IEEE Open Journal of
Intelligent Transportation
Systems

Received 26 January 2024; revised 20 June 2024; accepted 5 August 2024. Date of publication 9 August 2024; date of current version 20 August 2024.

Digital Object Identifier 10.1109/OJITS.2024.3441048

A Reference Architecture for Data-Driven Intelligent Public Transportation Systems

FRANCA ROCCO DI TORREPADULA[1], SERGIO DI MARTINO[1], NICOLA MAZZOCCA[1], AND PAOLO SANNINO[2]

[1]Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università degli Studi di Napoli Federico II, 80125 Naples, Italy

[2]Digital Development and Operations, Hitachi Rail STS S.p., 80147 Naples, Italy

CORRESPONDING AUTHOR: F. ROCCO DI TORREPADULA (e-mail: franca.roccoditorrepadula@unina.it)

This work was supported in part by the Italian PNRR MUR Project under Grant PE0000013-FAIR, and in part by the Spoke 9 *Digital Society & Smart Cities* of ICSC–Centro Nazionale di Ricerca in High-Performance-Computing, Big Data and Quantum Computing, funded by European Union - NextGenerationEU (PNRR-HPC) under Grant E63C22000980007.

ABSTRACT Smart cities include complex ICT ecosystems, whose definition requires the cooperation of several software systems. Among them, Intelligent Public Transportation Systems (IPTS) aim to effectively exploit public transit resources. Still, adopting an IPTS is non-trivial. Off-the-shelf IPTS are often tied to specific technologies and, thus, not easy to integrate within existing software ecosystems. Moreover, despite IPTS introduce several peculiar issues, there is a lack of domain-specific reference architectures, which would significantly ease the work of practitioners. To fill this gap, starting from the experience gained with the *Hitachi Rail* company in deploying a large-scale IPTS, we identify a set of requirements for IPTS, and propose a domain-specific reference architecture, compliant with these requirements, whose primary objective is facilitating and standardizing the design of IPTS, by providing guidelines to IPTS designers. Consequently, it eases also the interoperability among different IPTSs. As an example of an IPTS obtainable from the architecture, we present a solution currently deployed by Hitachi in a major Italian city. Still, being independent from the specific considered urban scenario, the architecture can be easily instantiated in different cities with similar needs. Finally, we discuss some research challenges which should be further investigated in this domain.

INDEX TERMS Intelligent public transportation systems, reference architectures, practical guidelines, field trials.

I. INTRODUCTION

THE current tendency towards urbanization is bringing many challenges to cities, as the pressure on their resources and infrastructures is increasing. Therefore, the concept of Smart City is becoming a widely recognized strategy to mitigate these challenges and to provide a better quality of life to its citizens [1], [2].

Such a trend is empowered by the continuous development of the Internet of Things (IoT), which allows the collection and transmission of big amounts of data, in different domains [3], [4].

From a technical standpoint, smart cities are complex ecosystems whose definition requires the cooperation of a multitude of software systems, regarding key aspects of the society (*e.g.,* mobility, economy). For instance, one of the main pillars of smart cities is represented by Intelligent Public Transportation Systems (IPTS), a class of Intelligent Transportation Systems (ITS) devoted to better managing public transport [5]. In this context, the IoT and the Internet of Vehicles (IoV) [6] enable data-driven solutions, allowing the collection of both vehicle data (such as location and speed), and passenger data (such as the number of passengers boarding a public vehicle or entering/leaving a station) [7], [8]. These data can be used to obtain new information, on top of which it is possible to develop several use cases [9], [10], with the general goal of exploiting the available public transport resources in a smarter, more effective, and even proactive way [5], [11]. To ease the

The review of this article was arranged by Associate Editor Erik Jenelius.

development and integration of multiple software systems for smart cities, many efforts are being made towards the definition of standards. In this direction, the European project *GAIA-X*[1] provides an infrastructure and data ecosystem for cities, municipalities, and districts to increase interoperability and interconnection of different systems, by leveraging existing standards and open technologies. The *FIWARE*[2] foundation is working on defining a standard framework for smart cities, allowing the creation of flexible, interoperable, and portable solutions.

Still, practitioners aiming at deploying a new IPTS have to face a number of difficulties:

- Off-the-shelf products are typically closed or tied to very specific data sensing solutions, thus being difficult to extend and/or customize;
- The development of a new solution, for the scenario at hand, poses significant challenges, also due to the nature and the amount of data to handle and analyze. As a result, the number of alternative solutions is overwhelming, in terms of computational models (cloud, edge, fog, etc.), technological solutions, architectural choices, etc.

To make things worse, architectures proposed for generic IoT solutions do not fit well in the IPTS domain, as this is characterized by a number of peculiar issues, such as distributed and moving sensors, resulting in a spatio-temporal nature of both the problem and the data, with significant impacts on the overall software structure. Furthermore, from our analysis of the state of the art, the architectures proposed for IPTS are typically too vertical with respect to a specific service/technology, or too generic, being therefore not easy to instantiate.

The U.S. Department of Transportation has recently proposed the *Architecture Reference for Cooperative and Intelligent Transportation (ARC-IT)*,[3] which includes several use cases for various subdomains within the ITS field, including also public transport systems. While being an important contribution to the ITS domain, this proposal delves into the definition of standardized interfaces among different modules. However, it still leaves the burden of the identification of technological solutions and computational models on the IPTS practitioners.

Hence, in our vision, there is a lack of domain-specific reference architectures for IPTS, which would significantly ease the work of practitioners. More in detail, a reference architecture is the definition of software elements and the data flows among them, designed to fulfill a set of identified functionalities/requirements [12]. It is the abstraction of software architectures that can be exploited for different scenarios and use cases, designed for facilitating and standardizing the development of concrete architectures.

[1]https://www.data-infrastructure.eu/GAIAX/Navigation/EN/Home/home.html

[2]https://www.fiware.org/

[3]https://www.arc-it.net/

Based on the experience we acquired within an academic-industrial collaboration, in the present paper we first identify a set of unique requirements for the data-driven IPTS architectures, and then we propose a reference architecture for IPTS specifically based on IoT/IoV data, but providing at the same time more accurate guidelines to system designers (*e.g.,* narrowing the field of suitable technologies). Furthermore, we propose a catalog of technologies that can be used for the implementation of these components. Finally, as a use case scenario, we present an instance of the proposed reference architecture, which is being employed by the Hitachi Rail company to deliver IPTS services in a real-world smart city.

We believe that this proposal and experience can be useful for IPTS architects in the development of their solutions.

II. PRELIMINARIES AND RELATED WORKS

A. DEFINITION OF REFERENCE ARCHITECTURE

According to [12], a *reference architecture* is the definition of software elements and the data flows among them, designed to fulfill a set of identified functionalities/requirements. More in detail, a *concrete architecture* is designed for a specific context, using specific technologies, and reflects concrete business goals of the stakeholders. Instead, a reference architecture is an abstraction of concrete ones, designed with two major goals, namely:

- the *facilitation* of the design of concrete architectures, aiming at providing guidelines and best practices for the design of systems;
- the *standardization* of concrete architectures, aiming at improving interoperability [13], [14].

The main difference between a reference and a concrete architecture lies in the generalization of the former, which implies its applicability in different scenarios, reflecting the requirements of the stakeholders in these contexts. This generic nature is achieved by designing them at higher levels of abstraction, thus abstracting from minor differences due to specific issues/technologies at hand [13]. As a consequence, a reference architecture is neutral with respect to technological solutions.

B. ARCHITECTURES FOR IPTS

The aim of this paper is to present a generic reference architecture for the IPTS domain. The rationale behind this work is that IPTS are characterized by very specific issues, such as complex spatio-temporal data collected from moving sensors, high variability in the type and frequency of the analyses to perform, the type and quality of the sensors to account for, the pervasive distribution of the components, and so on. These issues pose severe technological challenges to practitioners willing to develop a new IPTS. To make things worse, in the literature, there are no reference architectures or guidelines for this class of complex software systems. At the same time, architectures for generic IoT systems are not fully suitable for the IPTS domain, as they usually lack

supporting one or more of the above-highlighted aspects. Indeed, as summarized by Omoniwa et al. [15], most of the architectures presented for IoT systems are composed of three to five generic layers, namely a *perception*, a *network* and an *application* layer, optionally along with a *service* and a *business* layer. They often do not provide explicit support for the key concept that, in IPTS, sensors are mostly moving objects, distributed over a wide area, which escalates the complexity of deploying, managing, and monitoring such systems [16]. Moreover, due to the mobility of most IPTS components, wireless communication links are predominant in this kind of system and are known to be prone to a variety of attacks [17]. Additionally, analytics tasks on spatio-temporal data can be challenging, requiring ad-hoc technologies and solutions (such as a database management system able to cope with effective filtering and retrieving of this type of data).

