Hindawi
Advances in Civil Engineering
Volume 2021, Article ID 4913394, 9 pages
https://doi.org/10.1155/2021/4913394

*Research Article*

# IFC and Monitoring Database System Based on Graph Data Models

**Luka Gradišar and Matevž Dolenc**

*Faculty of Civil and Geodetic Engineering, University of Ljubljana, Ljubljana 1000, Slovenia*

Correspondence should be addressed to Luka Gradišar; luka.gradisar@fgg.uni-lj.si

Received 13 August 2021; Revised 6 October 2021; Accepted 13 October 2021; Published 31 October 2021

Academic Editor: Sang-Bing Tsai



An efficient database management system that supports the integration and interoperability of different information models is a foundation on which the higher levels of cyber-physical systems are built. In this paper, we address the problem of integrating monitoring data with building information models through the use of the graph data management system and the IFC standard (Industry Foundation Classes) to support the need for interoperability and collaborative work. The proposed workflow describes the conversion of IFC models into a graph database and the connection with data from sensors, which is then validated using the example of a bridge monitoring system. The presented IFC and sensor graph data models are structurally flexible and scalable to meet the challenges of smart cities and big data.

## 1. Introduction

Data management (DM) can be defined as a set of best practices, architectural techniques, and tools that enable consistent data access and delivery for a wide area of all applications and business processes or, in the context of the Internet of Things (IoT), the foundation on which the higher levels of cyber-physical systems are built. In order to take full advantage of the Construction 4.0 [1] concepts and use cases, we must first design an efficient data management system that will support integration of different information models, enable data interoperability, and support integration of real-time data streams. Different information models include, but are not limited to, building information model (BIM) and sensor data models.

Mannino et al. [2] in their literature review on the integration of BIM and IoT integration for facility management (FM) concluded that there are still a number of challenges that require research and development. One of the most important research challenges is the increasingly important issue of data interoperability and improvements to the Industry Foundation Classes (IFC) [3] standards for the support of FM. Moreover, with the increased deployment of various IoT systems, linking building information models with the real-time data streams is becoming an important research topic [2]. Currently, most solutions employ an external database system to store the sensor data. This external database system is then connected to the modelling or collaboration tools for analysis and visualization [4–7]. While using multiple databases is a working solution to a problem, it also presents some challenges, such as deploying and managing multiple database systems and scalability issues. To address these challenges, our goal was to propose a data management system that supports multiple data types and allows pairing various data models to the real-time sensor data. One of the possible solutions to the design problem is to use a graph-based database management system [8] to support both the model information and the monitoring data to facilitate future use cases (for example, digital/smart cities) that will require efficient processing and querying of diverse datasets (e.g., model data, sensor information, time series, and locations).

This paper provides a description of a developed graph-based database management system that enables efficient and scalable storage solution for diverse datasets (as introduced above) as well as a platform for various analyses and visualizations required by different use cases. A bridge

monitoring case study is used for demonstrating and assessing the proposed database system.

## 2. Related Work

Although several various database management systems have been proposed to support IFC data models, given the constant evolution in the field of data management and processing, we will only mention recent research that focuses on the area of BIM data interoperability and storage.

For example, Li et al. [9] proposed an object-relational storage model to support IFC models without loss of information. In their workflow, they proposed rules for mapping between IFC data types and the Oracle database where the parsed IFC data are stored in the corresponding tables. The workflow was validated on a building model, showing that no information was lost during the exchange of data.

In the field of building performance management, Zhang et al. [10] presented a scalable cyber-physical platform based on the NoSQL database, in which they integrated building information modelling and sensor network into a unified structure. The proposed platform supports flexible structure and horizontal scalability to deal with the challenges of big data.

Jeong et al. [11] proposed a bridge monitoring data management framework based on the NoSQL database system Apache Cassandra, in which they stored the bridge information model (BrIM), sensor data from the on-site computer, and inspection information. In their workflow, they used an XML-based bridge information model structure that was appropriately parsed and mapped to the NoSQL column families to store the bridge geometry, information, and engineering model along with sensor information and data. The workflow was validated on a real bridge, and they concluded that the proposed system enabled easier user queries and the use of large data for monitoring purposes.

On the topic of graph databases, Ismail et al. [12] proposed a methodology for developing graph data models based on the IFC standard. In their workflow, they used web services that use scripts to automatically parse IFC models based on the EXPRESS schema and generate CSV files containing the graph model data. These can be transferred to the Neo4j graph database using the generated commands and CSV files. The proposed graph data models were validated on a case study model where they showed simple queries, analysis of emergency routes from the spaces included in the IFC model, and the possibility to compare between different versions of the same model to find out the differences that occurred during the modelling.

Similarly, Zhao et al. [13] employed a graph database to improve the merging of IFC data. They proposed an IFC graph data structure that preserves the relationships between IFC instances. Their graph data model is based on three types of nodes that distinguish between entities, single values, and list values. This supports easier data mining, which is required for their proposed data merging. They showed that IFC data can be clearly represented by graphs and retain the structure of IFC models.

