Using BCF API and IFC-graph in a system for digital inspection plans to support a circular construction sector

8th of October 2023

Final report of one-year studies in building information modelling

Student: Martin Wiss (martin@wiss.se)

Department of Manufacturing and Civil Engineering

Norwegian University of Science and Technology (NTNU)

Supervisors: Knud Mohn (NTNU), Fabian Ståhl (Boverket)

Abstract

Inspection plans are required for most building construction and demolition measures subject to a permit or notification according to the Swedish building act (Boverket, 2021). The inspection plan not only contains information on the inspections to be done and their results, but also the types and amounts of building products that can be reused, the management of reusable building products, the types and amounts of waste produced, the management of waste, including a description of possibilities for recovery of materials and safe treatment of toxic substances, etc. (Riksdagen, 2010).

The national board of housing building and planning in Sweden (Boverket) have been investigating the feasibility of a national system for digital inspection plans (Finansdepartementet, 2022) and have presented the results in a report called “digitalization of inspection plans” (Boverket, 2023). Boverket mentions that “one of the most interesting solutions (for such a system) is based on the BCF API” (“en av de mest intressanta lösningarna bygger på BCF”).

To support a transition to a circular economy in the construction sector, it is important that a system for digital inspection plans can communicate information about reusable building products and waste management required by legislation.

One aim of this study was to see if the BCF API can be used to communicate any information regarding reusable building products and management of waste. If such a possibility existed, it was also interesting to know what kind of such information the BCF API can communicate, and how.

To test the possibilities of the BCF API, a BCF API server was developed using the FastAPI framework for python and the graph database Neo4j. The server was tested using Solibri Office as BCF client.

The result shows that a national system for digital inspections plan cannot be built using the BCF API only. At least not using the existing version of the BCF API, version 3.0. The BCF API was not designed to communicate the information necessary for such a system. The BCF API server could not be adopted to make Solibri Office send information required for digital inspection plans.

To create a national system of digital inspection plans it is necessary to use a dedicated API, such as an inspection plan API or a building permit API, for communicating the information of building permits and digital inspection plans. Such a system could also implement the BCF API services to let BCF clients view building models in the system, but this would be an extra feature, and not the core functionality of the system.

The other aim of this study was to see if there are benefits of using IFC-graph to store building models and building information in a system for digital inspection plans. It was shown that IFC-graph can be used for building information and query, also together with graph data from the BCF API. IFC-graph can extend the functionality of BCF API by providing additional building information for building components that have been communicated to the BCF API server.

There could be potential benefits of using IFC-graph in a system for digital inspection plans, this also includes benefits for supporting a circular construction sector. However, there are simpler solutions to access building information compared to IFC-graph. Which solution to recommend for building information access and query in a specific system depends on several factors, that are described later in this study.

Table of contents

List of figures

Glossary

AEC, Architecture, Engineering and Construction

AJAX, Asynchronous JavaScript and XML.

API, Application Programming Interface.

BCF, BIM collaboration format.

BIM, Building Information Modelling.

Boverket, the national board of housing building and planning in Sweden.

CDE, Common Data Environment.

GUID, Globally Unique Identifier (UUID is used instead).

IFC, Industry Foundation Classes.

LPG, Labeled Property Graph.

OGM, Object Graph Mapper.

OIR, Organizational Information Requirements.

TLS, Transport Layer Security.

UUID, Universally Unique Identifier (almost same as GUID).

Introduction

Background and purpose

Inspection plans for reuse and waste management

Inspection plans are required for most building construction and demolition measures subject to a permit or notification according to the Swedish building act (Boverket, 2021). An inspection plan contains information about the inspections needed to ensure that requirements of regulations are met. The inspection plan is often a checklist which states what is to be inspected, who is responsible for conducting the inspection, how and when the inspection is to be done etc. (Boverket, 2021).

In the year 2020, additional regulations with further requirements on information in inspection plans were added (Regeringen, 2020). Since then, inspection plans not only have to contain information about hazardous waste, but also information about the types and amounts of building products that can be reused, the management of reusable building products, the types and amounts of waste produced, the management of waste, including a description of possibilities for material recovery and safe treatment of toxic substances, etc. (Riksdagen, 2010).

Flaws on implementation of new regulations on inspection plans

A survey study (Fager & Johansson, 2022) indicates some flaws in the implementation of this new legislation for inspection plans. The survey study stresses the need for better knowledge and awareness about possibilities for reuse and recycling of building products and material to better implement the legislation and to increase reuse and recycling in the construction sector. Better knowledge is especially important for the person in charge of the inspections (“kontrollansvarig”) (Fager & Johansson, 2022).

Inspection plans today

Today inspection plans commonly consist of paper documents or digital documents such as email, excel spreadsheets, word or pdf documents. In such documents there can be inventories of building products and building materials that can be reused, including information about weight, volumes or numbers, as well as information about the management of such products and materials.

While such documents can be stored on electronic media such as hard drives, the information contained in such documents is not easily accessible, because the information is not structured and not aggregated. To retrieve aggregated statistics about reuse, it would be necessary to search through all unstructured data of each document in each system in each municipality.

A national system of digital inspection plans

The national board of housing building and planning in Sweden (Boverket) have been investigating the feasibility of a national system for digital inspection plans (Finansdepartementet, 2022) and have presented the results in a report called “digitalization of inspection plans” (Boverket, 2023). Boverket mentions that “one of the most interesting solutions (for such a system) builds on the BCF API” (“en av de mest intressanta lösningarna bygger på BCF”). This solution is described in the report as follows:

”Den tekniska underlagsrapporten visar på möjligheter att använda några typer av format som lämpar sig olika bra för olika former av distribution. En av de mest intressanta lösningarna bygger på Bim Collaboration Format (BCF). BCF är ett slags meddelandeformat som ursprungligen är framtaget för att skicka strukturerade meddelanden inom BIM-projekt för samordning och kvalitetsgranskning. Erfarenheten av att arbeta med formatet är att det fungerar allra bäst i en servermiljö där det finns

en mellanlagring eller meddelandeserver. Med den typen av lagringslösning skulle alla aktörer få tillgång till kontrollpunkter oavsett systemstöd."

"The technical report shows that there are opportunities to use various kinds of formats suitable for various kinds of distribution (of information). One of the most interesting solutions is based on the Bim Collaboration Format (BCF). BCF is a kind of "information delivery format" originally developed to send structured information for coordination and quality review in BIM projects. Experience from working with this format shows that it works best as a server solution. With a server solution, all users would have access to inspection plans regardless of the individual choice for system support."