The U.S. Department of Transportation has recently proposed the *Architecture Reference for Cooperative and Intelligent Transportation (ARC-IT)*. This proposal includes several *packages* to implement use cases for various subdomains within the Intelligent Transportation Systems (ITS) field, including commercial vehicles, parking, traffic management, and also public transport systems. It is worth noting, however, that this proposal is mostly focused on defining standardized communication interfaces among different modules, without delving into the specifics of how an IPTS should be implemented, its functionalities, or the technologies to be used. Thus, while being an important contribution to the ITS domain, still the ARC-IT solution leaves the burden of the identification of technological solutions and computational models on the IPTS practitioners.

There is a small number of papers where, among other points, IPTS actual architectures are presented. For example, Luo et al. [4] propose a framework focused on the scheduling problem of public transport based on IoT. The system consists of three different layers: (1) the *perception* layer involves sensors for data collection; (2) the *network* layer is responsible for transmitting the information from the previous layer to the successive one; (3) the *application* layer processes the received data and provides applications for passengers or transportation system's managers. In [3] the authors propose a solution for real-time data processing, based on an architecture consisting of three components: (1) *big data organization and management*, which involves data acquisition and pre-processing; (2) *big data processing and analysis*, using the *Hadoop* ecosystem and the *Apache Spark* processing engine; (3) *big data service management*, where decision-making is performed. Guerreiro et al. [18] describe a solution entirely dedicated to big data processing on ITS, employing technologies such as *Apache Spark*, *SparkSQL* and *MongoDB*. The proposal consists of four modules, respectively for data collection, data cleaning and harmonization, data storage, and data visualization. An architecture for conducting big data analytics in ITS is proposed in [11], and it involves three layers: (1) the *data collection* layer, where several kinds of data sources are located; (2) the *data analytics* layer, concerning data storage, management, mining and analysis; (3) *application* layer, where ITS-related tasks are performed. The architecture proposed in [19] aims at analyzing transportation data with Hadoop along with Spark, to handle real-time transportation data. The system is further divided into four layers, namely (1) *data collection and acquisition*, (2) *network*, (3) *data processing*, and (4) *application*. Amini et al. [20] define an architecture for real-time traffic control, where *Apache Kafka* is used as the layer that decouples publishers and subscribers from the analytics engine. Moreover, the *Hadoop Distributed File System (HDFS)* is employed as a data warehouse for posterior analysis (of raw data). Instead, the analytics engine gets input from all the publishing topics and performs data analysis. In our opinion, the architectures underlying the works described in [3], [11], [18], [19], [20] are very vertical, as they are typically designed and implemented to cope with a limited number of specific tasks, thus with low margins for generalizability to broader scenarios.

III. REQUIREMENTS FOR AN IPTS REFERENCE ARCHITECTURE

In the following, we propose a reference architecture for IPTS intended to facilitate the design of concrete IPTS-related architectures and provide guidelines for the system design. To this purpose, we first identify the principal requirements needed to be fulfilled by an IPTS, and then we define a reference architecture that maps the identified requirements into suitable software components. The identification of the requirements stems from the extensive experience we gained from several academic and industrial projects in the field of IPTS. Furthermore, we performed a thorough analysis of the existing literature, as described in Section II-B.

A reference architecture for IPTS should be designed to be generic enough to accommodate a wide number of use cases, such as vehicle scheduling (*e.g.,* [21], [22]), route planning (*e.g.,* [23], [24]), passenger load/flow prediction (*e.g.,* [25], [26]), or vehicle arrival time prediction (*e.g.,* [27], [28]).

In addition, we identified a set of further requirements specific to the IPTS domain, reported in Table 1, together with a description of the rationale behind each of them.

Let us note that none of the architectures reviewed in Section II-B can fully cover all the defined non-functional requirements. Table 2 reports whether a considered architecture covers the identified requirements. More in detail, the presence of a check mark means that the architecture covers the requirement; the letter *P* means that the architecture partially covers the requirement; the dash indicates that the architecture does not cover the requirement. To briefly summarize, all the architectures allow data collection from multiple and different devices, whereas none contemplates the use of edge computing to limit costs and bandwidth

TABLE 1. The identified IPTS-specific requirements with the correspondent rationales.

Req.	Brief Description	Rationale
R1	Acquisition from multiple and heterogeneous spatio-temporal data sources	An IPTS should be fed by data coming from a multitude of IoT/IoV sensors, both on- and off-board, characterized by different data types, formats, sampling rates, and so on. This is because such sensors are typically realized by different vendors, and therefore operate in different fashions. To complicate this scenario, is not uncommon that data collected from sensors meant to support a given use-case are used for different goals (*e.g.* cameras installed for onboard surveillance purposes can be exploited also for vehicle crowding monitoring). Thus, the reference architecture should simplify the connection to a number of IoT/IoV sensors, even not currently foreseen.
R2	Connection to multiple third-party data sources of interest	Many IPTS-related services could require/benefit from combining data acquired from IoT/IoV sensors, under the control of the system, with third-party data sources, such as network maps, vehicle scheduling information (*e.g.* GTFS (General Transit Feed Specification)), weather conditions, and so on. Thus, the reference architecture should enable a flexible connection to several external data sources.
R3	Internal storage/management capabilities for massive spatio-temporal data	An IPTS is typically fed with spatio-temporal data. To ease data analysis and processing, the acquired data (both from internal and third-party data sources) should be locally stored in a database explicitly supporting spatio-temporal data retrieval/filtering.
R4	Support for multiple data processing paradigms, both in batch and stream	An IPTS can offer a multitude of services, which may require different processing paradigms. Indeed, on the one hand, the batch processing paradigm allows to compute a significant amount of data, in a given time span; on the other hand, stream processing enables the dynamic and ongoing elaboration of streams of data, continuously and immediately, as they become available. Therefore, the reference architecture should support both the aforementioned processing paradigms.
R5	Loose coupling of internal components, to enable future addition/replacement of components/services	IPTS-related services may change over time. For this reason, an IPTS reference architecture should ease the plug-in of new services or components, or the replacement of the current ones.
R6	Interoperability with external services/applications, by means of standard communication protocols	The services produced by an IPTS, being valuable both for passengers and transport operators, should be accessed using different kinds of applications and/or devices. Furthermore, the system could be part of a more complex software ecosystem, (*e.g.* a smart city solution). Thus, interoperability APIs should be provided to enable external systems to interact with IPTS.
R7	Scalability, with respect to the number of active IoT/IoV devices	Since IPTS may deal with a variable number of data sources (*e.g.* vehicles, stops, etc.) the volume of the collected data can be very significant, going towards massive spatio-temporal datasets. As a consequence, it is fundamental to keep scalability aspects in mind, in order to effectively manage also a wide number of vehicles.
R8	Exploitation of edge computing resources	In IPTS massive data are acquired through different sensors, installed in different locations. To reduce network bandwidth, latency, and computational costs, IPTS should allow processing on the location where data are acquired, which is typical of the edge computing paradigm.

TABLE 2. Coverage of the identified requirements in the reviewed architectures.

	R1	R2	R3	R4	R5	R6	R7	R8
[3]	✓	-	P	✓	-	-	-	-
[4]	✓	-	-	-	-	-	-	-
[11]	✓	-	P	-	-	-	-	-
[18]	✓	✓	P	P	-	✓	✓	-
[19]	✓	-	P	✓	✓	-	✓	-
[20]	✓	-	P	✓	✓	-	✓	-

consumption. Moreover, while many proposals envision a storage component to store and manage historical data, none explicitly considers solutions with spatio-temporal capabilities.

To fill this gap, we propose a reference architecture that satisfies all the identified requirements, leveraging at the same time prominent aspects from the other proposals available in the literature. In particular, we identify the suitable software components forming the architecture, together with their connections, and then we describe how these components contribute to fulfilling the identified requirements. This process is reported in Section IV.

IV. A REFERENCE ARCHITECTURE IMPLEMENTING THE DEFINED REQUIREMENTS

In this section, we detail the reference architecture we defined, in terms of its operating environment, subsystems, components, and connections.