## 3. Graph-Based IFC and Sensor Representation

*3.1. Data Schema of Industry Foundation Classes.* The common format of IFC files is STEP Physical Format (IFC-SPF), a compact text format that stores data as a sequence of instances, each containing an instance number, a class name, and its attributes, describing the geometric and semantic information of a BIM model [14]. The data schema is defined by EXPRESS data models, which contain the specification of classes and data types used in BIM modelling. Each class is defined by the keyword ENTITY followed by the name and class properties, which include parent-child relationships and attributes specific to that entity (see Figure 1).

The parent-child relationship is specified by supertype and subtype, which specify the inheritance of properties, i.e., a single class inherits all the properties pertaining to each parent class in the hierarchy, plus a specific class property if it has one [15]. For example, in Figure 2, the first four attributes of the IfcBeam instance belong to the class IfcRoot, which is the first in the hierarchy, the next attributes are inherited from IfcObject, IfcProduct, and IfcElement, while the last attribute PredefinedType is the class attribute of IfcBeam. The attributes can be represented with a single value or an aggregation of values that can either be a reference to another instance, for example, the OwnerHistory attribute is given as the instance number of the IfcOwnerHistory instance, or a simple data type such as real, integer, number, logical, Boolean, binary, or string.

IFC data models, as defined in the EXPRESS language, are inherently object-oriented. Their focus is on describing objects, broken down into components and organized into classes that are interconnected [16]. Because of their complex structure, which is difficult for humans to read, it makes sense to use graph data models for their representation since they already use a graph structure in their definitions and graphs are easier to understand [17].

*3.2. Graph Database Management System.* Graph database management system or practically known as the graph database is a platform where you can use create, read, update, and delete operations on graph data models represented as nodes and relationships connecting them. The most common form of graph is the labelled property graph, where entities are represented as nodes connected by directed edges (relationships), both of which can be labelled and can contain any number of key-value attributes (see Figure 3). This structure is expressive and general, which allows to describe many real-world scenarios in graph forms [8].

Compared to relational databases, graphs can handle connected data with better performance, even as the amount of data increases, because queries are limited to only a portion of a graph. In addition, they provide a flexible structure since we can include new types of nodes or relationships without changing the existing function or queries

```
EXPRESS SPECIFICATION:
Entity name:  ENTITY IfcBeam
Hierarchy:    SUPERTYPE OF (IfcBeamStandardCase)
              SUBTYPE OF (IfcBuiltElement);
Attributes:    PredefinedType : OPTIONAL IfcBeamTypeEnum;
              WHERE
                ...
              END_ENTITY;
```

Figure 1: EXPRESS specification for the IfcBeam class.

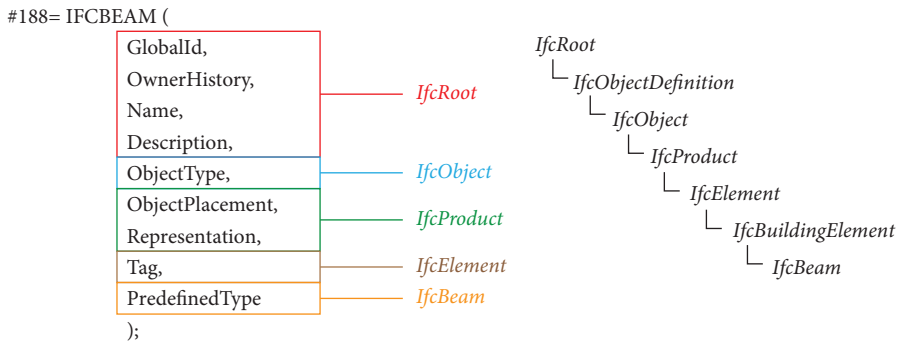


Figure 2: IFC attribute inheritance.

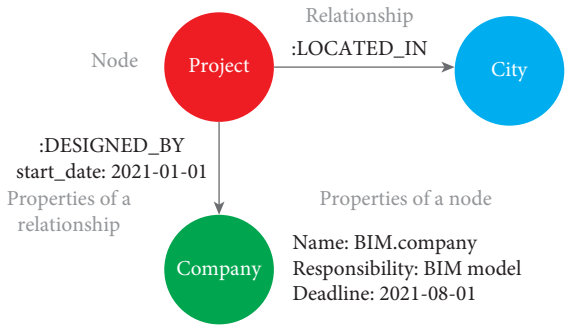


Figure 3: Example of the labelled property graph.

[8]. Although the use of graph models makes sense for highly interconnected data, there are some general advantages: the simplicity of graph representation; the graph data structure enables the modelling data as it is represented in the real world and can store all information in a node or in a relationship; queries can be used directly on the graph structure, from which a shortest path or various sup-graphs can be extracted; already developed graph algorithms and storage structures allow data to be efficiently stored and called upon [17]. Though there are many graph database systems, for this study, the Neo4j platform was used, which is known as a native graph database as it uses graph models for both storage and processing [18].

*3.3. IFC Graph Models.* To facilitate IFC data with graph models, we use a labelled property graph to semantically describe IFC entities as labelled nodes and attribute references as directed and labelled relationships. Our IFC graph data model is designed to preserve the IFC structure so that it can be mapped back to the IFC STEP model. However, it can be easily modified by adjusting the node and relationship structure in the Python script, and other authors [12, 13] proposed slightly different IFC graph models depending on the use case.

