Open Access

Open Journal of Internet of Things (OJIOT)
Volume 6, Issue 1, 2020

http://www.ronpub.com/ojiot
ISSN 2364-7108

ParkingJSON: An Open Standard Format for Parking Data in Smart Cities

Gowri Sankar Ramachandran[A], Jeremy Stout[B], Joyce J. Edson[B], Bhaskar Krishnamachari[A]

[A] Viterbi School of Engineering, University of Southern California, 3650 McClintock Ave, Los Angeles, CA 90089, USA, {gsramach, bkrishna}@usc.edu
[B] Information Technology Agency, City of Los Angeles, 200 N Main St 1400, Los Angeles, CA 90012, USA, {jeremy.stout, joyce.edson}@lacity.org

Abstract

Data marketplaces and data management platforms offer a viable solution to build large city-scale Internet of Things (IoT) applications. Contemporary data marketplaces and data management platforms for smart cities such as Intelligent IoT Integrator (I3), Cisco Kinetic, Terbine, and Streamr present a middleware platform to help the data owners to provide their data to the application developers. However, such platforms suffer from adoption issues because of the interoperability concerns that stem from heterogeneous data formats. On the one hand, the IoT devices and the software used by the device owners follow either a custom data standard or a proprietary industrial standard. On the other hand, the application developers consuming data from multiple device owners expect the data to follow one common standard to process the data without developing custom software for each data feed. Therefore, a common data standard is desired to enable interoperable data exchange through data marketplace and data management platforms while promoting adoption. We present our experiences from developing a city-scale real-time parking application for a smart city. We also introduce ParkingJSON, a new open standard format for parking data in smart cities, which could help the parking data providers to cover all types of parking infrastructures through a single JSON schema. To the best of our knowledge, this is the first parking data standard proposed that a) covers a wide range of parking spaces and structures, b) integrates spatial information, and c) provides support for data integrity and authenticity.

Type of Paper and Keywords

Regular research paper: *IoT, Parking, Smart City, Interoperability, Data Standard, ParkingJSON, Data Marketplace*

1 Introduction

Data marketplaces and data integration platforms are being considered as a solution for connecting hundreds of IoT devices with the application developers. Intelligent IoT Integrator [17, 12] (I3), Cisco Kinetic[1], Terbine[2], and Streamr[3] are examples of data marketplaces. Such solutions offer a middleware platform to let the device owners share their data with the application developers. This application development model provides a promising solution for building large city-scale IoT applications.

This paper is accepted at the *International Workshop on Very Large Internet of Things (VLIoT 2020)* in conjunction with the VLDB 2020 conference in Tokyo, Japan. The proceedings of VLIoT@VLDB 2020 are published in the Open Journal of Internet of Things (OJIOT) as special issue.

[1] https://www.cisco.com/c/en/us/solutions/index.html
[2] https://www.terbine.io/
[3] https://streamr.network/

Such IoT marketplaces and data management platforms make data available to the application developers from a wide range of IoT deployments, including parking sensors, weather stations, and solar monitoring systems. In this model, the data standard followed by the IoT devices and the device owners may vary depending on the hardware, software, and other business and policy constraints, including privacy. Consider an application developer interested in building a parking monitoring system for a city using a data marketplace. For this application, the application developer has to buy the parking data from multiple data providers, including the transportation department of cities, counties, and towns and various private garage owners. If each parking provider follows a custom and proprietary data format, then the application developers have to convert the data coming from such different sources into a single consistent format to aid their application. Given this friction, the lack of common data standard may even prevent some application developers from buying data from the marketplace, which, in turn, would reduce the revenue for the device and data owners, ultimately affecting the adoption of data marketplaces. These considerations lead to the following requirement:

Data marketplace and data management platforms for smart cities must support data standards for each application to help the device and data owners to easily provide data and the application developers to build applications.

Contemporary literature on data standards and IoT interoperability have articulated the need for a common standard [1, 7, 2]. Many existing efforts in this space either focus on enabling interoperability between messaging protocols such as MQTT [3] and CoAP [1, 7] or emphasize the need for interoperability at networking and MAC layers [2]. Other approaches for addressing data standardization include semantic interoperability [9, 10, 13]. Such methods lead to the standardization of networking and messaging protocols, but the data standardization remains an application-specific problem.

In this work, we consider the interoperability challenges in developing a real-time parking application for a smart-city using a data marketplace. In particular, we illustrate through multiple parking deployment scenarios why a new parking data standard is desired to interconnect heterogeneous and real-time parking feeds to a data marketplace. Based on the review of existing real-world parking feeds, we propose ParkingJSON, a new parking data standard for city-scale IoT applications. To the best of our knowledge, this is the first parking data standard proposed that a) covers a wide range of parking spaces and structures, b) integrates spatial information, and c) provide support for data integrity and authenticity.

The rest of the paper is structured as follows: Section 2 motivates the need for city-wide real-time parking applications involving an IoT data marketplace. The architecture of a marketplace-based parking application and its interoperability challenges are discussed in Section 3. Section 4 introduces ParkingJSON, our newly proposed parking data standard. We provide an example of data following our new standard in Section 5. The evaluation results is presented in Section 6. Finally, Section 7 concludes the paper with pointers to future work.

2 Motivation

2.1 Parking Application

The vehicle population is continuously increasing in metropolitan cities, which also increases the demand for parking spaces [8]. Multiple community members, including the government transportation agency, garage owners, and other private organizations, address the parking demands of the vehicle owners. However, studies suggest that the vehicle drivers are spending tens of hours searching for parking in each year[4]. Another study indicates that searching for parking costs $73 Billion for Americans[5]. Minimizing the searching time has the potential to reduce fuel usage and cost, while immensely reducing the drivers' stress. Gathering real-time parking information from all the parking providers in the city and making that data available to drivers in real-time is essential to enhance the driver's parking experience.

