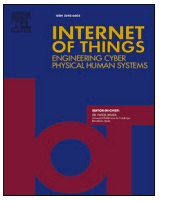
Internet of Things 25 (2024) 101125

Contents lists available at ScienceDirect

Internet of Things

journal homepage: www.sciencedirect.com/journal/internet-of-things



Lightweight data bridge for connecting self-service end-user analytic tools to NGSI-based IoT systems☆

Rui Humberto Pereira

Polytechnic of Porto/CEOS.PP, Porto, Portugal
University of Maia, UNICES, Maia, Portugal
LIACC - Artificial Intelligence and Computer Science Laboratory, Porto, Portugal

ARTICLE INFO

Keywords:
FIWARE
Internet of Things
NGSI
NGSI-LD
Self-Service Analytics Tools

ABSTRACT

The FIWARE and related projects aim to provide an open, sustainable ecosystem based on public standards and royalty-free that accelerates the development of Smart Solutions. The broker (Orion, Scorpio and Stellio) and additional components, named Generic Enablers (GE), provide means for collecting IoT data and feeding a data layer. The NGSI standard has a structural role in terms of interoperability, thus promoting the development of new GEs. However, this standard is not being adopted in analytic tools. In general, these tools connect to the IoT backend databases, which poses problems regarding data access control management and requires specialised IT support. Thus, it is not well suited for enterprise scenarios like, for instance, smart agriculture or smart industry, in which self-service tools, such as Power BI, Tableau and Qlik, are frequently used.

We propose a lightweight connector for data analytic tools and IoT ecosystems based on NGSI-v2/NGSI-LD/NGSI-TSDB. The proposed system provides web services based on JSON and CSV formats that those tools use without applying complex transformations in the data. Thus, non-technical business users can consume IoT data on demand and use all the analytical capabilities of these tools. We tested the system in an educational scenario where dashboards and analytics processes were developed with minimal effort. The system has implemented a mechanism for access control management at the service/tenant level and several querying features. The system is now being implemented in a real public service scenario. In future versions, we intend to improve the system in terms of performance, security and add more query features.

1. Introduction

The things that we use are generating more data than ever before. However, this vast amount of data is noisy, inconsistent and has variable rate flows, as well as other characteristics commonly named 5Vs [1]. Thus, in an Internet-of-Things (IoT) ecosystem, first, the data is collected, and then it must be cleaned and organised before being used and integrated with other sources of data, enabling the so-called Smart Solutions. However, IoT device manufacturers tend to apply their proprietary data models, APIs and platforms in their portfolio of products and systems, as well as IoT data also tend to be organised in silos, making it difficult to reuse it in interoperable

☆ Please cite this article as: Pereira, Rui H., Lightweight data bridge for connecting self-service end-user analytic tools to NGSI-based IoT systems, Internet of Things; Engineering Cyber Physical Human Systems (2023), https://doi.org/0000000

E-mail address: rhp@iscap.ipp.pt.

https://doi.org/10.1016/j.iot.2024.101125

Available online 13 February 2024

smart solutions. In many cases, their devices are dependent on solutions based on the software as a service (SaaS) model, thus causing some restrictions on the type of device due to hardware compatibilities, as well other limitations such as access control management, potential software and hardware vulnerabilities, periodic payment of licenses and limited access to sensor data for analysis [2].

To address these problems, the European Commission has funded several projects (*e.g.,* FIWARE, FI-Core, FI-Next, SmartSDK) either for developing and enhancing the FIWARE,[1] a Future Internet (FI) platform, as well as using it in pilots, such as City Platform as a Service - CPaaS.io[2] [3]. The goal of these initiatives is the creation of an open, sustainable ecosystem around public, royalty-free and implementation-driven software platform standards, easing the development of smart solutions and supporting organisations in their transition into smart organisations, thus enabling the development of solutions free of that lock-in vendor strategy and benefitting from data interoperability.

One of the notable aspects of the FIWARE ecosystem is its API based on the Next Generation Service Interfaces (NGSI) standard, which evolved from a set of specifications [4] originally issued by the Open Mobile Alliance™ (http://www.openmobilealliance.org).

Although those NGSI-based systems, and others, enable large volumes of IoT data to be generated and used in business processes in several use cases of data analytics, this is not always a simple task. According to Plazas et al. [5], traditional data-driven business intelligence might not be enough to compete in the context of Industry 4.0 since the collection and analysis of data from the IoT requires a more responsive approach. In [6], is presented an industrial case study in which are identified two limitations in the data analytic processes: (1) Automation - the processes are very dependent on the manual intervention based on MS Excel files; and (2) Scalability – due to lack of integration and support for large volumes of data when using MS Excel. Regarding the first limitation, the authors observed that the lack of automation leads to a considerable manual effort for end-users to build, adapt and apply analytic processes over IoT data, thus being very time-consuming for them.

Self-service business intelligence and analytics tools, such as Power BI, Tableau and Qlik, empower non-technical users to create reports and analyses without depending on IT teams [7] and significant cost savings [8], thus addressing some of the mentioned problems of lack of automation and work productivity. We notice that, according to Gartner, some of those analytic tools (Power BI, Tableau and Qlik) are market leaders, providing collaborative, self-service and user-friendly visual analytic capabilities, enabling non-technical users to explore data and get insights by developing low-code or no-code, automation workflows and applications [9].

As found in the literature, the support for self-service data analytic tools is available through several FIWARE Generic Enablers. However, these solutions are designed for big data environments, are complex, and generally store context data in backend databases or include their own visual data analytic features. When using self-service visual analytic tools, final users must connect to these backend databases, which may pose problems of access control management and other security issues because, in general, due to security policies, the set of users/systems that can access backend database systems is very restricted. Furthermore, the specific data models of these FIWARE components may also be too complex to be used by non-expert users. Thus, with the use of this type of tool to explore IoT data, the aforementioned limitations (in terms of responsive approaches [5] and automation [6]) are just partially mitigated because despite how easy it is to use these visual analytic tools to build analytic dashboards, the access to IoT data could continue to be a major problem.

The present research work began with the formulation of the following research question: RQ – *Which is the best approach for enabling end-users who are not IT and IoT experts to develop data analytic processes in NGSI-based ecosystems using self-service visual data analytic tools?* In this research work, a Design Science Research Methodology (DSRM) [10,11] was used.

In order to address those issues and answer our research question RQ, we propose a middleware that provides a table-oriented structure view of IoT entities' data in the current state and their history over time. The proposed system operates in any IoT ecosystem using the dialects NGSI-V2 and NGSI-TSDB, or NGSI-LD. With the present work, we expect to contribute to demonstrating the need for such type of component in the FIWARE catalogue, as well as making it freely available to the community.[3]

In the following section, a short overview of the FIWARE ecosystem is presented. In Section 3, the existing solutions based on FIWARE components (Generic Enablers) that provide means for developing analytic processes are discussed. Section 4 presents the methodology used in the present work. In Section 5, the proposed system is presented in terms of architecture, features and interfacing. Next, in Section 6, the system is evaluated based on an educational use case, in which students develop Power BI dashboards, RapidMiner analytic processes and PySpark-based programmes directedly connected to the IoT ecosystem using the proposed prototype. The paper is finished with the presentation of conclusions and future work proposals.

2. FIWARE ecosystem

2.1. Context brokers

In a FIWARE ecosystem, the context broker plays a major role in allowing the management of the entire lifecycle of context information, including updates, queries, registrations and subscriptions. It provides a REST API based on the Next Generation Service Interfaces (NGSI) standards [12,13], with two possible communication models: (1) Subscription/notification [14] and (2) Forwarding. Depending on the version of the NGSI supported by the broker API, some operations may not be available, as well as a broker may not implement the entire NGSI specification in the used version.

[1] https://www.fiware.org
[2] https://cpaas.bfh.ch
[3] Source code available at https://github.com/rhrp/iotbi

A context broker should be designed to provide means to produce, gather, publish and consume context data at a large scale and exploit it, thus enabling the reuse of IoT data over applications and platforms, aiming to transform solutions into truly smart applications [15].

In the FIWARE catalogue [15], one can find the context brokers Orion [16,17], Scorpio [18] and Stellio [19]. Currently, the Orion has two implementations: (1) Orion [16], based on NGSI-V2 [13], and (2) Orion Context Broker with Linked Data Extensions – Orion LD [17], based on NGSI-LD [12,20]. Both Scorpio and Stellio context brokers are based on NGSI-LD.

The NGSI-V2 is in maintenance mode, while the NGSI-LD specification is regularly updated and published by the European Telecommunications Standardization Institute (ETSI)[4]. At the present date, the latest specification is in version 1.7.1 [20], which was published in June 2023. Despite Orion LD, Scorpio and Stellio having distinct levels of implementation of the NGSI-LD specification, at the present date, they are aligned with version 1.6.1. However, it is important to note that Orion LD does not implement the NGSI-LD-based temporal API.

As part of FIWARE, several components of a catalogue, also named Generic Enablers, are available for dealing with three main aspects: (1) interfacing with the IoT devices, robots and third-party systems; (2) context Data/API management, publication, and monetisation; and (3) processing, analysis and visualisation of context information [15].

Since the communication model is based on the NGSI standard, this approach enables and encourages the growth of the catalogue in terms of the number of Generic Enablers developed by a community of members and partners, as well as its evolution in terms of the development of portable and interoperable smart solutions in several Smart* domains, such as Smart Cities, Smart Industry, Smart Energy and Smart Water. Thus, the context broker and generic enablers can be assembled together and combined with other third-party platform components to build new applications, easing the development of those smart solutions. In Section 3, we will discuss some of these components that are related to our work.

Context data is represented through values assigned to attributes that characterise the entities which are relevant to applications. An entity is a Digital Twin which digitally represents a real-world physical asset (*e.g.*, a bus in a city, a milling machine in a factory) or a concept (*e.g.,* a weather forecast, a product order). In FIWARE with linked data extensions, each Digital Twin: (1) is universally identified with a URI (Universal Resource Identifier); (2) belongs to a well-known type (*e.g.,* the Bus type or the Room type) also universally identified by a URI and (3) is characterised by several attributes, which in turn are classified as Properties, holding data (*e.g.,* the "current speed" of a Bus, or "max temperature" in a Room), and Relationships [21]. The data of these entities can be obtained from several types of sources, such as existing systems and users, through mobile apps or sensor networks. The context broker mediates the interactions applied to data entities in contexts, making them available through REST APIs. These interactions can be CRUD operations over context data, including geolocation queries, subscription of changes on data or performing the registration of external entities following an approach very similar to a proxy mechanism.