The types of nodes are derived from two sources: one directly from the instance classes (e.g., IfcBeam, IfcOwnerHistory, and IfcMaterial) and the other from the instance attributes labelled according to the EXPRESS schema (e.g., IfcLabel, IfcText, and IfcValue). The IFC schema differentiates between entities that are derived from IfcRoot, which have a global identification number, owner history, name, and description, and those in the resource layer that have no identity [16]. In our graph structure, we assign identity attributes to node properties as key-value pairs, rather than creating separate nodes for these values. All other nodes can have only one property (excluding the ID number), the value of simple data type attributes.

To define the relationship between two nodes, each node is assigned the ID number based on which they are connected. The node of an instance class inherits the exact instance number (e.g., #188), while the nodes derived from the attributes receive an additional number based on the order position of the attribute (e.g., #188_5). Each instance is

then either connected to another instance if the attribute contains the referenced instance number or connected to a created attribute node if it contains a simple value type. In addition, each relationship is assigned a label based on the attribute key name defined in the IFC EXPRESS schema (e.g., OwnerHistory, Tag, and Representation) and an ID number based on the position of the attribute (see Figure 4).

To address attribute aggregations, an additional node is created, from which the aggregation is then separated into single values, and the process is repeated, either creating relationships to other instances or creating new nodes with the separated values.

*3.4. Sensor Graph Models.* Sensor graphs are similar in structure to IFC graphs. A single sensor is represented by a sensor node that contains the name and description, while other attributes, such as sensor type, are defined as a separate node and connected to the sensor node to provide an organized structure and simplify querying.

Each sensor data measurement is represented as a node containing the date, time, and value of the measurement and is connected to a single node called sensor data, which in turn is then linked to an associated sensor node (see Figure 5). Depending on the purpose, they can be divided into nodes either by year or by any other measure.

*3.5. Transformation Methodology.* The developed methodology proposes a workflow to transform IFC models into graph data and combine them with monitoring data from sensors. The workflow can be divided into three parts: first, the IFC model is converted into a graph form, then the sensor entities are added and connected to the corresponding IFC object, and finally, the sensor data are added in batch or real-time capture and attached to the sensor entity (see Figure 6). With this approach, we are able to store different IFC models along with the sensor data in a single graph database.

The conversion of IFC models to graph data models is based on the IFC EXPRESS schema. This workflow employs Python scripts to automate the parsing of IFC STEP files and, using the Python dictionary for IFC entities, converts instances and their attributes into graph nodes and relationships with correct labels and properties. These are exported as CSV files and imported into Neo4j using the batch admin import function, which is suitable for large datasets [19].

Python dictionaries are created for each IFC class and all its attribute key and value names as described in the selected IFC EXPRESS schema retrieved from the buildingSMART website [20]. Since, in the IFC EXPRESS schema, each class contains only its specific class attributes without inherited properties to avoid repetition, the script iterates over the hierarchical structure and appends all parent attributes to the specific class. Figure 7 shows the Python dictionary of the class IfcBeam generated from the EXPRESS schema. It can be seen that IfcBeam contains 9 attributes, which will match the number of IfcBeam instance attributes in the IFC STEP

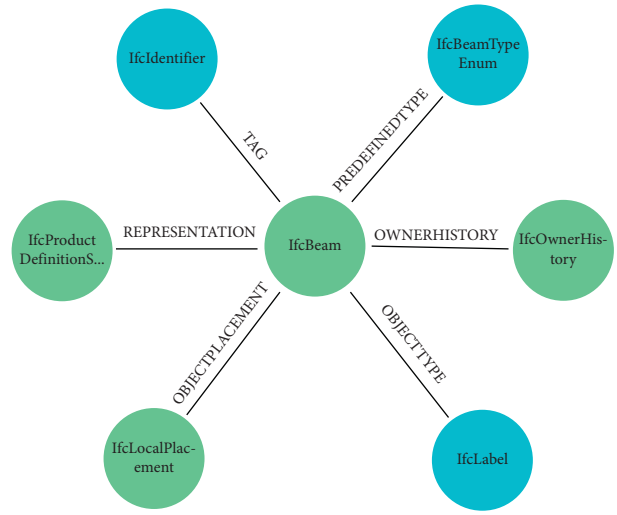


Figure 4: Graph data model for the IfcBeam entity.

file. This step is performed once to create a dictionary of IFC classes based on the selected IFC schema.

Mapping the IFC model to the graph form consists of parsing each instance in the IFC STEP file for instance number, class name, and attribute values. For each IFC instance, a node instance is created using the reference number as ID, the class name as the label, and the identity attributes as the node properties if the instance has any. Next, the script goes through each instance attribute, and if it is a reference number, it creates a relationship between the current instance and the referenced instance, containing a label derived from the attribute name, which is quickly found using the Python dictionary from earlier. If an attribute value is of a single data type, the script creates a node with a label of the attribute value type and a relationship connecting an instance and an attribute with a label of the attribute name, which again is found in the dictionary (see Figure 8). Created node and relationship instances are exported to CSV files, which are brought into the Neo4j database using the Admin Import function.

