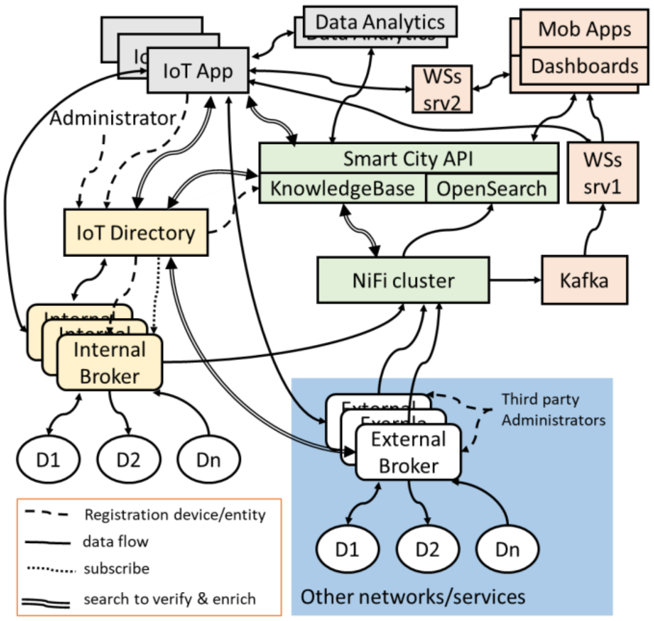

Figure 2: Architecture with details on automated device/model registration and data exploitation, it is a more detailed version of Figure 1, while authentication and authorisation modules are not represented here, see Figure 3 for them.

To make the consultation of registered IoT Devices easier, they are shown in a table where users can manipulate only the ones they have created, regardless of any generation process. In a list of accessible data, users can also see public devices of the same organization, together with devices outsourced to him/her. On the other hand, any general administrator has full visibility of every device belonging to all the organizations.

Figure 2 shows data flows during a platform usage, thus stressing both flows for event-driven and historical data usage (which will be validated in Section V about performance). There are several tool areas which generate and consume data messages and each of them may be present in multiple instances, connecting at the same time and requesting/producing real time data streams. Three tool areas may also consume historical time series data, or other information. Such main tool areas are:

- **External Services** providing data in pull and receiving them in push. Data are acquired via IoT Apps and pushed into the platform via some brokers, data are provided via IoT App as well.
- **IoT Devices/Data Channels** produce data in push and thus are connected to brokers, not only to NGSI brokers but also to MQTT and other kinds of broker. Data Messages are passed from broker(s) to Ni-Fi, thus reaching KB and storage, becoming part of historical data which can be accessed and queried (via SCAPI) from IoT App, Data Analytic and Dashboards. Data arriving in push to Ni-Fi can be also produced in push on Dashboards and IoT Apps via Kafka and WS secure server 1 (webSocket) to reach all the subscribed Dashboards/Apps via webSocket connection which manages multiple connections.
- **IoT Apps** may send a message in push to (i) a Broker to reach a Device, or to (ii) several Dashboards and their widgets via the WSs server 2 (managing multiple connections). A IoT App message can reach an IoT Device to act on it, it can reach the storage to be saved, or it can reach another IoT App to establish communication with and act, it can reach Dashboards to provide data to be represented, etc. If a message is sent by a sensor-actuator (Internal or External), its Broker broadcasts it to Ni-Fi, which spreads it in turn, thus also saving the acted messages and rendering them in real time.
- **Dashboards** may produce messages towards multiple IoT Apps via the WSs server 2, which manages multiple connections. These messages can be regarded as Virtual IoT Devices to act on some sensors/actuators or simulate them [45]. The IoT Apps in turn can forward this piece of information to internal or external brokers and connected devices.
- **IoT Directory** may generate a new message towards an IoT Broker (and may also read the last message sent from broker). The generation of messages from IoT Directory is typically used to check if Broker is alive and working correctly, and if IoT Device messages are accepted.

E. External Brokers and their Devices/Entities

To become easily interoperable with legacy brokers of third-party networks, we have defined a solution and process for the registration of External Brokers and their entities. At the first registration of an External Broker, thousands of devices should be discovered. In fact, Devices registered on a never connected **External Brokers**, are not registered on the IoT Directory and KB and as a consequence, Ni-Fi is not prepared to manage new data messages.

To perform on IoT Directory a manual registration of devices inherited from a legacy External Broker could be very time consuming. Moreover, External Orion Brokers may be multi-tenancy with service paths for tenants.

