Multi-platform data search and access method to compose digital twins using metadata

Ryota Yabe
Intelligent Systems Laboratory, SECOM CO., LTD
ryo-yabe@secom.co.jp

Abstract

Various real-world applications are implemented to make life comfortable, convenient, and safe. The data and applications are tightly coupled. Application implementation costs are high because data reusability is low. Data reusability is increased by implementing a mechanism that collects data according to the requirements of real-world applications, builds a digital twin (DT) from the collected data, and combines the data with real-world applications. Therefore, platforms providing data to creating DTs are emerging. However, DT data platforms have the following problems. As DT data platforms proliferate, they are distributed across multiple platforms. Consequently, it is difficult for real-world application implementers to obtain data for creating a DT. Data reprocurement increases application implementation costs. Herein, methods for solving data reusability problems are proposed. A function for outputting metadata representing the spatial structure was implemented on spatial data platforms. Data delivery interfaces are implemented in data platforms that manage data related to spatial data. The method for accessing the related data is described in the metadata section. In this study, prototype systems for building information modeling (BIM) data were implemented using the proposed method. The industry foundation classes (IFC) model server, which exports metadata, was implemented. The API Server, which manages the building of the 3D model data, was implemented. The endpoints for accessing the 3D data API server are described in the metadata.

Keywords
Digital Twin, BIM, IFC, IFC model server, Linked Data, metadata

1. Introduction

Various real-world applications have been developed for improving comfort, convenience, and safety. For example, autonomous cars can reduce congestion and resolve traffic bottlenecks in the mobile sector. In the field of disaster prevention, the formulation of evacuation plans using damage-prediction simulations also increases safety. Thus, the implementation of various real-world applications to solve tasks is important. Implementation of real-world applications has become increasingly important in the development of smart cities and buildings. Currently, the high cost of data generation is a barrier to implementation in real-world applications. Reducing the cost of creating data to implement real-world applications is difficult, because creating data requires surveying and drawing. Therefore, reusing the existing data is necessary to reduce the cost of implementing real-world applications. However, reusing data to reduce the cost of data creation is difficult. Real-world data are used to implement specific real-world applications and are not expected to be reused in other real-world applications. Therefore, data must be created each time real-world applications are implemented. Tight coupling between data and real-world applications inhibits data reuse. The Smart City Guidebook [1] published by the Japanese Cabinet Office highlights this issue. Loose coupling of data and real-world applications is required to increase data reusability.
A digital twin (DT) can be considered a means of increasing the reusability of data. A DT is a technology that replicates the real world in the cyber world by using data collected from the real world. Data reusability is increased by implementing a mechanism that collects data according to the requirements of real-world applications, builds a DT from the collected data, and combines the data with real-world applications.

Figure 1 Creating DTs from various data to implement application
Currently, data platforms that manage data to create DTs are being developed. For example, data platforms for creating DTs include PLATEU [2] and Azure Digital Twins [3]. Data platforms to create DTs will continue to grow.

Figure 2 Implement real-world applications using a DT created from platform data
However, the proliferation of data platforms has reduced their discoverability. As data platforms proliferate, data related to specific locations become scattered across many data platforms. An application implementer must search many data platforms whose data are suitable for the DT to implement the application. Failure to discover the data increases the cost of application implementation because the data must be regenerated.

This study attempts to solve data discoverability issues to improve the reusability of data for creating DTs. This study proposes the following method. A function for outputting metadata representing the spatial structure was implemented on spatial data platforms. Interfaces for data provision were implemented on the related data platforms. The procedure for accessing the interfaces to provide related data is described in the Metadata section. In the proposed method, it is possible to repeatedly search for available data on data platforms using metadata. Consequently, the reusability of the data is improved.

Figure 3 Cross-platform data discovery using metadata
The main feature of the proposed method is the construction of a data catalog that performs spatial and semantic searches. A data catalog is a collection of metadata for a data search. In
the proposed method, users can search for data to create a DT by traversing the semantic connections of the spatial elements. For example, the data user can search for related data based on a query such as “data related to all office rooms on the second floor of this building.”

In this study, prototype systems for building information modeling (BIM) of spatial data were implemented using the proposed method. Based on the proposed method, a spatial data management server and a related data management server were implemented and linked using metadata. A spatial data server was implemented that had a function to manage BIM data compliant with industry foundation classes (IFC) and a function to export metadata from stored BIM data. For the related data server, an API server was implemented with a function to manage the 3D building model data. The endpoints for accessing the 3D data API server are described in the metadata.