In a project funded by Svenska Byggbranschens Utvecklingsfond (SBUF) a prototype system for digital inspection plans was developed as a web application called "Automatiserade Standardiserade Kontroller" (ASK) (BIMformation AB, 2020). The possibilities of using files with IFC data in this system was discussed but not considered possible to achieve within the budget ("*Att bygga en drag-och-släpp-funktion för filer med IFC data och läsa av GUID är dock något som kräver betydlig budget och har inte kunnat åstadkommas inom ramarna för detta projekt*." *"To build drag and drop functionality for files with IFC data to read GUID requires a substantial budget and has not been possible within the limits of this project."*) (BIMformation AB, 2020).

Supporting a circular construction sector

The built environment requires significant resources and accounts for approximately 50 percent of all extracted material in the EU. The construction sector accounts for over 35 percent of the EU's total waste generation (European Comission, 2023). The construction sector also accounts for more than 20 percent of Sweden's total carbon dioxide emissions from a life cycle perspective (Boverket, 2023).

Boverket has been assigned the task of contributing to a circular economy in the building sector. For example, by investigating how digitalization can be used for this purpose (Finansdepartementet, 2022).

Better statistics on reuse

Today statistics of waste generated in the construction sector, is mainly based on statistics of waste received at waste facilities, but this kind of statistics does not show if waste is generated from construction or demolition measures and does not contain any information about reuse of building products or materials (Sundqvist, 2022).

A national system of digital inspection plans, where structured information from all digital inspection plans could be accessed through an API, would allow the creation of some statistics on reuse and some statistics on waste generation from both construction and demolition measures separately. Statistics on reuse and waste generation could then serve as one of several indicators that measure the progress towards a circular economy in the construction sector.

Better knowledge for reuse

A national system of digital inspection plans that contain information about reusable building products and generation of waste, could potentially help the developer, as well as the person in charge of the inspections ("kontrollansvarig"), on proper management for reuse, recycling, and recovery of building materials, and on proper management of building materials containing toxic substances.

A common task in demolition projects is the identification of building materials containing toxic substances and the process of taking decisions on how to safely handle and take care of such building materials. Information about toxic substances identified in past demolition measures could

potentially be used to train machine learning algorithms that give advice on where to look for various kinds of toxic substances in future demolition measures (Pei-Yu, Sandels, Mjörnell, Mangold, & Johansson, 2022).

A national system of digital inspection plans could provide all the structured and digital information needed to improve the training of such a machine learning model, and this model could in turn potentially feedback advice to the system, to the person in charge of the inspections ("kontrollansvarig") and to the developer of the construction or demolition measure.

Match making

A paper on barriers, success factors, and perspectives for the reuse of construction products in Norway (Knoth, Seilskjaer, & Mamo Fufa, 2022) stresses that match-making mechanisms to match supply and demand need to be put in place to connect those with a surplus of materials and reusable products to those who might need them. Match making initiatives are also identified as important to establish effective markets for secondary building materials in a report by the Danish environmental protection agency (Environmental Protection Agency, 2019).

If developers consent to sharing information in digital inspection plans, about reusable building products and waste that needs to be taken care of in a construction or demolition measure, then this information could potentially be used for matchmaking between stakeholders on the waste, recycling and reuse markets. Information about future available reusable products could potentially maybe also be used as input for the design of new buildings. A system of digital inspection plans for construction and demolition measures could potentially be used to provide information needed to create such match making mechanisms.

The BCF API for digital inspection plans

As can be seen from previous examples, to support the transition to a circular economy in the construction sector, it is important (for many reasons) to facilitate the communication of digital information on reusable building products and waste management. This kind of information should already be included in every inspection plan according to existing legislation. I.e., the information is already produced, it is just not available in a format which allows the practical production of statistics, and not available in a format that allows practical analysis of the information. The sharing of this information in ways that benefit a circular construction sector is also not possible.

As mentioned earlier in this study, the BCF API was seen as an interesting alternative for a national system of digital inspection plans in a report by Boverket (Boverket, 2023). It is hence interesting to understand if the BCF API can facilitate communication of the information that an inspection plan is supposed to contain according to legislation. To support the transition to a circular economy in the construction sector, it is especially interesting to understand if the BCF API can be used to communicate information about reusable building products and management of waste. This question can be formulated as "can the BCF API be used to meet any reasonable organizational information requirements (OIR) of a national system of digital inspection plans" (Norsk standard, 2019).

BCF API was not designed with an intention to facilitate communication of information about reusable building products and management of waste (buildingSMART, 2021). BCF API is used to communicate information about topics related to issues in digital building models, and those topics in turn can have viewpoints and comments and so on.

This study aims to see if the BCF API can be used to communicate information about reusable building products and management of waste, required by national legislation on inspection plans. If

such a possibility exists, it is also interesting to know what kind of such information the BCF API can communicate, and how.

Relating BCF data to IFC data

As will be explained later in this study, the amount of building information relevant for digital inspection plans that can be communicated between Solibri Office and any BCF API server is limited. The information that can be communicated in this way is limited both by the BCF API specification, by the capabilities of the BCF API server and by the capabilities of Solibri Office as a BCF-client.

But building information important for decisions on reuse, recycling or waste management of building products and materials (such as GTIN-number), can also be stored as properties in property sets in files with IFC data. Solibri Office can transfer UUID of building components related to a topic according to the BCF API specification. If the BCF API server has access to the same file with IFC data as Solibri Office, then the server could potentially find all kinds of interesting information in properties of related entities in this file, by using the building component UUID transferred from the BCF client.

Solibri Office can merge several building models from several files with IFC data into a federated building model to perform clash detection and cross-discipline issue management. This federated model must then be saved as a Solibri Model Checker (.smc/SMC) file. Solibri Office allows download of files with IFC data from servers that implement the Documents API. Solibri Office does, however, not allow uploading files with IFC data to Documents API servers. Solibri Office can only upload building models after they have been saved as SMC -files and there are no known supported ways of round-tripping data in SMC-files back to its original IFC data. SMC is a proprietary file format.

The IFC standard was created to facilitate the exchange of building information between software. However, Solibri Office converts open standard IFC data from STEP files into the proprietary file format SMC and there is no way to convert data in SMC files back into open standard IFC data. This makes the exchange of building information from Solibri to other software more difficult.