As a ***first approach*** to cope with this issue, the Snap4City IoT Directory harvests the External Orion Broker to collect a list of devices/entities belonging to the known tenants by using service paths. A periodic Discovery/harvest is needed, because if a new Device is added, the IoT Directory has to identify it for registration, in order to accept messages from it. Thus, a refresh time for periodic harvest is needed. By means of such harvesting process, IoT Directory can recover the information needed for registering and partial indexing devices into KB, since NGSI provided metadata are not complete for a full indexing on KB.

According to a ***faster approach***, we may suppose that the Snap4City Platform knows a set of Data Models (IoT device Models, FIWARE SDM, etc.). Subsequently, the harvesting process may recognize any device model (from a quick analysis of message format, device type and ID). Therefore, if IoT Directory recognizes Data Model, variable value names, data types and *unitCode* of each attribute, it can register them in KB properly and as a consequence, each message arriving on Ni-Fi can be validated and ingested. The risk of mixing variables with different units is very high, for example, adding Volt and KVolt, euro and Meuro, Celsius and Fahrenheit, Joule and BTU, etc. Conversion rates might not be always clear, since the *unitCode* is actually a string custom provided, and not imposed from a formal precise Dictionary.

To this purpose, a precise mapping from each Data Model including its attributes is needed and a simple analysis by similarity does not work for precise indexing and execution on business intelligence tools. Please note that IoT Directory can query Orion Broker to get a device model, while the model itself does not provide details to solve any mismatch of Value Type, Value Unit and Data Type (e.g., Temperature, Celsius, Float). *Therefore, IoT Directory needs to know the Data Model and the mapping of attributes to the Dictionary, in order to enable any registration and fast data ingestion, indexing, storing, etc*.

Therefore, the first release of the Directory harvester has provided a list of non-recognised devices, for which registration was not possible, leaving to the administrator the issue to solve the mismatch by means of a user interface. When thousands of new devices are discovered, the process become unmanageable, and this happens every time a new installation occurs and the registration of External Brokers as legacy FIWARE Orion brokers is mandatory.

In order to solve this problem, an ***automated harvesting approach*** of Devices/Entities on External Brokers has been designed as described in the next section. The registration of devices from External Brokers is one of the most innovative aspects addressed by IoT Directory which is capable of (i) harvesting brokers for device discovery, (ii) resolving semantic gaps on IoT device attributes/variables, (iii) registering devices, thus shortening the data ingestion and interoperability processes, see the following Subsection.

IV. Automated Harvesting of Data Models

In order to automatize the discovery and registration of devices/entities into the Directory (and KB) which are already registered on legacy external brokers, we have created an automated process, which can be scheduled to get updates, since devices on External Brokers are registered by third-parties. The registered devices on External Brokers can be: Case (i) custom made, Case (ii) compliant with some SDM version, and Case (iii) derived from some SDM versions.

Therefore, as a *preliminary step,* IoT Directory harvests periodically the definition repository of the FIWARE SDM from *github*. The collection of SDMs is classified per domain. The SMD harvesting starts by making a local copy of the SMDs collection. Then, each SDM is formally validated against a corresponding *schemaInterpreter,* usually all of them pass the validation, since they have been already validated during the publication phase. The validation procedure is also useful to detect Cases (iii), where an SDM has been customized.

In **Case (i) of full custom devices/models,** data models are totally unknown and so is the definition of their attributes. This means that the platform, before accepting messages, should at least: (a) model a Device with its attributes and then (b) register it. Thus, each new message, according to that device, can be recognized as belonging to that registered device.

In **Cases (ii) and (iii),** some information about models can be recovered from SDM definitions and schemas, while some attributes can be also not so well defined according to some problems listed in **Section III.E**. In fact, due to a mismatch that may occur with different usages of SDM in FIWARE Orion Brokers, it may happen that the same SDM is used with different *unitCode*. To this end, Snap4City Platform allows to define mapping rules on device/entity attributes to assign *{value type, value unit, data type}*, according to contextual conditions composed by the: broker, SDM name, device type, value name (attribute name), etc.

Therefore, **as a final consideration** the main problems during such broker harvesting and acceptance of new messages from devices of External Broker deal with the attribute matching with contextual information regarding both unknown and also already known Entity/device models (SDM, Custom, IoT Device Models of Snap4City, etc.). For this reason, in the phase of External Broker harvesting for each attribute a *semantic query* is performed on KB and Dictionary, so as to verify the presence of a full match in terms of contextual conditions.

The match may be as follows: (A) success: the data model with each attribute is recognized and a mapping is available, thus the new device/entity can be automatically registered, or (B) failure: some model attributes are not recognized; thus, a new mapping rule has to be produced and suggested to users.