To persist data about the entities and other functional aspects, the context broker uses a database such as MongoDB,[5] PostgreSQL[6] or TimescaleDB[7] (in the case of Orion/Orion LD, Scorpio and Stellio). Additionally, both Scorpio and Stellio also use Kafka[8] to decouple communication inside the broker. Fig. 6 presents a simplified overview of the architecture of these ecosystems.

Regarding security, the Orion context broker does not provide native security and authentication mechanisms, requiring additional components. As far as we know, this is also the case with Scorpio, where these issues are still on the roadmap.[9] In the case of Stellio, an API for authentication is available.

2.2. NGSI and context information management (CIM)

In NGSI-V2, the entities are represented in JSON. On the other hand, the NGSI-LD standard is an extended subset of JSON-LD [22] to define entities and their relationships. Thus, in terms of data modelling, the NGSI-V2 is simpler than NGSI-LD because the latter follows the layered approach with three levels of data abstraction, following the Context Information Management (CIM) model [20]. The current information model of CIM is based on RDF [23] standards to capture high-level relations between entities and properties of entities. The model's objects are RDF subclasses following the structure presented in Fig. 1 (*e.g.,* NGSI-LD Entity, Relationship, and Property are subclasses of `rdfs:Resource` [24]).

As mentioned, the CIM model defines three levels of data abstraction. At the first level, the NGSI-LD Core Meta Model represents the real world's entities (things and concepts). The attributes can be of two types: Properties and Relationships (in NGSI-V2, the attributes have values and associated metadata). The entities are represented as objects using JSON-LD, thus contextualising data and respecting compliance with other linked data. Each entity should specify a `@context` statement for binding the data representation to a vocabulary supporting the upper abstraction data layers. These vocabularies linked to entity instances are published. Fig. 2, on the right, presents an example of this bind.

At the second level of data abstraction, cross-domain ontology provides commonly used constructs, such as time, geographical

[4] https://www.etsi.org
[5] https://www.mongodb.com
[6] https://www.postgresql.org/
[7] https://www.timescale.com
[8] https://kafka.apache.org/
[9] https://github.com/ScorpioBroker/ScorpioBroker/blob/development-quarkus/docs/roadmap.md#scorpio-broker-roadmap (Accessed: Oct. 30, 2023)

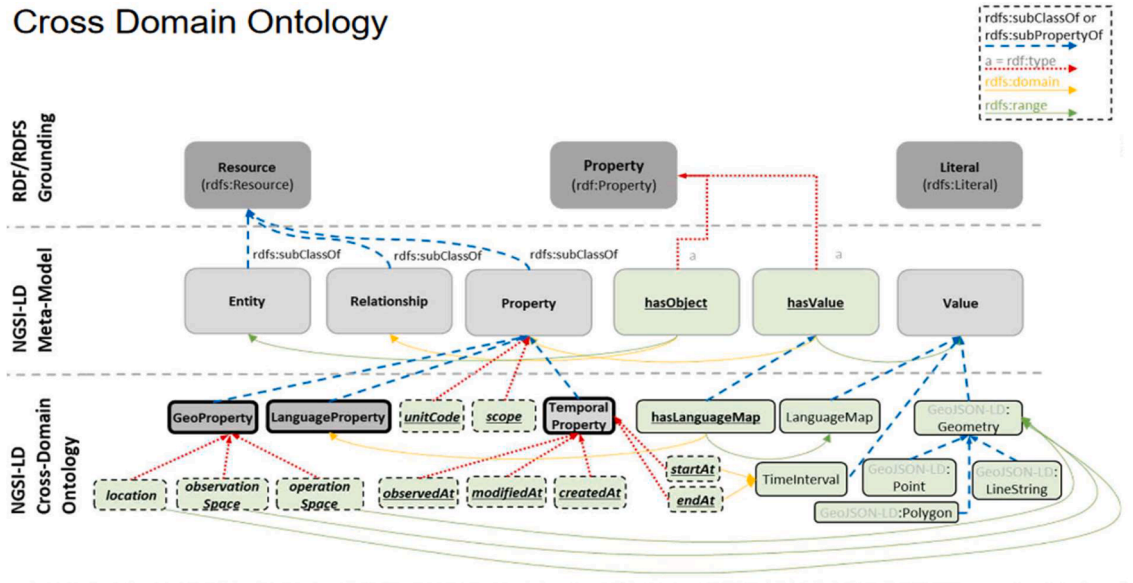


Fig. 1. Cross-domain NGSI-LD Information Model (source: [12]).

NGSI-V2

```
{
  "id": "TemperatureSensor:001",
  "type": "TemperatureSensor",
  "description": {
    "type": "Text",
    "value": "Temperature Gauge 1",
    "metadata": {}
  },
  "category": {
    "type": "Text",
    "value": "sensor",
    "metadata": {}
  },
  "controlledAsset": {
    "type": "Reference",
    "value": "Building:farm001",
    "metadata": {}
  },
  "dateObserved": {
    "type": "DateTime",
    "value": "2023-07-15T14:50:54.226Z",
    "metadata": {}
  },
  "temperature": {
    "type": "Number",
    "value": 20,
    "metadata": {
      "unitCode": {
        "type": "Text",
        "value": "CEL"
      }
    }
  }
}
```

NGSI-LD

```
{
  "@context":"http://context/ngsi-context.jsonld",
  "id":"urn:ngsi-ld:TemperatureSensor:001",
  "type":"TemperatureSensor",
  "description":{
        "value":"Temperature Gauge 1",
        "type":"Property"
  },
  "category":{
        "value":"sensor",
        "type":"Property"
  },
  "controlledProperty": {
        "value":"temperature",
        "type":"Property"
  },
  "controlledAsset":{
        "object":"urn:ngsi-ld:Building:farm001",
        "type":"Relationship"
  },
  "temperature":{
        "value":20,
        "type":"Property",
        "unitCode":"CEL",
        "observedAt":"2023-07-15T14:50:54.226Z"
   }
}
```

Fig. 2. TemperatureSensor entity in NGSI-V2 and NGSI-LD.

location and language, which generally apply to many domains. The top layer extends the previous one by defining domain-specific ontologies in any domain such as industry, building, cities or agriculture. In short, the cross-domain layer aims to describe common concepts, situations or constraints to avoid redefining them separately (and probably in a different way) in each domain-specific ontology.

In Fig. 1, these common concepts in ontology are illustrated. For instance, in the case of geo localisation, the model aims to distinguish between (a) the location of the entity, (b) its observation space and (c) operation space. For example, a camera located in a geographical point can observe a certain space, distinct from its localisation. Another example is a crane that has a location but can operate in a certain operation space. We should notice that this layer can also be represented in NGSI-V2 by means of a metadata dictionary, but it does not ensure the interoperability of these metadata, as we will discuss next.

Fig. 2 presents an example of an entity in both formats, NGSI-V2 and NGSI-LD, which helps to clarify these differences. Comparing the two data structures in the example, one can find three main differences between them: (1) In NGSI-LD, the identifier (`id`) of the

entity contains the prefix "urn:ngsi-ld"; (2) in NGSI-V2, each attribute contains a dictionary of metadata properties, in case of "temperature" the property unitCode, while in NGSI-LD this metadata is standardised, in our example the unitCode for expressing units and observedAt for expressing timestamps; and (3) in NGSI-LD, there is an additional element @context. This element references the NGSI-LD Meta-Model, providing an unambiguous definition and allowing the systems to interpret the attributes with more clarity and depth, avoiding needing a metadata dictionary of properties, as in the case of NGSI-V2. For the given example, the domain-specific specification of the temperature attribute (available at https://smartdatamodels.org/dataModel.Environment/temperature) defines it as a property of type number. We should note that there are other differences between the standards, such as the NGSI-V2 geo:json type, which has been renamed to GeoProperty. In [25], one can find more information about this issue.

It is desirable that these ontologies should be globally accepted in order to guarantee the interoperability of applications. The FIWARE Smart Data Models initiative,[10] launched by the FIWARE Foundation, provides a library of data models based on JSON/JSON-LD. Data models published under the initiative are compatible with schema.org and comply with other existing de-facto sectoral standards when they exist [21]. The Smart Data Models initiative aims to define a harmonised representation of formats and semantics that will be used by applications both to consume and publish data. Such interoperable data structures enable Data Spaces in which data may be shared across applications as-is on those Smart* domains referred to above [21,26].

In NGSI-LD, a temporal API that enables the management of entities' historical data was introduced. Using this API, users can apply several types of temporal criteria to obtain time series regarding the entities' attributes in passed states. For the case of NGSI-V2-based brokers, such as Orion, and those that do not implement the NGSI-LD-based temporal API, such as Orion LD, additional mechanisms (based on NGSI-TSDB) are required to manage the entities' historical data. The following section will discuss how this issue is handled in the FIWARE ecosystem.

3. Related work

In order to find related literature that presents approaches for implementing data analytic processes in NGSI-based ecosystems, a systematic literature review (SLR) methodology was used. At the Identification step, the following search string was applied to electronic scientific databases suggested in [27] and others also available in the search engine (http://www.b-on.pt/) used by the author. The result of this search was updated on January 24, 2024.

```
AB ( "ngsi" OR "fiware" )
AND AB ( "analytics" OR "business intelligence" )
AND AB( "internet of things" OR "iot" )
```

After removing all duplicates, twelve papers were selected. Due to the low number of papers, no additional inclusion and exclusion criteria were imposed. In the Screening step, all the papers' abstracts were read and analysed by the author in order to verify their relevance to this study. All papers that propose an approach for supporting data analytics over IoT data were considered eligible. On the other hand, the papers that only discuss the capabilities of FIWARE components were not eligible. Thus, only eleven papers were selected for the present research work.

The founded literature suggests that there are two main categories of approaches for implementing data analytic processes supported on NGSI-based ecosystems.

The first category, [28–32] present systems that implement their own analytic modules, which interact directly with the context broker for obtaining data; in these cases, the Orion context broker and NGSI-LD-based context brokers in the latter case. These studies were not included in the present work because they address a distinct problem from the one stated in RQ. On the other hand, the proposals in the second category of papers use FIWARE components to implement data streams and data sinks, which feed backend databases or build static replicas of the IoT data. This data is then used by the analytic modules. These six papers, [33–38], are discussed next.