Sensor entities are added to the graph database using the Neo4j query language Cypher [6], which can be connected to Python via the Neo4j Python driver. Sensor nodes and relationships connecting them to the IFC sensor element can be simply created using the MERGE command and statements to create nodes (variable:label {name:"value"}) and edges (node 1)-[:LABEL]-(node 2). Similarly, sensor data are added in real time by executing the Cypher commands or in batch by using the CSV file and the LOAD CSV function in Neo4j.

## 4. Case Study

IFC and monitoring database system based on graph data models was illustrated with a simulated test problem involving the BIM model of a bridge with monitoring sensor elements at various positions on the beams (see Figure 9). The monitoring elements represent strain gauge sensors and

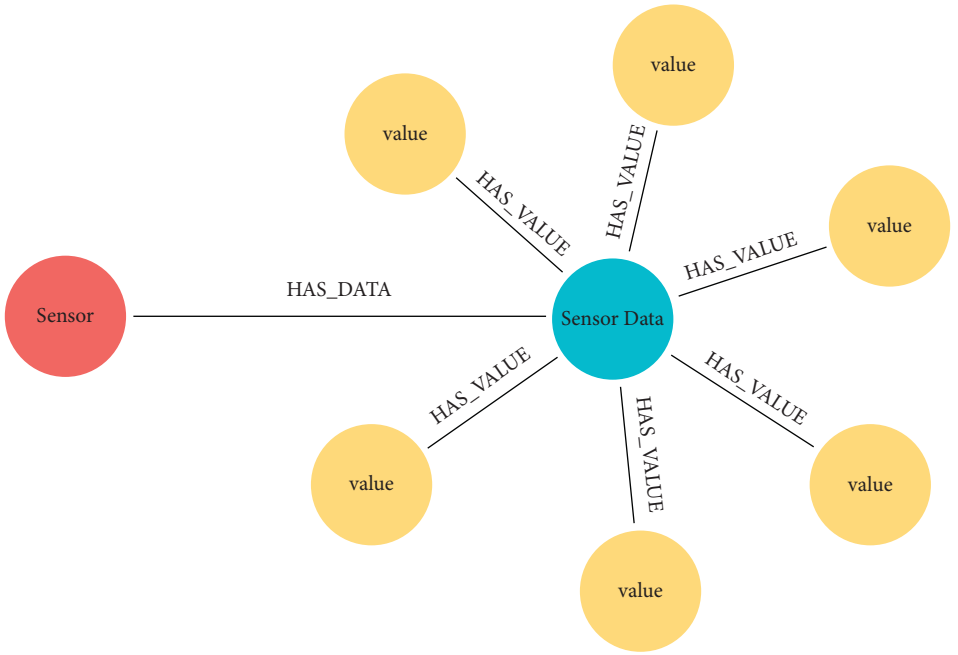


Figure 5: Graph data model for the sensor and its measurements.

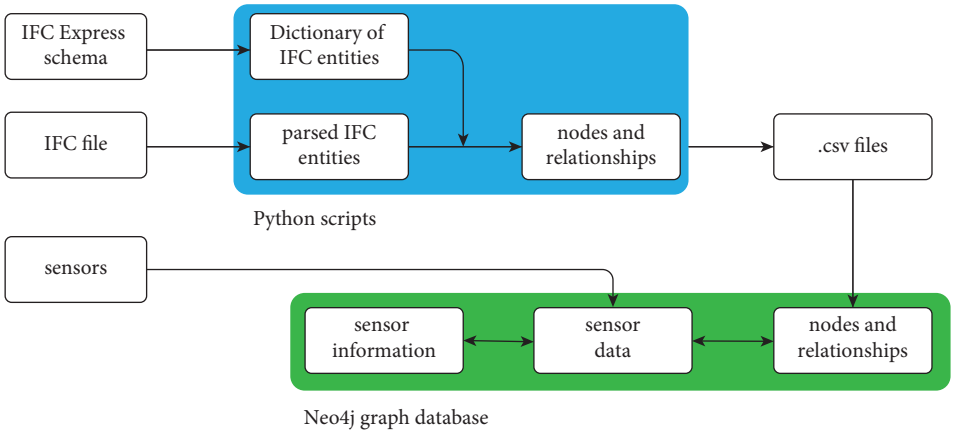


Figure 6: Workflow for converting IFC and sensor data into a graph database.

```
IFCBEAM:
 { GlobalId : IfcGloballyUniqueId,
 OwnerHistory : IfcOwnerHistory,
 Name : IfcLabel,
 Description : IfcText,
 ObjectType : IfcLabel,
 ObjectPlacement : IfcObjectPlacement,
 Representation :IfcProductRepresentation,
 Tag : IfcIdentifier,
 PredefinedType : IfcWallTypeEnum}
```

Figure 7: Python dictionary of class IfcBeam with its attribute key-value pairs.