A. THE OPERATING ENVIRONMENT

We envision a modern IPTS to be deployed among four major platforms, as depicted in Figure 1:

1) Vehicles and/or other locations under the control of the solution (*i.e.,* sensors installed at bus stops), where data are collected.
2) Third-party data/service providers, with which the IPTS interacts to obtain information such as the digital map of the road network, the weather, and so on. This platform thus lies outside the boundaries of the architecture.
3) A back-end, namely the execution platform where the core part of the solution is intended to be executed. It could be a local server farm or a Cloud infrastructure.
4) A multitude of end-points, for different intended users. For example, we can foresee applications for intended

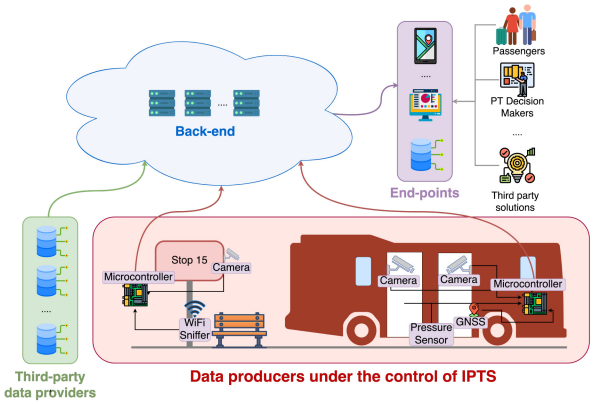

FIGURE 1. The operating environment.

users running on smartphones, external Web servers or even data centers can leverage the information generated by the solution, etc. In particular, we foresee at least the following potential end-users:

- Passengers that are interested in obtaining information regarding the PT quality of service [29] (*e.g.,* estimated time of arrival [30] or predicted crowding [31]). They can use such services through a mobile app or a website;
- PT decision-makers that want to optimize route planning and vehicle allocation to effectively exploit the available resources. In this case, the service can be accessed through a specific dashboard.
- Third-party software solutions that can exploit the information generated by the IPTS to realize their use cases.

It is worth noting that, in some settings, interoperability is a mandatory requirement for software in use with public administrations. For example, this is one of the key points introduced in the New Interoperable Europe Act.[4] Of course, in the presence of settings where interoperability is not required or a commercial benefit, the corresponding modules could not be implemented.

B. SUBSYSTEMS AND THEIR CONNECTIONS

The operational environment generically depicted in Figure 1 is reported in much more detail in this section. In particular, the reference architecture is presented in Figure 2. Since we are dealing with a reference architecture for data-driven IPTS, the role of data is crucial, as well as the ability of the architecture to deal with a multitude of data formats.

In the following, we describe each of its composing subsystems.

The **Data Producers subsystem** is firstly responsible for handling spatio-temporal data collection from multiple and heterogeneous IoT solutions, as stated by requirements R1 in Table 1. This is achieved within the *perception module*, which involves different kinds of sensors, enabling the acquisition of data of interest, such as those about vehicle state (*e.g.,* location, speed or acceleration) or passengers (*e.g.,* regarding how many people are on a vehicle).

In the presence of massive and/or complex data, it is not viable to stream them towards the back-end, due to cost and bandwidth limitations. For instance, the use of cameras for crowding detection requires video streams to be processed through an object detection phase, to estimate the number of on-board passengers. If this task were performed on the back-end, it would require the transmission of a massive amount of data from the cameras to the back-end, with huge requirements of network bandwidth and also leading to several issues, such as transmission delay and packet loss. To face this issue, we envision that an on-board preprocessing step may be necessary, as stated by requirement R8, to perform data preprocessing or aggregation before forwarding them to the remote back-end [32]. These computational steps are under the responsibility of the *preprocessing module* of the DPS, which should run on one or more microcontrollers deployed close to data sources, according to the edge computing paradigm [33].

The **Ingestion subsystem** is responsible for handling the data ingestion and/or homogenization towards the rest of the system. Indeed, as reported in Table 1, an IPTS should support data acquisition from a multitude of sensors (R1), as well as from third-party data providers (R2). Thus, data are collected in different formats. At the same time, an IPTS should be ready to accept future extensions of the system towards additional typologies of sensors, even not currently foreseen. For these reasons, a homogenization step is needed.

More in detail, within this subsystem we envision two classes of components, namely the *Data Ingestors* and the *External Data Connectors*, dealing respectively with the Data Producers subsystem and with third-party data/service providers, like other ICT solutions composing the Smart City ecosystem. Each component acts as an *adapter*, formatting the received heterogeneous data in conformity with an internal messaging solution (*e.g.,* a message broker) to be available for the rest of the system. Due to the variety of both sensors and third-party data sources, we foresee a different data ingestor or external data connector for each class of sensors/sources. The main differences between data ingestors and external data connectors lie in the mode and frequency of the interaction with the sources. Indeed, Data Ingestors handle continuous streams of data received from the Data Producer subsystem through a message broker (examined in Section V-A). On the other hand, External Data Connectors are meant to integrate external sources at data or at an application level. An example of interoperability at data level is the access to GTFS files describing the routes, while for the application level, an example can be the interaction with the APIs of an existing Automatic Vehicle Location (AVL) system, or road traffic monitoring services. Thus, these connectors interrogate the corresponding external data sources on demand, typically with a much lower frequency.

[4]https://ec.europa.eu/commission/presscorner/detail/en/IP_22_6907

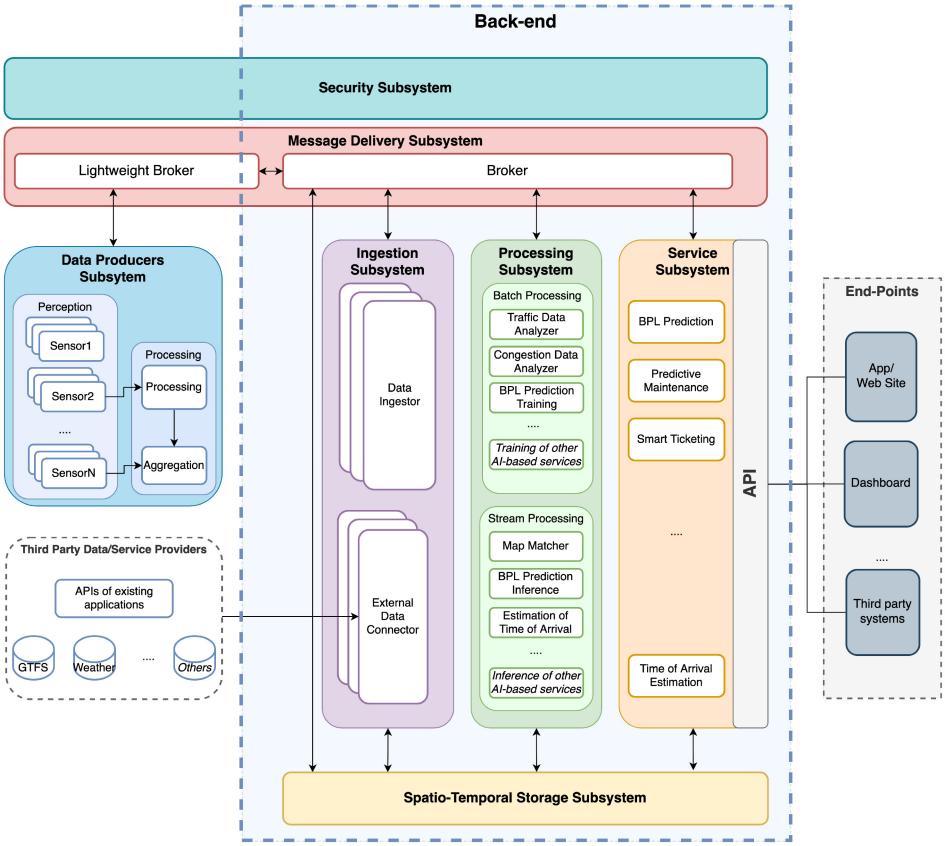

FIGURE 2. The Proposed Reference Architecture for Intelligent Public Transportation Systems.

The **Processing subsystem** deals with big data processing paradigms, where a well-known distinction involves [34]:

- *Batch processing*, mainly used to elaborate a significant amount of data, in a given time span;
- *Stream processing*, which refers to the dynamic and ongoing elaboration of streams of data, continuously and immediately, as they become available.

In the proposed reference architecture, given the nature of the considered data and offered services, both batch and stream processing modules are supported. Batch processing allows the analysis of historical and massive spatio-temporal data, collected from a multitude of IoT/IoV devices, to extract new meaningful knowledge, such as trends regarding passengers' demand or transport supply. This analysis can be conducted also in an apriori fashion. Moreover, since the IoT/IoV devices mostly generate continuous streams of data, it is also possible to elaborate them with lower latency, through stream processing, to act promptly or even proactively.

Finally, when the volume of incoming data is huge, but not necessarily continuous, micro-batch processing can be leveraged too, being a middle ground between the aforementioned paradigms. Differently from batch processing, here data are divided into manageable batches, potentially allowing for more efficient utilization of processing resources.