In [34], an IoT cloud-based solution for controlling power quality in energy management systems is presented. In the proposed system, IoT data is managed by the Orion context broker and Quantumleap [39] for historical data. At the data layer, the databases MongoDB and CreateDB handle the persistence of data. Also, in the energetic sector, Haghgoo et al. [37] present a cloud-based platform for analysing power distribution. This approach is similar to the previous one, using the same FIWARE components. In [38], a third use case in the energetic sector, the authors present a system based on Orion context broker and STH Comet [40], another FIWARE component for supporting data persistence. The collected data is then used by analytic modules.

Munoz-Arcentales et al. [36] propose an open-source reference implementation for providing context-aware data analytics capabilities to IoT-based smart environments. The authors' proposal is based on the Orion LD context broker, Draco [41], Cosmos [42] and other FIWARE components for data visualisation and applications to develop operational dashboards which are highly customisable by end-users.

Frangella et al. [35] propose a FIWARE-based IoT architecture for supporting real-time data acquisition and enabling the creation of a digital twin. The proposed system is based on FIWARE Cygnus [43], a connector between Orion Context Broker and other external systems, such as databases. In the authors' proposal, a MySQL database supports the data used in the analytic processes.

In [33], the data analytics are supported by a distributed file system that saves information among several machines. The authors have employed the Hadoop Distributed File System (HDFS) for this purpose. Another FIWARE component, the Cygnus [43], is in

[10] https://smartdatamodels.org/

charge of taking data records from the context broker and saving them in Hadoop. Then, a data analytics module uses the data stored in the Hadoop.

We observed that in these related works, the analytic modules are based on FIWARE components to collect and store IoT data in backend databases or data repositories. This data is then used by analytic modules based on other components, such as Apache Spark, Grafana or Wirecloud. We claim that, in the same way, these databases can be consumed by end-users using self-service visual data analytic tools for implementing the data analytics processes as they require. Therefore, in order to systematise our discussion, we will continue to focus on these FIWARE components that provide means for supporting any type of analytic module.

Depending on the NGSI version, the context broker manages the historical data on distinct approaches. Thus, we start the discussion from the perspective of the historical data entity management in order to store it in backend databases or export it as datasets. Next, the available means for end-users to extract IoT data are discussed. This section is finished by discussing the limitations found in these related works.

3.1. Management of historical data of entities

The NGSI-V2-based context brokers, such as Orion, provide means for managing data entities at the current state of their twins. In order to preserve and manage the history of the spatial-temporal context data, the FIWARE catalogue has available components that can be added to solutions, providing them with both views: the current state of context data and the historical state in the form of time series [15]. On the other hand, the NGSI-LD standard includes the specification of a temporal API, thus eliminating the need for additional components for managing historical data relating to the past states of entities. However, as mentioned above, the NGSI-LD-based context broker Orion LD does not implement this temporal API.

The Cygnus Generic Enabler [43] provides the means for creating a historical view of the context by injecting streams of data into multiple data sinks, including some popular databases like PostgreSQL, MySQL, MongoDB or AWS DynamoDB, as well as big data platforms such as Hadoop, Storm, Spark or Flink [15]. Internally, Cygnus is based on Apache Flume[11]. This is a data flow system based on the concepts of flow-based programming that supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic, enabling the automation of data flows between systems.

Draco [41] provides the means for persisting historical context data for big data analysis. It is developed on top of Apache NiFi[12], which supports powerful and scalable directed graphs of data routing, transformation, and system mediation logic. One of the several possible scenarios consists of using the NGSIToMySQL processor in order to persist data into a MySQL database. In short, Draco subscribes to NGSI events that are attended by a flow implemented using components of types connector, controller and processor, such as NGSIToMySQL, NGSIToPostgreSQL or NGSIToMongo. This tool also provides a GUI that helps users to build the flow processes.

The Cosmos Generic Enabler is a set of tools that help achieve streaming and batch processing tasks over context data. At the moment, users can implement processes based on Spark and Flink. The data can be collected from the context broker by means of the subscription of changes, as NGSI events, and as a data source. This approach enables historical context data to be handled as needed in analytic tools. Additionally, it provides expert users with all the power of these platforms (Spark and Flink) in order to generate datasets that can be used by non-technical users.

The FIWARE PySpark Connector proposes an approach for data bridging between the context broker and PySpark. This connector includes a parser function that enables the creation of NGSIv2 and NGSI-LD entities ready to use in a custom PySpark algorithm. However, the support for NGSI-LD temporal API is still in the long-term roadmap. Thus, the system must subscribe to changes in the context broker and store data in a repository or database to keep the several states of each entity. Like in the previous cases, datasets can be produced to be consumed by end-users.

Orion-LD implements the Temporal Representation of Entities (TRoE) in an experimental state [17], taking care of populating the TRoE databases. Despite temporal operations being part of NGSI-LD API specification [44], when invoking a temporal query, one receives the message: "*Not Implemented in Orion-LD, please use Mintaka for this operation*". The Mintaka[13] component handles the temporal queries by implementing the NGSI-LD temporal retrieval API. It relies on the Orion-LD Context Broker and a TimescaleDB installation (including the Postgis extension) that Orion-LD populates.

QuantumLeap [39] Generic Enabler is an alternative to STH Comet [40]. Both can store entity history as time series data. It allows the raw data to be queried and aggregates to be calculated. QuantumLeeap and STH Comet get the entity data via context broker subscription notifications. The QuantumLeap persist data into time-series databases, such as CrateDB and Timescale, but is prepared to support multiple types of backend. It provides a REST API service, named NGSI-TSDB [45], for storing, querying and retrieving NGSI-V2 and NGSI-LD (experimental support) spatial-temporal data. In contrast, STH—Comet does not support NGSI-V2/NGSI-LD, nor multiple database backends, being strongly tied to MongoDB.

3.2. Data bridging

As discussed in the previous section, both the current state and historical data of entities are stored in backend databases following a data model specific to the platform (*e.g.,* QuantumLeap) or designed by an IoT expert. This data is then available to be used by self-

[11] https://flume.apache.org/
[12] https://nifi.apache.org
[13] https://github.com/FIWARE/Mintaka

service visual analytic tools. However, this approach may suffer from two limitations when end-users need to consume these data: (1) they become dependent on the data model, which may not include all entities and/or attributes, or it may be complex to this user profile; (2) another problem regards to security restrictions that may exist to connect to the backends databases. In general, these systems are not accessible to end-users, posing access control management problems.

In the FIWARE catalogue, we find five combinations of components that allow the consumption of the current state and historical data of entities using NGSI-based interfaces to connect the analytic tools. Table 1 presents these combinations:

Using one of these combinations of RESTful-based APIs, one can consume context data by connecting a self-service analytic tool capable of ingesting JSON, thus bypassing the security and data model issues referred to above. However, there is an object-relational impedance mismatch between the JSON structures provided by these APIs and the table-oriented structures in general required by analytics tools. Thus, the user must manually define an ETL process to transform complex JSON (NGSI) into one or more tables. This scenario is not feasible, even for experienced users. Next, this problem is discussed in detail.

The NGSI V2/LD APIs provide an optional parameter for defining their output JSON format variant as normalised, key/value (simplified), or, in the case of NGSI-LD, as concise format [46,47]. In short, these format variations differ in terms of metadata details. In the simplified format, context data properties are provided in a plain and simple structure very similar to tabular, enabling a simple ETL process for data ingestion. Fig. 3 depicts an example of a `Person` entity composed of five properties and one relationship (spouse) to another Person. In this simple format, only the values of attributes (property or relationship) are presented, but the metadata is not available.

On the other hand, the normalised format (presented in Fig. 4) provides additional metadata such as `type`, `unitCode`, `observedAt`, `createdAt` and others, which enrich the data to be consumed. However, in this case, the ETL design process is not trivial and is very time-consuming.

This impedance problem is much more problematic in the case of NGSI-TSDB and temporal NGSI-LD. In these APIs, time series are represented in the scope of its context entity JSON object as arrays of timestamps and values for each attribute.

Fig. 5 depicts a small part of NGSI-TSDB API specification, in which one can observe pairs of time stamps and values of an attribute following the same order in separate arrays. The temporal NGSI-LD API follows a similar approach using the `EntityTemporalList` and `EntityTemporal` data structures [20]. Thus, it is very complex, if not impossible, for final users to implement an analytic process supported by Power BI or similar tools. Indeed, it becomes particularly problematic in the case of different numbers of samples from one entity or attribute to another.

3.3. Current limitations in NGSI-based ecosystems for data analytics

As far as we know, the main approaches for implementing data analytics in NGSI-based ecosystems are based on FIWARE components that persist data in backend databases or other types of storage, such as HDFS. This data is then used by the analytic modules to produce data visualisations and all types of analysis, as well as it can also be consumed by self-service visual data analytic tools to produce outputs according to the users' needs. We consider that the use of these self-service visual tools presents a significative advantage over those related works that provide pre-defined or customisable dashboards. Using these tools, end-users have the ability to develop any type of dashboard as much as they need. Of course, nothing prevents the two approaches from coexisting and even complementing each other.

However, despite all the advantages of those self-service visual tools, easy and flexible access to data could still be a major challenge in such environments. We present two drawbacks: (1) When using these tools, end-users must connect to those backend databases, which may pose problems with access control management and other security issues because, in general, due to security policies, the set of users/systems that can access backend database systems is very restricted; (2) Another problem, regards to the high complexity for end-users. The specific data models of the FIWARE components in some cases are complex (*e.g.,* Stellio or Scorpio) or do not follow the relational model (*e.g.*, MongoDB in the case of Orion/Orion LD), thus making end-users dependent on IT and IoT experts. Therefore, in scenarios of local enterprise IoT-based solutions (*e.g.*, Industrial and Agricultural), these components may not be well-suitable for analytical processes based on those self-service tools.

4. Methodology

Considering the nature of the addressed problem and the research question RQ, the used methodology applies the Design Science Research (DSR) method [10] following the Hevner *et al.* guidelines [11]. This methodology used to define, implement, and validate the proposed system comprises the following activities:

4.1. Activity 1: problem identification and motivation

The author of the present work intends to solve a practical problem of end-users (who are not IT or IoT experts) when developing analytics processes, such as IoT data visualisations. The context of this work is the FIWARE-based ecosystems or any others that provide NGSI-based mechanisms for interoperability. Thus, the research question RQ was formulated: *Which is the best approach for enabling end-users who are not IT and IoT experts to develop data analytic processes in NGSI-based ecosystems using self-service visual data analytic tools?*

Answering this question by proposing a new FIWARE component, the limitations found in the related works can be overcome because this new system can help end-users extract IoT data and develop analytics processes, autonomously without relying on IT or

Table 1
Combinations of generic enablers.

