Dynamic Geospatial Data Integration: A Case Study of Moving Objects in Munakata City, Japan Using OGC API Moving Features and Sensorthings API

Thunyathep Santhanavanich[1,2] *, Rushikesh Padsala[1,3], Matthias Betz[1], Volker Coors[1]

[1] Center for Geodesy and Geoinformatics, Stuttgart University of Applied Sciences
Schellingstrasse 24, 70174 Stuttgart, Germany
(thunyathep.santhanavanich, rushikesh.padsala, matthias.betz@hft-stuttgart.de, volker.coors)@hft-stuttgart.de

[2] Faculty of Environmental Sciences Technical University Dresden, 01062 Dresden, Germany

[3] Department of Building, Civil, and Environmental Engineering, Concordia University
1515 St. Catherine St. West Montreal, QC, H3G 2W1 Canada

KEY WORDS: Open Geospatial Consortium, Moving Features, SensorThings API, Intelligent Transportation Systems, Smart Cities

ABSTRACT:

The effective tracking and analysis of moving objects within urban environments presents a complex challenge that necessitates robust geospatial data integration. Open Geospatial Consortium (OGC) APIs offer standardized approaches to managing dynamic geospatial information. This paper presents a case study of real-time moving object tracking including buses and trains in the city of Munakata, Japan, utilizing two prominent OGC APIs: OGC API Moving Features and OGC SensorThings API. The study explores the implementation of both APIs, examining their strengths and limitations in handling real-time location updates and associated sensor data generated by moving buses. The research provides insights into the practical suitability of each API model for dynamic object tracking, offering valuable guidance for practitioners seeking to optimize geospatial data integration within smart cities and intelligent transportation systems.

1. INTRODUCTION

The emergence of smart cities and intelligent transportation systems (ITS) underscores the critical need for efficient management and analysis of dynamic geospatial data in tracking moving objects such as vehicles and pedestrians within urban environments. This demand entails real-time location updates alongside the integration of various sensor data to fully harness the capabilities of applications designed for disaster management, traffic services, security, navigation (robotic, aviation, maritime), and wildlife conservation (Richter et al., 2020, Asahara et al., 2015a). Acknowledging these requirements, the Open Geospatial Consortium (OGC), a leading entity in developing open geospatial standards, acknowledged the need to formulate open standards for incorporating dynamic geo-datasets and web-based frameworks, thus enhancing the precision of spatial decision-making procedures. Consequently, for a long time, OGC has provided a standardized and refined framework for a variety of web services, including the Catalogue Service, Web Feature Service (WFS), Web Coverage Service (WCS), Web Mapping Service (WMS), Web Processing Service (WPS) (OGC, 2015) and the OGC Sensor Web Enablement initiative (SWE) (Reed et al., 2013). In 2017, the OGC released the OGC API Whitepaper (Percivall et al., 2017), outlining the need for APIs to be more web-oriented and accessible to non-experts. A new paradigm emerged, emphasizing web-centricity, developer-friendliness, lightweight specification development, and a shift from service-oriented to resource-oriented models. This transition involved modular specification development, including core and extension specifications, facilitating easy implementation and adoption for the mass market and the web, marking the inception of the new generation of OGC API. The new generation of OGC APIs are crafted to facilitate the effortless provision and utilization of geospatial data on the web, enabling seamless integration with diverse types of information. These standards extend the foundation laid by the OGC Web Service Standards (such as WMS, WFS, WCS, WPS, etc.), yet diverge by establishing resource-centric APIs that leverage contemporary web development methodologies. Instead of adhering to traditional service-oriented approaches, these standards embrace a modular ”building blocks” framework, allowing for the creation of innovative APIs tailored for web-based access to geospatial content. The objective is to streamline accessibility and foster adaptability in the utilization of geospatial data across a wide spectrum of applications and contexts. OGC APIs are evolving rapidly over time, and numerous OGC APIs are already under development for example Styles, Maps, Routes, Joins, 3D GeoVolumes, Processes, Features, Environmental Data Retrieval and many others.

This paper focuses on the dynamic geospatial data integration utilizing two such OGC API standards – OGC API Moving Features and OGC SensorThings API – to explore their suitability for tracking moving features: buses in the city of Fukuoka, Japan. A comparative analysis of the implementation process will highlight the advantages and potential drawbacks of each API for handling dynamic geospatial data. The ultimate goal of this paper is to exploit and optimize the potential of these next-generation OGC APIs, by facilitating the integration of moving feature data from a multitude of sources, promoting data exchange initiatives that are essential for broadening the market and enhancing the applications of geospatial information and analysis extensively.