2. Related Works

This section describes the related work. The proposed method searches for related data to construct a DT by using spatial metadata from various platforms.

2.1. 3D City DB

This section presents related work linking the geometric and attribute data of spatial data. An example of a method for associating spatial data with related data is 3D City Database (DB). The 3D City DB is open-source software for server-side applications to manage CityGML [5] data. CityGML is a data format used in a GIS and is based on XML. Because CityGML contains not only geometric data but also complex semantic data, such as object type, it is difficult to store in a spatial database. This section describes the method for obtaining information on the API of a related data platform to provide related data from the searched metadata. Users receive attribute data values describing how to access related data from the obtained metadata. Data users then access the related data based on the data access method obtained from the metadata attributes. The purpose of the 3D City DB is to provide an environment that allows both attribute value-based and spatial search functions to be used with the CityGML. Therefore, the 3D City DB approach does not solve the data discoverability problem.

Users who want to use CityGML data must know that there is a spatial database that manages the geometric data of a CityGML file and a CSV file that manages the attribute data of a CityGML file. In addition, the data user must know the method of accessing the database and CSV files.

The purpose and approach of our proposed method differ from those of a 3D City DB. The purpose of the proposed method is to create an environment for searching and collecting data across various spatial data platforms and to construct a DT. Therefore, in our proposed method, the 3D City DB is also one of the search targets of the data management system for building a DT.

2.2. BIMServer

This section describes the BIMServer [6] as a related work. BIM is an integrated architectural process that uses 3D building information. Three-dimensional building model data used in the BIM process are referred to as BIM data. BIM data have a format called IFC, which is an international standard data format for data exchange. The IFC was developed by building SMART International (bSI), an ISO certified international standardization body for BIM. Among the systems that manage BIM data, a system that manages IFC-compliant BIM data is called an "IFC model server," and research has been conducted to realize it. The problem in implementing the IFC model server is storing the IFC data in the database. Because the IFC data have a complex object-tree structure, storing IFC data in a database is challenging.

BIMServer is an IFC model server implemented to address this problem. It is open-source software for server-side applications to manage IFC data. BIMServer is a monolithic application that not only stores, manages, and reads IFC data, but also provides various functions necessary for the building lifecycle, such as clash detection.

The purpose and approach of our proposed method are different from those of the BIMServer. The purpose of the proposed method is to create an environment for searching and collecting data across various spatial data platforms and to construct a DT. Therefore, in the proposed method, the data managed by the BIMServer are included in the search targets.
2.3. Ontologies for spatial metadata

This section describes the spatial metadata ontologies. Ontology is a means of formally describing knowledge and the relationships between this knowledge. An ontology was used to describe the schemas of metadata on the Web. Ontologies have been proposed to describe the schemas of the spatial elements in BIM data.

A Building Topology Ontology (BOT) [7] is an example of an ontology that describes the concept of spatial elements. BOT is an ontology that represents a simplified spatial structure formulated by bSi and the World Wide Web Consortium (W3C) to simplify and express IFC-compliant BIM data. BOT represents spatial elements such as instances of site, building, storey, space, and element classes. Spatial elements are connected by properties such as “containsZone” in BOT.

Next Generation Service Interface-Linked Data (NGSI-LD) [8] is another example of an ontology for describing the concept of spatial elements. NGSI-LD is an ontology based on the Next Generation Service Interfaces (NGSIs) [9], which is a data specification formulated for use in DTs and smart cities, etc. NGSI-LD was standardized by the European Telecommunications Standardization Institute (ETSI). NGSI-LD represents spatial elements as instances of the building class, which includes instances of sensor and human classes.

The Digital Twin Definition Language (DTDL) [10] is primarily used to express the data of DTs managed by Azure Digital Twins. Additionally, several domain-specific ontologies based on DTDL have been proposed. An example is the Real Estate Core [11], which is an ontology defined to represent smart buildings. The Real Estate Core represents spatial elements such as instances of land, buildings, level, and room classes. These classes are linked by “isPartOf” property. DTDL is an ontology for representing DTs created with reference to ontologies such as NGSI-LD.

These ontologies are defined to structurally express the relationships between spaces and the IoT devices installed in those spaces. However, these ontologies are not intended for cross-platform data discovery. The proposed method uses metadata conforming to these ontologies to search for data and create a DT. Our proposed method allows searching for related data across platforms by outputting metadata that conform to such ontologies from spatial data platforms and writing the endpoint information of the related data into the metadata. The proposed method effectively utilizes the metadata described by these ontologies.