The automated production of mapping Rules is based on similarities from unknown device models discovered, pieces of information in KB and Dictionary, while relaxing some conditional constraints such as the broker, the SDM, the organization, etc. To this purpose, the set of active mapping rules is queried. The resulting queries can be browsed and corrected via a visual interface for non-technical experts. Formally, mapping rules R are defined as follows:

R:= IF <condition> THEN <action list>
<condition>: = <c> | <c> AND <condition>
<c> := <variable> <op> <constant>
<variable> := “device name” | “context broker” |
    “device type” | “modelname” | “Value Name” |
    “service” | “servicepath” | “organization” | “version”
<op> := “==” | “!=” | “in”
<constant> := integer | float | string | list | “null”
<action list> := <a> | <a>, <action list>
<a> := <action variable>: <action constant>
<action variable> := “Data Type” | “Value Type” |
    “Value Unit” | “Editable” | <Healthiness value> |
    <Coded Healthiness criteria>
<action constant>:= string

Rules are structured as an if-<condition>-then-<action list>. The <condition> describes the context of rule application in terms of joined constraints on broker, service, model, device type, organization, etc. In fact, a rule can be functional for an organization or broker and not for others. In IoT Directory, it is possible to search, edit, activate and deactivate rules. Please note that in a given context multiple rules firing may be present. In this case, devices fired with multiple rules are proposed to the administrator for his/her decision about rule application. On the contrary, if a unique firing condition is obtained, the rule is applied, all the conformant devices are automatically registered, and their messages are accepted for ingestion purposes (thus shortening the exploitation of External Brokers). The <action list> is a list of assignments to complete and/or solve the mismatch about device/entity attribute definitions. Actions are used at each device registration to enrich the provided data to have full device/entity information, so as to perform a complete registration and semantic indexing. An example of a Rule can be:

IF “context broker” == “Brk45” AND
    “Value Name”==“aPower”
  THEN “Value Type”:“activepower”, “Value Unit”:“KW”,
    “Data Type”:“float”

V. Performance Assessment and Validation

In the above sections, we have demonstrated how Snap4City solution satisfies R1, R2, and R7 in connecting and exploiting data related to different models coming from different kinds of internal and external brokers, controlling message conformance with a given model, and verifying message correctness (R3 and R4). The semantic interoperability, R5, is provided by the services of both KB and Dictionary. The harvesting of devices on brokers and the automated registration of them cover R8 and R9. As to R10 and R11, evidence is reported in [34], while R6 is certified by the fact that Snap4City is an official platform and solution of FIWARE, since the automated deployment of Orion Brokers is one of its mandatory features.

In order to assess any effective strong point of the proposed solution, a specific set of performance experiments has been performed. The *non-functional requirement* regarding the performance has to be verified, since the interoperability may have a relevant impact on those aspects, and in particular on performance for: (i) broker harvesting and device registration, (ii) data message ingestion and data consumption which is (iii) responding to Smart City API providing data to clients, (v) passing data driven data from broker directly into user interface consumption.

A. Harvesting Performance

As discussed above, harvesting External Brokers may take into account one or more rules to recognize attributes and data models, registering and indexing the devices/entities. This will help in strongly shortening the time to connect external brokers, recovering their data and performing manual 1:1 or in bulk registration (if models are known). Moreover, the proposed solution can dynamically add new Devices/entities as soon as they are registered on External Broker, which reduces a lot the gap between using Internal and External Brokers.

The validation of this approach has been performed in a condition of operative workload on Snap4City.org with 20 Internal Brokers and 7 external brokers, for the arrival of about 22.000 new devices (the reference architecture is depicted in **Figure 2**). A regular user on IoT Directory GUI takes about (i) 2.5 minutes to **create a custom device** with 9 attributes, and the platform takes 0.7s to register the device/entity (ii) 2 minutes to **create a device model** with 9 attributes. Once the model is created, we have two cases to perform device registration from the model: (a) manually, the user takes about 0.95s which is an error prone high repetitive operation, (b) automatically, by using an IoT App which takes about 0.623s and then produced manually via visual programming in about 30 min. Technically, the platform core takes 0.3s for each new device registration (saving data on IoT Directory, registering on KB, performing all needed verification to avoid ID duplications, etc.).

As to the registration of devices coming from a multi-tenant External Broker with 22000 devices with 15 models, the related harvesting time has been of an average of 32s, using automatically generated rules. When custom Rules for a specific SDM are applied, the system takes around 1.99min to harvest all of them, controlling rules at each event. After that, 14406 devices matching those rules have been automatically passed, thus becoming ready for data ingestion. The time for device registration is the same as in case (a). Furthermore, when the system harvests on the same External Broker, it can show updates of already registered Devices.