The rest of this paper is organized as follows. Section 2 provides an overview of the research background, focusing on the two

* Corresponding author

OGC API standards. Section 3 details the implementation of both standards for our specific use cases in the city of Munakata, Japan. Subsequently, in section 4, a discussion on the implications and findings of the implemented standards will be presented. Finally, section 5 summarizes the research outcomes and provides insights for future research directions.

2. BACKGROUND

2.1 OGC API Moving Features

The OGC Moving Features standards (Hayashi et al., 2017) offers a rigorous framework for encoding, accessing, and managing dynamic geospatial data with a focus on moving objects. Moving features are defined by their time-dependent geometries and potentially time-varying non-spatial attributes. This standard supports point (0-dimensional), linear (1-dimensional), areal (2-dimensional), and volumetric (3-dimensional) representations, enabling the modeling of object trajectories and interpolated positions. Researchers (Asahara et al., 2015a) also found that the OGC Moving Features model can be effectively used to integrate many types of location data and applications. It facilitates sophisticated analysis of movement patterns, with the flexibility to incorporate both location- and time-based queries.

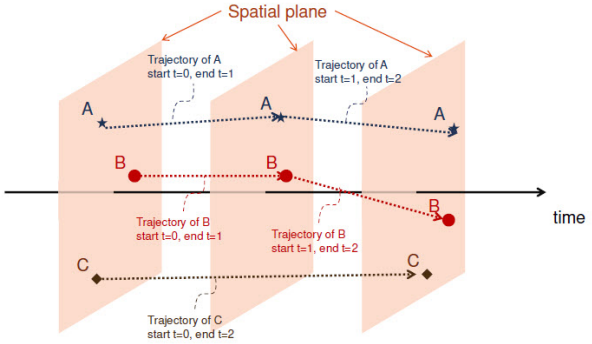

Figure 1. OGC Moving Features - Foliation model describing trajectories of moving objects (Asahara et al., 2015a).

The foliation data model in the OGC Moving Features has been illustrated in Figure 1, derived from ISO19141:2008. It depicts the trajectories of three points (A, B, and C) over time, where the horizontal axis represents time and three spatial planes capture temporal snapshots. Each trajectory links two such snapshots and is defined by a start and an end time. Initially, at t=0, points A and C begin their movements, while B remains stationary. By t=1, A alters its course, and B commences its movement. The model records the trajectories of A and B across different intervals, and C's continuous movement until t=2, including instances where no movement occurs, like B's initial stationarity. This approach ensures all state changes, including position, velocity, and other attributes, are captured and sequenced by time, allowing for partial data loading while still understanding all feature states up to that point.

The OGC Moving Features API built upon a foundational data model, offers various encoding mechanisms for their representation. These include XML/GML (Asahara et al., 2019) and JSON (Kim and Ishimaru, 2020) for comprehensive encoding, and simpler CSV (Asahara et al., 2015b). While these encodings share the same standard and data model, they exhibit notable discrepancies that significantly impact the types of information that can be modeled. Each of these formats has its unique strengths and limitations. First, CSV encoding is beneficial for its support of temporal gaps in observations. However, it is verbose, with redundant information such as start and end time and location for each segment. It only supports linear interpolation and handles moving points exclusively. Furthermore, its usability in GIS is limited due to the missed opportunity to use WKT for geometry representation. Next, XML Encoding shares CSV's advantage of supporting temporal gaps in observations and extends this by handling complex geometries. It also supports static or non-temporal descriptive object properties. Despite these advantages, XML encoding is verbose, similar to CSV, and requires synchronization between temporal geometry and attributes. It also applies the same interpolation for geometry and all attributes. Finally, the JSON encoding provides the most compact representation and can handle complex geometries. It models timestamps of location and attribute changes independently, eliminating the need for synchronization. It allows for individual specification of interpolation modes for each attribute and supports non-temporal descriptive attributes. However, JSON encoding presents multiple options for encoding the same situation, leading to potential confusion regarding the bounds of time periods. Additionally, it does not support temporal gaps in observations which could be overcome and handled directly at the application level. Overall, while each encoding format has its advantages, it also comes with inherent limitations. The choice of encoding format should be guided by the specific requirements of the task at hand. In this research, we will be focusing on using the OGC Moving Features API in JSON encoding which is the most light-weight format and has the fewest limitations.

Regarding the open-source implementation of the OGC Moving Feature, MobilityDB[1] is an open-source database management system that extends PostgreSQL and its spatial extension, PostGIS, to manage and analyze temporal and spatiotemporal objects. It is designed to efficiently process complex queries on moving object trajectories, providing significant functionalities for OGC Moving Features. In the past, it has been used in various fields including transportation, urban planning, environmental monitoring, and location-based services (Sakr et al., 2023, Brandoli et al., 2022, Godfrid et al., 2022).