To let a BCF API server access building information from a file with IFC data used by Solibri Office, then the file containing IFC data must first be put on the server by a tool other than Solibri Office. But when this IFC data from the file already exists on the BCF API server, then this IFC data can be used to extend the building information that will be possible to communicate using Solibri Office and the BCF API.

IFC-graph

The STEP file format is the most used file format for storing IFC data. For software to make sense of the data in a STEP file, it must parse all the lines in the STEP file and create corresponding data structures in computer memory. It is not possible to understand the data in a STEP file by reading line by line from start to end, because a line in the beginning of the file can have a link to a line at the end of the file, and vice versa. Dedicated software libraries to parse IFC STEP files exist. STEP files tend to be relatively large, especially for complex models of large buildings. There are normally several hundred thousand lines or sometimes even several million lines in a STEP file with IFC data. Hence unnecessary amounts of data must be parsed and loaded into computer memory to access even very small amounts of information from a building model saved as a STEP file.

An alternative approach for building information access and query of IFC data, is to convert the IFC data from the STEP file format into a labeled property graph (LPG). A paper written by Junziang et al. provides an algorithm for converting IFC data into LPG using py2neo for the Neo4j graph database. The resulting LPG is called IFC-graph (Junziang, Peng, & Xiang, 2023).

When IFC data exists as IFC-graph in a Neo4j database, the Cypher query language can be used to query and access building information from this IFC-graph. Neo4j enables complex queries using Cypher and implements optimized algorithms for searching through vast amounts of complex LPG data at high speeds to return results in practical data structures (such as JSON etc.).

This study aims to see if IFC-graph can be used to facilitate access and query of building information useful for decisions on reuse, recycling or waste management of building products and materials in a system for digital inspection plans. Especially if used in conjunction with data collected in a server using the BCF API.

Problem statement

Using the BCF API

The first aim of this study is to see if the BCF API can be used to communicate information about reusable building products and management of waste, required by national legislation on inspection plans. If such a possibility exists, it is also interesting to know what kind of such information the BCF API can communicate, and how.

Using IFC-graph

The second aim of this study is to see if IFC-graph can be used to facilitate access and query of building information useful for decisions on reuse, recycling or waste management of building products and materials in a system for digital inspection plans. Especially if used in conjunction with data collected in a server using the BCF API.

Method

Creating a BCF server

To test the capabilities of information exchange using the BCF API, a BCF API server was considered necessary. At the time of this study there was no open source BCF API server available. This is why a BCF API server had to be developed. A BCF API server implementing the BCF API specification (buildingSMART, 2021) was developed using python, the FastAPI framework (FastAPI, 2023) and Neo4j.

Python was selected because it is one of the most common programming languages and often used in research and data science. Neo4j was selected for storing BCF data to make potential integrations between BCF data and IFC-graph possible. FastAPI uses pydantic models to convert data between python objects and JSON and to validate data used for API requests and responses. FastAPI routes and pydantic models were autogenerated using OpenAPI (formerly Swagger) specifications for BCF API (buildingSMART, 2023).

Implementing the Foundation API and secure communication

The Foundation API must be implemented before the BCF API can be implemented (buildingSMART, 2021). The Foundation API requires OAuth2 and secure encrypted communications over HTTPS. The authorization code flow of OAuth2 was implemented using JWT and self-destructing nodes in Neo4j as sessions. The self-destructing nodes were implemented using TTL functions in the APOC library for Neo4j (Neo4j, 2023).

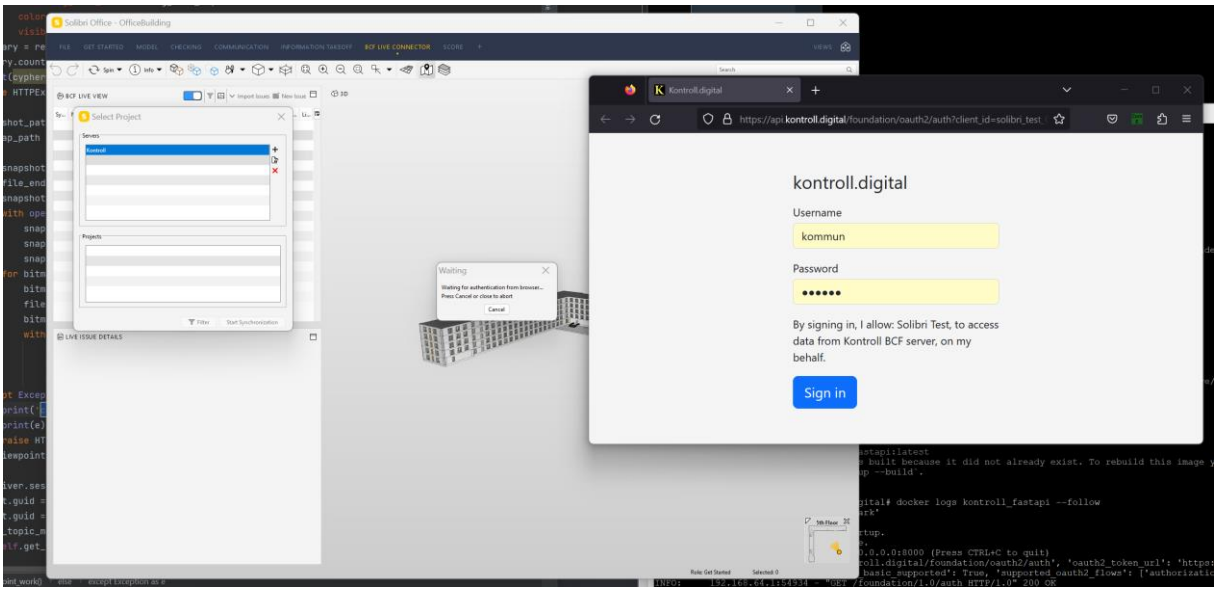


Figure 1 The process of logging in to the server using Solibri Office, Oauth2 and the web user interface developed for the Foundation API.

To enable safe encrypted communication over HTTPS it was necessary to register a domain name and acquire a TLS certificate for that domain name. The domain name “kontroll.digital” was purchased and a TLS certificate for this domain name was acquired from “Let’s encrypt”.

Deployment and test setup

The BCF API server was deployed in one container for the API and another container for the database (Neo4j) using Docker Compose on a Linux server, more specifically an Ubuntu server. Certbot was used for autogeneration of TLS certificates (Electronic frontier foundation, 2023) for the domain “kontroll.digital”. Nginx was used as a reverse HTTPS proxy to connect the Docker internal network to the remote web.