2.2 The role of a Smart City Data Platform

IoT data marketplaces and data management platforms have been developed to let the device and data owners in the community provide their data to the application developers. Examples of IoT data marketplaces include Intelligent IoT Integrator (I3) [12, 17], Terbine.io, and Streamr. Another example of a data management platform that is not based on a marketplace model is Cisco Kinetic. Such platforms enable the cities to develop large-scale city-wide IoT applications by leveraging the data sources provided by the community members. Following a marketplace-based application model, the city administration and the government agencies need not deploy and manage hundreds of IoT devices throughout the city for gathering sensor data. Instead, the community members deploy, manage, and make their IoT devices and their sensor data available

[4] https://www.usatoday.com/story/money/2017/07/12/parking-pain-causes-financial-and-personal-strain/467637001/

[5] https://inrix.com/press-releases/parking-pain-us/

to the city and the other application developers because the data marketplace provides an incentive for the sellers [15].

In the next section, we describe how a real-time parking application can be developed by using a data marketplace. The same concepts can be readily extended to other data management and integration platforms.

3 A Real-time Parking Application Using an IoT Data Platform

The City of Los Angeles' Information Technology Agency (ITA) is considering a real-time data-driven parking application to help the community members make an informed parking decision. A parking application is being developed in collaboration with researchers at the University of Southern California and other government and academic partners. In this section, we will present the architecture of this application.

3.1 Architecture of Data-driven Parking Application

Figure 1 shows the architecture of the real-time parking application that is currently under development at the City of Los Angeles. The key stakeholders and their roles in this application are described below.

3.1.1 Parking Data Providers

These are owners and managers of public and private parking establishments. Most of the parking sites in metropolitan cities have a system in place to gather data about the parking availability in real-time. Currently, the parking information is mainly posted at the entrances of each parking site. However, the parking systems let the garage managers share this information with other entities, including the city administrators. Through an incentive-oriented application development model, we encourage the independent garage owners and managers to share their parking data in real-time with other application developers via the data marketplace [15], as shown in Figure 1. One of the incentives is that the garage owners could attract more vehicles to their parking garages since the application developers would present their parking data through a city-wide unified application to help the vehicle drivers easily find parking spots. Besides, the parking data providers could also charge a small fee to consume their data.

3.1.2 I3 Data Marketplace

I3 [12, 17] is the open-source data marketplace developed at the University of Southern California in collaboration with a number of government, industry,

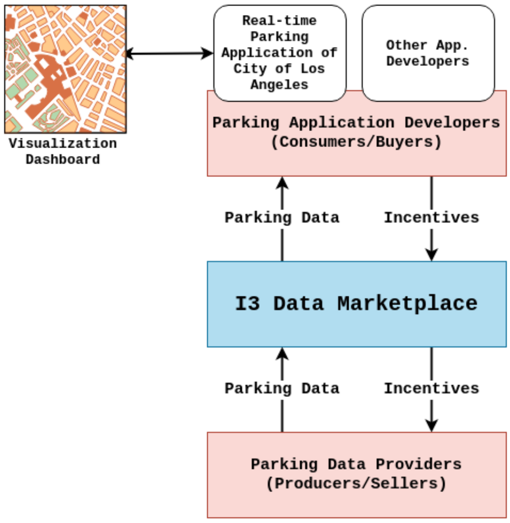

Figure 1: The Architecture of Marketplace-based Parking Application that is Currently Under Development at the City of Los Angeles

and academic partners, including the City of Los Angeles. In Figure 1, I3 marketplace middleware is used to bridge the data providers with the application developers. The key functionalities of the marketplace middleware include user and device management, authentication and access control, ratings, data exchanging protocol, among other things. For more information, we refer the reader to works describing I3 [12, 17]. Besides, open-source software is also made available to the researchers and marketplace enthusiasts at the following link: `https://github.com/ANRGUSC/I3-Core`. For readers interested in understanding the marketplace functionalities through a demonstration, we have released a video here: `https://youtu.be/qFee7mlhriE`.

3.1.3 Parking Application Developers

These are community members, including government agencies, private organizations, and other individuals interested in building an IoT application using the data sources available in the data marketplace. Figure 1 shows how the application developers would receive parking data from the I3 data marketplace by agreeing to data usage polices and providing incentives. There can be tens to hundreds of application developers at the north end of the data marketplace, focusing on various IoT applications. In our parking application use case, the City of Los Angeles creates a real-time parking application using the I3 data marketplace.

Table 1: Examples of Data Formats Used by the Parking Providers

Neighborhood	Parking Type	Fields	Notes
Downtown Los Angeles (Parking Occupancy)	Street Parking	Spaceid, Eventtime, Occupancystate	This API only provides occupancy details for each spaceids. To translate the spaceid into location, another API (see the next row) should be issued.
Downtown Los Angeles (Parking Inventory)	Street Parking	Spaceid, Blockface, Metertype, Ratetype, Raterange, Timelimit, Parkingpolicy, Latlng	This API should be used in combination with the above API.
Los Angeles International Airport (LAX)	Multi-level Parking Structure	Lotdescription, Lot 1 Occupancy, Parkingid, Parkingname, Totalparkingspaces, Occupied, Freespaces, Fullcapacity, Color, Dataexportdatetime, Long (Longitude), Lat (Latitude)	This API provides parking data for all the parking structures at LAX airport. But, it does not provide parking status for each floor (or level).
Parking Deployment at Los Angeles	Multi-level Parking Structure	lotName, lotID, totalSpots, availableSpots, occupiedSpots, percentOccupied, percentAvailable, occupancyLevel, availabilityLevel, parkingPolicies, spotAvailability	This API provides parking availability for each multi-level parking structure. And, it includes a field for parking policies and spot availability. Through spot availability field, this API notifies free parking spots available for handicapped.