Broker / NGSI version	Historical data	Comments
Orion / NGSI-V2	QuantumLeap / NGSI-TSDB	QuantumLeap has full support for Orion
Orion LD / NGSI-LD	QuantumLeap / NGSI-TSDB	QuantumLeap is experimental in Orion LD
Orion LD / NGSI-LD	Mintaka / NGSI-LD	Non-stable specifications and implementations
Scorpio / NGSI-LD		The broker supports temporal NGSI-LD API
Stellio / NGSI-LD		The broker supports temporal NGSI-LD API

NGSI-LD specification version 1.7.1 was published in June 2023.

```
{
   "@context": [
       "https://fiware.github.io/tutorials.Step-by-Step/example.jsonld",
       "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.4.jsonld"
   ],
    "id": "urn:nsgi-ld:Beatle:John_Lennon",
    "name": "John Lennon",
    "born": "1940-10-09",
    "spouse": "urn:nsgi-ld:Person:Cynthia_Lennon",
    "age": 40,
    "location": {
        "type": "Point",
        "coordinates": [-73.975, 40.775556]
    }
}
```

Fig. 3. Simplified format (source: [46]).

```
{
   "@context": [
      "https://fiware.github.io/tutorials.Step-by-Step/example.jsonld",
      "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.4.jsonld"
    ],
   "id": "urn:nsgi-ld:Beatle:John_Lennon",
   "type": "Beatle",
   "age": {"type": "Property","value": 40, "unitCode": "ANN"},
   "name": {"type": "Property","value":"John Lennon"},
   "born": {"type": "Property","value":"1940-10-09"},
   "spouse": {
       "type": "Relationship",
       "object": "urn:nsgi-ld:Person:Cynthia_Lennon"
    },
    "location": {
      "type": "GeoProperty",
      "value": {
          "type": "Point",
          "coordinates": [-73.975, 40.775556]
      }
    }
}
```

Fig. 4. Normalized format (source: [46]).

```
100   IndexArray:
101     type: array
102     items:
103       type: string
...
113   ValuesArray:
114     type: array
115     items: {}
...
123   IndexedValues:
124      type: object
125     properties:
126       index:
127         $ref: '#/definitions/IndexArray'
128       values:
129         $ref: '#/definitions/ValuesArray'
```

Fig. 5. Part of NGSI-TSDB API specification [45] that defines the `IndexedValues` object.

IoT experts.

4.2. Activity 2: define the objectives for a solution

As previously mentioned, the developed artefact should enable end-users to obtain IoT data from NGSI-based ecosystems on-demand and develop analytics processes without the help of IT or IoT experts. This objective encompasses the following requirements for the artefact to be developed: (1) A fast learning curve for using the proposed system; (2) simple mapping between IoT entities and extracted data; (3) a standard interface enabling the support for self-service visual analytics tool and (4) overcome the control access limitations of backend systems usually imposed by security policies.

The present objectives are qualitative. At this stage, quantitative objectives, such as system performance indicators, are not considered.

4.3. Activity 3: design and development

In this activity, the artefact is created. Section 5 discusses the developed prototype of a data bridge for connecting self-service end-user analytic tools to NGSI-based IoT systems.

4.4. Activity 4: demonstration

In Section 6.1, a set of experiments and results are presented. We argue that these presented results demonstrate that the developed artefact addresses the stated problem in RQ.

4.5. Activity 5: evaluation

This activity complements the previous one, observing and measuring how well the artefact supports a solution to the problem. This activity involves comparing the objectives of a solution to actual observed results from the use of the artefact in the demonstration. Thus, in Section 6.2, the results of the experiments presented in Section 6.1 are compared with the objectives formulated in Section 4.2, as well as its limitations are discussed in Section 6.3.

4.6. Activity 6. communication

In this last activity, the problem and its importance, the artefact, its utility and novelty, the rigour of its design, and its effectiveness are communicated to researchers and other relevant audiences, such as practising professionals.

5. Proposed system

In this section, is presented a lightweight data bridge that accelerates the ETL process creation in table-oriented self-service data analytic tools, providing a simple and uniform API that non-technical end-users can use to consume IoT data on-demand and build their analytic no-code processes in Power BI, Tableau, Qlik or any other analytic tool capable of ingest table-oriented data in JSON or CSV.

Using the proposed prototype, end-users can select their own datasets according to their needs, applying temporal, geolocation and other selection criteria for filtering context data stored in NGSI-based platforms in the combinations presented in Table 1. They can optimise data loading and transformation processes by applying differential loading approaches in terms of temporal and geolocation. These filter capabilities are particularly important in big data use cases due to large volumes of context data and increments of data produced in real-time by IoT devices.

The prototype supports JSON and CSV output formats in a way that does not require complex and time-consuming data transformations. Without coding, data is available for users as tables in a single step. Empowered by the self-service data analytic capabilities of visual tools, users can potentially perform as many kinds of analyses as they need.

5.1. System architecture

The proposed prototype is implemented on NodeJS using several libraries, such as Express,[14] for implementing our REST API, a modified version of NGSI-JS[15] for handling NGSI-based connection with context broker, Turf[16] for geolocation and json2csv[17] for converting JSON in CSV. As far as we know, there is no library for NGSI-TSDB support; thus, we developed all the necessary code from scratch.

[14] https://expressjs.com/
[15] https://github.com/ficodes/ngsijs /
[16] https://turfjs.org/
[17] https://juanjodiaz.github.io/json2csv/

In Fig. 6, one can observe a global perspective of the entire system architecture, from data collection through sensors, which publish data in the MQTT broker [14], to the RESTful API endpoint used by the analytic tool (*e.g.,* the Power BI). This figure does not include other applications that also consume the same context data through NGSI-based interoperability with context broker and QuantumLeap, as well as a Hadoop system used for storing static data (*i.e.,* old data collected from sensors), because these additional systems are not relevant to the proposed approach.

This figure presents a generic scenario in which one or more brokers can be used. In our use case, we tested the prototype within an environment with four brokers, Orion, Orion LD, Scorpio and Stellio, each in a distinct tenant.

In the case of Orion/Orion LD, a MongoDB database is used to store the entities that represent each Digital Twin in the real world in their last known state. The historical context data (collected by QuantumLeap by means of update subscriptions) is stored in the CrateDB database. On the other hand, the brokers that provide the temporal NGSI-LD API store both types of data, the current world state and historical data, in a single database: PostgreSQL and TimescaleDB, for Scorpio and Stellio, respectively.

As one can observe in Fig. 6, the data bridge prototype mediates the interaction between the brokers and QuantumLeap on the left side and the analytic tool on the right.

The context document (http://context/ngsi-context.jsonld), which contains the specification of the custom entities, is hosted in a web server depicted in Fig. 6.

The required connection parameters, such as IP addresses, TCP ports, NGSI versions (of the brokers and QuantumLeap) and URLs of the context document, are defined in a configuration file presented in Fig. 11- System configuration file. In Section 5.4, we discuss these settings in some detail.

5.2. Interface for analytic tools

The proposed system provides a uniform API for data bridging, enabling users to be unaware of the context broker they use. In Appendix A, the API and its parameters (Table A.1) for spatial-temporal querying and other options are presented.

This API reuses and adapts parameters of NGSI-V2, NGSI-LD and NGSI-TSDB APIs in order to get a uniform interface and extends these with three additional parameters: `format`, `extended` and `join`. The `format` parameter allows the user to choose the output format as JSON or CSV. The `extended` parameter is Boolean; If set to true, additional columns containing the context broker's metadata are added.

By default, the objects are flattened as multiple columns, each per object's attribute, to produce a table-oriented data structure. This data transformation is presented in the following example of a `Building` entity modelled according to the diagram in Fig. 7. The context broker provides the NGSI-LD structure presented in Fig. 8 on the left side. Then, this data is consumed by the proposed system, which produces the entity's columns at the output as the table-oriented data structure presented in Fig. 8 on the right for the possible data structures: normal and extended.

In the extended format, new columns for `@context`, the `address_verified` flag, location latitude and longitude are added. In the case of entity `TemperatureSensor`, the temperature attribute may get new ones, such as `temperature_unitCode` filled with the value "Celsius"; `temperature_type` filled with "Float" and `temperature_observedAt`. Additionally, inspired in the QuantumLeap, for the location still, there is an additional column containing the geographic location centroid.

Finally, this output can be represented as JSON or CSV structures according to the `format` parameter and then consumed by the analytic tool.

Regarding error responses, the error message payload is a JSON object, including an error code and optional description. Appendix A - Table A.2 depicts the types of HTTP responses and payloads.

The CRUD operations for context data modification, data aggregation per attributes (*e.g.,* sum, average), or types of queries other than temporal and geolocation make no sense in our system. Notice that the proposed system is a data bridge to enable the connection of end-user self-service data analytic tools to the IoT ecosystem. Nevertheless, query operations have a major relevance. In the next section, the querying features are discussed.

5.3. Querying

The self-service analytics tools provide visual, easy and powerful features of data filtering, aggregation and many other analytic operations available to users. The proposed system should provide a simple query syntax in order to be used by non-technical users. Thus, the proposed system is focused on providing four levels of data selection for extracting the datasets: (1) service/tenant, (2) entity level, (3) temporal and (4) geographic. Additionally, it enables join operations between two entities when a one-to-many relationship exists.

In general, IoT data is produced in real-time and is handled by subscriptions of changes in data entities, as mentioned in Section 3.1. This requires that the analytic tool can handle these subscriptions, such as the case of the cloud-based version of Power BI integrated into Azure, which provides a REST endpoint for receiving data updates [48]. To address this limitation in the desktop analytic tools (that usually do not provide such API), users can use the proposed temporal and geographic filters to split the data into smaller chunks based on geographic areas and/or time intervals of old (static) and new data to optimise the ETL process execution in the desktop tool, thus, enabling dashboards to be updated on demand.