The effective advantage of the proposed harvester consisted in the automated production of Device Models and rules, which passes from 2 minutes to about 0.125s for each new model. The speed up obtained with manual model-based registration with respect to full manual registration has been of about 523,3 times; the speed up obtained by performing the automated registration via IoT App has been of about 686,5 times; and the speed up obtained by performing a fully automated process has been of 800,2 times.

B. Data Ingestion Performance via Broker

According to **Figure 2,** once a device is registered the Internal or External Broker directly sends new messages to Ni-Fi which performs some enrichment. Actually, each broker needs to provide an authentication/authorisation (A&A) filter (see **Figure 3**).

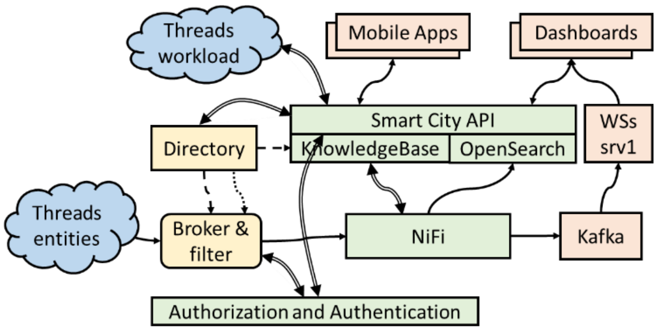

Figure 3: Architecture (a part) for performance assessment.

The Broker & Filter verifies the right to post a data message on the platform for the specific Device/Entity (the authentication is performed once, while the authorisation is performed at each new post/message, while cache from filter to the authorisation permits to increase the ingestion rate). Once the data are passed by filter, the broker pushes them on Ni-Fi, which in turn enriches them on the basis of KB info (performing a query and caching it), to finally post data on OpenSearch and Kafka for real time event-driven expositions on some front-end real time dashboards, IoT App and users who have subscribed to the WebSocket server 1 (WSs srv1 in **Figure 3**).

In order to assess the performances on data ingestion a separate installation based on Snap4City platform has been deployed according to the Micro X model. Snap4City platform can be **installed on premises,** according to a number of models ranging from MicroX based on single VM with a docker based deploy of all the internal processes, up to a DataCityLarge which provides a scalable multi VM solution for big data storage and high throughput of data ingestion. The MicroX VM was resourced with 16 cores of 2.1 Ghz, 32 GByte Ram. The assessment has been performed producing, from a variable number of devices (threads/entities), parallel input streams pushing message data on a single broker/filter.

According to **Figure 4**, the platform sustained a maximum of about 138msg/s (each of which with 10 variables, plus time stamp and GPS) using about 240 threads sending their data at the same time (which are 82800 new single data variables per minute). The result has been about 1.79msg/s per entity, which implies to be capable of ingesting about 12Million of complex msg/day. This kind of capability satisfies a medium size city, which may have about 124K devices sending updates every 15 minute (for traffic, pollutant, light status, etc.). Large cities may need to have multiple brokers, a cluster of Ni-Fi for the ingestion and a cluster of OpenSearch nodes for storage. Moreover, the sustained workload may also depend on the amount of data accesses performed from front end, as addressed in the next subsection.

The result has been obtained with a fine tuning of different tasks in the Ni-Fi and with the installation of an additional cache from Ni-Fi and KB, so as to avoid requesting data to enrich data messages for ingestion at each new insertion (from both internal or external brokers as well). The platform reached its max in ingestion performance when loaded with 240 threads; the CPU workload was at 65% and RAM memory at 11Gbyte, while the transfer rate on writing on HD turned out to be of 22Mbyte/s, which is far from the maximum allowed rate of the platform. From the graph in **Figure 4**, it is evident that the insertion rate has reached a saturation and such saturation was mainly due to the needed activities on A&A; the insertion of a cache did not solve the problem. Without the specific A&A calls the ingestion rate reached the 170msg/s. Moreover, higher performances can be reached with vertical scalability by increasing the number of cores and the amount of memory.

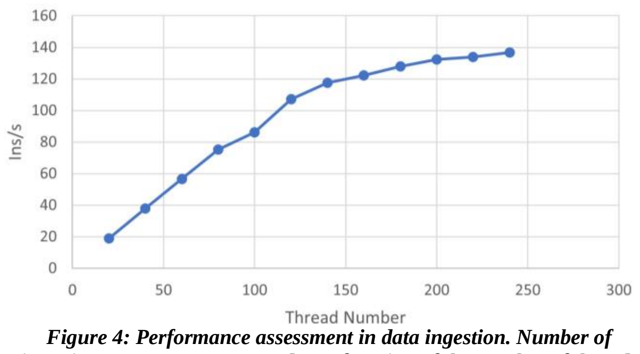

Figure 4: Performance assessment in data ingestion. Number of insertions/messages per second as a function of the number of threads (simultaneous devices sending data).