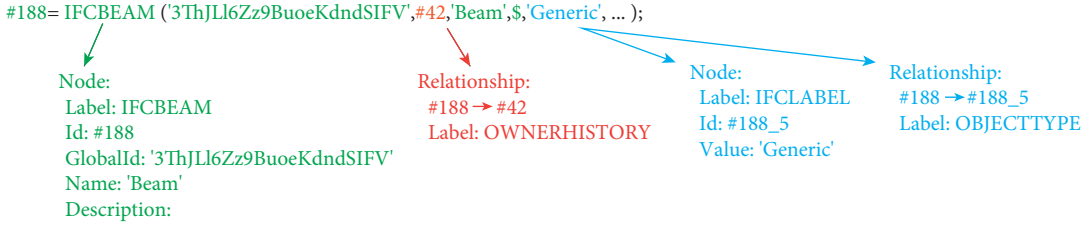


Figure 8: Example of nodes and relationships created from an IfcBeam instance.

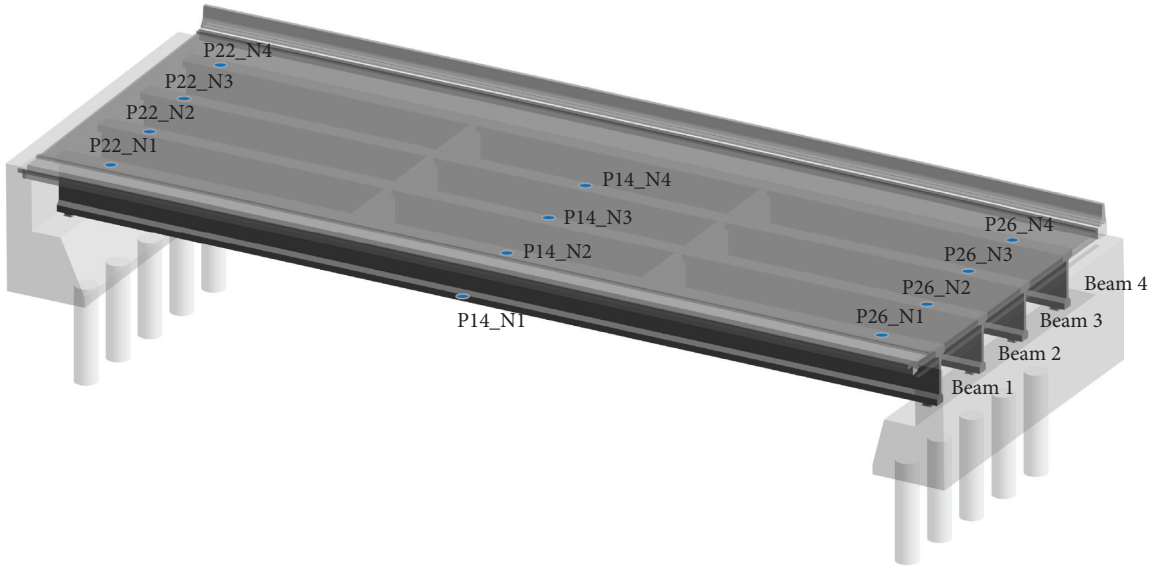


Figure 9: Example IFC model of a beam with sensors.

are accompanied with measurement data over several days at one-minute intervals.

The BIM model was created in Revit modelling software and exported using IFC4 Design Transfer View settings. The IFC model was mapped into nodes and relationships using Python scripts based on the IFC4 EXPRESS specification and exported to CSV files, which were imported into Neo4j using the Admin Import function to create IFC graph data models. The IFC graph structure can be seen in Figure 10, where three distinct clusters can be identified. The first cluster contains the semantic information of the model and the building elements, the second cluster represents information about the units used in the model, and the third cluster is the geometry information of the model.

The sensor graph models were created by executing the commands in the Cypher query language:

```
MERGE (s:Sensor{Name:"P14_N4_MM7-T",Description:"Weighing"})
MERGE (t:SensorType{Name:"Strain_Gauge"})
MERGE (m:Manufacturer{Name:"TML"})
MERGE (s)-[:HAS_PROPERTY]->(t)
MERGE (s)-[:HAS_PROPERTY]->(m)
```

The first three commands create a sensor instance node and attribute nodes for the type and manufacturer of the sensor, while the last two commands connect the instance with its properties.

Sensor data consisted of 10 days of measurements of one-minute average strains and the date and time of each measurement. The data were brought in a CSV file, batch transferred to the graph database using the LOAD CSV function, and attached to the sensor instance node. To simulate real-time data, a Python script was connected to the graph database via the Neo4j Python driver, from where Cypher commands were executed in real time using the MERGE command to first create a node with measurements and then connect it to the sensor data node.

To retrieve the beam or sensor element from the IFC graph model, a simple query was used:

```
MATCH p=(n:IFCBUILDINGELEMENTPROXY)-[*1]->(p1) RETURN p
```

This returns all nodes with a label IFCBUILDINGELEMENTPROXY to which the beam and sensor element belong because they were created with a custom generic family. In addition, the query retrieves all attribute nodes that are associated with the building proxy element node (see Figure 11).

Retrieval of sensor elements and associated measurement data can be done by the following query:

```
MATCH (s:Sensor{Name:"P14_N4_MM7-T"})
MATCH d=(n:SensorData{Name:"P14_N4_MM7-T"})-[:HAS_VALUE]->(v:Values)
WHERE date(v.measured) = date("2019-07-01")
RETURN n,s,d
```

Here, we can restrict the measurement data to a specific date or time by using the conditional statement WHERE and return the sensor element and its data measured on the specified date (see Figure 12). In this way, we can retrieve any sensor, its measured data, and its connection with the IFC element or a location of the sensor.