2.2 OGC SensorThings API

OGC SensorThings API provides a unified, geospatial-enabled way to interconnect Internet of Things (IoT) devices, data, and applications over the HTTP protocol (Liang et al., 2016). The SensorThings API is emerging as a pivotal integrator between urban infrastructures and pervasive sensor networks (Kotsev et al., 2018). This API architecture is based on the foundation of the RESTful principles, aiming to foster interoperability among Internet of Things (IoT) devices and geospatial data platforms, ergo catalyzing the amalgamation of IoT with smart city frameworks (Liang et al., 2016). Central to its utility in smart urban ecosystems is its capability for standardizing data management, facilitating the seamless interplay of geospatial information, enabling real-time data processing, and bolstering urban planning and management processes. The API's architectural design facilitates the dynamic linkage of a diverse array of sensor data with urban data systems, proving instrumental in a spectrum of municipal functionalities including, but not limited to, environmental surveillance, traffic orchestration, and public welfare

[1] https://mobilitydb.com/

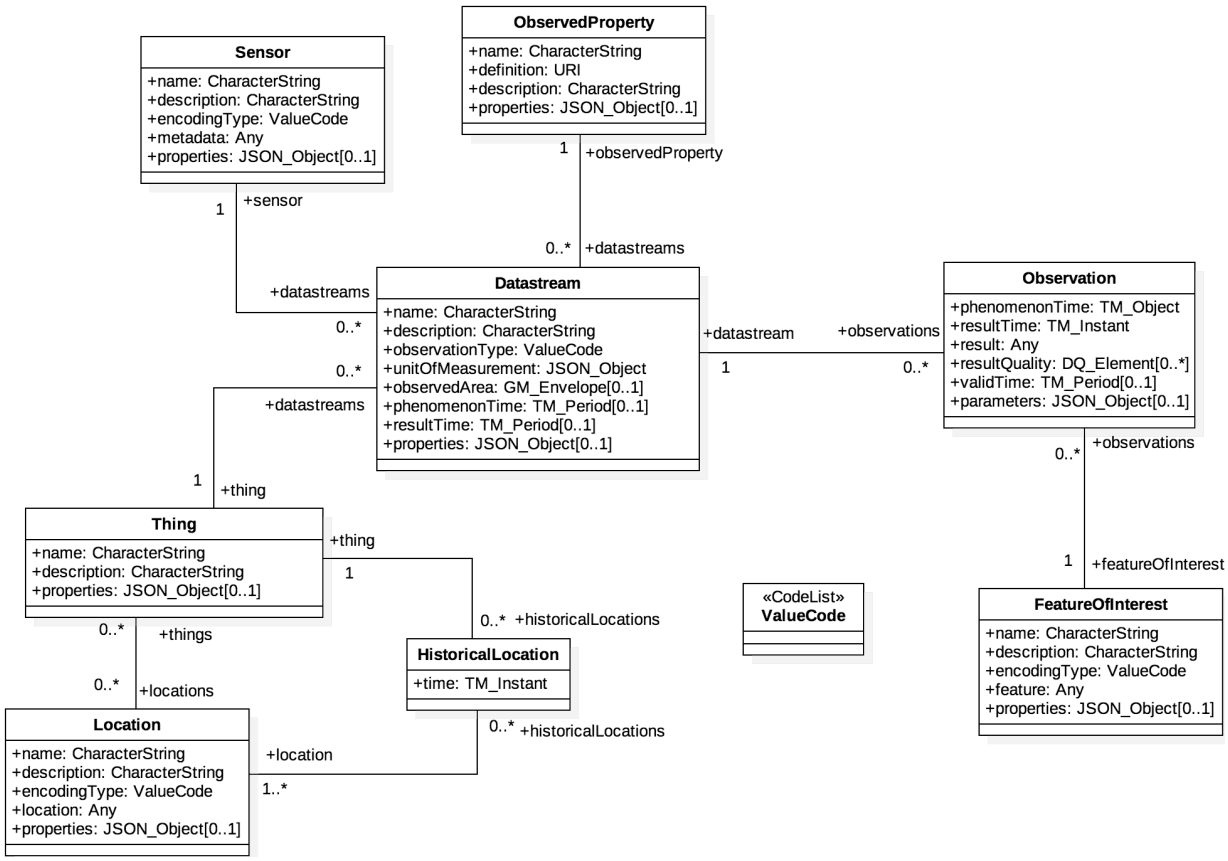

Figure 2. OGC SensorThings API (V1.1): Sensing Entities UML data model (Liang et al., 2021).

systems (van Der Schaaf et al., 2020). Such integration empowers urban administrators with augmented analytics, leading to more sagacious decision-making paradigms and efficacious urban governance (Liang et al., 2016).