C. IoT Data Access: Geo & Time Query Performance

Once IoT data are on storage, KB plays the role of spatial and semantic index, while Open Search keeps time series data. For this reason, performances responding to different kinds of queries could be different. Moreover, these activities of data access may influence the data ingestion performance. For example, accessing KB influences costs related to data warehouse in the data enrichment from Ni-Fi, while accessing some series acts on Open Search, which is also fed by data insertion.

In order to perform a realistic performance assessment, for both cases, storage has been loaded with about 300.000 entities/devices with 1 year of time series data, with samples of every 10 minutes and 10 variables per message (this case may represent a medium-large size city: light control, stoplights, traffic flow, pollutant, parking, etc.) (in one year it could accumulate 35Tbyte of data including indexes). The reference architecture for the front-end assessment is reported in **Figure 3,** where the simultaneous data access requests are represented by using a number of threads. This test has been performed without ingestion processes.

The spatial query on smart city API (engaging only KB) has been set to search for a category of entities in a ray of 1Km and collect about 30 results over some hundreds (the execution time does not quite change when resulting entities are in a range of 1-100). In **Figure 5 (a)**, a max of 5700 successful requests per minute has been obtained for spatial queries. The workload has saturated the resources provided for the above described VM MicroX, reaching 90% of CPU clocks.

The space and time series query on smart city API has been set to search for a category of entities in the area and have been limited to 1 result for the whole day, which is about 144 samples. The collection of the whole sequence does not quite change the query cost. This kind of query initially engages KB and then, according to results, it engages Open Search ordered time index. In this case, the saturation of VM resources provided (reaching 92% of CPU clocks) has been reached with 1600 successful requests per minute by using 25 tasks (see curve (b) on **Figure 5).**

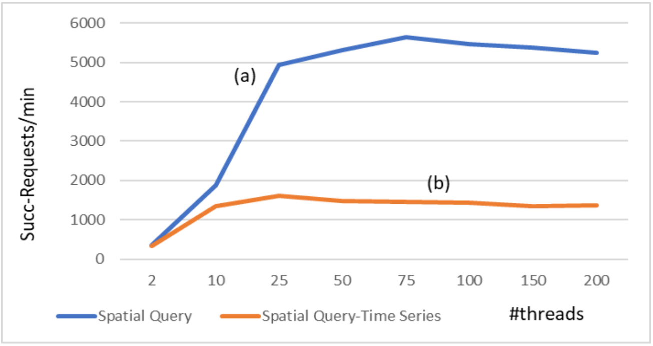

Figure 5: Performance assessment in data access, geo query. Number of API calls successful requests per minute as a function of the number of contemporary threads/requests (cases): (a) spatial queries, (b) spatial and time series queries).

D. Combining Ingestion and Access Workloads

Some performance tests have been carried out to combine different queries with data ingestion processes. Results have shown that when both workloads are reduced to ½ of the maximum, the single VM reached 90% of CPU saturating the resources, and the performance in data ingestion only decreased of 2.3% while the performance in data access effectively reduced to the 50%. This implies that a certain unbalance in scaling and decupling in a different way back-end ingestion processes (Broker and Ni-FI) has to be considered with respect to the from-end (KB and OpenSearch).

E. Assessing performance for end-to-end event driven messages.

When new events coming from devices need to be directly communicated to front-end (end to end secure), according to Figure 3, they need to pass through Ni-Fi (for enrichment, indexing and storage), and through Kafka/WebSocket to manage multiple clients. In order to reduce some workload of this real time channel, only real time messages (on the basis of their timestamps) are also forwarded to Kafka/WebSocket by Ni-Fi, thus avoiding any distribution of already loaded historical data, for example. In this first case, the maximum performances which could be obtained were slower than those for direct ingestion: the registered maximum was 16000 single data values changes (on value and GPS position) per minute.
The messages begun out of Dashboards to act on platform (devices or other entities) can (i) move from a WebSocket server to be distributed to multiple IoT Apps, or (ii) be directly posted on Broker (this would imply to have messages sent in Push into Ni-Fi, storage, etc. full round).