Let us note that, within the Processing Subsystem, any kind of batch or stream processing algorithm can be included. This means that the system eases the plug-in of novel and even unforeseen processing techniques, including new Artificial Intelligence (AI) solutions. In this case, since the training of AI models is typically performed offline, being quite time-consuming, the corresponding component, responsible for such a task, can be executed on the batch processing module. On the other hand, the inference phase of AI models usually requires lower processing resources and is performed on the fly, whenever new data are available. For this reason, it runs on the stream processing module.

However, it is worth noting that, in the case of tasks that require online/incremental training, AI model training is also carried out in the stream processing module.

The **Service subsystem** encapsulates a set of higher-level services, which can be mainly realized on top of the outputs from batch and/or stream processing components, to implement different use cases, varying in purpose and complexity. For instance, in this subsystem, there could be a service to compute the passenger load prediction of a bus at the stops of a given line. This would require information processed by multiple stream processors, a former performing the map matching of the live IoV data coming from the vehicle, and the subsequent one implementing the inference model for passenger load prediction [35]. Optionally, modules of the

Service subsystem could also exploit data from the storage subsystem and/or the internal message broker.

The functionalities implemented by the Service subsystem can be publicly exposed by means of APIs using standard formats, such as REST. These APIs act as a *façade*, to this subsystem, allowing both the access, in an easy and standardized way, to the offered IPT-related services, both the interoperability of the platform with external software solutions, such as those of a smart city ecosystem, or external IPTS.

The **Message Delivery Subsystem** is orthogonal to the aforementioned ones. It mainly involves brokers, whose main duty is to preserve a loose coupling among the different components of the system. This property is stated by requirement R5. Broadly speaking, a broker enables the communication between M data producers and N data consumers, which may not be aware of each other, usually according to a *publish-subscribe* pattern. Indeed, thanks to a broker, the *publishers* (*i.e.,* the data producers) do not send their messages directly to data consumers, but rather they publish them to the broker, which is responsible for their organization into appropriate *topics*/*contents*. The broker forwards the messages to the *subscribers* (*i.e.,* data consumers) that are interested in that specific topic/content. The key advantages of using such a design pattern are entity and time decoupling: publishers and subscribers do not need to know each other, and the communication can be *asynchronous*, as they do not need to produce and consume at the same time [36]. This is made possible by the broker, which is in charge of dispatching messages and storing them locally, until they are read by subscribers.

In our reference architecture, we recommend the use of two different types of message brokers, to fulfill different requirements. Indeed, the communication between the Data Producer subsystem and the Ingestion subsystem involves IoT devices, that typically have limited computational and energetic resources, and sometimes use metered connections (*i.e.,* wireless network connections with a limited amount of data usage per month). *Lightweight Brokers* are specifically designed for this scenario, in which devices exchange data in near real-time, while limiting network bandwidth usage [37]. Thus, these brokers implement IoT-oriented protocols for message delivery. On the other hand, within the back-end infrastructure, a more sophisticated broker can be employed for the communication among subsystems, where requirements on the available resources are less stringent, but other types of non-functional requirements are paramount, such as flexibility, scalability, fault-tolerance, and so on.

The **Spatio-Temporal Storage Subsystem** deals with data persistency, which is required by several use-cases within an IPTS (*e.g.,* reporting services or machine/deep learning models). For instance, predictive models for passenger load prediction, or for vehicles' predictive maintenance, need to be fed by a historical dataset, properly filtered from all collected data. Thus, a subsystem responsible for data storage and retrieval is essential. Furthermore, IPTS-related data are typically characterized by a spatio-temporal nature. Therefore, the Spatio-Temporal Storage subsystem should explicitly support effective spatio-temporal filtering/querying, on top of the massive collected datasets.

Finally, the **Security Subsystem** prevents the exploitation of potential vulnerabilities from malicious agents. In the following list, typical security features that an IPTS should consider are reported [38].

- *Confidentiality:* data must be made available only to authorized users. Several sensitive data (*e.g.,* on-board videos) are involved and transmitted among IPTS modules [39], and thus can be intercepted through attacks, such as *eavesdropping*. To complicate this scenario, the high degree of heterogeneity and the distributed nature of IPTS increase the complexity of monitoring and control of such a system. Moreover, many IPTS components, being installed on vehicles, communicate through wireless links, which are inherently insecure and prone to a variety of attacks [16]. Techniques to mitigate such vulnerabilities must be employed, like private networks and/or encryption methods, along with Identity and Access Management (IAM) capabilities.
- *Integrity:* data must not be improperly modified or deleted by unauthorized users. An example of an attack against integrity in IPTS is GPS spoofing, consisting of attackers broadcasting false GPS signals to cause travelers to change their trip, based on these manipulated data [16]. To prevent this, as well as other typical attacks, such as *malware, false data injection* or *node capture*, smart intrusion detection systems should be employed, to detect deviations from normal behaviors [40], [41].
- *Availability:* data and services must be available, whenever they are requested, for authorized users. Attacks such as *denial-of-service* (DoS) are critical for IPTS, leading to temporal unavailability or unacceptable delay of the service. Thus, they should be prevented, for instance by applying signature-based authentication or proof-of-work strategies [16].
- *Privacy:* the access and usage of confidential or personal information about an entity must be protected. In IPTS, sensitive data are exploited, ranging from financial information (*e.g.,* for payment purposes) to location information as we all citizens' habits. Thus, IPTS must have the ability to protect and anonymize personal information, also according to local privacy regulations, such as the European General Data Protection Regulation (GDPR). A well-known technique, for this purpose, is *differential privacy* [42], which introduces artificial noise, in query answers or directly in the data, in a controlled way.

V. A CATALOG OF TECHNOLOGIES FOR THE IMPLEMENTATION OF THE REFERENCE ARCHITECTURE

Practitioners willing to instantiate the proposed reference architecture must select one or more specific technologies

for each of the aforementioned subsystems. To support them in such a process, by narrowing the field of suitable technologies, in this section we present a brief catalog of available technological solutions for the domain-specific subsystems. For the sake of brevity, we omit discussion on the general-purpose modules, such as the security one.

A. MESSAGE BROKERS

In our reference architecture, we recommend the use of two different types of message brokers, to fulfill different requirements. Indeed, where IoT/IoV devices are involved, account should be taken of the limited resources at their disposal. Instead, at the back-end, further non-functional requirements should be considered, such as scalability and fault tolerance. In the following, we briefly presented a catalog of IoT-oriented protocols, and possible solutions for the broker employed on the back-end.

1) IOT-ORIENTED PROTOCOLS FOR LIGHTWEIGHT BROKERS

When IoT devices are involved in communication, it must be considered that these devices have limited resources, low power, and sometimes use a meter connection [37]. For this reason, the use of an IoT-oriented protocol for message delivery is recommended. Possible solutions are *MQTT (Message Queuing Telemetry Transport Protocol)*, *AMQP (Advanced Message Queuing Protocol)*, and *CoAP (Constrained Application Protocol)* [43]. They all support publish/subscribe message protocol, being at the same time designed for lightweight communication in a constrained network. The main difference regards the transport protocol: MQTT and AMQP are based on TCP, while CoAP is based on UDP. For this reason, the latter does not incur in the connection overheads inducted by TCP. Therefore, compared to MQTT and AMQP, it reduces message size/overhead, network bandwidth, and power consumption. Conversely, MQTT and AMQP are more oriented to reliability, at the cost of more bandwidth. These protocols are implemented by a number of open-source lightweight brokers, such as Apache Active MQ for MQTT and AMQP, Eclipse Mosquitto for MQTT, and Eclipse Californium for CoAP.

2) BROKERS FOR THE BACK-END

The publish/subscribe protocol within the back-end can be implemented by more sophisticated brokers, realizing a distributed message delivery system. Indeed, here the focus is on increasing scalability and fault tolerance, rather than on reducing network bandwidth. This is usually obtained by the distribution and/or replication of topics/contents across multiple machines. A number of open-source solutions implementing distributed message delivery services are available, such as *Apache Kafka*,[5] *RabbitMQ*,[6] or Apache Pulsar.[7]

As for cloud-based solutions, typical message brokers are *Google Cloud Pub/Sub*[8] and *Amazon Kinesis*.[9]

B. BIG DATA PROCESSING FRAMEWORKS

Several frameworks are available to support the implementation of batch/stream processing solutions. One of the first solutions supporting parallel batch processing is *Apache Hadoop*.[10] It is based on the *MapReduce* paradigm [44], and includes also the *Hadoop Distributed File System* (HDFS), which is a distributed file system able to store data on multiple servers, providing parallel access and fault tolerance. However, a known drawback of Hadoop is its lack of efficiency when repeatedly reusing the same set of data, as this can require multiple access to the HDFS (*i.e.,* on physical disks), which can negatively influence the overall execution time [45]. *Apache Spark*,[11] which runs on top of Hadoop, uses more advanced caching features to increase the processing speed of an application. However, if the processed dataset overflows the in-memory cache capacity, Spark can present the same limitations as Hadoop. Additionally, Spark can be employed for stream processing too, as it can divide the incoming streams of events into groups of micro-batches, that can be processed with lower latency. However, collecting events together to build a mini-batch is still a limiting factor for real-time data analysis [46].