To test the BCF API server, Solibri Office together with the BCF live connector was used as a BCF client. To make Solibri Office connect to the BCF API server at the “kontroll.digital” domain name, it was necessary to register this domain name in a beta version of Solibri Office. Solibri Office only connects to preregistered server.

During development, the python code repeatedly had to be deployed from the development platform to rebuilt Docker containers on the BCF API server before running tests using Solibri Office from the development platform. Bash scripts were created on Ubuntu to ease this deployment process, including rebuilding of Docker containers. Git was used for versioning during development. And the whole project was uploaded to Github on: https://github.com/marwiss/kontroll as open source with an AGPL v3 license.

Creating a Documents API server

When the BCF API server was set in place, it was shown that Solibri Office cannot upload nor download any files at all using the BCF API services. The BCF API does have a service intended for uploading and downloading documents, but this service is not used by Solibri Office. Solibri Office instead uses integrations with the Documents API to upload and download files to a server CDE (Solibri, 2023). An open-source implementation of the Documents API exists but is written in C# and Angular with a relational database as backend (not a graph database), so it could not be integrated easily with the BCF API written in python (Dangl, 2023). For this reason, also a Documents API implementation had to be developed using python, FastAPI and Neo4j.

The Documents API requires the development of a web user interface for file management. This web user interface was developed using Jinja2-templates for FastAPI, bootstrap for front-end user interface and jQuery for simple AJAX communication with the API.

Backend data storage and IFC-graph

To see if it would be possible to integrate any data from the BCF API with IFC-graph, Neo4j was used to store all data communicated to the server using every API, except binary files (such as images) which were stored on the Ubuntu file system. All data used for OAuth2 authentication, and all data communicated using Foundation API, BCF API and Documents API was stored as LPG and accessed with the official python driver for Neo4j using Cypher queries.

An existing algorithm for conversion of IFC data in STEP files into IFC-graph was implemented using the py2neo and IfcOpenShell libraries, because this algorithm was already written to utilize the py2neo library instead of the official python driver for Neo4j (Junziang, Peng, & Xiang, 2023).

Testing the BCF API services and Solibri together

To see if the BCF API can be used to communicate information about reusable building products and management of waste, required by national legislation on inspection plans, the BCF API server and its services were adjusted and tested using Solibri Office in various ways described later in this study.

Testing IFC-graph using a simple IFC data

To see if IFC-graph can facilitate access and query of building information useful for decisions on reuse, recycling or waste management of building products and materials etc., in a system for digital inspection plans, the following steps were taken.

A custom file upload feature was created using a Jinja2-template for the Documents API web user interface originally only intended for selecting and downloading files to the BCF client. A custom extra file uploading service was added to the server. The algorithm for converting IFC data in STEP files to IFC-graph was integrated with the file upload service, so that uploaded STEP file automatically is converted into IFC-graph.

A very simple building model with only three walls was created in Archicad. This model was exported to a IFC STEP file. Each IfcWall-entity had a property set “Reuse” with the property “GTIN”. This simple STEP file was uploaded using the custom extra file upload service and converted to IFC-graph on the server. Nodes containing properties with the value “GTIN” were found using Cypher in Neo4j remote browser. The path between any of these nodes and nearest IfcWall-node was discovered using Cypher. A Cypher query to find a property in any property set of an IfcProduct (such as IfcWall) identified by its GlobalId (UUID) was written in Cypher by looking at the structure of this path.

Combining BCF API and IFC-graph

The BCF API viewpoint service can upload information about selected components in the model together with information about their UUID. This service was tested together with IFC-graph. The selected components were saved to the LPG with relationships to viewpoint-nodes. The UUID of components were stored as properties to the component nodes.

A function that could connect each Wall (i.e., each IfcWall-component node) communicated using the BCF API with a relationship to its corresponding node in the IFC-graph, was developed. This function could also read the GTIN value of the corresponding walls in the IFC-graph and set the GTIN property of the IfcWall to this value. The function would also create a [SAME_AS]-relationship between each component node and its corresponding IfcWall node in the IFC-graph.

It was thus concluded that IFC-graph can be used to find GTIN-values of BCF-components using their UUID. The GTIN can in turn be used to find lots of interesting building information useful for decisions on reuse, recycling or waste management of building products and materials. This information can be retrieved from the same system or from another system/API.

A total of about 7000 lines of code was needed to implement all the three API and the features described in this section. The code mostly consisted of python and some Cypher.

Limitations

Solibri Office has some limitations as a BCF client because it does not implement all services and features of the BCF API. Nevertheless, Solibri Office was selected as a BCF client to test the BCF API server because Solibri Office is a common BCF client, and Solibri was available to use for free from the university. Solibri also provided a beta version of Solibri Office that was able to connect to the developed BCF API server. Other BCF clients generally only offer users the chance to connect to one specific BCF API server. This BCF API server is often a commercial server that requires some kind of subscription.

No suitable available BCF client plugin for Revit or Archicad to test the BCF API server could be found.

No BCF client was developed.

Only the present version of the BCF API specification, version 3.0 is tested and considered in this study. The next version, version 4.0, will probably add a possibility to add custom fields to topics. However, at the moment it is not possible to test this feature using any existing BCF clients.

Main part

Implementing the BCF API

The first practical step of investigating the possibilities of the BCF API was to try to create a server that implements the BCF API specification, and to test the functionality of this server using Solibri Office as a BCF client.

The BCF API specification does not enforce usage of a particular database type or database design. When implementing the BCF API specification, it is possible to save the data in any kind of database using any kind of database design or scheme (buildingSMART, 2021). To future proof the ability to integrate BCF API data with IFC-graph in the backend, the Neo4j graph database was selected for storing BCF API data as LPG.