F. Considerations

In general, the increment of performance could be obtained by means of increasing the number of cores (i.e., vertical scalability), while the amount of memory provided was not a limitation. Moreover, the architecture could be also horizontally scaled by adding more brokers, clustering Ni-Fi and Open Search, and/or providing a balanced front end for queries. All these scaling activities impact in different manner on the performance improvements and should be performed on the basis of the actual usage of the platform. For instance, in medium sized city with 300.000 entities the platform should be oriented to provide services to: (i) city users (for instance to 30.000 contemporary users, over a population of 200.000), or (ii) just to be used for decision makers and control room. In case (ii), a simple front-end can satisfy all the requests, while in Case (i) a cluster of Open Search and front-end in balance is needed. In the Snap4City.org platform, a 6 nodes OpenSearch cluster manages time series data and provides them to a front-end cluster providing cached results from API call requests.

When setting up a new IoT Smart City infrastructure, in most cases the historical data have to be ingested by collecting them from former storages, therefore the process cannot be performed in short time. This also means that the operative conditions of the platform should be able to process at the same time: (i) historical data ingestion processes, (ii) real time data ingestion, (iii) data access to provided services and hints (access from mobile, dashboard, data analytics, etc.).

As a limit case, returning to the data ingestion process, a medium-large size city with 300.000 entities/devices with samples every 10 minutes, having data to be recovered from other servers for the last 3 years, may lead to recollect and ingest more than 47 billion of data messages (which may actually become Tbyte on storage space according to the preferred redundances and number variables per message). The ingestion of these historical data via a single broker would take 416 days for manual device registration and when it comes to time series ingestion, about 4000 days at a rate of 12 million msg/day: this makes the process very unfeasible. The problem can be solved by means of an automated registration of devices and an architecture scaling (back-end and storage) with multiple brokers, Ni-Fi and OpenSearch, and in some cases, avoiding performing data ingestion of historical data by filtering them by A&A verifications.

VI. Conclusions

The proliferation of IoT devices, brokers, networks, data models, operators and tenants, makes the harmonization and management of IoT Platform a hard goal. This paper offers an analysis and a comparison among relevant existing platforms, and it points out the basic requirements to achieve such aims. These identified requirements are in most cases not addressed by main platforms which prefer to stay on their own end-to-end solutions with limited interoperability and capacity of exploiting legacy IoT networks in place, in terms of performance. The proposed solution addressed problems of (a) interoperability by reducing set up time to efficiently detect and learn how to process unknown data structures (devices, entities) distributed via brokers; (b) performance by dimensioning the front-end and back-end processes to reach high rates in a broker-based platform, while preserving full capabilities features of data warehouse.

As to interoperability, the main identified and solved problems are those related to a large variety of Data Models coming from non-controllable External Brokers. The issue has been solved by designing and implementing a harvester and reasoner that is capable to automatically recognize/understand and map the new data models/types into those already known by Knowledge Base. This approach, together with the definition of a comprehensive meta model and dictionary, has allowed to speed up the process more than 800 times. The harvesting and comprehension process can be periodically performed to keep the platform updated with any newly defined data models by third party brokers. Furthermore, the process is helped by Km4City ontology and Data Dictionary to recognize the new data types and models according to the semantic domain.

Moreover, any processes of data discovery, registration and ingestion also impact on performance. To this end, the proposed solution has been assessed in terms of performance in harvesting brokers, discovering and registering devices, collecting messages and data access; thus, providing evidence of the maximum performance which can be obtained by each single front-end / back-end component/area and how they are influenced each other in the whole architecture. This study has led us to a number of considerations regarding platform scaling and usage, especially when the former have to be used to harvest and ingest legacy data coming from External Brokers.

Future work can be oriented on enforcing stronger encryption mechanisms which may impact on the protection of data and connections as mentioned in section II.A. An activity in this direction could be to investigate the enforcement of blockchain solutions on specific IoT devices.

The reported study and IoT Directory have been developed in the framework of the Herit-Data Project and it is currently used in the Snap4City infrastructure, made of more than 18 tenants, and billions of data. Snap4City is an open source IoT platform for Smart Cities and Industry 4.0, official FIWARE platform, compliant with the Smart Data Model of FIWARE, EOSC, and lib of Node-RED.

Acknowledgment

The authors would like to thank the MIUR, the University of Florence and all the companies involved for co-funding the Herit-Data project. We express our thanks also to Chiara Camerota and Giuseppe Parrotta for some development and early experiments on the harvesting aspects of the platform. Km4City and Snap4City (https://www.snap4city.org) which are open technologies and research of DISIT Lab. Sii-Mobility is based on and has contributed to the Km4City open solutions.

References

[1] P. Bellini, P. Nesi, G. Pantaleo, IoT-enabled smart cities: A review of concepts, frameworks and key technologies, Applied Sciences. 12 (3) (2022) 1607.