Figure 2 shows the visualization dashboard of the parking application that is currently under development. The real-time parking information is available for seven different neighborhoods, including Downtown Los Angeles, parts of Hollywood, Los Angeles International Airport (LAX), and Long Beach. This application shows parking availability for streets and parking garages. The community members can check the parking availability in real-time based on the color-coding. For a particular parking garage or a street, if the number of free parking spots exceeds 50%, then it is denoted in yellow color. Otherwise, the map presents the results in red color. Besides, the users can click each color-coded block to see how many parking spots are currently available, including its total parking capacity. At the time of writing this paper, a demonstration of the parking application was hosted online here: `https://findmeaspot.lacity.org/`.

3.1.4 Interoperability Challenges

Although the architecture of the parking application presented in Figure 1 provides a platform for the City of LA to gather parking feeds from the various neighborhood, there were several interoperability challenges, with respect to the City of LA from taking feeds via the I3 data marketplace platform. In this section, we will describe the interoperability challenges and why a custom application with custom standards is not sustainable and scalable.

Protocol Inconsistency: All the parking data providers in the city use REST-based services for sharing their parking data. Protocols following the request-reply messaging model are not scalable [14] for data marketplaces, which is the reason behind the selection of MQTT, a publish-subscribe system, as a messaging protocol for our I3 data marketplace. However, it is very easy to build and deploy simple gateway scripts that can

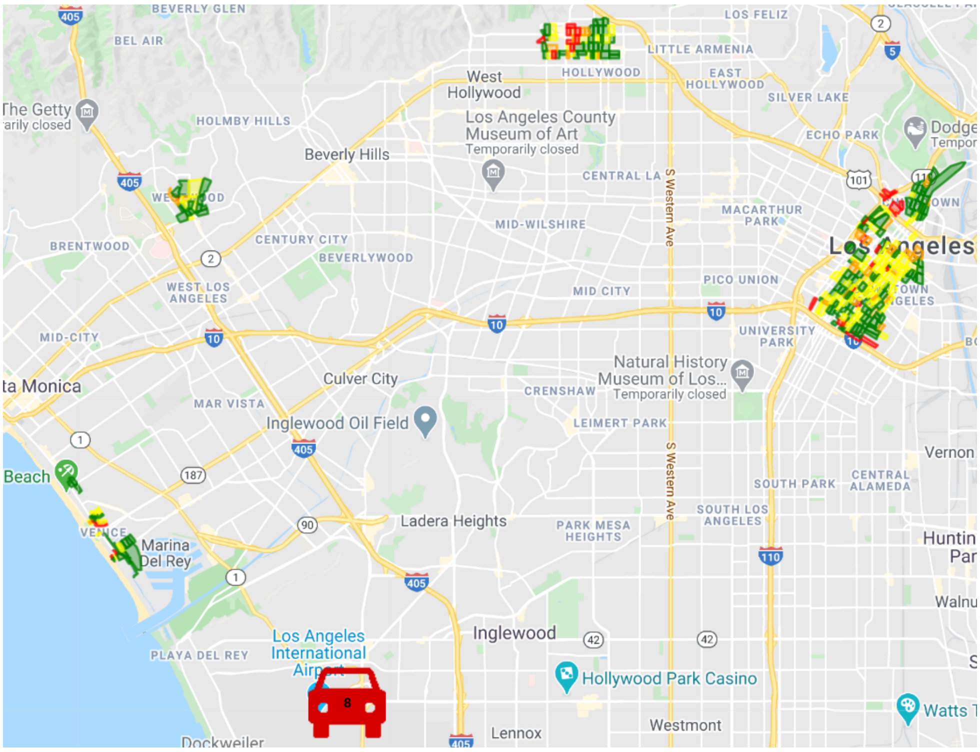

Figure 2: A Visualization Dashboard of Real-time Parking Application developed by the City of Los Angeles

convert between a REST API feed from a parking data provider and an MQTT stream to interface with the data marketplace.

Heterogeneous Data Format: Each parking data provider follows a different data format. Therefore, the application developers cannot consume the data without writing a custom data parser for each parking provider. Table 1 provides examples of parking data formats from real-world deployments.

Continuous Management of Custom Software: To deal with protocol and data format inconsistencies, custom software can be developed. However, such software has to be employed either at the data provider's end or the application developer's end. Running custom software for each data provider at the marketplace middleware is not scalable, and it leads to continuous development and management of software for each protocol and data format, which is not sustainable. Alternatively, the application developers could also deploy custom software as part of their application. Still, this model creates friction and may discourage application developers from adopting the marketplace-based application model.

These challenges show that the parking monitoring infrastructure deployed and managed by various government and private agencies are not interoperable, which hinders their effectiveness and utility. Note that many current parking availability system deployments have limited use because they are typically only used for displaying the parking availability information at the entrances of the parking garages and nearby streets. Therefore, the vehicle drivers are still required to drive close to the digital displays to check parking availability. Creating a city-wide real-time parking application accessible through a mobile interface, therefore, offer a promising alternative, but it requires a set of common standards.

We review the parking data standards followed by real-world parking deployments and propose a new parking data standard, ParkingJSON, to mitigate the data interoperability challenges at data marketplaces and other large-scale city-wide parking applications.