3. Proposed method

In this section, the proposed method is described. The proposed method comprises the following four elements: The first is the metadata output of the spatial information platform. The second element is the method of associating metadata with information regarding the endpoint to access related data. The third element is a method for spatially and semantically searching for the metadata of a spatial element whose related data are retrieved from the metadata group by traversing the metadata graph. The fourth element is a method for obtaining information to access related data from the searched metadata.

Figure 4 presents an overview of the proposed method. The elements that comprise this method are spatial data platforms and data platforms that manage related data in space. Examples of related data include 3D model data for visualization, and 2D floor plan data. Spatial data platforms use APIs that expose metadata to represent spatial structures. Data platforms that manage related data have APIs to provide the data. Metadata contains information about endpoints for accessing data about spaces represented by metadata.
3.1. Spatial structure metadata

This section describes the metadata that represent the spatial structure. The spatial structure metadata were tree structured. Each node in the tree represents a spatial component. The edges that comprise the tree represent the connection and inclusion relationships between the spatial components. The spatial structure metadata represent the relationships between spatial elements as edges.

The purpose of representing metadata as a graph data model is to ensure compatibility with existing spatial metadata data models such as BOT, DTDL, and NGSI-LD, which represent the spatial structure as graph data.

Metadata is created by the spatial platform from stored spatial data and is provided by the metadata provisioning API.

3.2. Linking method from metadata to API providing related data

This section describes the linking method between the metadata and API. In this method, metadata contain attributes containing information about the API endpoint of data platforms that manage data about spaces represented by metadata. For example, a floor metadata attribute for association contains information regarding the API endpoint of the data platforms that manage floor data, such as 3D model data and floor maps.

3.3. Method for semantic and spatial search

This section describes a method for spatially and semantically searching for the metadata of a spatial element whose related data must be retrieved from the metadata by traversing the metadata graph. Data users search for spatial element metadata from which they want to obtain related data from a group of metadata outputs using a spatial data platform by traversing the edges that represent semantic connections between nodes representing a spatial element.

For example, a user might wish to retrieve data related to a room on the first floor of a building. First, the user retrieves floor instances of the building from the building instance by traversing the semantic connections between the building and floor. Next, metadata representing the first floor were retrieved from the retrieved floor metadata group. Subsequently, the metadata representing the room associated with the first floor were obtained by tracing the semantic relationship between the floors and rooms.

3.4. Method to access API for obtaining related data

This section describes a method for obtaining information regarding the API of a related data platform to provide related data from the searched metadata. Users receive attribute data values describing how to access related data from the obtained metadata. Data users then access the related data based on the data access method obtained from the metadata attributes.

Figure 4 Overview of proposed method
4. Implementation

This section describes the implementation of a prototype system based on the proposed method. Figure 5 shows an overview of the prototype system. This implementation primarily deals with BIM data as spatial data. The prototype consisted of two systems. The first was an IFC model server. The second system manages 3D data generated from the IFC data, provides functions for exporting 3D data in various CG data formats, and spatially searches for 3D objects. The second system will be referred to as the “IFC geometry server.” The Resource Description Framework (RDF) [12], which is the metadata description framework of the Semantic Web, was adopted to write metadata to link these systems. Metadata were provided by the metadata API of the IFC model server.

![Spatial Structure Metadata](image1)

**Figure 5 Overview of prototype systems**

4.1. IFC model server implementation

This section describes the implementation of the proposed IFC model server. Figure 6 shows the components of the prototype system. The IFC model server was implemented using Python 3.9 and ArangoDB [13]. The function of the IFC model server as a Web API server was implemented using Flask [14]. IFC model server uses IfcOpenshell-python [15] to interpret IFC data. The IFC model server uses RDFlib [16] to export metadata to RDF. In our implementation, the IFC model server stored the IFC data in a graph database. This prototype uses ArangoDB as a graph database to store IFC data. To access ArangoDB from Python, our prototype system used Python-Arango [17] as the database driver.

![Components of IFC model server](image2)

**Figure 6 Components of IFC model server**

4.1.1 Process to upload IFC file

This section describes the process of uploading the IFC files. The process of uploading an IFC file to the IFC model server is shown in Figure 7. The IFC model server uses the following procedure to upload the IFC files.