The SensorThings interface is designed to enable easy access to dynamic sensor data in a standardized manner. It is crucial for this interface to be effectively promoted and well-documented in order to address the complexity of the model. Several organizations and projects in Europe, including IoT Sensors in Hamburg[2], beAware[3], Heracles[4], and the European Environment Agency's air quality data management system[5], have already adopted SensorThings as their data access layer (van Der Schaaf et al., 2020). Additionally, organizations such as SensorUps[6] and FraunhoferIOSB[7] have publicly provided comprehensive documentation and best practice guidelines for using the SensorThings API, covering everything from the first step to implementation within an organization. Distinguished from conventional time-series databases, SensorThings excels by offering expansive support for metadata encapsulation alongside the dynamic orchestration of 8 entities including Things, Sensors, Datastreams, Locations, HistoricalLocations, Observations, FeaturesOfInterest, and ObservedProperties as shown in Figure 2. This dynamic ensemble is crucial for the meticulous monitoring and documentation of temporal and spatial variations in phenomena or environmental metrics (van Der Schaaf et al., 2020, Santhanavanich et al., 2018, Zhang et al., 2023). Contrary to the limited focus on time-stamped data points prevalent in traditional databases, the SensorThings API enriches data interpretation by incorporating extensive metadata, thereby facilitating a comprehensive analytical perspective where each data instance is augmented with pertinent metadata, enhancing the depth and comprehension of the dataset.

3. METHODOLOGY

As a part of the project UDigit4iCity[8], we implemented two OGC API interfaces: the OGC API - Moving Features and the OGC SensorThings API to facilitate access to moving object data in Munakata city. Our objective is to examine and contrast the viability of both standards as potential standardized access portals for moving objects such as buses and trains within the city. In order to achieve this, the overall workflow had been structured as shown in Figure 3

3.1 Data Inputs

In the present study, a diverse array of GIS and sensor data within Munakata City, Japan, is utilized, and sourced from project PLATEAU[9]. This dataset encompasses a range of data types,

[2] https://iot.hamburg.de/v1.0
[3] https://beaware-project.eu/
[4] http://www.heracles-project.eu/
[5] https://airquality-frost.docker01.ilt-dmz.iosb.fraunhofer.de/v1.1
[6] https://developers.sensorup.com/docs
[7] https://www.iosb.fraunhofer.de
[8] https://www.hft-stuttgart.de/forschung/projekte/aktuell/icity-2-udigit4icity
[9] https://www.mlit.go.jp/plateau

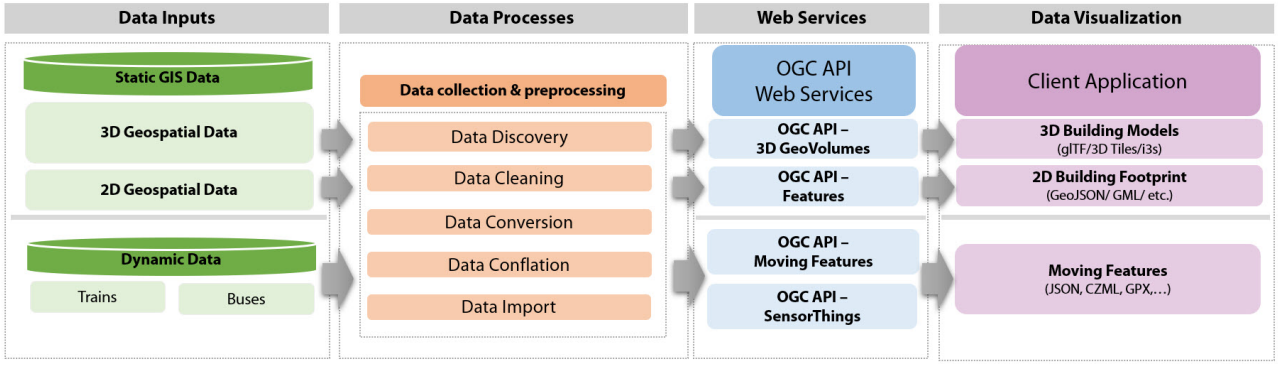

Figure 3. Interoperable Geospatial System Architecture with OGC Web Services.

including static sensor data (e.g., open-source environmental datasets), dynamic mobility data (e.g., sensor data from buses and trains), and varied GIS data both in 2D (e.g., roads, land use patterns) and 3D formats (3D building models available in CityGML and 3D Tiles formats and building types).