## 5. Discussion

Using graph data models, we have shown that they are able to handle different types of data involved in facility management and structural health monitoring. Using graphs to facilitate IFC models and any real-time monitoring data can be seen as an integrated alternative to the

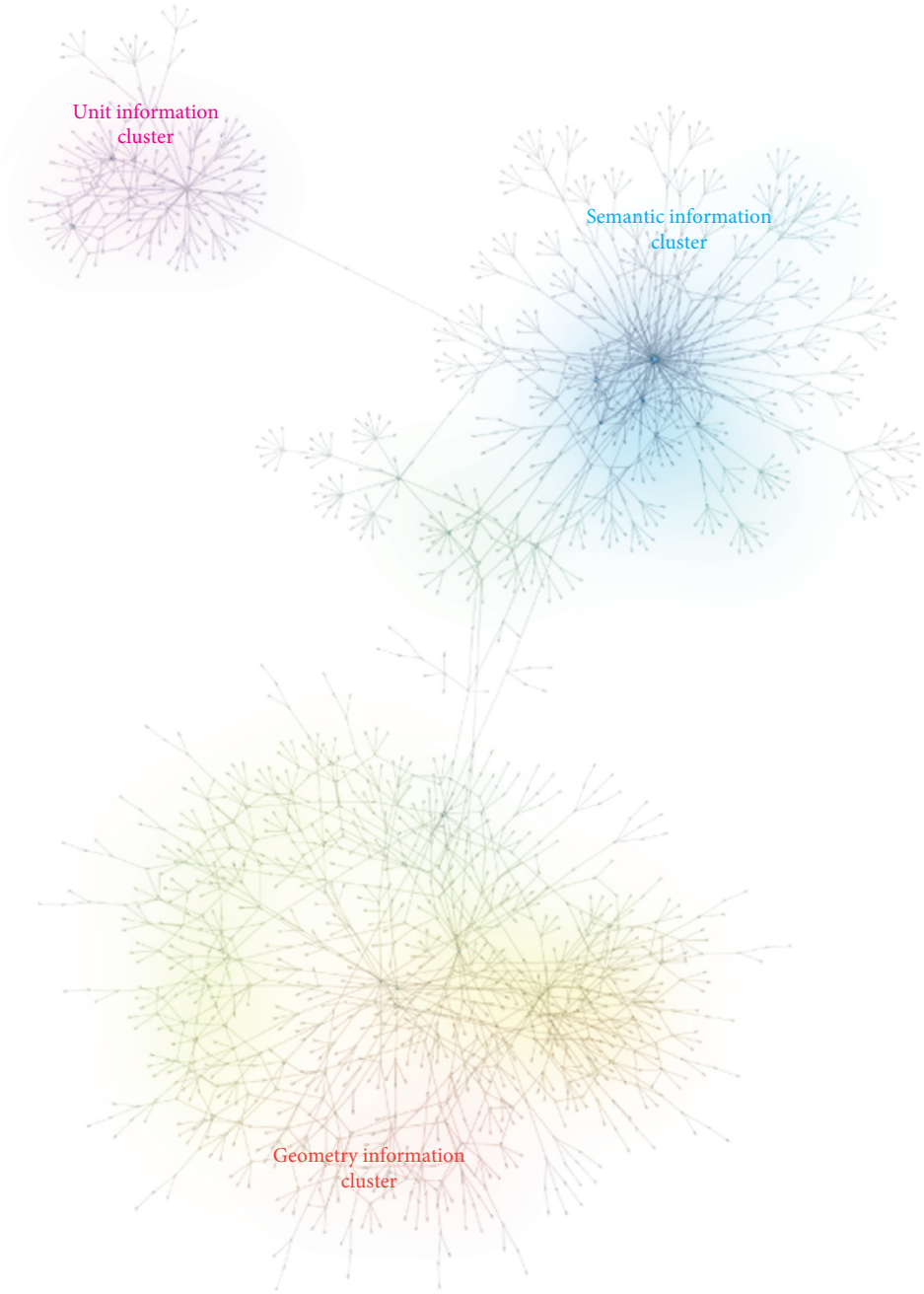


Figure 10: Graph representation of the IFC model of a single BIM element.

current practice of using isolated systems, each managing one type of data. The deployment of a single flexible management system can be seen as an important step towards digital twins and smart cities as advances in facility management and sensor technologies will increase the number of management applications and data generated, increasing the demand for efficient data sharing, storage, and interoperability.