1. The IFC model server converts the uploaded IFC file into a Python object tree using IfcOpenshell-Python.
2. IFC model server stores this tree data in ArangoDB by using Python-Arango.
4. 1. 2 Process to provide data

This section describes the data provision process that uses the API of the IFC model server. The process of providing data using the API of the IFC model server is shown in Figure 8. The IFC model server provides data using the API through the following steps.
1. Receive request through API implemented by flask.
2. Query data in ArangoDB using Python-Arango and read IFC data.
3. Convert data from IFC data to Python object tree.
4. Convert python object tree to JSON and provide JSON data through API.

4. 1. 3 Process to provide metadata

This section describes the metadata-provision process using the API of an IFC model server. The process of providing metadata using the API of the IFC model server is shown in Figure 9. The IFC model server provides metadata using the API through the following steps.
1. Receive request through metadata API implemented by Flask.
2. Query data in ArangoDB using Python-Arango and read IFC data.
3. Convert data from IFC data to Python object tree.
4. Convert a Python object tree into JSON and provide JavaScript Object Notation for
Linked Data (JSON-LD) [18] through an API.

Figure 9 Provide metadata process through IFC model server API

4.2. IFC geometry server implementation

This section describes the implementation of an IFC geometry server. Figure 10 shows the components of the prototype system. The IFC geometry server was implemented using Python 3.9 and PostGIS. The function of the IFC geometry server as a Web API server was implemented using Flask. IFC geometry server uses IfcOpenshell-Python to interpret IFC data. The IFC geometry server used trimesh [19] to export various 3D CG formats. This prototype uses PostGIS [20] as a spatial database to store the 3D mesh data. To access PostGIS in Python, our prototype system used psycopg2 [21] as the database driver.

Figure 10 Components of IFC model server

4. 2. 1 Process to Upload IFC file

This section describes the process of uploading an IFC file to an IFC geometry server. The process of uploading an IFC file to the IFC geometry server is shown in Figure 11. The IFC model server uses the following procedure to upload IFC files. This IFC geometry server uses the following procedure to upload the IFC files.
1. The IFC geometry server converts the uploaded IFC file into Python objects that represent the geometry using IfcOpenshell-Python.
2. IFC geometry server stores these Python object data in PostGIS using psycopg2.
4.2.2 Process to provide geometry data

This section describes the process of obtaining geometric data using the API of the IFC geometry server. The process of providing the geometric data as a JSON file using the API of the IFC geometry server is shown in Figure 12. The process of providing geometric data as a GLB file using the API of the IFC geometry server is shown in Figure 13. The IFC geometry server provides data using the API through the following steps.

1. The IFC geometry server receives requests through an API implemented in a flask and reads the geometric data from PostGIS by querying using psycopg2.
2. IFC geometry server converts data from database query results to Python object.
3. IFC geometry server converts Python objects to JSON and expose them through API.

In the response format, the IFC Geometry Server API supports the GLB format, which is a binary version of the GL Transmission Format (GLTF) [22] used to create content that runs on web browsers. If the specified response data format is in the GLB format, the trimesh converts the Python objects.
4.3. Metadata data model

In this section, the proposed metadata model is described. In this implementation, the BOT was used as the metadata data model to represent the spatial structure. BOT uses it to represent spatial metadata because it has a high affinity for spatial data managed by the IFC model server. The metadata provided by the IFC model server API include site class, building class, story class, space class, and element class instances. These classes have an attribute definition called “has3DModel” to represent the associated 3D model. In this implementation, an IFC geometry server endpoint for accessing associated 3D model data is described as an attribute value of “has3DModel.” Metadata is created each time that metadata API is called.

4.4. Process to upload IFC file

This section describes the process of data upload to these two servers. The process of uploading the IFC file to the two servers is shown in Figure 14. The files were uploaded to the two servers using the following four steps.
1. An IFC data owner uploads an IFC file to the IFC model server.
2. The IFC model server stores the uploaded IFC data and generate a model ID.
3. The IFC model server returns the model ID to the data owner.
4. The owner uploads the same IFC data to the IFC geometry server using the model ID issued by the IFC model server.
5. The IFC geometry server stores the geometric data generated from the uploaded IFC data associated with the model ID.

Once processes 1–5 are completed, the metadata can be output from the IFC model server. The metadata contain information regarding the endpoint for accessing geometric data. Information about the endpoint is registered as a value of the “has3DModel” attribute.