3.2 Web Services

To deliver the data to the application client, the OGC API web services had been developed and deployed at our university's server and are available to be accessed publicly. To better handle multiple instances and secure the server a nginx reverse proxy is employed to handle different URLs for different services or instances. The non-http(s) access is also restricted to avoid hostile takeover events and restrict database access to local services only. The web services include:

OGC SensorThings API In this work, the OGC SensorThings API Part 1: Sensing Version 1.1 (Liang et al., 2021) implementation from the Fraunhofer Opensource SensorThings-Server[10] (FROST) had been deployed. This offers a standardized way to manage and retrieve time series of moving objects and their associated metadata. After the deployment of the STA service, the data source needs to be mapped to the STA model (Figure 2), and the observation data cannot be published directly to the STA without proper registration of Datastream, Sensor, ObservedProperty, Thing, and initial Location, except for the FeatureOfInterest which is the optional entity and will point to the Location entity by default. For example, Table 1 illustrates how to map the moving bus to the SensorThings entity model.

```
GET
↪ <STA_Root_URL>/<Version>/<SensorThings_Entity_Name>
↪ HTTP/1.1
```

Listing 1. Example SensorThings Endpoint URL

After the initial registration of the SensorThings entities, the data collected from the moving sensors can be transmitted to the SensorThings server using either MQTT or HTTP protocols. In the current configuration, the data is transmitted through HTTP POST requests, which are managed by the sensor aggregator middleware provided by the sensor provider. When a change in the geolocation of a moving object is detected by the sensor, a request is triggered to update the Location entity endpoint, as illustrated in Listing 2. In this scenario, the Location entity

[10] https://github.com/FraunhoferIOSB/FROST-Server

SensorThings Entity	Encoded data
Thing	Metadata of the bus
Location	The latest geo-Locations of the Bus
HistoryLocation	Historical time-series of the bus's geo-Locations
Sensor	Metadata of Sensor model
Observed-Property	The observed properties from the sensor model such as Speed, Temperature
FeatureOf-Interested	The feature geospatial data [point/line/polygon] of the phenomenon captured by the sensor. In most cases, it is the bus itself.
Datastream	Bus datastreams measured from the Sensor
Observation	Time-series of the bus's Datastream over time

Table 1. Example mapping of SensorThings Entities for moving object: bus.

```
POST <STA_Root_URL>/V1.1/Locations HTTP/1.1
Accept: application/json, */*
Content-Type: application/json

{
  "name": "Bus - 101",
  "description": "Updated BUS-101 locations",
  "encodingType": "application/vnd.geo+json",
  "location": {
    "type": "Point",
    "coordinates": [130.5410, 33.8055]
  }
}
```

Listing 2. HTTP POST request to SensorThings's Location entity.

represents the most recent location of the objects, while the previous geolocations will automatically be transferred to store in the HistoricalLocations entity as time-series data.

For other types of sensor data, such as environmental data related to the bus, the data is posted to the Observations entity in relation to the Datastreams. For instance, Listing 3 demonstrates an example of an HTTP POST request to update the temperature in Celsius for Datastream ID 5, with a reported value

```
POST <STA_Root_URL>/v1.1/Datastreams(5)/Observations
↪ HTTP/1.1
Accept: application/json, */*
Content-Type: application/json

{
  "phenomenonTime": "2024-02-01T10:00:00.000Z",
  "resultTime" : "2024-02-01T10:00:00.000Z",
  "result" : 25
}
```

Listing 3. HTTP POST request to SensorThings's Location entity.

of 25 at the timestamp of 2024-02-01T10:00:00.000Z.

OGC API – Moving Features designed for accessing dynamic moving objects, has been developed at our university. Data is accessible in JSON format. In contrast to the SensorThings API, the Moving Features service aligns directly with the OGC APIs standard featuring an Open API interface that enables effortless exploration and retrieval of dataset collections and their associated metadata by both machines and humans via HTTP request as shown in listing 4. With various implementations of the Open API interface utilizing HTML templates (Qiu, 2017), it is feasible to present HTML catalog responses to clients when exploring data through a web browser (as shown in Figure 4) and to deliver data in JSON format when accessed by machines.

To incorporate a new object with dynamic data into the system, its Moving Feature collection and associated metadata must first be registered on the server by the HTTP POST request to the server with a payload as in Listing 5. This contrasts with the SensorThings API, where multiple entities (Datastream, Sensor, Thing, Location, ObservedProperty) must be created prior to data submission. After the collection is established, moving feature data can be submitted as a time series as shown in a JSON payload shown in Listing 6. The Moving Feature service accommodates both temporal geometries, which track the evolution of geometric attributes (e.g., location, orientation), and temporal properties, which represent other attributes (e.g., sensor readings, speed). Interpolation types ('Discrete', 'Step', 'Linear', 'Regression') are stored alongside both temporal geometries and properties.

```
GET <OGC_Moving_Feature_landing>/collections  HTTP/1.1
```

Listing 4. HTTP GET request to OGC API Moving Features collection.

OGC API – 3D GeoVolumes for accessing and transferring 3D geospatial content over the internet. It enables users to discover and access a variety of 3D content from different providers, available in multiple 3D formats (e.g., 3D Tiles, I3S). This service has been deployed as a central service that can be consumed by multiple research projects.