Table inner joins are particularly useful in data models like the one in Fig. 7. The `TemperatureSensor` entity contains the temperature in a building but does not contain the location and other data about the building where the temperature was measured. In order to address this use case, the parameter `join` (see Table A.1) lets the user include the attributes of the related table, providing a

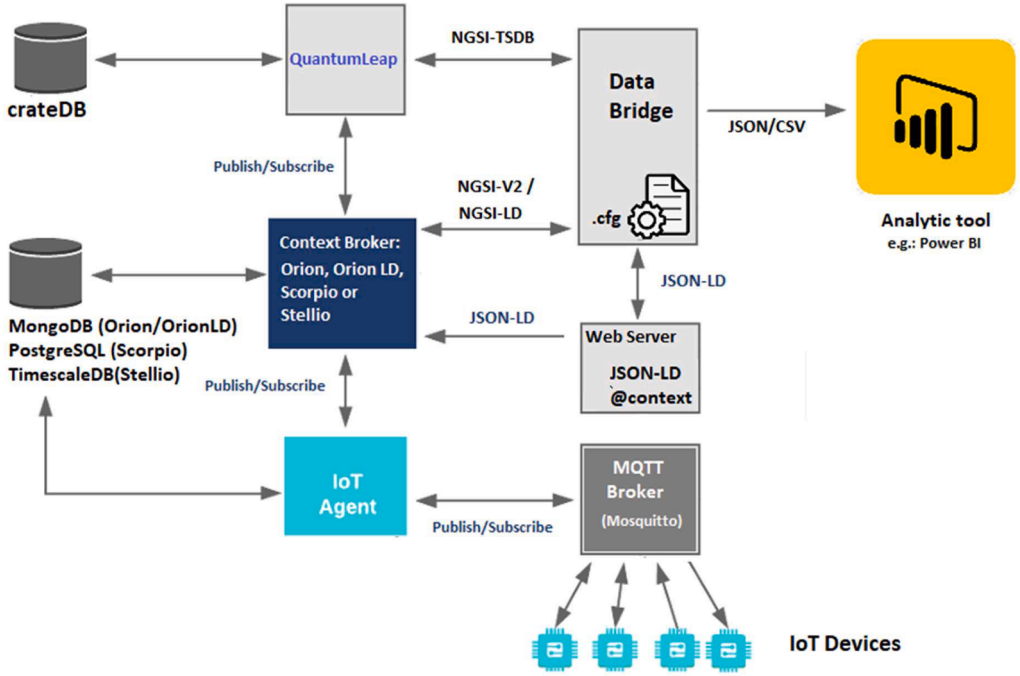


Fig. 6. System architecture.

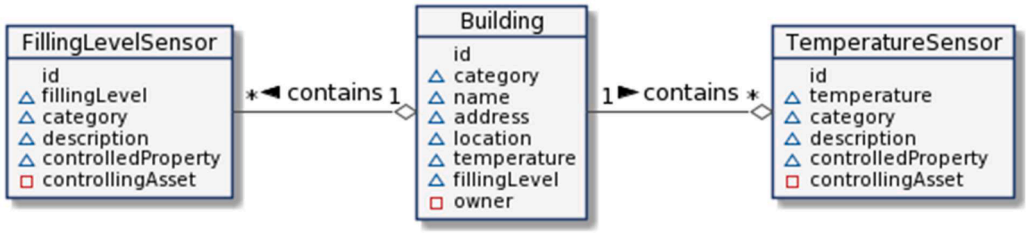


Fig. 7. Entities within the example Farm Management Information System (FMIS) (Extracted from [49]).

view of the two tables, as in the example presented in Fig. 9. In this example, is presented a tuple with the current state of an instance of `TemperatureSensor` joined with the controlled asset (building). The query predicate selects all instances of the `TemperatureSensor` entity in the service/tenant "test_ld". Because the `extended` parameter is set to true, all building attributes are also presented. Using the `attrs` parameter, users can select fewer columns to reduce the width of the dataset. Using the `entityId` path parameter or the `idPattern` parameter, users can reduce the height of the dataset to a smaller number of rows.

Fig. 10 presents another example in which the parameter `fromDate` selects all temperature values of the entity `urn:ngsi-ld:TemperatureSensor:001` from May 18th to the present moment. Due to space limitations, a single instance (table row) is presented, as well as some columns are omitted. As we can observe, three new columns provide the data and time (in two formats, `_timeIndex` and `_dateTime`) and the temperature measure at that time. If extended and join options are disabled, then the dataset just contains the first four columns: `_timeIndex, _dateTime, entityId` and `temperature`. We should notice that in the extended version (this example), two values of the temperature are presented per row: one regarding the sample (May 18th) and another in the current state (May 26th).

Additionally, in the case of a broker that supports NGSI-LD temporal API (*e.g.*, Sttelio), the `instanceId` field provides a reference to the record that contains the temperature at the observed time.

As previously mentioned, this query API was developed with simplicity as a design principle. It should enable users to select their datasets on-demand, only applying geographic and temporal criteria, as well as it should provide means for differential updates. All other complex analyses, such as filtering and data aggregation by attributes, should be addressed in the scope of the visual analytic tools.

```
## Building entity
{
    "id": "urn:ngsi-ld:Building:farm001",
    "type": "Building",
    "category": {
        "type": "Property",
        "value": ["farm"]
    },
    "address": {
        "type": "Property",
        "value": {
            "streetAddress": "Großer Stern 1",
            "addressRegion": "Berlin",
            "addressLocality": "Tiergarten",
            "postalCode": "10557"
        },
        "verified": {
            "type": "Property",
            "value": true
        }
    },
    "location": {
        "type": "GeoProperty",
        "value": {
            "type": "Point",
            "coordinates": [13.3505, 52.5144]
        }
    },
    "name": {
        "type": "Property",
        "value": "Victory Farm"
    },
    "@context": "http://context/ngsi-context.jsonld"
}'
```

```
## Prototype output: list of columns in the normal format

_entityId                   "urn:ngsi-ld:Building:farm001"
address_streetAddress       "Großer Stern 1"
address_addressRegion       "Berlin"
address_addressLocality     "Tiergarten"
address_postalCode          "10557"
name                        " Victory Farm "
category                    "farm"
location_centroid           "52.5144, 13.3505"
```

```
## Prototype output: list of columns in the extended format

_entityId                   "urn:ngsi-ld:Building:farm001"
_context                    "http://context/ngsi-
context.jsonld"
address_streetAddress       "Großer Stern 1"
address_addressRegion       "Berlin"
address_addressLocality     "Tiergarten"
address_postalCode          "10557"
address_verified            true
name                        "Victory Farm"
category                    "farm"
location_centroid           "52.5144, 13.3505"
location_centroid_lat       52.5144
location_centroid_lon       13.3505
```

Fig. 8. Data transformations.

```
## Query:  /api/v1/test_ld/TemperatureSensor/?appKey=secret&extended=true&join=Building

_entityId                                   "urn:ngsi-ld:TemperatureSensor:001"
_context_0                                  "http://context/json-context.jsonld"
_context_1                                  "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
description                                 "Temperature Gauge 1"
category                                    "sensor"
controlledAsset                             "urn:ngsi-ld:Building:farm001"
controlledProperty                          "temperature"
temperature                                 25.3
temperature_observedAt                      "2023-05-26T15:42:35.329Z"
temperature_unitCode                        "CEL"
controlledAsset__context_0                  "http://context/json-context.jsonld"
controlledAsset__context_1                  "https://uri.etsi.org/ngsi-ld/v1/ngsi-ld-core-context-v1.7.jsonld"
controlledAsset_address_streetAddress       "Großer Stern 1"
controlledAsset_address_addressRegion       "Berlin"
controlledAsset_address_addressLocality     "Tiergarten"
controlledAsset_address_postalCode          "10557"
controlledAsset_address_verified            true
controlledAsset_name                        "Victory Farm"
controlledAsset_category                    "farm"
controlledAsset_location_centroid           "52.5144, 13.3505"
controlledAsset_location_centroid_lat       52.5144
controlledAsset location centroid lon       13.3505
```

Fig. 9. Join of two entities.

```
## Query:/api/v1/test_ld/TemperatureSensor/urn:ngsi-ld:TemperatureSensor:001
         ?appKey=secret&extended=true&join=Building&fromDate=2023-05-18

_timeIndex                  1684420830000
_dateTime                   "2023-05-18T14:40:30.000+00:00"
_entityId                   "urn:ngsi-ld:TemperatureSensor:001"
temperature                 26
temperature_type            "Property"
temperature_value           25.3
temperature_instanceId      "urn:ngsi-ld:Instance:282bd2d0-de26-44dd-ad6b-c186adc793db",
temperature_observedAt      "2023-05-26T15:42:35.329Z"
temperature_unitCode        "CEL"
description                 "Temperature Gauge 1"
……
```

Fig. 10. Temporal query using a join of two entities.

5.4. System definitions

In the current version of the prototype, the security can be enabled and handled in the scope of a FIWARE service/tenant, requiring an access key per user. These issues, and others, are defined in a configuration file with the structure presented in Fig. 11. For development and test purposes, the security can be turned off. If enabled, each user must use a key that grants him a daily limit of

requests to one or more FIWARE services/tenants.

Each FIWARE service/tenant requires connection parameters (*e.g.*, IP and port) for a context broker (*e.g.,* Orion LD or Scorpio) and optionally to QuantumLeap. In this example, four services/tenants are configured for each type of supported context broker. The first two context brokers share the same QuantumLeap in port 8668. Because Scorpio and Stellio support the temporal NGSI-LD API, the connection to QuantumLeap is unnecessary.

All client accesses are logged into a log file according to the definition in these config settings.

5.5. Security – authentication and control access management

Our approach, based on middleware, provides a uniform mechanism for authentication and control access management, enabling users to be unaware of each context broker they use. At present, these mechanisms are minimal to work as proof of concept. On the internal side, due to the distinct security mechanisms of each context broker, distinct approaches are required. In the current implementation, our prototype is based on a trust relationship with each context broker.

6. Evaluation and results discussion

The proposed system aims to provide a simple interface for connecting data analytic tools to NGSI-based IoT ecosystems. We argue that enterprise solutions in smart domains, such as Smart Agriculture and Industry, could potentially benefit from the proposed system, enabling non-technical business users to consume on-demand IoT data and use it in their daily tasks. Thus, in order to evaluate the proposed system, we simulated these enterprise scenarios in an educational use case.

Using the brokers Orion, Orion LD, Scorpio and Stellio, and QuantumLeap, we implemented a real FIWARE ecosystem depicted in Fig. 6. The air quality sensors collect data from the Open Weather Map[18] public platform. The system has been working using the Orion LD and QuantumLeap since January of 2023, collecting data from 26 geographical points in the city of Maia in Portugal. The physical measurements collected were the temperature, humidity and wind speed.

Regarding computational resources, the system was implemented using four docker-based network environments running on a shared Linux virtual server with four cores and 16GB of RAM. The first two groups of dockers support the Orion and Orion LD. Each has a QuantumLeap, a CrateDB, a MongoDB, an IoT Agent, a Mosquitto and a web server in the Orion LD's docker; the third group of dockers for Scorpio, PostgreSQL and Apache Kafka and a web server and the last for Stellio, TimescaleDB and Apache Kafka and another web server. Outside the dockers, in the main host, our prototype and the context data providers (*i.e.,* the sensors) were installed. This environment was used to test the prototype and conduct its evaluation.