The use of a graph-based IFC management system can provide the BIM manager with many advantages. First, we can address the complex and difficult-to-read schema of raw IFC files. By employing graph data models to represent IFC data, we are able to see their interrelated structure and use graph algorithms to perform complex queries to traverse the subgraphs and modify their data while still preserving the original structure. Additionally, the graph structure is flexible and allows changing the data structure and adding new data points and connections. Moreover, in our proposed workflow, we have included semantic enrichment of IFC graph data models by making use of the IFC EXPRESS schema to add names and types to attribute nodes and connections, making it easier for the user to understand and find the information pertaining to IFC elements. Second, graph data management systems can be used as an underlying system for structural health monitoring or other higher levels of cyber-physical systems, allowing BIM

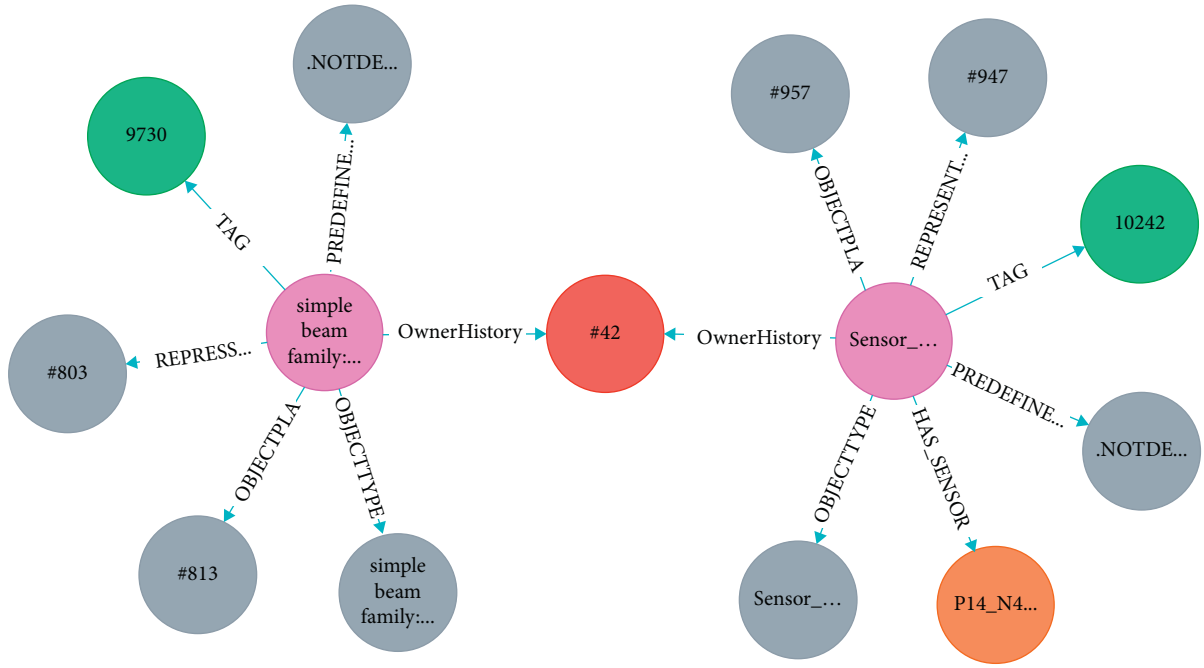


Figure 11: Getting the beam and sensor instances and their attributes.

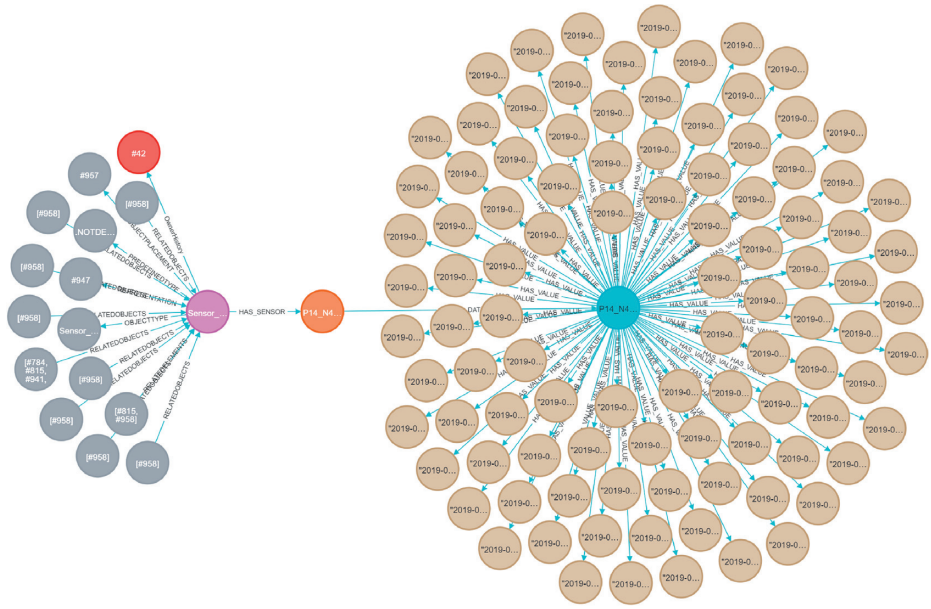

Figure 12: Retrieval of the sensor element and data for a certain date.

managers to store the data in one place and access them efficiently for other applications. However, as others have shown, graph data models are not limited to use as storage, but allow BIM managers to use them for other applications, such as comparing models to find changes between different versions [12], analysis of emergency routes [12], and IFC data merging [13]. Third, to ensure interoperability, both IFC and Neo4j graph data models follow open standards and can be used by other applications, allowing for collaborative work without closed environments.