3.3 Data Visualization and interoperablities

In this study, the moving feature object data from both OGC SensorThings API and OGC API Moving Features are visualized on the web client which is developed with the CesiumJS-based 3D web client as illustrated in Figure 5. The data from both APIs can be easily loaded and viewed on the client by

```
{
  "links": [
    {
      "rel": "self",
      "href": "{root}/collections/bus_1",
      "type": "application/json",
      "title": "OGC API - Moving Features collections
      ↪  of bus_1"
    }
  ],
  "collections": [
    {
      "id": "bus_1",
      "title": "Bus Route 1",
      "description": "Bus Route Collection...",
      "itemType": "movingfeature",
      "updateFrequency": 1000,
      "extent": {
        "spatial": {
          "bbox": [
            [
              130.5089055259,
              33.7766570373,
              130.586648516,
              33.8137539159
            ]
          ],
          "crs": "..."
        },
        "temporal": {
          "interval": [
            "2023-11-12T12:22:11Z",
            "2023-11-24T12:32:43Z"
          ],
          "trs": "..."
        }
      },
      "children": [],
      "links": [
        {
          "title": "Bus Route 1 - Collection",
          "href": "{root}/collections/bus_1",
          "rel": "self",
          "type": "application/json"
        },
        {
          "title": "Bus Route 1 - Moving Features",
          "rel": "items",
          "href": "{root}/collections/bus_1/items",
          "type": "application/json"
        }
      ]
    }
  ]
}
```

Listing 5. OGC API Moving Features collection in JSON.

making an HTTP GET request that returns JSON responses, as depicted in Listing 6 (Moving Features) and Listing 3 (SensorThings).

Along with the moving geospatial data, the 3D city model is also loaded to the client via the OGC API - 3D GeoVolumes.

4. DISCUSSION

4.1 API Implementation

In terms of real-world implementation, the SensorThings API currently offers greater accessibility and support. Existing open-

```
POST <MF_root>/collections/bus_1/items  HTTP/1.1
Accept: application/json, */*
Content-Type: application/geo+json

{
  "type": "Feature",
  "id": "mf_1",
  "properties": "...",
  "temporalGeometry": {
    "type": "MovingPoint",
    "datetimes":[
        "2023-11-12T12:22:11Z",
        "..."
    ],
    "orientations":[
        {"scales": [1,1,1],"angles": [0,0,0]},
        "..."
    ],
    "interpolation": "Linear",
  },
  "temporalProperties": [
        {
            "datetimes":[
                "2023-11-12T12:22:11Z",
                "..."
            ],
            "temperature": {
                "type": "Measure",
                "values": [25, "..."],
                "interpolation": "Linear"
            }
        }
    ]
}
```

Listing 6. Example JSON payload for the moving item with integrated temperature temporal data in OGC API - Moving Features.

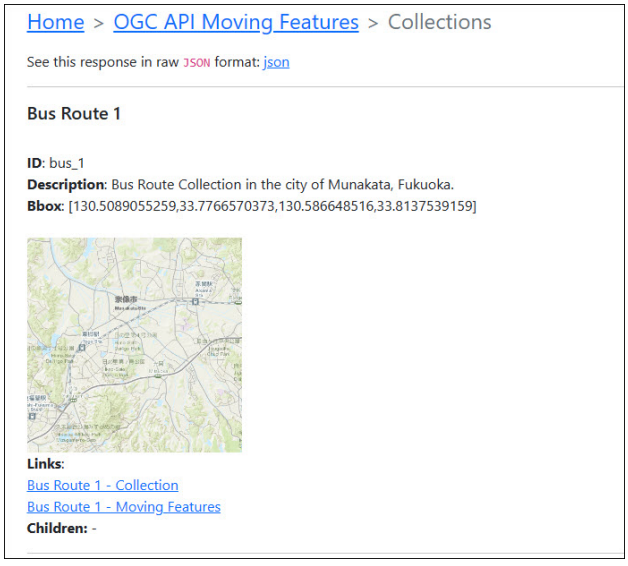

Figure 4. OGC API - Moving Features's collections in HTML encoded shown by default when accessed by humans via web browsers

source server implementations like FROST-Server and GOST[11], along with online learning resources[12], provide organizations with a smoother adoption process. In contrast, the OGC API