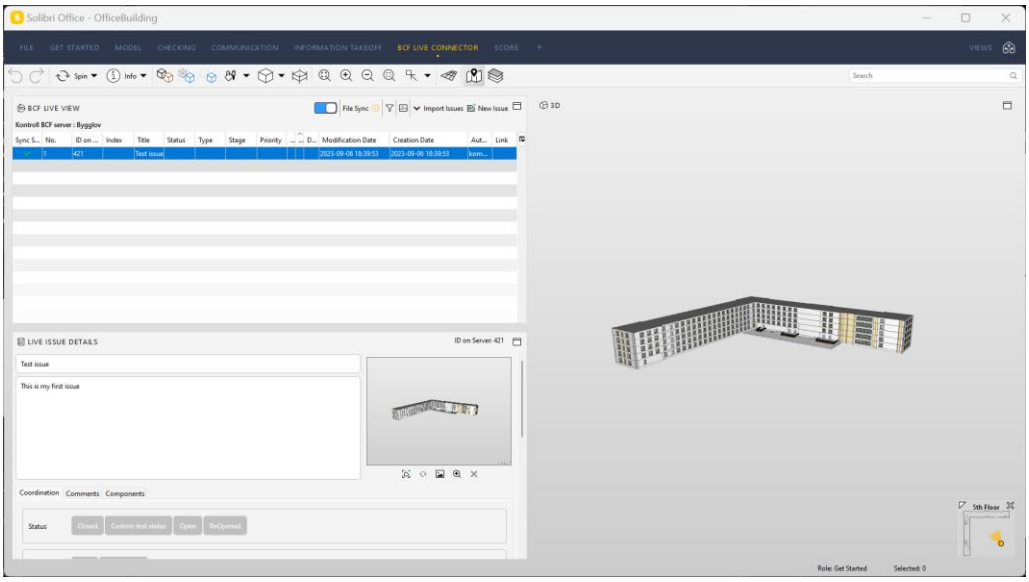


Figure 2 An issue has been synchronized to the BCF API server using Solibri Office.

A graph scheme for storing BCF data was suggested by (Schulz, Oraskari, & Beetz, 2021). A similar but more complete scheme was used in this case. Neo4j is a schemeless database. However, some kind of virtual scheme must nevertheless be used, even if this scheme only exists in the mind of the developer, so that data stored in the database later can be retrieved by the API in a predictable way.

The schemeless nature of Neo4j makes it precarious to manipulate IFC-graph data, because there is always a possibility of creating IFC-graph data that violates the IFC EXPRESS schemes. This will not be valid IFC data. This is why IFC-graph in Neo4j should probably mainly be used for the purpose of reading and analysis of IFC data, not for creating, writing, exporting or manipulating IFC data. The picture below shows basic instance BCF data visualized using the Neo4j browser.

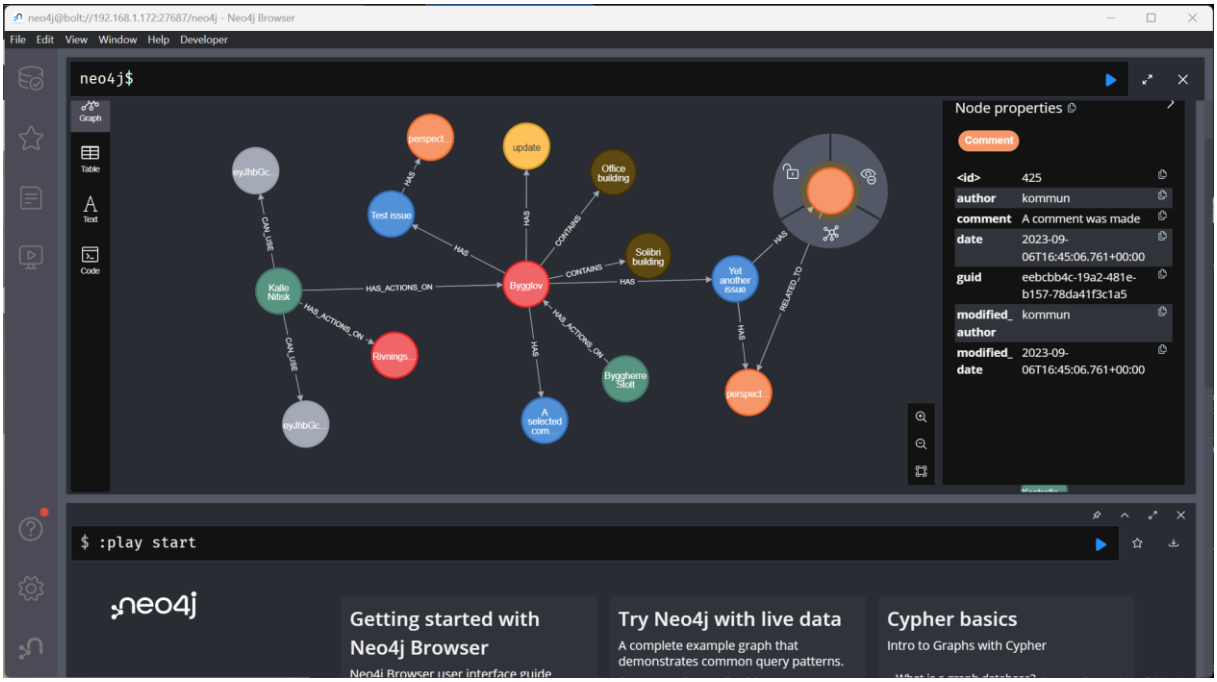


Figure 3 Some BCF data have been added to the LPG by the API. Three blue issues connected to a project "Bygglov" and a comment related to one of the viewpoints.

For the purpose of manipulating an IFC-graph, a graph database with a type system enforcing a scheme (preferably an EXPRESS scheme) would be a safer option. However, there are no widely used available typed graph databases that can validate instance LPG data to EXPRESS schemes. TypeDB is a strongly typed graph database that can enforce a scheme defined using TQL. TypeDB schemes do not allow multiple inheritance like EXPRESS. However, even though IFC schemes are written in EXPRESS, the IFC schemes does not use multiple inheritance (buildingSMART). It would be interesting to see to what extent the IFC EXPRESS schemes could be implemented using TypeDB TQL scheme.

Implementing the Foundation API

Before implementing the BCF API it is necessary to first implement the Foundation API. The Foundation API specifies a small number of services and a few conventions that are common to all OpenCDE API. Implementers of BCF API should start by implementing the Foundation API and only then continue to implement the BCF API (buildingSMART, 2021).

One drawback of using Neo4j for implementing the Foundation API and the BCF API is the lack of libraries that can convert OData v4 filter queries to Cypher. To fully implement the BCF API using Neo4j you would have to write your own library (in any programming language of your choice) for parsing OData v4 filter queries and converting them into Cypher. However, this functionality and such a parser is not necessary for the purpose of this study.

The Foundation API is used to establish a common gateway for clients to find out which versions of various OpenCDE API the server supports, and to establish common services for authentication and user identification. The Foundation API requires authentication using OAuth2. No known open-source graph implementations of OAuth2 authentication could be found so this had to be coded in python. The OAuth2 authentication process was facilitated using self-destructing nodes in Neo4j for sessions. The self-destructing nodes were implemented using the TTL functions in the APOC library for Neo4j (Neo4j, 2023).