The experiment discussed next was conducted using the data collected from January of 2023 until now using the first docker-based network based on Orion LD and QuantumLeap.

6.1. Experiments and results

Although this experiment is based on an educational use case involving a class of business sciences master's students (with non-technical backgrounds), the goal was to simulate a real scenario involving users without non-technical skills who were asked to consume IoT data and implement analytical processes using it. The experiment started by presenting the prototype's API and the available time series of temperature, humidity and wind speed in the geographic points to a group of 14 students. Each student was then asked to implement the following five Power BI dashboards applying temporal and geographic criteria over the data of those three physical measurements: (1) the number of daily collected samples; (2) the average temperature in several points of a map (depicted in Fig. 13); (3) the daily temperature variations (*i.e.,* the temperature variation from the previous day – depicted in Fig. 14); (4) line graphic visualisation of the three time-series, one per each physical measurement and (5) the average, maximum and minimum physical measurement per each hour of the day as a bar graph.

At the end of the experiment, all students successfully imported the IoT data to the Power BI. Their difficulties only occurred during the design phase of the dashboards. We should note that all students already have basic knowledge of ETL processes involving MS Excel, CSV and (simple) JSON files. Thus, there was only a slight learning curve regarding the API and their parameters.

As one can observe in Fig. 12, a few ETL steps were required in the case of Power Query Editor, even in the JSON format option. Indeed, most of them were generated automatically in a few seconds. As discussed in Section 3.2, this step would be much more complex or impossible to achieve without the proposed system. Alternatively, one may consider the hypothesis of connecting the analytical tool to the backend database (in this scenario: CrateDB, PostgreSQL or TimesaleDB). Although this is a simple step, there are two major problems: (1) In general, in real production scenarios, these backend databases have very restrictive security policies, posing problems in terms of control and access management for the IT support teams; (2) additionally, such an approach could be more time consuming for users, due to the specificities of the data model of the broker. For instance, the QuantumLeap stores data in the CrateDB as one table per entity. However, in the case of Scorpio and Stellio, the data model in the database is much more complex, in which an entity is supported on multiple tables containing its attributes as JSON-based columns. Thus, in order to explore this alternative approach, we asked the 14 participants in the study to extract data directly from the backend systems CrateDB, used by Quantumleap

[18] https://openweathermap.org/

```
// Security
config.security={enabled:true,appKeys:{}};
config.security.appKeys['secret']={name:'Public Key',limitDay:100,scopes:['owm_v1','test_ld']};
config.security.appKeys['e395….7r6s']={name:'RHP Key',limitDay:1000,scopes:['owm_v1','test_ld']};

// Log
config.logger = {};
config.logger.access={file:'/path to/access_pbim.log',format:'combined'}

// Orion Context Broker
config.broker = {};
config.broker.servers= {};
config.broker.servers['owm_v1']={host:'172.22.0.3',port:'1026',https:false,
                                 ngsi:'v2',broker='Orion'};
config.broker.servers['test_ld']={host:'172.21.0.4',port:'2026',https:false,
                                  ngsi:'ldv1',broker='OrionLD',
                                  ngsildContext:'http://context/ngsi-context.jsonld',
                                  jsonldContext:'http://context/json-context.jsonld'};
config.broker.servers['urn:ngsi-ld:test_scorpio']={host:'172.19.0.6',port:'9090',https:false,
                                 ngsi:'ldv1',broker:'Scorpio',
                                 ngsildContext:'http://context/ngsi-context.jsonld',
                                 jsonldContext:'http://context/json-context.jsonld'};
config.broker.servers[' urn:ngsi-ld:tenant:default']={host:'172.20.0.4',port:'8080',https:false,
                                 ngsi:'ldv1',broker:'Stellio',
                                 ngsildContext:'http://context/ngsi-context.jsonld',
                                 jsonldContext:'http://context/json-context.jsonld'};
// QuantumLeep
config.quantumleap = {};
config.quantumleap.servers= {};
config.quantumleap.servers['owm_v1']={host:'172.22.0.5',port:'8668',https:false};
config.quantumleap.servers['test_ld']={host:'172.21.0.6',port:'8668',https:false};
```

Fig. 11. System configuration file.

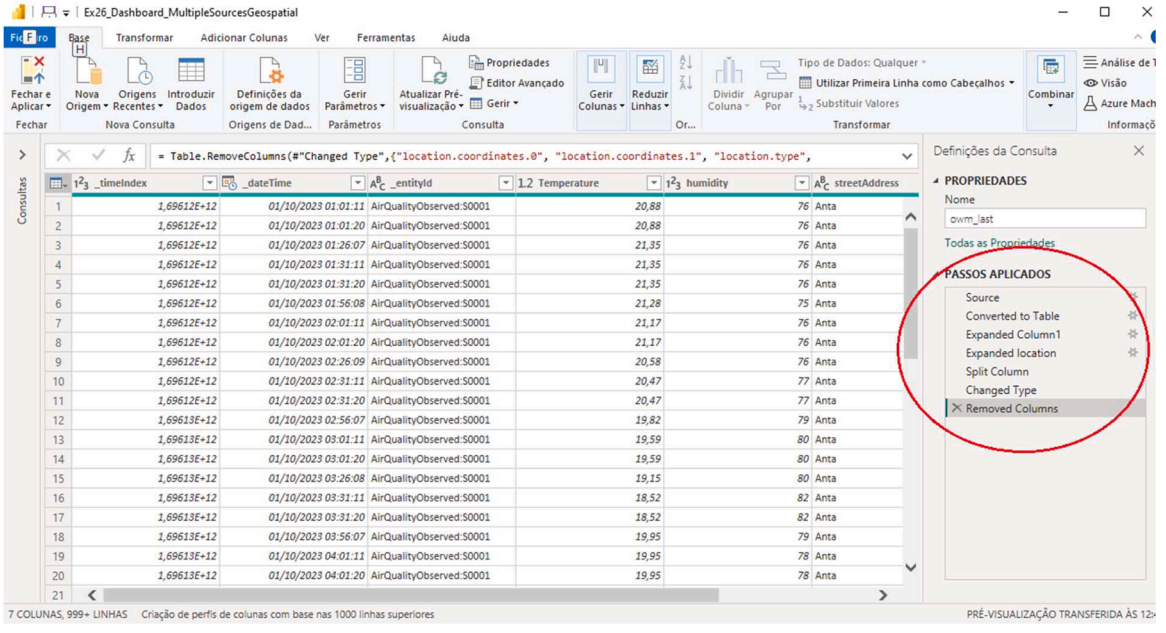


Fig. 12. Power query editor.

and PostgreSQL, used by Stellio. All the participants required technical support to establish the connection to these tested databases, contrasting with instantaneous connection using the proposed system. Using the web interface provided by the CrateDB, all participants successfully manually imported data to Power BI. However, in the case of PostgreSQL, none of them was able to finish the task. They justified their difficulties with the complexity of the data structures based on several tables and columns in the JSON format.

These results allow us to claim that the proposed system can accelerate the implementation of ETL processes, which is particularly useful when using self-service visual analytical tools because end-users have a simple interface that maps an IoT entity in a single table; thus, all complexity regarding the data model is handled in the context of the visual analytic tool. Additionally, the proposed approach still helps end-users overcome the necessary access restrictions imposed by the security policies of the IoT ecosystem. As a result, end-users can do their tasks, minimising the technical support given by the IT teams.

A second experiment involved the PySpark, a Python API for Apache Spark. This API enables users to perform real-time, large-scale data processing in a distributed environment using Python. It also provides a PySpark shell for interactive data analysis. Applying the

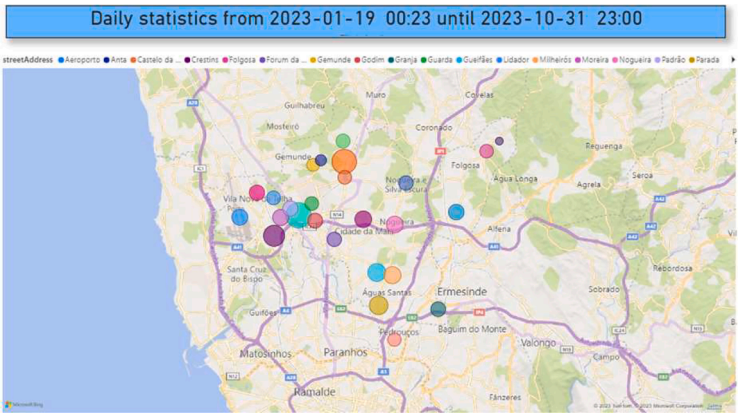


Fig. 13. Average temperature in geographical points.

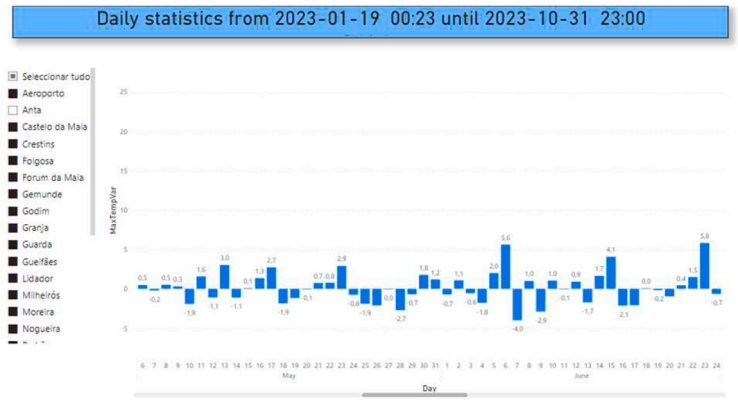


Fig. 14. Average temperature daily variations.

CSV format prototype's option, the few lines of code presented in Fig. 15 were enough to download the IoT data, load it into a DataFrame[19] and then print the average temperature per geographic place to the screen.

In this second experience, we reached the same conclusions that we had obtained in the previous one using Power BI. The end-users can be more productive and agile, less dependent on the IT support teams, focusing their effort on the analytics process in the scope of a visual analytic tool or programming environment, such as Python.

6.2. *Discussion of the results*

In the previous section, we demonstrated that the proposed system enables end-users to extract IoT data from NGSI-based ecosystems. Based on these experimental results, we now justify how the objectives defined in Section 4.2 were achieved.