[11] https://github.com/gost/server
[12] https://developers.sensorup.com/docs

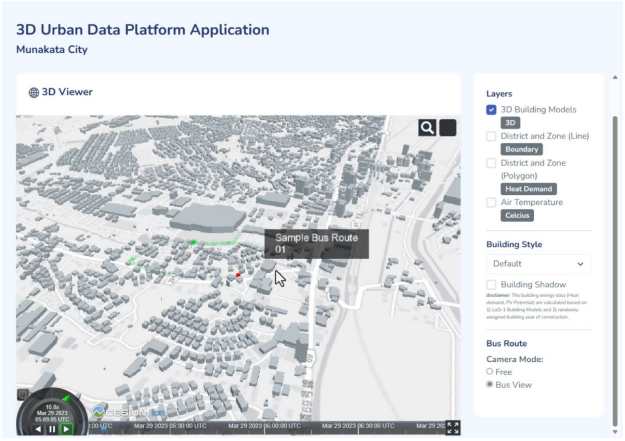

Figure 5. 3D Munakata City Model with Dynamic Bus Route Visualization, built with Cesium-JS library.

- Moving Features lacks readily available open-source implementations and dedicated learning resources. Organizations considering its use would likely need to undertake in-house development. Nonetheless, the Moving Features standard's foundation in OpenAPI suggests potential future integration with open-source OGC API implementations like PyGeoAPI[13].

Adopting either standard effectively necessitates a learning curve for developers and GIS professionals. This investment can be significant, especially when experience with similar standards is limited. The nascent stage of OGC API - Moving Features adoption further compounds this challenge, as fewer public implementations and learning resources are available compared to the SensorThings API. Organizations should consider providing training and support to facilitate successful implementation.

4.2 Data Accessibility

Both the OGC SensorThings API and OGC API - Moving Features support spatial (bounding box) and temporal (date/time) filtering of data. However, the SensorThings API offers more granular filtering capabilities, allowing queries focused on specific observed properties, sensors, and things. This precision provides greater flexibility in data retrieval but might increase query complexity for developers unfamiliar with the SensorThings structure. In contrast, Moving Features prioritizes ease of access and filtering. Additionally, SensorThings' reliance on separate datastreams for each observed property can necessitate multiple requests for comprehensive data. The Multidatastream extension can simplify this process. Moving Features, on the other hand, delivers both geometric and all temporal data within a single request by default.

In terms of data discovery and developer experience, the OGC API - Moving Features holds an advantage by leveraging the OpenAPI specification. This provides standardized API definitions with JSON and HTML responses, along with readily available open-source interface implementations, as an example shown in figure 4. These tools make it easier for developers to explore and understand the API structure. In contrast, SensorThings API lacks server built-in interfaces, often requiring developers to either directly navigate JSON data or create custom interfaces for data exploration. The modular design principles promoted by OGC API standards further enhance developer experience by enabling flexible implementation and potential extensions to the Moving Features API.

[13] https://pygeoapi.io/

4.3 Dynamic data handling

The OGC API - Moving Features specification offers built-in mechanisms for representing detailed movement characteristics that are not natively supported by the SensorThings API. This includes the ability to specify interpolation types (discrete, linear, or potentially more advanced methods) to define how movement is represented between observed positions. Additionally, Moving Features allows encoding the state of a moving object (e.g., start, stop, or custom states like "accelerating"). It can even capture the orientation data (roll, pitch, yaw) of the moving object. While similar information could be included in the SensorThings API, this would necessitate adding custom properties to entities like Observations or registering them as extra ObservedProperties. This alternative approach likely results in a less standardized and more complex implementation compared to the dedicated constructs provided by the OGC API - Moving Features specification.

One potential limitation of the OGC API - Moving Features lies in its handling of real-time data streams. While technically feasible to update moving feature data, the current specification typically requires re-posting the entirety of an item's data via HTTP POST requests whenever a change occurs. This approach can lead to increased payload sizes, potentially hindering performance, especially in high-frequency, real-time scenarios where numerous updates occur rapidly. In contrast, the SensorThings API offers a more efficient approach for real-time data updates by natively supporting streaming protocols like HTTP and MQTT. These protocols are specifically designed for efficient data transfer, making them well-suited for real-time data transmission and reducing the strain on network resources compared to repeated full data re-posting via HTTP POST.

5. CONCLUSION

This study has found that both the OGC API - Moving Features and OGC SensorThings API support the representation and analysis of moving features, though they cater to different use cases and present contrasting implementation considerations. The OGC API - Moving Features excels with its specialized focus on moving objects, providing built-in constructs for temporal geometry, state, and orientation. This focus streamlines the handling of complex movement data. Furthermore, its adherence to the OpenAPI specification enhances developer experience and ease of implementation. In contrast, the SensorThings API takes a broader IoT perspective, accommodating both stationary and moving sensor data alongside device tasking capabilities. While it offers the flexibility to model detailed moving object characteristics, this customization requires more effort and may lead to less standardization compared to the Moving Features standard.