Using the services of the BCF API

Using Solibri Office as a client it was possible to try out various services of the BCF specification, such as the projects services, the topics services, the BIM snippet services, the comments services, the viewpoints services, the documents services and the project extension service.

BIM snippet services

The BIM snippet services of the BCF API have no detailed explanation in the official BCF API documentation. According to online forum discussions related to BCF, the BIM snippet services were originally intended to allow collaboration on modifications of files with IFC data. This intention however contradicts how files with IFC data are commonly used in the AEC industry today.

The usual workflow, when using commercial BIM software like Revit or Archicad, is to create, work in and save building models in the proprietary file format of the software. It is not possible to create or edit a building model using the STEP file format with IFC data. When a modification of a file with IFC data is requested using BCF, then the modifications are made in the proprietary file format of the software, and then a new STEP file with IFC data must be exported before it can be shares. This means that files with IFC data are not modified directly, even though it would be theoretically possible to do so. With this workflow it will not be possible to reach information management maturity level 3 of ISO 19650-1 (Norsk standard, 2019). As to my knowledge, no software nor any server has ever implemented the BIM-snippet feature of the BCF specification (buildingSMART, 2021).

BlenderBIM is a relatively new opensource BIM software that can create, save and open building models as IFC data natively. This means that there is no need to export data from a proprietary file

format to IFC. Building models are instead created as IFC data from the beginning and saved as IFC data. This workflow, where no exports are necessary, prevents the risk of data loss during exports. By following three additional technical requirements for saving IFC data as STEP files BlenderBIM also enables versioning for IFC data text files. BlenderBIM could thus easily implement the BIM snippet service as intended if used as a BCF client. The three additional technical requirements needed to implement versioning for IFC native capable software are (Postle, 2022):

- IFC entities must be written in the same format as received, with the same numeric IDs as before. Sorting the lines by numeric ID is recommended.
- Attribute changes to entities must be written in-place, preserving the numeric ID of the entity.
- Numeric IDs of deleted entities must not be reused for new entities.

The BIM snippet services could potentially also be used to send some other kind of information. However, this is not the intended purpose of the BIM snippets services, and since no known BCF client nor any server have ever made any use of the BIM snippet services, this possibility is not of interest in this study limited to using Solibri Office as BCF client. The BIM snippet service is not tested because no client supports this service. To test this service, it would be necessary to develop a custom BCF client that makes unintended use of the BIM snippet services, and that's outside the scope of this study.

The path for the unused PUT BIM snippet service is:

```
PUT /bcf/{version}/projects/{project_id}/topics/{guid}/snippet
```

Documents services

Another way to send information about reusable building products and waste etc. could potentially be to utilize the Documents services of the BCF API. Any kind of information could potentially be encoded into JSON documents and then uploaded to the server using the Documents services.

The Documents services of the BCF API are however not used by Solibri Office at all. Solibri Office does neither use the BCF API to upload nor to download documents (such as STEP files or other documents) even though this functionality probably should be implemented by a BCF API 3.0 compliant client. Instead Solibri Office uses the Documents API for file uploads and downloads. Technically, using the Documents API is a more complicated and maybe sometimes even a better way of sharing documents between a client and a common data environment (CDE).

The path for the unused POST Document service is:

```
POST /bcf/{version}/projects/{project_id}/documents
```

Project extensions service

The Project extensions service is used by the BCF API server to define the possible values that can be used in topics and comments for a BCF client, for example topic labels/fields. Any such values defined using the project extensions service could then later be transferred from the client to the server as "metadata" for the topic using the topics services. The possible values defined using "project extensions" can be changed on the fly. The most recent extensions state which values are valid for labels/fields of a topic at a given moment, at least for newly created topics.

One limitation of the project extension service is that possible values of labels/fields can only be defined as lists of possible values. The user of the client then must select one or several of the given

values from this list. It was not possible to let a client user enter free values using project extensions. At least not using Solibri Office as BCF client.

To test the possibilities of project extensions, a custom project extension response, that extends the default labels/fields and values of labels/fields of topics was defined on the server. However, the custom extension value only showed up in Solibri Office in the predefined labels/fields already used by Solibri Office. Any extra labels/fields added using the project extensions service did not show up in Solibri Office. It seems that Solibri Office does not support adding new custom labels/fields to topics. Solibri Office only allows extensions to the existing labels/fields that Solibri Office already implements.

This was the project extensions response:

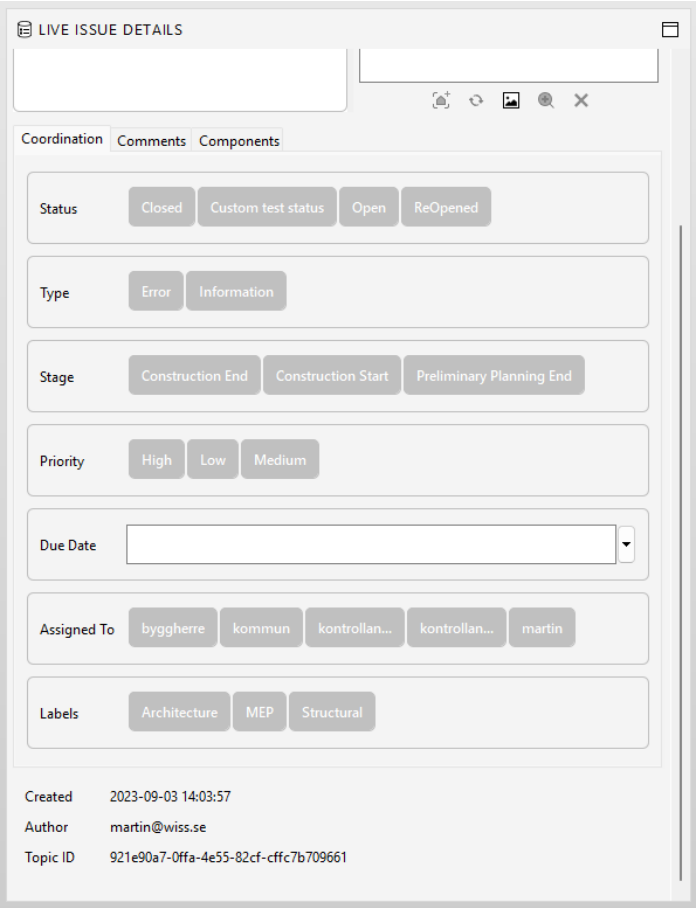

```
Endpoint:  "project_extensions_get"
Request:  {"project_id": {"py/object": "uuid.UUID", "hex": "c8e9cccea4af11edb9df0242ac120003"}}
Response:  {"topic_type": ["Information", "Error"], "custom_information": ["Custom value 1", "Custom value 2"
], "topic_status": ["Open", "Closed", "ReOpened", "Custom test status"], "topic_label": ["Architecture", "Str
uctural", "MEP"], "snippet_type": [".ifc", ".csv"], "priority": ["Low", "Medium", "High"], "users": ["kommun"
, "byggherre", "kontrollansvarig", "kontrollant", "martin@wiss.se"], "stage": ["Preliminary Planning End", "C
onstruction Start", "Construction End"], "project_actions": ["update", "createTopic", "createDocument"], "top
ic_actions": ["update", "updateBimSnippet", "updateRelatedTopics", "updateDocumentReferences", "updateFiles",
 "createComment", "createViewpoint"], "comment_actions": ["update"]}
INFO:     192.168.64.1:33574 - "GET /bcf/3.0/projects/c8e9ccce-a4af-11ed-b9df-0242ac120003/extensions HTTP/1.
0" 200 OK
```