(1) A fast learning curve for using the proposed system – The population that participated in the experiment does not have technical skills in the areas of IT or IoT. Despite this, their knowledge of the tools used (Power BI and PySpark) was sufficient to be able to connect to the NGSI-based ecosystem and extract data. We should notice that this task just takes a few moments (see Fig. 12). This contrasts with related work, in which end users must use different technical methods for data extraction, depending on the FIWARE component;
(2) Simple mapping between IoT entities and extracted data – The proposed system maps an IoT entity in a single table as in the relational model. Optionally, enables inner join operations on two related entities. Furthermore, the proposed approach allows users to be oblivious to the underlying FIWARE component and its NGSI dialect. Thus, end-users only manipulate data tables following the relational model. We consider this approach as simple as possible;

[19] https://spark.apache.org/docs/latest/sql-programming-guide.html

```
response=None
try:
   url=http://127.0.0.1:5000/api/v1/owm_v1/multiSensor?appKey=secret&fromDate=2023-05-01&toDate=2023-05-31
   response = urllib.request.urlopen(url)
except Exception as ex:
   print(ex)
else:
   open("/tmp/multiSensor.json", "wb").write(response.read())
   df = spark.read.json("/tmp/multiSensor.json")
   df.group by('streetAddress').avg('Temperature').show()
finally:
   if response is not None:
```

Fig. 15. A PySpark-based example for obtaining the average of the temperature per the streetAddress field.

(3) Standard interface enabling the support for self-service visual analytics tool - The system provides two possible output formats, JSON and CSV. These formats are well known and used in all leading tools, such as Power BI, Tableau and Qlik, as well as in Python-based environments;
(4) Overcome the control access limitations of backend systems usually imposed by security policies – As discussed in Section 5.1, the proposed architecture mediates the interaction between the NGSI-based ecosystem and the tools used by end-users. This approach only exposes a REST API to the outside world, avoiding access to those backend systems, as well as the context brokers. The proposed system provides an authentication mechanism and access control per FIWARE tenant (see Section 5.5).

We should highlight that there are also other benefits in terms of productivity. As discussed above, a simple task in a few seconds is required to extract data about an IoT entity. This is not the case with related works, in which end-users are dependent on IT and IoT experts, have to deal with complex database connection methods and have to apply data transformations on extracted data, thus compromising their work productivity.

In the next section, the limitations of the proposed system and future work are discussed.

6.3. Limitations and future work

The current version of the prototype does not implement any mechanism of caching, as well as little concern for code optimisation was taken into account. Thus, the real performance evaluation is out of the scope of the present work. Despite that, we performed some tests using datasets with 500.000 records (of those three physical measurements), which we consider a small volume of data compared with a real big data scenario. For this size of the dataset, in the presented docker-based environment, the execution average time of a query was 4.57 s. This is an average value that was calculated using the metrics provided by the system after ten consecutive queries. We consider this average time as satisfactory in terms of user experience during our system evaluation. However, using a larger number of records could be a significant performance issue.

Regarding security and access control, the applied security model was shown to be effective. The proposed system enabled users to consume IoT data of four distinct tenants in four distinct systems and versions of NGSI: Orion/ QuantumLeap, Orion LD/QuantumLeap, Scorpio and Stellio. We should notice that by using our system, the IoT ecosystem can be locked from the outside world. Only the endpoint discussed in Section 5.2 is available from the corporative network or the Internet. However, other scenarios involving public and corporate communities, as well as monetised approaches, will require more sophisticated security models.

In the present security model, the data granularity access control is defined at the tenant level, thus granting access to all entities in a tenant. A more granular access control at the entity level is required. Each user should have access only to a set of entities of one or more tenants.

Finally, regarding the querying features, the prototype enabled the implementation of ETL processes based on extracted data by imposed temporal and geolocation criteria. The proposed system also enables the join operation at the data extraction moment, enabling data of two entities to be ingested as a single table. These table join operations can be done in the scope of the self-service analytic tool using separated datasets (one per entity); nevertheless, this option can be particularly useful in the case of NGSI-LD data, in which entities may contain relationships with others. Despite these satisfactory results, additional criteria parameters could be needed in other complex use cases.

Thus, in terms of limitations, three types of issues should be the subject of research and improvements in future versions of the prototype: (1) performance, (2) security and access control management, and (3) new querying options.

7. Conclusions

In NGSI-based ecosystems, IoT data can be consumed by self-service visual analytic tools following two approaches. In the first, using FIWARE components (discussed in Section 3) to store data in backend databases. However, for the end-users, this approach poses problems of security and dependence on IT support. The second one is based on queries to the context brokers. These systems have available NGSI-based APIs, which provide means to query data according to spatial and temporal criteria, enabling the users to get the temperature, humidity or any other type of value measured by a sensor, serving their purposes when using analytic tools. However, these APIs have complex data structures that invalidate direct connection to those tools due to object-relational impedance mismatch between JSON structures provided by these APIs and the table-oriented structures, which, in general, are consumed by data analytics

tools.

We presented a prototype of a lightweight data bridge for self-service data analytic tools, such as Power BI, Tableau, Qlik and RapidMiner, programming environments, such as PySpark or any other type of tool that consumes table-oriented data in JSON or CSV formats, enabling them to connect directly to NGSI-V2/NGSI-LD/NGSI-TSDB based systems in order to consume real-time produced data, as well as historical data in form of time series.

The proposed system addresses those problems by exposing a simple and uniform API for context brokers (based on NGSI-V2 and NGSI-LD) and QuantumLeap (based on NGSI-TSDB), supporting their spatial and temporal query capabilities. Additionally, transforms data to a table-oriented structure and adds new query options. Finally, provides authentication and access control mechanisms. In the current version of the prototype, a user authenticates using a unique key, which grants him access to all entities of a service/tenant; for future versions, the entity type granularity control access is planned.

In order to evaluate our proposal, a FIWARE ecosystem was mounted and used in experiments conducted by a class of business sciences master's students who implemented Power BI dashboards and programs in PySpark. This experiment demonstrated that end-users could easily consume data from an IoT ecosystem on-demand and autonomously without requiring the support of specialised IT teams, contrasting with related works in terms of work productivity and agility. This is an important feature in several use cases, such as smart industrial or agricultural. The proposed system is being implemented in a real use case of a public service organisation for managing air quality data.

The presented system is still a prototype developed without major concerns about performance and security. Thus, in terms of limitations, three types of issues will be subject to improvements in future versions of the prototype: (1) performance, (2) security and access control, and (3) new querying options.

This work presents a system that makes communication between humans and machines easy.

CRediT authorship contribution statement

Rui Humberto Pereira: Writing – review & editing, Writing – original draft, Validation, Supervision, Software, Methodology, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Rui Humberto Pereira reports financial support, administrative support, and article publishing charges were provided by Polytechnic Institute of Porto Center for Organizational and Social Studies.

Data availability

Data will be made available on request.

Acknowledgements

This work is financed by Portuguese national funds through FCT - Fundação para a Ciência e Tecnologia, under the project UIDB/05422/2020.

Appendix A

API Endpoint

```
/v1/:firwareService/:entityType[/:entityId]
```

Table A.1
- API parameters.

Parameter	Type	Mandatory	Description
`appKey`	Q/H	No	The key used for access control
`fiwareService`	P	Yes	Selects the FIWARE tenant/service
`fiwareServicePath`	Q	Yes	Selects the Orion hierarchical scope of context data, *e.g.*, a city or region
`entityType`	P	Yes	Selects the entity to be exported as a table
`entityId`	P	No	Optional parameter for filtering data from a single instance of an entity
`fromDate`	Q	No	Optional parameter for limiting data from a certain date
`toDate`	Q	No	Optional parameter for limiting data till a certain date
`join`	Q	No	Join operator similar to the one in the relational model. Applicable only for NGSI-LD when the selected entity has a one-to-many relationship with another entity.
`idPattern`	Q	No	Optional parameter for filtering the entity Id using regular expressions

(*continued on next page*)

Table A.1 (*continued*)

Parameter	Type	Mandatory	Description
attrs	Q	No	Optional parameter to restrict the displayed attributes passing a comma-separated list of attribute names
georel	Q	No	Geolocation filtering. The same parameters used in the NSGI-based APIs [44,45]
coords	Q	No	
minDistance	Q	No	
maxDistance	Q	No	
limit	Q/H	No	Limit the number of rows to be exported
format	Q/H	No	JSON | CSV (default JSON)
extended	Q/H	No	If "true", additional columns containing metadata are added to the dataset

In column type, Q means query string parameter, P for a path parameter and H for HTTP Header. Some of these parameters are mandatory.

Table A.2
- Type of responses and errors.

HTTP code	Output	Meaning
200	A dataset in JSON or CSV	Success
400	JSON Object	Invalid syntax or unknown parameter
403	{	Invalid appKey for the given FIWARE service/tenant or the daily limit was exceeded
404	error:"",	Invalid FIWARE service/tenant
413	description:""	The dataset is too large. The user should try to increase the limit parameter.
500	}	Internal error, *e.g.*, while connecting to the broker