The most suitable choice between these APIs depends on the specific application requirements. If the primary goal is the in-depth tracking and analysis of moving objects, the OGC API - Moving Features offers a more direct and standardized solution. When a broader IoT context necessitates diverse sensor data integration, device tasking, and robust real-time streaming capabilities, the SensorThings API presents a more versatile framework.

ACKNOWLEDGEMENTS

The research was conducted at the Centre for Geodesy and Geoinformatics, HFT Stuttgart, within the framework of project UDigit4iCity (grant agreement No 13FH9I06IA) and project DiaOpt4iCity (grant agreement No 13FH9I10IA). Funding for both projects was provided by the German Federal Ministry of Education and Research (BMBF). The geospatial dataset utilized in this study was sourced from project PLATEAU[14], supported by the Ministry of Land, Infrastructure, Transport and Tourism, Japan.

REFERENCES

Asahara, A., Hayashi, H., Ishimaru, N., Shibasaki, R., Kanasugi, H., 2015a. International standard "ogc moving features" to address "4vs" on locational bigdata. *2015 IEEE International Conference on Big Data (Big Data)*, IEEE, 1958–1966.

Asahara, A., Shibasaki, R., Ishimaru, N., Burggraf, D., 2015b. OGC® Moving Features Encoding Extension: Simple Comma Separated Values (CSV).

Asahara, A., Shibasaki, R., Ishimaru, N., Burggraf, D., 2019. OGC® Moving Features Encoding Part I: XML Core.

Brandoli, B., Raffaetà, A., Simeoni, M., Adibi, P., Bappee, F. K., Pranovi, F., Rovinelli, G., Russo, E., Silvestri, C., Soares, A. et al., 2022. From multiple aspect trajectories to predictive analysis: a case study on fishing vessels in the Northern Adriatic sea. *GeoInformatica*, 26(4), 551–579.

Godfrid, J., Radnic, P., Vaisman, A., Zimányi, E., 2022. Analyzing public transport in the city of Buenos Aires with MobilityDB. *Public Transport*, 14(2), 287–321.

Hayashi, H., Asahara, A., Kim, K.-S., Shibasaki, R., Ishimaru, N., 2017. OGC® Moving Features Access.

Kim, K.-S., Ishimaru, N., 2020. OGC® Moving Features Encoding Extension - JSON.

Kotsev, A., Schleidt, K., Liang, S., Van der Schaaf, H., Khalafbeigi, T., Grellet, S., Lutz, M., Jirka, S., Beaufils, M., 2018. Extending INSPIRE to the Internet of Things through SensorThings API. *Geosciences*, 8(6), 221.

Liang, S., Huang, C.-Y., Khalafbeigi, T., 2016. OGC® SensorThings API Part 1: Sensing, Version 1.0.

Liang, S., Khalafbeigi, T., van der Schaaf, H., 2021. OGC® SensorThings API Part 1: Sensing, Version 1.1.

OGC, 2015. OGC® standards and supporting documents— ogc.

Percivall, G., Holmes, C., Wesloh, D., Heazel, C., Gale, G., Christl, A., Lieberman, J., Reed, C., Herring, J., Desruisseaux, M., Blodgett, D., Simmons, S., de Lathower, B., 2017. OGC® Open Geospatial APIs - White Paper.

Qiu, Y., 2017. The openness of open application programming interfaces. *Information, Communication & Society*, 20(11), 1720–1736.

Reed, C., Botts, M., Percivall, G., Davidson, J., 2013. OGC® sensor web enablement: Overview and high level architecture.

Richter, A., Löwner, M.-O., Ebendt, R., Scholz, M., 2020. Towards an integrated urban development considering novel intelligent transportation systems: Urban Development Considering Novel Transport. *Technological Forecasting and Social Change*, 155, 119970.

[14] https://www.mlit.go.jp/plateau

Sakr, M., Zimányi, E., Vaisman, A., Bakli, M., 2023. User-centered road network traffic analysis with MobilityDB. *Transactions in GIS*, 27(2), 323–346.

Santhanavanich, T., Schneider, S., Rodrigues, P., Coors, V., 2018. Integration and visualization of heterogeneous sensor data and geospatial information. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 4, 115–122.

van Der Schaaf, H., Mossgraber, J., Grellet, S., Beaufils, M., Schleidt, K., Usländer, T., 2020. An environmental sensor data suite using the OGC sensorthings api. *Environmental Software Systems. Data Science in Action: 13th IFIP WG 5.11 International Symposium, ISESS 2020, Wageningen, The Netherlands, February 5–7, 2020, Proceedings 13*, Springer, 228–241.

Zhang, M., Yue, P., Hu, L., Wu, H., Zhang, F., 2023. An interoperable and service-oriented approach for real-time environmental simulation by coupling OGC WPS and SensorThings API. *Environmental Modelling & Software*, 165, 105722.