3.2 Design Requirements

Table 1 shows the different parking data formats currently employed by parking management systems. In particular, the table highlights the following issues:

- **Multiple Parking Types:** Multiple parking modalities are presented in a city. These range from street-side parking, single-level parking garages, to multi-level parking structures.

- **Diverse Reporting Model for Parking Availability:** Some parking establishments provide individual spot status using a Boolean variable that switches between *occupied* and *vacant*. In the case of multi-level parking structures, the information about the *total spots* and *total availability* is reported, and some deployments could also provide parking availability at each level.

- **Inconsistent Metadata:** Building a map-based visualization or to understand the freshness of the data, it is important to receive information including the GPS coordinates and timestamp associated with the last update. In the case of street-side parking, the app. developers may also need information about specific parking spots. Some parking deployments provide both the metadata and the parking availability information through a single API. But, in some cases, multiple APIs are required to interpret the parking data for a particular location. As shown in Table 1, the parking availability data is reported using spaceids for Downtown LA neighbourhood, and the application developer is required to issue an additional API to gather location information associated with each spaceid, when processing and plotting the data on a map.

These issues must be taken into account when creating a new parking data standard. We list down the requirements for a parking data standard below:

- **R1** The parking data format should flexibly cover a wide range of parking infrastructures.

- **R2** All the relevant metadata should be part of the parking data standard, and it should not require multiple data queries or messages to interpret the parking data.

- **R3** The spatial data should be embedded within the data standard to help the application developers create map-based visualizations.

- **R4** The data should follow a programmer-friendly data schema.

- **R5** Some mechanisms should be added to ensure integrity and the authenticity of the data.

Related Work on Parking Data Standard: To the best of our knowledge, the topic of *parking data standard* has not been discussed in the literature. Existing literature describes how IoT and wireless sensor networks can be used to create smart parking applications [16, 4]. The *Alliance for Parking Data Standards*[6] is looking into standardizing the parking data through a consortium of government and industrial partners. But, there is no open-source parking data standard available to help the parking data providers and application developers.

4 PARKINGJSON: AN OPEN STANDARD FORMAT FOR PARKING DATA IN SMART CITIES

We propose ParkingJSON, a new parking data standard satisfying the requirements elucidated in Section 3.2. The key features of ParkingJSON are discussed below.

4.1 Capturing Spatial Relationships through a Hierarchical Layering Schema

Figure 3 shows the hierarchical layering scheme followed by our parking data standard, ParkingJSON.

Within a city, there are multiple areas or neighborhoods. For example, the Los Angeles International Airport (LAX), Downtown Los Angeles, and an university campus are considered as areas. In our parking data standard, an *area* covers one particular neighborhood. Formally, we can denote the area as $\mathcal{A}$. Within a city, there can be $\ltimes$ areas, which can be represented as $a_1, a_2, ..., a_{\ltimes}$.

Within each area, a_i, there could be multiple parking lots—for example, Lot 6 within LAX airport area or City Center Parking in the Downtown Los Angeles area. In our parking data standard, a *lot* covers a particular parking structure, which may have tens to hundreds of parking spaces across one or more levels (or floors). Formally, we can represent lots as $\mathcal{L}$. Within each area, a_i, there can be $\ltimes$ parking lots, which can be represented as $l_1, l_2, ..., l_{\ltimes}$.

Within each lot, l_i, there could be multiple sections (or floors). For example, section 2 (denotes level 2 or floor 2) of Lot 6 within LAX airport area or section 5 (denotes level 5 or floor 5) of City Center Parking in Downtown Los Angeles area. In our parking data standard, a *section* covers a particular segment or a level of a parking lot, wherein each section may have tens of parking spaces. Formally, we can represent sections as $\mathcal{S}$. Within each lot, l_i, there can be $\ltimes$ sections, which can be represented as $s_1, s_2, ..., s_{\ltimes}$.

Within each section, s_i, there could be multiple individual parking spots. For example, the parking spot 12 in section 2 (denotes level 2 or floor 2) of Lot 6 within LAX airport area or the parking spot 28 in section 5 (denotes level 5 or floor 5) of City Center Parking

[6] https://www.allianceforparkingdatastandards.org/

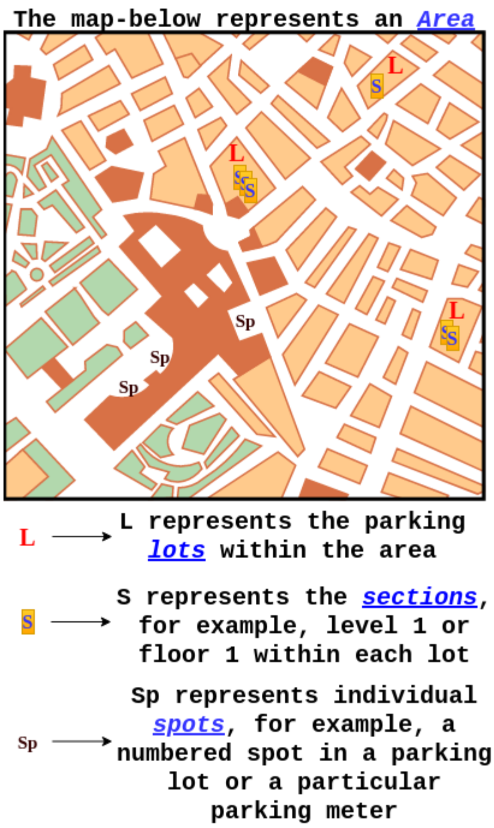

Figure 3: Hierarchical Layers followed by ParkingJSON, Our Parking Data Standard

in Downtown Los Angeles area. In our parking data standard, a *spot* refers to the individual parking spot, which is the lowest granularity level. Formally, we can represent spots as $\mathcal{P}$. Within each section, s_i, there can be $\ltimes$ individual spots, which can be represented as $p_1, p_2, ..., p_{\ltimes}$.