Let us note that there are also processing frameworks that are natively designed to support stream processing [47]. An example is *Apache Storm*.[12] Here, the architecture is similar to Hadoop, but Storm works on unbounded streams of data. It provides a master node that assigns tasks to worker nodes, monitored by supervisors. Alternatively, *Apache Flink*[13] is a native stream processing framework that supports also batch processing. It is based on a master-worker model, as Storm, and uses the snapshot mechanisms to maintain the status of jobs in distributed checkpoints, to recover it in case of failure [48]. Apache Flink includes also several libraries to support scalable batch and streaming workloads. For example, *PyFlink* is a Python API to build custom ML/AI pipelines, while *FlinkML* provides implementations of standard ML algorithms.

It is worth noting that the aforementioned big data processing frameworks do not provide native support for spatial data. Nevertheless, several efforts have been made to extend such platforms towards spatial data processing [49]. For instance, *SpatialHadoop* [50] enhances Apache Hadoop by providing a spatial language, data types, indexes, and operations. Similarly, *GeoSpark* [51] extends Apache Spark with spatial data types, such as points, rectangles, and polygons, as well as a geo-spatial query processing layer. Similar contributions have also been made for stream processing platforms. Among

[5] https://kafka.apache.org/
[6] https://www.rabbitmq.com/
[7] https://pulsar.apache.org/
[8] https://cloud.google.com/pubsub
[9] https://aws.amazon.com/it/kinesis/
[10] https://hadoop.apache.org/
[11] https://spark.apache.org/
[12] https://storm.apache.org/
[13] https://flink.apache.org/

them, *GeoFlink* [52] is a spatial extension of Apache Flink, that enables spatial indexing of vector data and spatial queries.

C. MACHINE AND DEEP LEARNING MODELS

As highlighted in Section IV, the proposed reference architecture eases the plug-in of ML/DL models, which can be exploited with the general purpose of enhancing public transports.

In particular, Recurrent Neural Networks (RNNs) have been widely adopted given their natural ability to deal with time series problems, such as passenger flow prediction, bus arrival time, and so on [53]. These models are often combined with Graph Neural Networks (GNNs), which can be exploited to examine spatial correlations. Indeed GNNs treat transportation networks as graphs and employ filters to capture spatial dependencies [54]. Recently, also Large Language Models (LLMs) have made their entrance in IPTS. LLMs are pre-trained on extensive datasets and can be fine-tuned on different specific tasks, related to the investigated scenario. Broadly speaking, LLMs can extract complex linguistic patterns from intricate datasets [55]. Hence, within the IPTS domain, they can be exploited to understand textual information (e.g., tweets related to the movement or habits of people), generate human-readable explanations of ML models, or provide reports [56].

D. SPATIAL DATABASE MANAGEMENT SYSTEMS

As data collected for IPTS are mostly characterized by a spatial nature, they require specific solutions able to support their effective storage and retrieval. In this direction, traditional Databases Management Systems (DBMS) have been extended by integrating ad-hoc spatial data types, query operators, and indexing techniques [57]. Broadly speaking, spatial DBMS can be categorized into two major families: relational and NoSQL ones [58]. Relational DBMSs are well-established solutions used for storing and retrieving large structured datasets, based on the relational model. Examples of relational DBMS, which support extensions for spatial data, are PostGIS[14] (plug-in of PostgreSQL) or Oracle Spatial Database.[15] They enable the definition of attributes using geometry data types (*e.g.,* point, polygon, line-string), support the Open Geospatial Consortium (OGC) standard SQL extensions for spatial queries, and support different index types, to speed up spatial data retrieval. While being a technology widely employed, the horizontal scaling (*i.e.,* the distribution over multiple servers) of relational DBMS is still a critical issue.

As for NoSQL DBMS, they are mostly used to manage unstructured or semi-structured data and are designed for easy horizontal scaling. Differently from relational DBMS, NoSQL ones usually do not provide full support to the ACID properties (Atomicity, Consistency, Isolation, Durability). In all the domains where the fulfillment of these properties is not mandatory, this enables the support for an easier distribution over multiple servers, thus allowing to effectively handle massive datasets [58]. As an example in the IPTS domain, information about ticket usage should be handled with an ACID-compliant DBMS, given their socio-economic importance, while high-frequency GPS data can be managed also by a NoSQL, given the scarce relevance of a single record.

Let us note that many of the spatial big data processing platforms described in Section V-B often rely on NoSQL databases, to better face challenges due to data volume, variety, and velocity. Similarly to relational DBMS, also NoSQL DBMS typically provide only minimal support to spatial data (such as Neo4j,[16] Cassandra,[17] and Redis[18]), but can be often extended to this purpose by means of plug-ins [59]. It is worth noting that MongoDB[19] natively includes spatial data types and operators, as it allows to store spatial data as GeoJSON objects, supporting also complex spatial queries and geospatial indexing techniques.

In conclusion, relational DBMS can be a suitable solution when the enforcement of ACID properties is crucial. Otherwise, whether it is possible to give up consistency for the benefit of data distribution, and thus of parallelization, NoSQL DBMS can be exploited.

VI. AN EXAMPLE OF IPTS IN A REAL-WORLD SCENARIO

In this section, we present an IPTS deployed by *Hitachi Rail* in a real smart city, as a field trial. This is an example of actual architecture obtainable from the proposed reference architecture (through appropriate implementation and technological choices), fulfilling all the requirements described in Table 1.

A. FEATURES OF THE IPTS

The system we describe, deployed in a major Italian city, is meant to support transport operators in achieving smarter bus fleet planning and management. This is obtained thanks to a combination of services, including also AI-based predictions of bus crowding. Such Hitachi Rail platform is accessible through a dashboard, depicted in Figure 3, which shows the map of the city and the actual location of the monitored buses, along with their travel direction, represented by small red arrows. Further information on traffic levels and in-vehicle crowding can be provided, by enabling the *traffic* and *crowding* options on the top right of the panel. As a result, red lines highlight road segments with heavy traffic, while a red circle around a bus indicates overcrowding conditions. As depicted in Figure 4, by clicking on a given bus, it is possible to obtain information about:

- its real-time location;

[14]https://postgis.net/
[15]https://www.oracle.com/it/database/spatial/
[16]https://neo4j.com/
[17]https://cassandra.apache.org/index.html
[18]https://redis.io/
[19]https://www.mongodb.com/

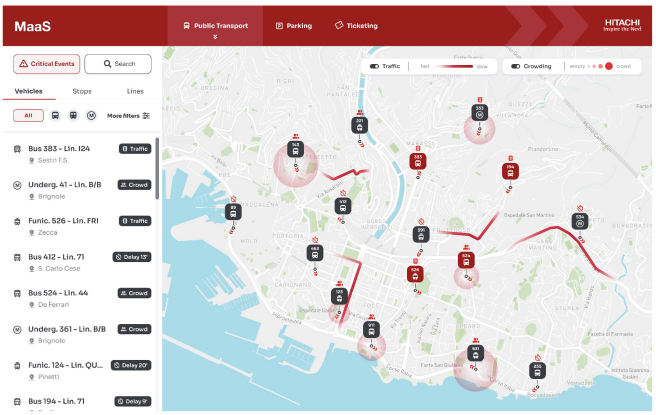

FIGURE 3. The dashboard of the Hitachi Rail platform.

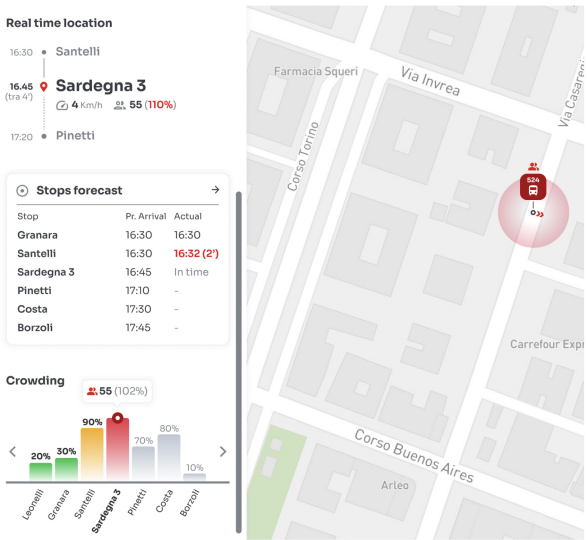

FIGURE 4. The BPL prediction and the estimation of time of arrival services.