## 6. Conclusions

In this research work, a flexible and scalable IFC graph data model is proposed, focusing on preserving the IFC data structure and supporting the linkage between information models and associated real-word monitoring data. In particular, the proposed workflow describes the transformation of IFC models into a graph database where sensor entities link the building information model with sensor data. This approach was validated through a case study of a bridge IFC

model linked to monitoring data, showing that IFC data can be represented as a collection of nodes and relationships.

Proposed graph data models can be modified by introducing other schemas into the workflow. Together with the unified database system, this provides a flexible and scalable structure that can be adapted to future use cases and provides a consistent foundation for cyber-physical systems.

To increase the validity of this research, future work will address higher levels of cyber-physical systems by utilizing the graph database as the underlying system from which data analysis and visualizations will be performed on a use case of a real-world structure with a large sensor network.

## Data Availability

The IFC file and monitoring data used in this research are available at https://github.com/IFCManager/IFC_Graph_Database.

## Conflicts of Interest

The authors declare no conflicts of interest.

## Acknowledgments

This research was supported by the Slovenian Research Agency under the Young Researcher Funding Program.

## References

[1] R. Klinc and Ž. Turk, "Construction 4.0 – digital transformation of one of the oldest industries," *Economic and Business Review*, vol. 21, no. 3, pp. 393–410, 2019.

[2] A. Mannino, M. C. Dejaco, and F. R. Cecconi, "Building information modelling and Internet of Things integration for facility management - literature review and future needs," *Applied Sciences*, vol. 11, no. 7, pp. 25–3062, 2021.

[3] "Industry foundation classes," 2021, https://www.buildingsmart.org/standards/bsi-standards/industry-foundation-classes/.

[4] Z. Riaz, E. A. Parn, D. J. Edwards, M. Arslan, C. Shen, and F. P. Mora, "BIM and sensor-based data management system for construction safety monitoring," *Journal of Engineering, Design and Technology*, vol. 15, no. 6, pp. 738–753, 2017.

[5] J. M. D. Delgado, L. J. Butler, I. Brilakis, M. Z. E. B. Elshafie, and C. R. Middleton, "Structural performance monitoring using a dynamic data-driven BIM environment," *Journal of Computing in Civil Engineering*, vol. 32, no. 3, p. 17, 2018.

[6] D. Kazado, M. Kavgic, and R. Eskicioglu, "Integrating building information modeling (BIM) and sensor technology for facility management," *Journal of Information Technology in Construction*, vol. 24, pp. 440–458, 2019.

[7] X. Yin, H. Liu, Y. Chen, Y. Wang, and M. A. Hussein, "A BIM-based framework for operation and maintenance of utility tunnels," *Tunnelling and Underground Space Technology*, vol. 97, no. 12, Article ID 103252, 2020.

[8] I. Robinson, J. Webber, and E. Eifrem, *Graph Databases*, pp. 1–32, O'Reilly Media, California, U S A, 2013.

[9] H. Li, H. Liu, Y. Liu, and Y. Wang, "An object-relational ifc storage model based on Oracle database," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLI-B2, pp. 625–631, 2016.

[10] Y. Zhang, K. Kang, J. R. Lin, J. P. Zhang, and Y. Zhang, "Building information modeling–based cyber-physical platform for building performance monitoring," *International Journal of Distributed Sensor Networks*, vol. 16, no. 2, p. 21, 2020.

[11] S. Jeong, Y. Zhang, S. O'Connor, J. P. Sohn, and K. H. Law, "A NoSQL data management infrastructure for bridge monitoring," *Smart Structures and Systems*, vol. 17, no. 4, pp. 669–690, 2016.

[12] A. Ismail, A. Nahar, and R. Scherer, "Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard," in *Proceedings of the 24th International Workshop on Intelligent Computing and Engineering*, Nottingham UK, July 2017.

[13] Q. Zhao, Y. Li, X. Hei, and M. Yang, "A grap-based method for IFC data merging," *Advances in Civil Engineering*, vol. 2020, Article ID 8782740,, 15 pages, 2020.

[14] "IFC formats," 2021, https://technical.buildingsmart.org/standards/ifc/ifc-formats/.

[15] "The EXPRESS definition language for IFC development," 2021, https://standards.buildingsmart.org/documents/Implementation/The_EXPRESS_Definition_Language_for_IFC_Development.pdf.

[16] A. Borrmann, J. Beetz, C. Koch, T. Liebich, and S. Muhic, "Industry foundation classes: a standardized data model for the vendor-neutral exchange of digital building models," in *Building Information Modeling*, pp. 81–126, Springer, Berlin, Germany, 2018.

[17] R. Angles and C. Gutierrez, "An introduction to graph data management," *Data-Centric Systems and Applications*, Springer, Berlin, Germany, pp. 1–32, 2018.

[18] "What is a graph database," 2021, https://neo4j.com/developer/graph-database/.

[19] "Neo4j admin import," 2021, https://neo4j.com/docs/operations-manual/current/tutorial/neo4j-admin-import/.

[20] "IFC specifications database," 2021, https://technical.buildingsmart.org/standards/ifc/ifc-schema-specifications/.