References

[1] Y. Demchenko, P. Grosso, C. de Laat, P. Membrey, Addressing big data issues in scientific data infrastructure, in: 2013 International Conference on Collaboration Technologies and Systems (CTS), IEEE, 2013, pp. 48–55, https://doi.org/10.1109/CTS.2013.6567203. May.
[2] T. Domínguez-Bolaño, O. Campos, V. Barral, C.J. Escudero, J.A. García-Naya, An overview of IoT architectures, technologies, and existing open-source projects, Internet Things 20 (2022) 100626, https://doi.org/10.1016/j.iot.2022.100626. Nov.
[3] F. Cirillo, G. Solmaz, E.L. Berz, M. Bauer, B. Cheng, E. Kovacs, A standard-based open source IoT platform: FIWARE, IEEE Internet Things Mag. 2 (3) (2019) 12–18, https://doi.org/10.1109/IOTM.0001.1800022. Sep.
[4] M. Bauer, et al., The context API in the OMA next generation service interface, in: 2010 14th International Conference on Intelligence in Next Generation Networks, IEEE, 2010, pp. 1–5, https://doi.org/10.1109/ICIN.2010.5640931. Oct.
[5] J.E. Plazas, S. Bimonte, M. Schneider, C. de Vaulx, and J.C. Corrales, "Self-service business intelligence over on-demand IoT data: a new design methodology based on rapid prototyping," 2020, pp. 84–93. 10.1007/978-3-030-54623-6_8.
[6] E. Lima, R. Bayot, P. Brito, N. Rodrigues, B. Ribeiro, N. Lopes, Business analytical framework for the manufacturing industry, in: 2021 IEEE 19th International Conference on Industrial Informatics (INDIN), IEEE, 2021, pp. 1–8, https://doi.org/10.1109/INDIN45523.2021.9557370. Jul.
[7] J. Passlick, L. Grützner, M. Schulz, M.H. Breitner, Self-service business intelligence and analytics application scenarios: a taxonomy for differentiation, Inform. Syst. e-Bus. Manage. 21 (1) (2023) 159–191, https://doi.org/10.1007/s10257-022-00574-3. Mar.
[8] P. a Schlesinger, N. Rahman, Self-service business intelligence resulting in disruptive technology, J. Comp. Inform. Syst. 56 (1) (2016) 11–21, https://doi.org/10.1080/08874417.2015.11645796. Jan.
[9] J.S.D.P.A.G.F.F.A.P.R.M.E.M.K.Q.C.L. Kurt Schlegel, "Magic quadrant for analytics and business intelligence platforms." Accessed: May 02, 2023. [Online]. Available: https://www.gartner.com/doc/reprints?id=1-2D773G95&ct=230411&st=sb.
[10] K. Peffers, T. Tuunanen, M.A. Rothenberger, S. Chatterjee, A design science research methodology for information systems research, J. Manage. Inform. Syst. 24 (3) (2007) 45–77, https://doi.org/10.2753/MIS0742-1222240302. Dec.
[11] March Hevner, Park, Ram, Design science in information systems research, MIS Q. 28 (1) (2004) 75, https://doi.org/10.2307/25148625.
[12] Bees D., Frost L., Bauer M., Fisher M., and Li W., "ETSI White Paper No. 31 - NGSI-LD API: for context information management." Accessed: Apr. 13, 2023. [Online]. Available: https://www.etsi.org/images/files/ETSIWhitePapers/etsi_wp31_NGSI_API.pdf.
[13] J.M.C. Fonseca, M. Márquez, and T. Jacobs, "FIWARE-NGSI v2 Specification." Accessed: Mar. 01, 2023. [Online]. Available: https://fiware.github.io/specifications/ngsiv2/stable/.
[14] A. Lazidis, K. Tsakos, E.G.M. Petrakis, Publish–Subscribe approaches for the IoT and the cloud: functional and performance evaluation of open-source systems, Internet Things 19 (2022) 100538, https://doi.org/10.1016/j.iot.2022.100538. Aug.
[15] FIWARE, "FIWARE components." Accessed: Mar. 01, 2023. [Online]. Available: https://www.fiware.org/catalogue/.
[16] "FIWARE Orion context broker." Accessed: Apr. 14, 2023. [Online]. Available: https://fiware-orion.readthedocs.io/en/master.
[17] "FIWARE Orion context broker (with Linked Data Extensions)." Accessed: Apr. 14, 2023. [Online]. Available: https://github.com/FIWARE/context.Orion-LD.
[18] NECTI J., "Scorpio NGSI-LD broker." Accessed: Apr. 14, 2023. [Online]. Available: https://github.com/ScorpioBroker/ScorpioBroker.
[19] "Stellio context broker." Accessed: Apr. 14, 2023. [Online]. Available: https://github.com/stellio-hub/stellio-context-broker.
[20] ESTI, "ETSI GS CIM 009 V1.7.1 - context information management (CIM); NGSI-LD API." Accessed: Aug. 08, 2023. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.07.01_60/gs_cim009v010701p.pdf.
[21] U. Ahle, J.J. Hierro, FIWARE for data spaces. Designing Data Spaces, Springer International Publishing, Cham, 2022, pp. 395–417, https://doi.org/10.1007/978-3-030-93975-5_24.
[22] Manu Sporny, Dave Longley, Gregg Kellogg, Markus Lanthaler, Pierre-Antoine Champin, and Niklas Lindström, "JSON-LD 1.1: a JSON-based serialization for linked data." Accessed: Apr. 19, 2023. [Online]. Available: https://w3c.github.io/json-ld-syntax.
[23] RDF Working Group, "W3C - resource description framework (RDF)." Accessed: Apr. 17, 2023. [Online]. Available: https://www.w3.org/RDF/.
[24] W3C, "RDF Schema 1.1 - W3C Recommendation 25 February 2014." Accessed: Apr. 17, 2023. [Online]. Available: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/.
[25] FIWARE, "Steps to migrate to JSON-LD." Accessed: May 09, 2023. [Online]. Available: https://fiware-datamodels.readthedocs.io/en/stable/ngsi-ld_howto/#steps-to-migrate-to-json-ld.
[26] "Smart data models." Accessed: Apr. 14, 2023. [Online]. Available: https://smart-data-models.github.io/data-models/.
[27] A. Carrera-Rivera, W. Ochoa, F. Larrinaga, G. Lasa, How-to conduct a systematic literature review: a quick guide for computer science research, MethodsX 9 (2022) 101895, https://doi.org/10.1016/j.mex.2022.101895.

[28] C.I. Valero, E. Ivancos Pla, R. Vaño, E. Garro, F. Boronat, C.E. Palau, Design and development of an AIoT architecture for introducing a vessel ETA cognitive service in a legacy port management solution, Sensors 21 (23) (2021) 8133, https://doi.org/10.3390/s21238133. Dec.
[29] A. Arman, P. Bellini, D. Bologna, P. Nesi, G. Pantaleo, M. Paolucci, Automating IoT data ingestion enabling visual representation, Sensors 21 (24) (2021) 8429, https://doi.org/10.3390/s21248429. Dec.
[30] K.M. Tsiouris, et al., Designing interoperable telehealth platforms: bridging IoT devices with cloud infrastructures, Enterp. Inf. Syst. 14 (8) (2020) 1194–1218, https://doi.org/10.1080/17517575.2020.1759146. Sep.
[31] G. Mastandrea, D. Mattia, L. D'Oriano, G.R. Rana, F. Nocera, M. Mongiello, IOT data-driven experimental process optimisation for kevlar fiberglass components for aeronautic. 2021 IEEE International Workshop On Metrology for Industry 4.0 & IoT (MetroInd4.0&IoT), IEEE, 2021, pp. 18–22, https://doi.org/10.1109/MetroInd4.0IoT51437.2021.9488447. Jun.
[32] L. Martín, L. Sánchez, J. Lanza, P. Sotres, Development and evaluation of Artificial Intelligence techniques for IoT data quality assessment and curation, Internet Things 22 (2023) 100779, https://doi.org/10.1016/j.iot.2023.100779. Jul.
[33] M.A. Zamora-Izquierdo, J. Santa, J.A. Martínez, V. Martínez, A.F. Skarmeta, Smart farming IoT platform based on edge and cloud computing, Biosyst. Eng. 177 (2019) 4–17, https://doi.org/10.1016/j.biosystemseng.2018.10.014. Jan.
[34] J. Garrido-Zafra, A.R. Gil-de-Castro, R. Savariego-Fernandez, M. Linan-Reyes, F. Garcia-Torres, A. Moreno-Munoz, IoT cloud-based power quality extended functionality for grid-interactive appliance controllers, IEEE Trans. Ind. Appl. 58 (3) (2022) 3909–3921, https://doi.org/10.1109/TIA.2022.3160410. May.
[35] J.L. Frangella, F. Longo, G. Mirabelli, A. Padovano, and V. Solina, "A FIWARE-based IoT platform for enabling digital twins in a greenfield smart factory: an application study on a repurposed manufacturing line," in XXVI Summer School Francesco Turco - Industrial Systems Engineering, 2021.
[36] A. Munoz-Arcentales, S. López-Pernas, J. Conde, Á. Alonso, J. Salvachúa, J.J. Hierro, Enabling context-aware data analytics in smart environments: an open source reference implementation, Sensors 21 (21) (2021) 7095, https://doi.org/10.3390/s21217095. Oct.
[37] M. Haghgoo, A. Dognini, A. Monti, A cloud-based platform for service restoration in active distribution grids, in: 2020 6th IEEE International Energy Conference (ENERGYCon), IEEE, 2020, pp. 841–846, https://doi.org/10.1109/ENERGYCon48941.2020.9236560. Sep.
[38] F. Terroso-Saenz, A. González-Vidal, A.P. Ramallo-González, A.F. Skarmeta, An open IoT platform for the management and analysis of energy data, Fut. Gen. Comp. Syst. 92 (2019) 1066–1079, https://doi.org/10.1016/j.future.2017.08.046. Mar.
[39] "QuantumLeap." Accessed: Nov. 24, 2023. [Online]. Available: https://quantumleap.readthedocs.io/en/latest/.
[40] "Welcome to the FIWARE Short Time Historic (STH) - Comet documentation." Accessed: Nov. 24, 2023. [Online]. Available: https://fiware-sth-comet.readthedocs.io/en/latest/.
[41] "FIWARE Draco." Accessed: Nov. 11, 2023. [Online]. Available: https://fiware-draco.readthedocs.io/en/latest/.
[42] "Welcome to cosmos", Accessed: Apr. 27, 2023. [Online]. Available: https://fiware-cosmos.readthedocs.io/en/latest/.
[43] "Cygnus." Accessed: Nov. 24, 2023. [Online]. Available: https://fiware-cygnus.readthedocs.io/en/latest/.
[44] "ETSI ISG CIM /NGSI-LD API specification." Accessed: May 22, 2023. [Online]. Available: https://forge.etsi.org/swagger/ui/?url=https://forge.etsi.org/rep/NGSI-LD/NGSI-LD/-/raw/master/spec/updated/generated/full_api.json.
[45] "QuantumLeap API - NGSI-TSDB." Accessed: Apr. 13, 2023. [Online]. Available: https://app.swaggerhub.com/apis/smartsdk/ngsi-tsdb.
[46] "Concise NGSI-LD." Accessed: Mar. 23, 2023. [Online]. Available: https://github.com/FIWARE/tutorials.Concise-Format.
[47] ETSI, "ETSI GS CIM 009 V1.6.1 (2022-08) cross-cutting Context Information Management (CIM); NGSI-LD API." Accessed: Mar. 01, 2023. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/CIM/001_099/009/01.06.01_60/gs_CIM009v010601p.pdf.
[48] "Real-time streaming in Power BI." Accessed: May 30, 2023. [Online]. Available: https://learn.microsoft.com/en-us/power-bi/connect-data/service-real-time-streaming.
[49] "FIWARE - linked data entity relationships." Accessed: May 24, 2023. [Online]. Available: https://github.com/FIWARE/tutorials.Entity-Relationships/.