- its planned and estimated time of arrival;
- its past and current crowding situation, as well as a prediction for the next stops.

B. IMPLEMENTATION DETAILS

In the following, we provide some details on technical aspects related to the implementation of the described framework. Bus location data are obtained by querying an existing AVL system, using a specifically designed *External Data Connector*. On the other hand, bus crowding data are collected by a custom *Data Producer Subsystem*, composed of on-board cameras, whose video streams are processed by an edge module. The obtained crowding information is then sent to the back-end, thanks to the interaction with the *Lightweight Broker*, which pushes this information to the main *Broker*. At this point, a tailor-made *Data Ingestor* component parses and then stores these ingested data in the *Spatio-Temporal Storage subsystem*. All these bus service data, accumulated over time, are then exploited by some AI-based components, to predict in-vehicle crowding and

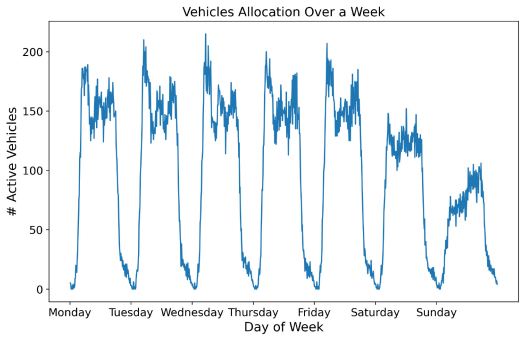

FIGURE 5. Number of vehicles allocated over a week.

journey time, for each monitored bus. This is implemented by means of different modules: *batch processors* for model training, and *stream processors* for the inference phase. These resulting predictions are then exposed by ad-hoc *REST APIs*, within the *Service subsystem*. In this way, the provided services are accessible from several locations, such as, for example, the dashboard depicted in Figure 3, as well as from other existing IPTS.

C. WORKLOAD ANALYSIS

As stated by requirements R7, an IPTS should be able to cope with a variable number of installed IoT/IoV sensors, and thus with a variable amount of data to handle. Indeed, the system should well tolerate the addition of further vehicles or stops/stations to be monitored, as well as peak situations, where more data are generated. In the following, we present a workload analysis, meant to get a better insight into the performances of the Hitachi Rail IPTS under real-world situations.

Figure 5 depicts the number of vehicles in service over a week, in May 2022. As expected, such a number is higher on weekdays, since more vehicles are active to serve a greater number of commuters. Instead, the number of working vehicles decreases during the weekend, especially on Sunday. Furthermore, focusing on a single day, a morning peak, around 8 AM, can be noticed, then a stable number up to about 11 PM, and later the number of running buses drops to almost zero, for night service.

As mentioned before, the employed vehicles are equipped with sensors, sending data about vehicle position and occupancy to the back-end. As a result, the way the fleet is allocated over a week affects the amount of data sent over that time, and thus the employed bandwidth.

Figure 6 highlights the pattern of data transmission rates from all the involved vehicles towards the back-end, over the same week considered before. This is a subset of all the exchanged data and is used, for instance, by the modules realizing the bus crowding prediction service. Indeed, such a service exploits vehicle positioning and crowding data, acquired through GNSS sensors and cameras, respectively, installed on the buses. Moreover, to avoid privacy issues and reduce network bandwidth and costs overhead, the object

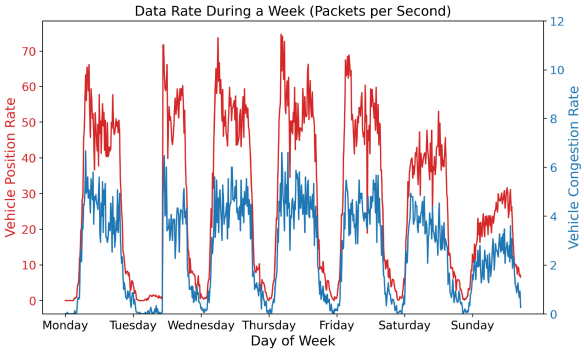

FIGURE 6. Number of packets per second transmitted from buses to the back-end, over a week. In blue are the packets with vehicle congestion data, and in red are those with vehicle position.

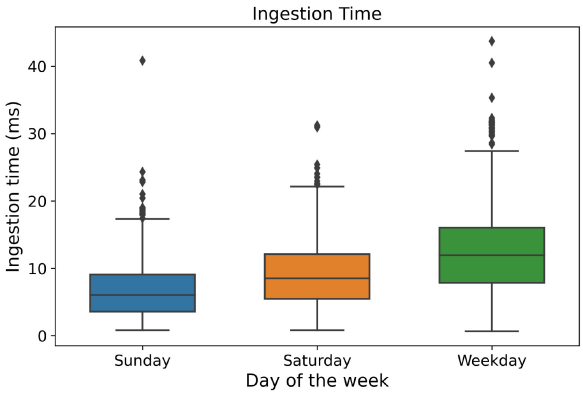

FIGURE 7. Performance of the Hitachi Rail IPTS, in terms of data ingestion time (ms), over the week.

detection phase is performed on the vehicle, according to the edge-computing paradigm. From Figure 6, we can notice that the number of packets containing monitored information is higher on weekdays. On Saturday and Sunday, instead, the transmission rate gradually decreases. Furthermore, when considering a single day, there is a morning peak, around 8 AM, then a stable number of data transmitted up to about 11 PM, and later the number of packets drops to almost zero. This is in line with the pattern shown in Figure 5.

Let us note that these mobility patterns impact the performances of the considered system. For instance, in Figure 7 we report how the ingestion time (in milliseconds), as extrapolated from the logs of the cloud provider, is affected by the variation of active vehicles, and thus of sent packets. From these box plots, we can see that the system is easily able to cope with the variability of the workloads, well tolerating the working days' peaks. For instance, according to Figure 6, on weekdays, on average, the rate of transmitted packets is twice the one of Sunday, while the increment in terms of ingestion time is less than twice, thus reflecting this ratio roughly in a sub-linear way.

VII. OPEN CHALLENGES

We believe that the proposed reference architecture can support a more effective development of IPTS. Still, we see a number of open challenges in this domain. In the following, we briefly summarize them.

Lack of standard formats for public transport data. The General Transit Feed Specification (GTFS) is a data format employed worldwide to describe fixed-route transit services, such as schedule times, routes, and bus stops [60]. It represents data about public transport planning and scheduling. An equivalent standard format for operational data (*e.g.,* actual stop arrival time or passenger load) is missing. For this reason, as highlighted in Section III, an IPTS is typically fed by data coming from heterogeneous IoT/IoV sensors, and therefore characterized by different formats. As a result, a homogenization step is required, leading to the necessity to include a *Data Ingestors* subsystem, where a different connector/adapter should be realized for each different acquisition format. In our vision, a standardization effort for operational data could be beneficial to ease this task and simplify the architecture.

Data privacy/security. Data-driven IPTS are fed by sensitive data (*e.g.,* videos of passengers acquired from monitoring cameras), transmitted and processed by different modules. Thus, access to such information must be protected, through accurate data anonymization techniques, data/channel encryption methods, IAM procedures, and so on. However, when IoT devices are involved, their limited processing, storage, and energy capabilities must be taken into account (for instance, they can limit the applicability of common cryptographic approaches). Moreover, it is well-known that such devices can be vulnerable to a number of physical and side-channel attacks. To cope with such issues, the scientific community and/or international organization (*e.g.,* the IoT Security Foundation[20]) should enforce minimum security standards in heterogeneous IoT products/applications [61].

Deployment of AI at the edge (EdgeAI). According to the edge computing paradigm, moving processing tasks closer to data sources leads to benefits in terms of network bandwidth, latency, and computational cost. However, moving the inference of AI models at the edge poses peculiar challenges, especially in the case of Deep Neural Networks (DNNs). Indeed, IoT/edge devices are characterized by limited capabilities, which contrast with the huge computational costs and memory occupancy of DNNs. Thus, further strategies to cope with these limitations should be investigated, such as model compression techniques, aiming at reducing models and fulfilling edge requirements [62].

Definition of common benchmarks for evaluating architecture effectiveness. An IPTS practitioner can be in trouble in objectively assessing the impact of different design choices on the performance of the system. Indeed, to date, to the best of our knowledge, there are no widely accepted benchmarks for this class of software systems. Moreover, traditional benchmarks for data-intensive systems available

[20]https://www.iotsecurityfoundation.org/

in the literature (*e.g.,* TPC[21]) [63] are not suitable, being both not focused on the very specific nature of the data involved in IPTS, nor representative of IPTS-related workloads. As a result, in this field comparing different architectural or software proposals is still an open challenge, and the definition of benchmarking datasets/workloads should be investigated.