Figure 4 The was the project extension service response from the server, containing some custom extensions underlined (see below)

The interesting parts of this response is underlined below:

"topic_type": ["Information", "Error"],
"custom_information": ["Custom value 1", "Custom value 2"],
"topic_status": ["Open", "Closed", "ReOpened", "Custom test status"],
"topic_label": ["Architecture", "Structural", "MEP"],

The result in Solibri Office show that the “Custom test status”-value of the “topic_status” label/field was successfully added. However, the “custom_information” label/field with its values, was not added. This label/field does not show up on the following screenshot:



Adding new labels/fields with the project extensions service can only be used if the client supports this. And Solibri Office did not support adding new labels/fields this way. Solibri Office only supported adding new value choices to existing labels/fields.

Suppose that results of inspections, according to digital inspection plans, were to be communicated as topics/issues using BCF API. Then it would be difficult to let users of the system add any extra information about this topic besides the general description of the topic. It would have been easier to use the BCF API to communicate information on inspections, if it was possible to add extra free text labels/fields to a topic, indicating any result of the inspection etc.

Such functionality is considered for the next version of the BCF API specification (version 4.0) but does not exist as of today (buildingSMART, 2020) (buildingSMART, Add properties to schema, 2022).

The path for the GET Project extensions service is:

```
GET /bcf/{version}/projects/{project_id}/extensions
```

BCF version 4.0

The next version of the BCF API, version 4.0, implements several proposals. One issue that has been marked as “closed” in the Github repository for the BCF API specification, is “Allow Clients to show custom Topic Properties” (buildingSMART, Issues - Considered for Next Version - Is closed, 2023). This issue means changes to 4 files in the repository, the topic_GET.json, topic_POST.json, topic_PUT.json and the extensions_GET.json. Three new files are added that describe JSON schema for custom_fields. The README.md file is updated as follows:

“Projects may be configured to allow custom_fields`in a topic. This is a way of enabling clients and servers to embed custom data in a topic. Those custom fields can be sent when creating or updating a topic, and they will be returned by the server when retrieving topics.

Custom field values are always represented as strings. The type of custom field indicates how it should be parsed. The value null indicates that it is absent.

- *integer: A number that does not contain decimals*
- *decimal: A number than can contain decimals*
- *string: Any string*
- *boolean: The values 'true' or 'false'*
- *enum: A value from the predefined enumValues array*
- *date-time: ISO 8601 compliant date*

custom_fields are an optional array property that can be used by clients to attach custom data to a topic. The server should return the same array in the response. The array may be empty. It should contain the custom fields as defined in the project extensions. The id property of each custom field object is used to identify a field within a project. When creating or updating a topic, the `id` property from the extensions must be provided.”

Defining a custom field in the project extensions service may look like this:

```
"custom_fields": [
    {
        "id": "6e6bddaa-4c53-4fb8-b884-500e0d2dba6a",
        "name": "Price (in dollars)",
        "type": "decimal",
        "readonly": false,
        "required": true
    },
    {
        "id": "43673cf1-cd05-4544-b1ac-d299c58b8f9b",
        "name": "GTIN",
        "type": "string",
        "readonly": false,
        "required": true
    }
]
```

After custom fields have been defined in the project extensions service it will be possible to exchange information for such custom fields using a new custom_fields property of the topic GET, POST and PUT services. It may look like this:

```
"custom_fields": [
    {

         “id”: "6e6bddaa-4c53-4fb8-b884-500e0d2dba6a",
         “value”: “133.50”}

    },
    {

         “id”: "43673cf1-cd05-4544-b1ac-d299c58b8f9b",
         “value”: “0123456789012”

    }
]
```

Conclusion on possibilities of BCF services

No other ways of transferring building information related to inspection plans using the present version of the BCF API, version 3.0, were found. The present version of the BCF API specification, version 3.0, provides some hypothetical ways to transfer building information (such as using the Documents services or the BIM snippet services), however it was not possible to make use of Solibri Office as a client for sending such information by adjusting the BCF server. So, to use any BCF API services to send building information, you would probably need to develop your own client besides developing your own BCF API server. Creating a custom BCF client is outside the limitations of this study. If you need to create your own custom client just to implement some BCF API services in some creative but unsupported manner, then you could just as well create your own custom API instead.

The next version of the BCF API specification, version 4.0, will allow adding custom fields to topics. When this feature also gets implemented by BCF clients, it will present a way to customize the use of the BCF API for a wide variety of purposes that require additional exchange of information. This might make the BCF API more useful for implementing at least some parts of digital inspection plans. However, it is not possible to test the future BCF API version 4.0 using any existing BCF clients. This version has neither been released nor implemented yet, hence this feature will not be described further in this study.

Using IFC data from STEP files

An alternative approach for "sending" building information to the server from a BCF client is possible if this information is included in a STEP file with IFC data available to both the BCF client and the BCF API server at the same time. The BIM collaboration format was designed to communicate topics about a building model. Topics can have viewpoints. The viewpoint service of the BCF API can transfer a list of selected components to the server. The purpose of this component feature of the POST Viewpoints service, is to be able to select and show building components related to a topic/viewpoint. The components are supposed to be highlighted in the 3D-view from the viewpoint in the client. However, an alternative way of using this feature is if we imagine that a topic could contain a viewpoint that contains a list of building components that for example are supposed to be reused.