Figure 14 IFC file upload process to two systems

5. Verification of Prototype System Operation

This section describes the verification of the proposed method by using a prototype system. The purpose of this check is to verify that the endpoint of the IFC geometry server can be accessed using the metadata output from the IFC model server. The functionality of the prototype system was verified using the following procedure.
1. Upload a test IFC file to the IFC model server.
2. Upload a test IFC file to the IFC geometry server.
3. Retrieve metadata from the IFC model server.
4. Query the retrieved metadata.
5. Retrieve geometry data from the endpoint registered in the has3DModel attribute.

A test model is shown in Figure 15.

Figure 15 Test IFC model

5.1. Uploading IFC file to IFC model server

This section describes the uploading of IFC data to the IFC model server and the resulting upload. Figure 16 shows the test web UI for uploading an IFC file using the API of the IFC model server. The Web UI was developed by Swagger [23]. To confirm this, a test IFC file was
uploaded using the Web UI. Figure 16 shows the file upload results for the Web UI. The IFC model server results show that the test IFC file was successfully uploaded. In Figure 16, the IFC model server returns the model ID (“Ifcmodel_id”) and message (“upload finished”).

Figure 16 Swagger Web UI of IFC model server

Figure 17 Results of IFC file upload to IFC model server

5.2. IFC file upload to IFC geometry server

This section describes the uploading of IFC data to the IFC geometry server and the resulting upload. Figure 18 shows the test web UI for uploading an IFC file using the API of the IFC geometry server. For verification, the test IFC file was uploaded using a Web UI. Figure 18 shows the file upload results for the Web UI. The IFC geometry server results showed that the test IFC file was successfully uploaded. Figure 18 also shows that the model ID was passed as a parameter when the test IFC file was uploaded to the IFC geometry server. Figure 19 shows the results of uploading an IFC file to the IFC geometry server.
5.3. Retrieving Metadata

This section describes the results of metadata retrieval from the IFC model server. Access to the IFC model server API was provided to retrieve the metadata. Figure 20 shows a portion of the metadata retrieved by the IFC model server API. In this case, the endpoint was “http://localhost:8080/v1/ifcgeometry/upload".
IFC model server are described in JSON-LD format. Metadata instances contain an attribute representing the object type. The values of the object-type attribute indicate that the metadata instances are instances of BOT spatial element classes, such as Elements and Space. Metadata objects have other attributes such as type, Class, GlobalId, Name. Metadata instances also have a “has3Dmodel” attribute. The value of “has3Dmodel” is information about the endpoint of the IFC geometry server, but these attributes are tentatively defined and do not conform to BOT and any standard or published specification. This result shows that the IFC model server was able to export the metadata to obtain endpoint information for accessing related data and that the metadata could be retrieved via the IFC model server API.

Figure 20 Example of metadata exported from IFC model server API

5.4. Querying metadata

This section describes the results of querying metadata from the IFC geometry server based on the endpoint information described in the metadata. BOT metadata allow a spatial element to be searched using data exploration methods, such as the SPARQL Protocol and RDF Query Language (SPARQL) [24]. SPARQL is the standard query language and protocol for RDF data and RDF databases. Figure 14 shows the Python code-to-query metadata provided by the API of the IFC model server using SPARQL on the Jupyter notebook and the query result. This SPARQL query obtains the story- and space-class instances linked to these instances. The lower part of Figure 19 shows that the executed query was successful. This result shows the metadata exported from the IFC Model Server, which enables spatial and semantic searching of related data in the IFC Geometry Server.
This section describes the results of geometric data retrieval from the IFC geometry server based on the endpoint information described in the metadata. Figure 22 shows the Python code used to access the IFC geometry server endpoint described in the metadata attribute and to visualize the 3D data retrieved from the endpoint in a Jupyter notebook. In addition, Figure 23 shows the response of the IFC geometry server endpoint described in the metadata. The response from the IFC geometry server is a serialized JSON with attributes such as globally_unique_id, class_name, vertices, indices, normals, and face_colors. In the Python code shown in Figure 22, the JSON data are visualized using trimesh. The resultant visualization is shown at the bottom right, which indicates that the related data can be obtained by accessing the endpoint described in the metadata provided by the IFC model server.
5.6. Discussion

This section discusses the results of the operational verification of the prototype. A series of tests confirmed that the metadata in the IFC model server made it possible to search and retrieve related data. The behavior of the prototype systems shows that our proposed method enables the semantic and spatial search and retrieval of related data managed by related data platforms. For example, it is possible to search and retrieve data from platforms that manage spatial data with different expressions, such as floor plans targeting the same space.
6. Future work

This section describes the further work required to address issues of social implementation of the proposed method. In this study, the feasibility of the proposed method is demonstrated through implementation of a prototype system. The proposed method is effective in creating DT. To realize an environment that enables the creation of DTs by the semantic and spatial searching of data in the future, our proposed method has several issues that need to be solved. Four possible studies that address these issues are described below.