Green computing. Since IPTS are strictly tied to the IoT paradigm, the environmental problems posed by this technology should be borne in mind. For instance, IoT devices need power and energy to perform their tasks, and especially to share data with other devices, or with the back-end. To deal with these environmental issues, the concept of green computing (*i.e.,* the sustainable, environment-aware, and environment-friendly computing or IT [64]) should be considered when designing an IPTS, as well as further investigated by the scientific community. Additionally, given the aforementioned lack of benchmarks for IPTS, also the comparison of different proposals, in terms of energy efficiency, remains an open problem.

VIII. CONCLUSION

Smart cities are complex ICT ecosystems, whose definition requires the cooperation of a multitude of software systems. Among them, IPTS employ data acquired through IoT/IoV devices with the general goal of exploiting the available public transport resources in a smarter, more effective, and even proactive way.

However, the deployment of a new IPTS is a challenging task. Indeed, on one hand, off-the-shelf products are typically closed or tied to very specific data-sensing solutions, thus being difficult to extend and/or customize. On the other hand, the number of alternative solutions is overwhelming, in terms of computational models (cloud, edge, fog, etc.), technological aspects, architectural choices, and so on. The definition of reference architectures is a well-known strategy to facilitate the design of complex systems and to improve their interoperability. Unfortunately, from an analysis of the state of the art, we found a lack of reference architecture for IPTS. At the same time, standard reference architectures for IoT systems do not fit well the specific domain, missing to explicitly supporting some key factors of the IPTS domain.

To fill this gap, based on the experience we gained in an academic-industrial collaboration on real IPTS, in the present paper first we identified a set of requirements specific for IPTS, and then we proposed a novel reference architecture, intended as a collection of software components and their connections, meant to comply the stated requirements. More in detail, as described in Section IV, the proposed reference architecture combines, in a novel and original way, seven subsystems. The resultant framework leads to a number of interesting features. For instance, we suggest using two different Message Brokers: a Lightweight Broker, intended for managing and supporting wireless communication with IoT/IoV devices, and a traditional message broker, to handle the message exchange among the modules deployed at the back-end. Moreover, the reference architecture requires a specific storage subsystem, able to effectively support spatio-temporal data management, filtering, and retrieving. Finally, we explicitly consider the employment of edge-computing resources, to reduce network bandwidth, latency, and computational costs.

The feasibility of the proposal is proved by presenting an operational IPTS obtainable from the described model (*i.e.,* the Hitachi Rail IPTS platform), together with a workload analysis of the proposal in a real-world scenario.

Finally, we sketched up some open challenges that in our opinion are still hindering the development of this class of systems.

Currently, we are working on the evolution of the described IPTS to offer new services, for a different geographical context. In particular, we are devising software solutions to include smart ticketing, fleet planning and scheduling, and parking management. At the current state of development, all these new services fit well with the proposed reference architecture.

We believe that our proposal will be beneficial to the developers of IPTS, as it offers accurate guidelines about the system architecture, and narrows the field of suitable technologies for the system implementation.

[21] https://www.tpc.org/default5.asp

REFERENCES

[1] L.-M. Ang, K. P. Seng, G. K. Ijemaru, and A. M. Zungeru, "Deployment of IoV for smart cities: Applications, architecture, and challenges," *IEEE Access*, vol. 7, pp. 6473–6492, 2018.

[2] E. Ismagilova, L. Hughes, Y. K. Dwivedi, and K. R. Raman, "Smart cities: Advances in research—An information systems perspective," *Int. J. Inf. Manag.*, vol. 47, pp. 88–100, Aug. 2019.

[3] M. Babar and F. Arif, "Real-time data processing scheme using big data analytics in internet of things based smart transportation environment," *J. Ambient Intell. Humanized Comput.*, vol. 10, no. 10, pp. 4167–4177, 2019.

[4] X.-G. Luo, H.-B. Zhang, Z.-L. Zhang, Y. Yu, and K. Li, "A new framework of intelligent public transportation system based on the Internet of Things," *IEEE Access*, vol. 7, pp. 55290–55304, 2019.

[5] S. Elkosantini and S. Darmoul, "Intelligent public transportation systems: A review of architectures and enabling technologies," in *Proc. Int. Conf. Adv. Logist. Transp.*, 2013, pp. 233–238.

[6] L. Guo et al., "A secure mechanism for big data collection in large scale Internet of Vehicle," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 601–610, Apr. 2017.

[7] C. Mccarthy et al., "A field study of internet of things-based solutions for automatic passenger counting," *IEEE Open J. Intell. Transp. Syst.*, vol. 2, pp. 384–401, 2021.

[8] R. Seidel, N. Jahn, S. Seo, T. Goerttler, and K. Obermayer, "NAPC: A neural algorithm for automated passenger counting in public transport on a privacy-friendly dataset," *IEEE Open J. Intell. Transp. Syst.*, vol. 3, pp. 33–44, 2021.

[9] J. Hoppe, F. Schwinger, H. Haeger, J. Wernz, and M. Jarke, "Improving the prediction of passenger numbers in public transit networks by combining short-term forecasts with real-time occupancy data," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 153–174, 2023.

[10] F. Gallo, N. Sacco, and F. Corman, "Network-wide public transport occupancy prediction framework with multiple line interactions," *IEEE Open J. Intell. Transp. Syst.*, vol. 4, pp. 815–832, 2023.

[11] L. Zhu, F. R. Yu, Y. Wang, B. Ning, and T. Tang, "Big data analytics in intelligent transportation systems: A survey," *IEEE Trans. Intell. Transp. Syst.*, vol. 20, no. 1, pp. 383–398, Jan. 2019.

[12] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston, MA, USA: Addison-Wesley Prof., 2003.

[13] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Inf. Softw. Technol.*, vol. 54, no. 4, pp. 417–431, 2012.

[14] G. Muller, "A reference architecture primer," Eindhoven Univ. Techn., Eindhoven, The Netherlands, White Paper, 2008.

[15] B. Omoniwa, R. Hussain, M. A. Javed, S. H. Bouk, and S. A. Malik, "Fog/edge computing-based IoT (FECIoT): Architecture, applications, and research issues," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4118–4149, Jun. 2019.

[16] D. Hahn, A. Munir, and V. Behzadan, "Security and privacy issues in intelligent transportation systems: Classification and challenges," *IEEE Intell. Transp. Syst. Mag.*, vol. 13, no. 1, pp. 181–196, Apr. 2019.

[17] A. Perrig, J. Stankovic, and D. Wagner, "Security in wireless sensor networks," *Commun. ACM*, vol. 47, no. 6, pp. 53–57, 2004.

[18] G. Guerreiro, P. Figueiras, R. Silva, R. Costa, and R. Jardim-Goncalves, "An architecture for big data processing on intelligent transportation systems. An application scenario on highway traffic flows," in *Proc. IEEE 8th Int. Conf. Intell. Syst. (IS)*, 2016, pp. 65–72.

[19] B. Jan, H. Farman, M. Khan, M. Talha, and I. U. Din, "Designing a smart transportation system: An Internet of Things and big data approach," *IEEE Wireless Commun.*, vol. 26, no. 4, pp. 73–79, Aug. 2019.

[20] S. Amini, I. Gerostathopoulos, and C. Prehofer, "Big data analytics architecture for real-time traffic control," in *Proc. 5th IEEE Int. Conf. Models Technol. Intell. Transp. Syst. (MT-ITS)*, 2017, pp. 710–715.

[21] A. Andrade-Michel, Y. A. Ríos-Solís, and V. Boyer, "Vehicle and reliable driver scheduling for public bus transportation systems," *Transp. Res. B, Methodol.*, vol. 145, pp. 290–301, Mar. 2021.

[22] Y. Bie, M. Hao, and M. Guo, "Optimal electric bus scheduling based on the combination of all-stop and short-turning strategies," *Sustainability*, vol. 13, no. 4, p. 1827, 2021.

[23] S. Wang, Y. Sun, C. Musco, and Z. Bao, "Public transport planning: When transit network connectivity meets commuting demand," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 1906–1919.

[24] Z. Wang, S. Wang, and H. Lian, "A route-planning method for long-distance commuter express bus service based on OD estimation from mobile phone location data: The case of the Changping Corridor in Beijing," *Public Transp.*, vol. 13, no. 1, pp. 101–125, 2021.

[25] J. Zhang et al., "A real-time passenger flow estimation and prediction method for urban bus transit systems," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 11, pp. 3168–3178, Nov. 2017.

[26] P. Wang, X. Chen, J. Chen, M. Hua, and Z. Pu, "A two-stage method for bus passenger load prediction using automatic passenger counting data," *IET Intell. Transp. Syst.*, vol. 15, no. 2, pp. 248–260, 2021.