[2] S.Chen, H. Xu, D. Liu, B. Hu, H. Wang, A vision of IoT: Applications, challenges, and opportunities with china perspective, IEEE Internet of Things Journal. 1 (4) (2014) 349–359.

[3] K. Mekki, E. Bajic, F. Chaxel, F. Meyer, Overview of cellular LPWAN technologies for IoT deployment: Sigfox, LoRaWAN, and NB-IoT, in: 2018 Ieee International Conference on Pervasive Computing and Communications Workshops (Percom Workshops), IEEE, 2018: pp. 197–202.

[4] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J.P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture. 98 (2019) 289–330.

[5] B. Cheng, G. Solmaz, F. Cirillo, E. Kovacs, K. Terasawa, A. Kitazawa, FogFlow: Easy programming of IoT services over cloud and edges for smart cities, IEEE Internet of Things Journal. 5 (2) (2017) 696–707.

[6] M. Aboubakar, M. Kellil, P. Roux, A review of IoT network management: Current status and perspectives, Journal of King Saud University-Computer and Information Sciences. 34 (7) (2022) 4163–4176.

[7] C. Badii, P. Bellini, A. Difino, P. Nesi, Smart city IoT platform respecting GDPR privacy and security aspects, IEEE Access. 8 (2020) 23601–23623.

[8] T. Qiu, N. Chen, K. Li, M. Atiquzzaman, W. Zhao, How can heterogeneous internet of things build our future: A survey, IEEE Communications Surveys & Tutorials. 20 (3), (2018) 2011–2027.

[9] I. Zyrianoff, A. Heideker, L. Sciullo, C. Kamienski, M. Di Felice, Interoperability in open IoT platforms: WoT-FIWARE comparison and integration, in: 2021 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2021: pp. 169–174.

[10] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, T. Watteyne, Understanding the limits of LoRaWAN, IEEE Communications Magazine. 55 (9), (2017) 34–40.

[11] S. Valtolina, F. Hachem, B.R. Barricelli, E.G. Belay, S. Bonfitto, M. Mesiti, Facilitating the development of iot applications in smart city platforms, in: End-User Development: 7th International Symposium, IS-EUD 2019, Hatfield, UK, July 10–12, 2019, Proceedings 7, Springer, 2019: pp. 83–99.

[12] F. Cirillo, G. Solmaz, E.L. Berz, M. Bauer, B. Cheng, E. Kovacs, A standard-based open source IoT platform: FIWARE, IEEE Internet of Things Magazine. 2 (3), (2019) 12–18.

[13] AWS, AWS IoT, <https://aws.amazon.com/iot>, (accessed 17.05.23).

[14] Siemens, MindSphere Siemens, <https://siemens.mindsphere.io/en>, (accessed 14.12.22).

[15] MS Azure, MS Azure IoT, <https://azure.microsoft.com/en-us/overview/iot>, (accessed 17.05.23).

[16] IBM, IBM Watson IoT, <https://ww.ibm.com/watson>, (accessed 17.05.23).

[17] OMA SpecWorks, OMA SpecWorks <https://omaspecworks.org/>, (accessed 17.05.23).

[18] Open and Agile Smart City, Open and Agile Smart City, <https://oascities.org/>, (accessed 17.05.23).

[19] M. Botts, G. Percivall, C. Reed, J. Davidson, OGC® sensor web enablement: Overview and high level architecture, in: GeoSensor Networks: Second International Conference, GSN 2006, Boston, MA, USA, October 1-3, 2006, Revised Selected and Invited Papers, Springer, 2008: pp. 175–190.

[20] M. Compton, P. Barnaghi, L. Bermudez, R. Garcia-Castro, O. Corcho, S. Cox, J. Graybeal, M. Hauswirth, C. Henson, A. Herzog, others, The SSN ontology of the W3C semantic sensor network incubator group, Journal of Web Semantics. 17 (2012) 25–32.

[21] P. Guillemin, F. Berens, M. Carugi, H. Barthel, A. Dechamps, R. Rees, C. Cosgrove-Sacks, J. Clark, M. Arndt, L. Ladid, others, Internet of Things Global Standardisation-State of Play, in: Internet of Things Applications-From Research and Innovation to Market Deployment, River Publishers, 2022: pp. 143–197.

[22] M. Blackstock, R. Lea, IoT interoperability: A hub-based approach, in: 2014 International Conference on the Internet of Things (IOT), IEEE, 2014: pp. 79–84.