How does ParkingJSON cover the areas with only street-side parking or a single-level parking structure? Although our hierarchical parking data standard covers different types of parking establishments, not all areas in a city may have multi-level parking lots with multiple sections. Some areas may only have street-side parking or a single-level parking structure. In our proposed parking data standard, each higher level of the hierarchy could either be stand-alone or be itself a collection of lower levels. The following variations are valid data formats in our standard:

- An Area can be a collection of lots (e.g., LAX) or collection of spots (Downtown LA), or it may be just a stand-alone unit (no further subdivision).
- A lot can be a collection of sections, or collection of spots, or stand-alone.
- A section can be a collection of spots or stand-alone.
- A spot is always stand-alone.

We further explain the above variations with a practical example in Section 5, and more examples are available at the following GitHub repository: `https://github.com/ANRGUSC/ParkingJSON`.

4.2 Attributes

In this section, we present the data attributes of our parking data standard, ParkingJSON:

Area-specific Attributes: Table 2 presents the area-specific attributes used in ParkingJSON standard. The key represents the fields that should be included in the area segment of the data, and the attribute values and the data types for each attribute are also presented to help the parking application developers. Most of the fields are self-explanatory, except for AreaGeometry, which captures the shape of the area. For each area, we could draw a polygon or other geometric shapes by tracing a set of GPS coordinates. For example, a square-shaped area can be represented with four GPS coordinates, which is shown in Figure 5.

Lot-specific Attributes: The lot-specific attributes are similar to the area-specific attributes, in terms of the keys and values. It contains the following fields: Type: Lot, OwnerInfo, LotID, LotName, LotLatLong, LotGeometry, TimeStamp, TotalSpots, and OccupiedSpots. A table is not created for lot-specific attributes to avoid redundancy. Except for the *Type* field, all the other items are similar to area-specific attributes.

Section-specific Attributes: The section-specific attributes also follow a similar pattern. It contains the following fields: Type: Section, OwnerInfo, SectionID, SectionName SectionLatLong, SectionGeometry, TimeStamp, TotalSpots, and OccupiedSpots. Here, the *Type* field should contain the value "Lot".

Spot-specific Attributes: The spot-specific attributes have the following fields: Type: Spot, OwnerInfo, SpotID, SpotName, SpotLatLong, SpotGeometry, TimeStamp IsOccupied (True or False), SpotPolicy. Unlike other segments in the data format, the spot segment maintains a *Boolean* attribute called "IsOccupied", which is used to identify the status of a single parking spot. Additionally, there is also an attribute called "SpotPolicy", which is introduced to specify options such as Unrestricted, HandicapOnly or PermitOnly, and it could be extended further if needed.

Table 2: Area-specific Attributes

Attribute Key	**Attribute Value**	**Data Type for Attribute Value**
Area-specific Attributes		
Type	Area	String
OwnerInfo	Parking provider info	String
AreaID	Alphanumeric identifier	String
AreaName	Name of the area	String
AreaLatLong	Latitude and Longitude	Key-value Store
AreaGeometry	Spatial coordinates	Key-value Store
Timestamp	Last update timestamp	String (YYYY-MM-DDTHH:MM:SS.SSS)
TotalSpots	Total number of spots in the area	Integer
OccupiedSpots	Total number of occupied spots	Integer

All these attributes are maintained in a JSON document [5]. We chose JSON because it is easy to handle JSON documents in popular programming languages such as Python, Java, and JavaScript. Figure 4 provides a high-level overview of ParkingJSON schema.

4.3 Specifications for Geometry Attribute

Figure 5 shows the difference between *Point* and *Polygon* shapes. This attribute follows the *GeoJSON* data format [6]. We will show an example of Geometry attribute in Section 5.

4.4 Parking Payment Policies

An optional attribute is introduced to represent the parking payment policies, which is shown in Table 3. A “PolicyType” field is added to identify the pricing model for a given parking segment (area, lot, section, or spot). And, the timing field is used to represent the timing limitations, while the support for prepaid payment is expressed through a Boolean value. Both the *PolicyType* and *Timing* fields can be extended with custom attributes based on the payment modalities supported by the parking providers.

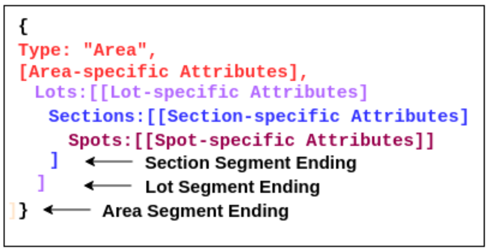

Figure 4: The High-level JSON Schema of ParkingJSON

4.5 Parking Reservation Policies

Some parking vendors may also provide support for reservation of parking spaces. We include an optional attribute in ParkingJSON to handle such circumstances, which is shown in Table 4. Note that the details of future parking availability and how to make reservations are relegated to the URL and not specified in this standard.

4.6 Security and Integrity of Parking Data

To gain a guarantee that the data comes from a valid source (source authentication) and has not been tampered with (integrity), a digital signature could be included in the JSON. The whole ParkingJSON file could be sent as the payload of a JWT [11] (Java Web Token), which adds a header and signature field (using JWS (Java Web Signature)). The header can include the certificate and cryptographic algorithms used for the signature.