[27] T.-H. Tsai, "Self-evolutionary sibling models to forecast railway arrivals using reservation data," *Eng. Appl. Artif. Intell.*, vol. 96, Nov. 2020, Art. no. 103960.

[28] B. Yu, W. H. Lam, and M. L. Tam, "Bus arrival time prediction at bus stop with multiple routes," *Transp. Res. C, Emerg. Technol.*, vol. 19, no. 6, pp. 1157–1170, 2011.

[29] Z. Liu, N. Wu, Y. Qiao, and Z. Li, "Performance evaluation of public bus transportation by using DEA models and Shannon's entropy: An example from a company in a large city of China," *IEEE/CAA J. Automatica Sinica*, vol. 8, no. 4, pp. 779–795, Apr. 2021.

[30] L. Dunne, F. R. Di Torrepadula, S. Di Martino, G. McArdle, and D. Nardone, "Bus journey time prediction with machine learning: An empirical experience in two cities," in *Proc. Int. Symp. Web Wireless Geograph. Inf. Syst.*, 2023, pp. 105–120.

[31] I. Peled, K. Lee, Y. Jiang, J. Dauwels, and F. C. Pereira, "On the quality requirements of demand prediction for dynamic public transport," *Commun. Transp. Res.*, vol. 1, Dec. 2021, Art. no. 100008.

[32] W. Yu et al., "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2017.

[33] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.

[34] K. Reddig, B. Dikunow, and K. Krzykowska, "Proposal of big data route selection methods for autonomous vehicles," *Internet Technol. Lett.*, vol. 1, no. 5, p. e36, 2018.

[35] F. Amato, S. Di Martino, N. Mazzocca, D. Nardone, F. R. di Torrepadula, and P. Sannino, "Bus passenger load prediction: Challenges from an industrial experience," in *Proc. Int. Symp. Web Wireless Geographical Inf. Syst.*, 2022, pp. 93–107.

[36] P. Dobbelaere and K. S. Esmaili, "Kafka versus RabbitMQ: A comparative study of two industry reference publish/subscribe implementations: Industry paper," in *Proc. 11th ACM Int. Conf. Distrib. Event-Based Syst.*, 2017, pp. 227–238.

[37] G. C. Hillar, *MQTT Essentials—A Lightweight IoT Protocol*. Birmingham, U.K.: Packt Publ. Ltd., 2017.

[38] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.

[39] H. Yang, J. Zhao, and M. Wang, "Optimal control of automated left-turn platoon at contraflow left-turn lane intersections," *J. Intell. Connected Veh.*, vol. 5, no. 3, pp. 206–214, 2022.

[40] T. Olovsson, T. Svensson, and J. Wu, "Future connected vehicles: Communications demands, privacy and cyber-security," *Commun. Transp. Res.*, vol. 2, no. 1, 2022, Art. no. 100056.

[41] W. Yu, J. Li, L.-M. Peng, X. Xiong, K. Yang, and H. Wang, "SOTIF risk mitigation based on unified ODD monitoring for autonomous vehicles," *J. Intell. Connected Veh.*, vol. 5, no. 3, pp. 157–166, 2022.

[42] P. A. Bonatti and S. Kirrane, "Big data and analytics in the age of the GDPR," in *Proc. IEEE Int. Congr. Big Data (BigDataCongress)*, 2019, pp. 7–16.

[43] B. Mishra and A. Kertesz, "The use of MQTT in M2M and IoT systems: A survey," *IEEE Access*, vol. 8, pp. 201071–201086, 2020.

[44] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[45] Y. Samadi, M. Zbakh, and C. Tadonki, "Comparative study between Hadoop and spark based on Hibench benchmarks," in *Proc. 2nd Int. Conf. Cloud Comput. Technol. Appl. (CloudTech)*, 2016, pp. 267–275.

[46] R. Sahal, J. G. Breslin, and M. I. Ali, "Big data and stream processing platforms for industry 4.0 requirements mapping for a predictive maintenance use case," *J. Manuf. Syst.*, vol. 54, pp. 138–151, Jan. 2020.

[47] H. Nasiri, S. Nasehi, and M. Goudarzi, "Evaluation of distributed stream processing frameworks for IoT applications in smart cities," *J. Big Data*, vol. 6, no. 1, pp. 1–24, 2019.

[48] M. A. Lopez, A. G. P. Lobato, and O. C. M. Duarte, "A performance comparison of open-source stream processing platforms," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2016, pp. 1–6.

[49] Y. Loukili, Y. Lakhrissi, and S. E. B. Ali, "Geospatial big data platforms: A comprehensive review," *KN J. Cartogr. Geograph. Inf.*, vol. 72, no. 4, pp. 293–308, Dec. 2022. [Online]. Available: https://doi.org/10.1007/s42489-022-00121-7

[50] A. Eldawy and M. F. Mokbel, "SpatialHadoop: A mapreduce framework for spatial data," in *Proc. IEEE 31st Int. Conf. Data Eng.*, 2015, pp. 1352–1363.

[51] J. Yu, J. Wu, and M. Sarwat, "Geospark: A cluster computing framework for processing large-scale spatial data," in *Proc. 23rd SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2015, pp. 1–4.

[52] S. A. Shaikh, K. Mariam, H. Kitagawa, and K.-S. Kim, "GeoFlink: A distributed and scalable framework for the real-time processing of spatial streams," in *Proc. 29th ACM Int. Conf. Inf. Knowl. Manag.*, 2020, pp. 3149–3156.

[53] Q. Jiang, "GMM clustering based on WOA optimization and space-time coupled urban rail traffic flow prediction by CEEMD-SE-BiGRU-AM," *Mobile Inf. Syst.*, to be published.

[54] J. Wang, Y. Zhang, Y. Wei, Y. Hu, X. Piao, and B. Yin, "Metro passenger flow prediction via dynamic hypergraph convolution networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 12, pp. 7891–7903, Dec. 2021.

[55] Y. Zhang, C. Wei, S. Wu, Z. He, and W. Yu, "GEOGPT: Understanding and processing geospatial tasks through an autonomous GPT," 2023, *arXiv:2307.07930*.

[56] L. L. L. Starace and S. Di Martino, "Can large language models automatically generate GIS reports?" in *Proc. Int. Symp. Web Wireless Geograph. Inf. Syst.*, 2024, pp. 147–161.

[57] E. Baralis, A. D. Valle, P. Garza, C. Rossi, and F. Scullino, "SQL versus NoSQL databases for geospatial applications," in *Proc. IEEE Int. Conf. Big Data (Big Data)*, 2017, pp. 3388–3397.

[58] D. Guo and E. Onstein, "State-of-the-art geospatial information processing in NoSQL databases," *ISPRS Int. J. Geo-Inf.*, vol. 9, no. 5, p. 331, 2020.

[59] Y. Gacha, M. A. B. Rhaiem, and T. Abdellatif, "Geospatial big data for a sustainable and green smart city," in *Proc. Int. Conf. Cyberworlds (CW)*, Oct. 2023, pp. 217–224. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10337569?casa_token=6mRINfyhBvMAAAAA:Y0Zdxjhxm0ldd2o-dBzSQGNgLUNUnpKhaoNz5edPOScdvohXguhs3KjSCMX6_d5sAdyYDfQR

[60] A. Antrim and S. J. Barbeau, *The Many Uses of GTFS Data—Opening the Door to Transit and Multimodal Applications*, Location-Aware Inf. Syst. Lab., Univ. South Florida, Tampa, FL, USA, 2013.

[61] P. Anand, Y. Singh, A. Selwal, M. Alazab, S. Tanwar, and N. Kumar, "IoT vulnerability assessment for sustainable computing: Threats, current solutions, and open challenges," *IEEE Access*, vol. 8, pp. 168825–168853, 2020.

[62] Y. Cheng, D. Wang, P. Zhou, and T. Zhang, "Model compression and acceleration for deep neural networks: The principles, progress, and challenges," *IEEE Signal Process. Mag.*, vol. 35, no. 1, pp. 126–136, Jan. 2018.

[63] S. D. Martino, A. Peron, A. Riccabone, and V. N. Vitale, "Benchmarking management techniques for massive IIoT time series in a fog architecture," *Int. J. Grid Utility Comput.*, vol. 12, no. 2, pp. 113–125, 2021.

[64] N. I. Sarkar and S. Gul, "Green computing and Internet of Things for smart cities: Technologies, challenges, and implementation," in *Green Computing in Smart Cities: Simulation and Techniques*. Cham, Switzerland: Springer, 2021, pp. 35–50. [Online]. Available: https://doi.org/10.1007/978-3-030-48141-4_3

Open Access funding provided by 'Universitá degli Studi di Napoli "Federico II"' within the CRUI CARE Agreement