[23] P. Desai, A. Sheth, P. Anantharam, Semantic gateway as a service architecture for iot interoperability, in: 2015 IEEE International Conference on Mobile Services, IEEE, 2015: pp. 313–319.

[24] M. Blackstock, R. Lea, Toward interoperability in a web of things, in: Proceedings of the 2013 ACM Conference on Pervasive and Ubiquitous Computing Adjunct Publication, 2013: pp. 1565–1574.

[25] S. Chun, S. Seo, B. Oh, K.-H. Lee, Semantic description, discovery and integration for the Internet of Things, in: Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015), IEEE, 2015: pp. 272–275.

[26] L. Hao, H. Schulzrinne, Goldie: Harmonization and orchestration towards a global directory for IoT, in: IEEE INFOCOM 2021-IEEE Conference on Computer Communications, IEEE, 2021: pp. 1–10.

[27] M. Jacoby, T. Usländer, Digital twin and internet of things—Current standards landscape, Applied Sciences. 10 (18), (2020) 6519.

[28] J. Conde, A. Munoz-Arcentales, A. Alonso, S. López-Pernas, J. Salvachua, Modeling digital twin data and architecture: A building guide with FIWARE as enabling technology, IEEE Internet Computing. 26 (3), (2021) 7–14.

[29] https://www.fiware.org/ Last Accessed 21-05-2023

[30] U. Ahle, J.J. Hierro, FIWARE for data spaces, Designing Data Spaces. (2022) 395. In: Otto, B., ten Hompel, M., Wrobel, S. (eds) Designing Data Spaces. Springer, Cham. https://doi.org/10.1007/978-3-030-93975-5_11.

[31] M. Jarke, C. Quix, Federated Data Integration in Data Spaces, Designing Data Spaces. (2022) 181. In: Otto, B., ten Hompel, M., Wrobel, S. (eds) Designing Data Spaces. Springer, Cham. https://doi.org/10.1007/978-3-030-93975-5_11.

[32] H. Fang, Managing data lakes in big data era: What's a data lake and why has it became popular in data management ecosystem, in: 2015 IEEE International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), IEEE, 2015: pp. 820–824.

[33] Snap4City, Snap4City Portal and service, <https://www.snap4city.org> (accessed 17.05.23).

[34] C. Badii, P. Bellini, A. Difino, P. Nesi, G. Pantaleo, M. Paolucci, Microservices suite for smart city applications, Sensors. 19 (21), (2019) 4798.

[35] M. Ammar, G. Russello, B. Crispo, Internet of Things: A survey on the security of IoT frameworks, Journal of Information Security and Applications. 38 (2018) 8–27.

[36] P.P. Ray, A survey of IoT cloud platforms, Future Computing and Informatics Journal. 1 (1-2), (2016) 35–46.

[37] D. Namiot, M. Sneps-Sneppe, On software standards for smart cities: API or DPI, in: Proceedings of the 2014 ITU Kaleidoscope Academic Conference: Living in a Converged World-Impossible without Standards?, IEEE, 2014: pp. 169–174.

[38] GDPR. Accessed: May. 21, 2023. [Online]. Available: https://en.wikipedia. org/wiki/General_Data_Protection_Regulation

[39] Mozaffari-Kermani, M., & Reyhani-Masoleh, A. (2009). Fault Detection Structures of the S-boxes and the Inverse S-boxes for the Advanced Encryption Standard. *Journal of Electronic Testing*, *25*, 225-245.

[40] Anastasova, M., Azarderakhsh, R., & Kermani, M. M. (2021). Fast strategies for the implementation of SIKE round 3 on ARM Cortex-M4. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *68*(10), 4129-4141.

[41] Bisheh-Niasar, M., Azarderakhsh, R., & Mozaffari-Kermani, M. (2021). Cryptographic accelerators for digital signature based on Ed25519. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *29*(7), 1297-1305.

[42] P. Bellini, D. Nesi, P. Nesi, M. Soderi, Federation of smart city services via APIs, in: 2020 IEEE International Conference on Smart Computing (SMARTCOMP), IEEE, 2020: pp. 356–361.

[43] C. Badii, P. Bellini, D. Cenni, A. Difino, P. Nesi, M. Paolucci, Analysis and assessment of a knowledge based smart city architecture providing service APIs, Future Generation Computer Systems. 75 (2017) 14–29.

[44] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, M. Paolucci, Automating IoT data ingestion enabling visual representation, Sensors. 21(24), (2021) 8429.

[45] P. Bellini, D. Cenni, M. Marazzini, N. Mitolo, P. Nesi, M. Paolucci, Smart city control room dashboards: big data infrastructure, from data to decision support, J. Vis. Lang. Comput. 4 (2018) 75–82.