4.7 Requirements Evaluation

Section 3.2 elicited the design requirements for a parking data standard. Here, we show how our proposed parking data standard, ParkingJSON, satisfies those requirements:

- **R1** is satisfied through the use of hierarchical layering schema, which covers from an entire area to an individual parking spot.
- Our parking data standard includes name, identifier, GPS coordinates, geometry, and a timestamp for the area, lots, sections, and spots. Therefore, an application developer can process the parking information using a single data feed, which fulfills **R2**.
- When visualizing parking data on a map, it is important to process information about the

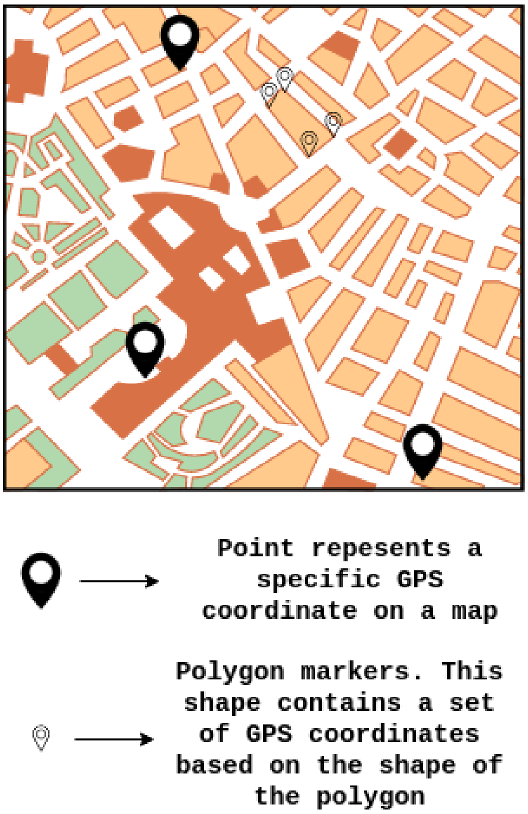

Figure 5: Geometry Attribute: Point vs Polygon

geometrical shape. Our parking data standard includes geometry for the area, lots, sections, and spots, which satisfies **R3**.

- Through the use of the JSON document, we satisfy requirement **R4**.
- ParkingJSON files could be secured through the use of JWT tokens, which meets **R5**.

4.8 Expanding this Standard to Other Applications beyond Parking

We understand that the proposed ParkingJSON standard is heavily tailored for parking applications. However, we believe that this standard can be used for other applications, including environmental monitoring (e.g., temperature, humidity, and wind speed), air quality sensing, and traffic management, by replacing the parking specific segments with new segments particular to each application, building on the hierarchical geospatial nature of this data standard. We plan to explore this in our future work.

Table 3: Optional Attribute for Payment Policies

Attribute Key	**Attribute Value**	**Data Type for Attribute Value**
Key: PaymentPolicy, Values are as follows:		
PolicyType	Free, FlatRate, UnitRate, TwoPhaseFlat (i.e. initially free for some time, then flat fee), TwoPhaseUnitRate (i.e. initially free for some time, then charged by the minute/hour)	String
Timing	MaxTime, TimeUnit (for UnitRate), FlatRatePrice or RatePerTime, InitialTime (for TwoPhase)	String
IsPrepaid	True or False	Boolean

Table 4: Optional Attribute for Reservation Policies

Attribute Key	**Attribute Value**	**Data Type for Attribute Value**
Key: ReservationPolicy, Values are as follows:		
IsReservable	True or False	Boolean
Max Reservation Time	Time in Minutes (How far in advance it can be reserved?)	Integer
ReservationURL	URL	String

5 A Practical Example

Appendix A shows an example of the ParkingJSON document for one of the USC campuses. Due to the space limitation, we represent partial data for two parking lots. But, a complete JSON document for this location would have multiple parking lots with tens of sections and hundreds of individual spots. Besides, it is important to note how the Geometry attribute is used to carry the spatial coordinates that are associated with the parking data. In this case, the areaGeometry attribute can be used to draw a polygon on the map to denote the area. To avoid redundancy, we have not included *Geometry* for all the data segments. We refer the reader to our GitHub repository for more real-world examples: `https://github.com/ANRGUSC/ParkingJSON`.

6 Evaluation

In this section, we evaluate how our parking data standard, ParkingJSON, influences the size. Table 5 shows the storage and communication costs.

To understand the storage cost, we measured the file sizes using Linux's *ls -lhS* command. This metric can be used to identify the storage requirements at the data provider's and the application developer's infrastructures. Although the storage overhead is in the order of *kilobytes*, the applications employing resource-constrained embedded devices may have to provide sufficient memory for storage based on the number of lots, sections, and spots handled by their application.

The serialization overhead is another essential metric since this determines the number of bytes that gets transmitted from the data provider to the marketplace middleware and from marketplace middleware to the application developer. From the data provider's perspective, this metric can be used to identify the bandwidth requirement for his/her deployment. Note that the file size is not the direct indicator of the bytes transmitted on the network. Contemporary programming languages such as Python, Java, and JavaScript provide a serialization library to encode the data into a transferable format. To understand the payload size after the serialization, we have used Python's built-in serializer, which is part of the *Json* package. Besides, we have used the SDK (`https://github.com/ANRGUSC/I3-SDK`) provided by the I3 data marketplace to publish data to an I3 marketplace instance and measured the payload sizes at the MQTT broker that is part of I3-v1 [17] middleware. Our evaluation shows that serialization reduces the payload size by approximately 50% when compared against the storage sizes, which is because of both some redundancy in the base format and the efficient "base64" encoding scheme employed by Python's JSON serializer.

The data for Downtown Los Angeles in their current custom parking format requires two data streams for processing the data; one stream provides the current parking status, while the other flow informs the metadata associated with each parking spot in the Downtown area. Unlike this, our parking data standard, ParkingJSON, uses a single stream to provide all the necessary details.