6.1. Standardization of platform API and schema of metadata

This section describes the work performed on data standardization. To realize data discovery and linkage across platforms using the proposed method, the interoperability of data and metadata is required. The lack of metadata interoperability across spatial data platforms makes the semantic and spatial discovery of data difficult because data users must understand the metadata model and rewrite queries to search for data for each platform. In addition, the different API data models for each platform make the creation of DTs costly because data users need to convert the data provided by the platforms. To solve these issues, standardization of the metadata model and platform API is required.

6.2. Standardization of data creation process

This section describes the work conducted to standardize the process of creating and uploading data to a platform in the future. The following three steps are necessary to implement our proposed method.

1. A process must be established to generate spatial data, from which metadata can be extracted and uploaded to the platform.
2. A process must be established to generate data related to the spatial data to be uploaded to the platform.
3. A process must then be established to generate metadata to link the spatial data uploaded to the platform with related data.

To realize the proposed method, the data generators must perform Steps 1–3. Therefore, it is necessary to formulate and standardize the process for each of Steps 1–3 so that data generators can generate data according to the standard processes.

6.3. Establishment of data quality description and search method

This section describes the work performed using a data quality description method. To implement an application using a DT, the data comprising the DT must satisfy the requirements. Application implementers want to know many requirements regarding data quality in advance, such as data format, accuracy, granularity, level of detail, etc. However, application implementers do not know the quality of data when using our proposed method.

To promote our proposed method, it is necessary to establish a method to describe the quality of data and to establish a method to search for data not only spatially and semantically but also with data quality requirements. In the future, the establishment of a data quality description and search method with data quality requirements will be necessary.

6.4. Approach to establish process and Standardization

In this section, we discuss the efforts required to solve the issues described in 6.1 to 6.3. Establishing data-creation processes and standardizing data models are the keys to solving these problems. For stakeholders to adopt the standards for the data model, data creation process, and data quality description, these standards should be practical and useful. Establishing practical and useful standards using knowledge extracted from Proof of Concept(PoC) experience is important.

To develop useful standard data models, processes, and data quality description methodologies, we propose the following four steps:

1. Gather stakeholders in a specific domain.
2. Conduct a PoC to implement the application.
3. Develop data models, processes, and data quality descriptions using the knowledge gained from the PoC.
4. Promote international standardization by disseminating successful cases and increasing the number of applications.

In the future, PoCs for application implementation using DTs should be more active. Our prototype system is open source [25]; therefore, it can be used as a reference for...
implementation examples when the PoC is activated.

7. Conclusion

In this study, a method is proposed to link spatial data platforms and related data platforms using spatial metadata to create DTs. Spatial data platforms are designed to output metadata representing spatial structures. Related data platforms have been designed with interfaces to provide the data. The procedure for accessing the interfaces to provide related data is described in the Metadata section. In this study, prototype systems were implemented based on the proposed method. The metadata and related data provided by the prototype system are verified. By querying the metadata output from the prototype system and retrieving data using metadata information, we showed that our proposed method can spatially and semantically search for the data necessary to construct a DT.

However, the proposed method has several limitations. The proposed method cannot ensure the interoperability of API specifications and metadata between platforms. It is necessary to standardize API specifications and metadata. It is also necessary to establish a standardized process for generating spatial data from which metadata can be exported and uploaded to DT platforms. To resolve these problems, it is important to formulate a process to generate spatial data that can be converted to metadata, develop specifications for metadata and API, and standardize the process and specifications. To develop useful standard specifications and processes, it is necessary to gather stakeholders in a specific domain, conduct a PoC to implement the application, and use the knowledge obtained from the PoC. We opened our prototype system because we believe that people can handle spatial information using the proposed method and can develop it into a standard method using various PoCs.

Additionally, the proposed method can be extended to IoT device data. Extending the proposed method to the IoT domain is a future task.

References