It is important to note that the benefits of interoperability provided by ParkingJSON come at a slight cost. Each parking segment for a lot, section, or spot adds approximately 1.2 kilobytes to the storage and 500 bytes to the serialized payload. We believe that this is an acceptable trade-off for improving the interoperability using such a standard.

Table 5: The Storage and Communication costs of ParkingJSON compared against the Custom Standards followed by Downtown Los Angeles and LAX Airport Deployments

Scenario	Size in KB	Serialized Payload Size in KB
ParkingJSONwith 1Area	1.2	0.5
ParkingJSONwith 1area1lot	2.3	1.0
ParkingJSONwith 1area1lot 1section	3.5	1.5
ParkingJSONwith 1area1lot 1section1spot	4.6	2.0
ParkingJSONfor DowntownLA with1area 3spots	4.2	1.9
ParkingJSONforLAX with1area 7lots	9.2	3.9
Parking Data Without Our Standard		
LAX currentdata standard	2.0	2.1
Downtown currentdata standard	0.26	0.27
Downtown inventoryfor currentdata standard	1.3	1.3

All the ParkingJSON files that were used for the evaluation are made available at the following GitHub repository: `https://github.com/ANRGUSC/ParkingJSON`.

7 Conclusion and Future Work

In this work, we have presented our experiences from developing a city-wide real-time parking application for the City of Los Angeles, involving I3, which is an open-source data marketplace developed at the University of Southern California. In particular, we have highlighted how the interoperability challenges could prevent the parking data providers and the application developers from adopting a marketplace-based application model. To enhance adoption, we have proposed a new parking data standard, ParkingJSON, which covers the many different types of parking infrastructures encountered in a city. We have provided an example parking data along with evaluation results highlighting the storage and communication costs of our parking data standard. To the best of our knowledge, this is the first parking data standard proposed that a) covers a wide range of parking spaces and structures, b) integrates spatial information, and c) provide support for data integrity and authenticity.

In our future work, we will work with the local parking data providers in the city of Los Angeles to convert their parking data format to ParkingJSON data standard. Subsequently, we plan to work with the City of Los Angeles to enhance their *findmeaspot*[7] parking application and identify the effectiveness of our proposed data standard in realistic settings. Lastly, we plan to investigate approaches to optimize the data format to reduce the payload size.

Acknowledgements

This work is supported by the USC Viterbi Center for Cyber-Physical Systems and the Internet of Things (CCI).

References

[1] B. Ahlgren, M. Hidell, and E. H. Ngai, “Internet of things for smart cities: Interoperability and open data,” *IEEE Internet Computing*, vol. 20, no. 06, pp. 52–56, nov 2016.

[2] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, “A mobile multi-technology gateway to enable iot interoperability,” in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, 2016, pp. 259–264.

[3] A. Banks and R. Gupta, “MQTT version 3.1.1,” 2014.

[4] R. E. Barone, T. Giuffrè, S. M. Siniscalchi, M. A. Morgano, and G. Tesoriere, “Architecture for parking management in smart cities,” *IET Intelligent Transport Systems*, vol. 8, no. 5, pp. 445–452, 2013.

[5] P. Bryan and M. Nottingham, “JavaScript object notation (JSON) patch,” Internet Requests for Comments, April 2013. [Online]. Available: https://tools.ietf.org/html/rfc6902

[6] H. Butler, M. Daly, A. Doyle, S. Gillies, S. Hagen, and T. Schaub, “The GeoJSON format,” Internet Requests for Comments, August 2016. [Online]. Available: https://tools.ietf.org/html/rfc7946

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

[8] N. Fulman and I. Benenson, “Establishing heterogeneous parking prices for uniform parking availability for autonomous and human-driven vehicles,” *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 1, pp. 15–28, 2019.

[9] M. Ganzha, M. Paprzycki, W. Pawłowski, P. Szmeja, and K. Wasielewska, “Semantic interoperability in the internet of things: An overview from the inter-iot perspective,” *Journal of Network and Computer Applications*, vol. 81, pp. 111 – 124, 2017.

[10] G. M. Honti and J. Abonyi, “A review of semantic sensor technologies in internet of things architectures,” *Complexity*, vol. 2019, 2019.

[11] M. Jones, J. Bradley, N. Sakimura, and Token, “JSON web token,” Internet Requests for Comments, May 2015. [Online]. Available: https://tools.ietf.org/html/rfc7519

[12] B. Krishnamachari, J. Power, S. H. Kim, and C. Shahabi, “I3: an IoT marketplace for smart communities,” in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*. ACM, 2018, pp. 498–499.

[13] M. Noura, M. Atiquzzaman, and M. Gaedke, “Interoperability in internet of things: Taxonomies and open challenges,” *Mobile Networks and Applications*, vol. 24, no. 3, pp. 796–809, 2019.

[14] P. R. Pietzuch and J. M. Bacon, “Hermes: a distributed event-based middleware architecture,” in *Proceedings 22nd International Conference on Distributed Computing Systems Workshops*, 2002, pp. 611–618.

[15] G. S. Ramachandran and B. Krishnamachari, “Towards a large scale iot through partnership, incentive, and services: A vision, architecture, and future directions,” *Open Journal of Internet Of Things (OJIOT)*, vol. 5, no. 1, pp. 80–92, 2019. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:101:1-2019092919345869785889

[7] https://findmeaspot.lacity.org/

[16] M. M. Rathore, A. Ahmad, A. Paul, and S. Rho, "Urban planning and building smart cities based on the internet of things using big data analytics," *Computer Networks*, vol. 101, pp. 63–80, 2016.

[17] X. Zhao, K. Karyakulam Sajan, G. Ramachandran, and B. Krishnamachari, "Demo abstract: The intelligent iot integrator data marketplace — version 1," in *Proceedings of the 5th ACM/IEEE Conference on Internet of Things Design and Implementation*, ser. IoTDI '20, 2020.

APPENDIX A

An example ParkingJSON document for one of the USC campuses with two parking lots is presented below.


```
{
  "Type": "Area",
  "Attributes":[
  {"OwnerInfo":"USC"},
  {"AreaID":"usc5428"},
  {"AreaName":"USC UPC Campus"},
  {"AreaLatLong":[-118.39,33.94]},
  {"AreaGeometry":{
                "type": "Polygon",
                "coordinates": [
                    [
                        [
                            -118.39,
                            33.94
                        ],
                        [
                            -118.40,
                            33.94
                        ],
                        [
                            -118.40,
                            33.94
                        ],
                        [
                            -118.39,
                            33.94
                        ],
                        [
                            -118.39,
                            33.94
                        ]
                    ]
                ]
  }
  },
  {"Timestamp":"2019-12-07T21:22:48.1
     20"},
  {"TotalSpots":2023},
  {"OccupiedSpots":1949}
  ],
  "Lots":[
  {
    "Type":"Lot",
    "OwnerInfo":"USC",
    "LotID":"Lot1",
    "LotName":"LotDowney",
    "LotLatLong":[-118.39,33.94],
    "LotGeometry":{
      "Notes": "Not shown in this
         example due to space
         restrictions. But it will
         follow the format presented
         in previous area, lot, and
         section segments"
         },
    "Timestamp":"2019-12-07T21:22:48.
       120",
    "TotalSpots":455,
    "OccupiedSpots":"324",
    "Sections":[
      {
      "Type":"Section",
      "OwnerInfo":"USC",
      "SectionID":"usclot1floor1",
      "SectionName":"698floor1",
      "SectionLatLong":[-118.39,33.94
         ],
      "SectionGeometry":{
                "Notes": "Not shown
                   in this example
                   due to space
                   restrictions. But
                   it will follow the
                    format presented
                   in previous area,
                   lot, and section
                   segments"
         },
    "Timestamp":"2019-12-07T21:22:48.
       120",
    "TotalSpots":102,
    "OccupiedSpots":78
    }
    ]
    },
    {
      "Type":"Lot",
      "OwnerInfo":"USC",
      "LotID":"Lot2",
      "LotName":"LotShrine",
      "LotLatLong":[-118.39,33.94],
      "LotGeometry":{
```

```
                "Notes": "Not shown
                    in this example
                    due to space
                    restrictions. But
                    it will follow the
                     format presented
                    in previous area,
                    lot, and section
                    segments"
            },
    "Timestamp":"2019-12-07T21:22:48.
        120",
    "TotalSpots":655,
    "OccupiedSpots":"344",
    "Sections":[
      {
      "Type":"Section",
      "OwnerInfo":"USC",
      "SectionID":"usclot2floor1",
      "SectionName":"438floor1",
      "SectionLatLong":[-118.39,33.94
          ],
      "SectionGeometry":{
                "Notes": "Not shown
                    in this example
                    due to space
                    restrictions. But
                    it will follow the
                     format presented
                    in previous area,
                    lot, and section
                    segments"
            },
    "Timestamp":"2019-12-07T21:22:48.
        120",
    "TotalSpots":148,
    "OccupiedSpots":78,
    "Spots":[
        {
          "Type":"Spot",
          "OwnerInfo":"LADOT",
          "SpotID":"defgh456",
          "SpotName":"DT124",
          "SpotLatLong":[-118.39,33.94],
          "SpotGeometry":{
                    "Notes": "Not shown
                        in this example
                        due to space
                        restrictions. But
                        it will follow the
                         format presented
                        in previous area,
                        lot, and section
                        segments"
                },
        "Timestamp":"2019-12-07T21:22:48.
            120",
        "IsOccupied":"True",
        "SpotPolicy":"HandcicapOnly"
        }
        ]
        }
        ]
        }
        ]
}
```


Example 1: A ParkingJSON Document for USC Campus with two parking lots. Each parking lot has one section, wherein one of the parking lots include a single spot-specific data. This example is created to show how the area, lots, sections, and spots would be represented following our parking format.

We encourage the readers to review the other examples provided in the following GitHub repository: `https://github.com/ANRGUSC/ParkingJSON`.

Author Biographies

Gowri Ramachandran is a senior postdoctoral research associate at the Center for Cyber Physical Systems and the Internet-of Things (CCI) at the University of Southern California. He received his Ph.D. from imec-DistriNet, KU Leuven, Belgium. His research interests include Internet-of-Things (IoT), smart cities, and blockchain.

Jeremy Stout is currently a Programmer/Analyst with the City of Los Angeles Information Technology Agency (ITA). He develops and maintains/supports applications used by City Departments and the general public. His interests include learning, tinkering, and applying new/emerging technology to cultivate in a digital age.

Joyce Edson is currently the Executive Officer and Deputy CIO with the City of L.A.'s Information Technology Agency (ITA). She provides City Departments with Application development and consultation for digital solutions. Her interest is in the application of new/emerging technology to effectively, efficiently and sustainably improve the government operations, and improve the quality of life for the public government serves.

Bhaskar Krishnamachari received his Ph.D. degree from Cornell University, Ithaca, NY, USA, in 2002. He is currently a Professor with the Department of Electrical and Computer Engineering, and Director of the Center for Cyber-Physical Systems and the Internet of Things. His research interests include the design and analysis of algorithms and protocols for the Internet of Things, Wireless Networks, and Distributed Systems.