The path for the POST Viewpoints service is:

```
POST /bcf/{version}/projects/{project_id}/topics/{guid}/viewpoints
```

In BCF API specification version 4.0 this way of sending information would probably be better implemented using a custom field to a topic. It would be possible for a user of a client to associate a topic with some selected objects in a building model, if the following two extra custom field types were allowed in BCF API version 4.0.

- GlobalId: A string containing a GUID representing any instance of IfcRoot in the model
- Array[GlobalId] (to let the user associate a topic with a list of selected objects in the model)

It would be good to be able to attach information about lists of objects to the topic itself, not only to the viewpoint, because viewpoints have a visual purpose and are not intended for transfer of information with other purposes. A BCF viewpoint captures a viewpoint of a model that highlights an issue. Still, the concept of sending a list of components using viewpoints will be discussed hereafter.

Relating BCF-data to IFC data using component UUID

Each component in the list of components transferred as part of a viewpoint using the BCF API contains a UUID. All objects in the model that inherits IfcRoot has a property GlobalId that contains

this UUID. And all components in a building model should inherit IfcRoot. The UUID is a unique ID in the model.

When the list of components with UUID has been transferred from the client to the server, the server can identify components with the same UUID in a building model from a STEP file on the server, to see if these identified components have other properties that could be interesting. For example, interesting for the purpose of reusing building components. This means that such information, such properties, can be accessed by the server even in cases when the BCF client cannot communicate such information, or such properties, from the client to the server using the BCF API.

For this method to work, there must be a way to share STEP files with IFC data from the user/client to the server, so that the server and the user/client can share the same STEP file with IFC data and can read the same building information from this STEP file. Solibri Office does, however, not implement the Documents services of the BCF API specification. Solibri Office can thus not use BCF API to send nor to download any kind of files. Instead Solibri Office uses the Documents API to send models and download files. For a BCF API server to share files with Solibri Office, then this BCF API server must also implement the Documents API.

Implementing the Documents API

Solibri Office does not allow the upload of files with open IFC data using the Documents API. In order to upload a model from Solibri Office, the model must first be saved as a Solibri model checker (SMC/.smc) file. Solibri Office uses the SMC-format to store a federated model that can consist of multiple models from several imported files with IFC data of various disciplines. However, the SMC-format is not an open file format, it is a proprietary file format. There is no known or supported way to round-trip an SMC-file back into its original IFC data. This means that files with IFC data can only be downloaded from the Documents API server using Solibri Office as a client, never uploaded.

To get files with IFC data from the client/user to the server in the first place, another method must be used. One such method would be to implement a custom file upload service or to write a custom web app client that utilizes the Documents API. In this study, to enable uploading files with IFC data, a simple separate custom web file upload service was created.

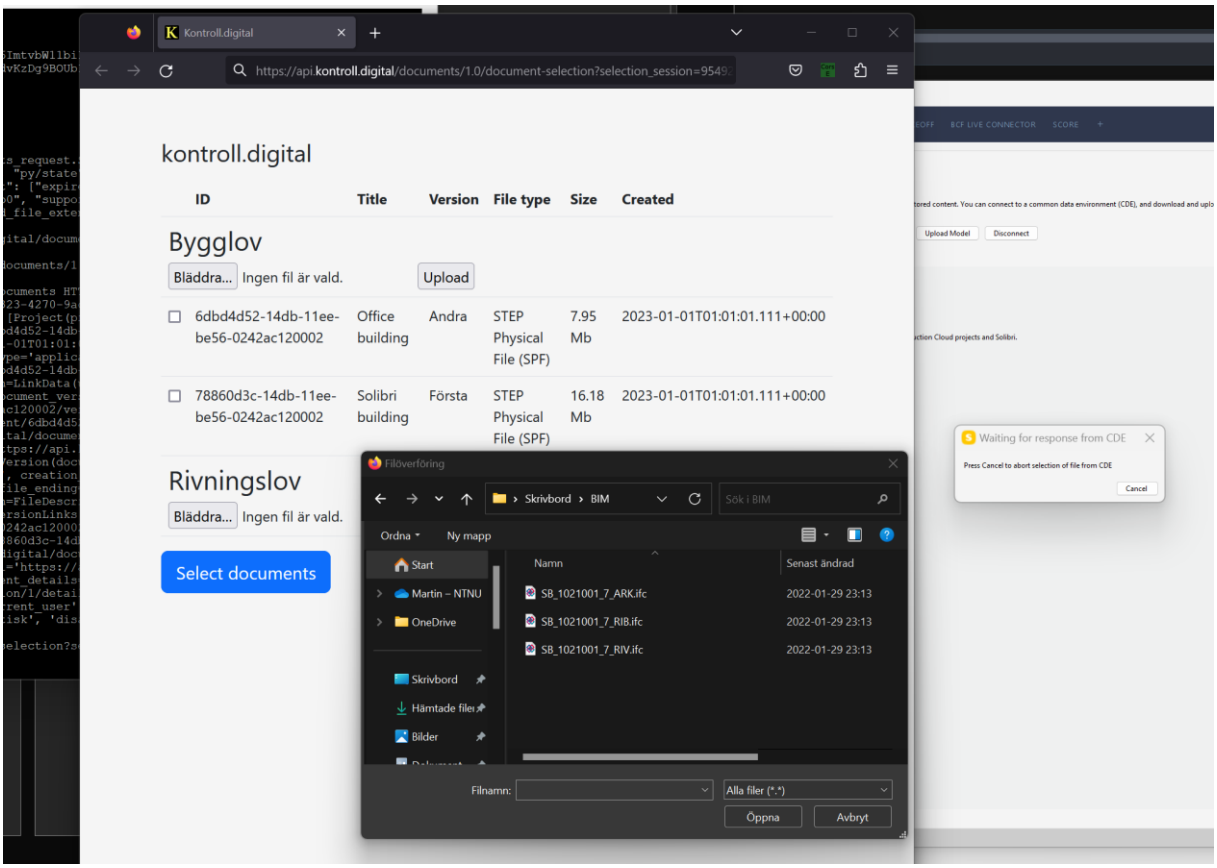


Figure 5 A view of the custom web file upload service integrated in the the Documents API web user interface for selecting files for download to Solibri Office.

Parsing STEP files with IFC data on the server

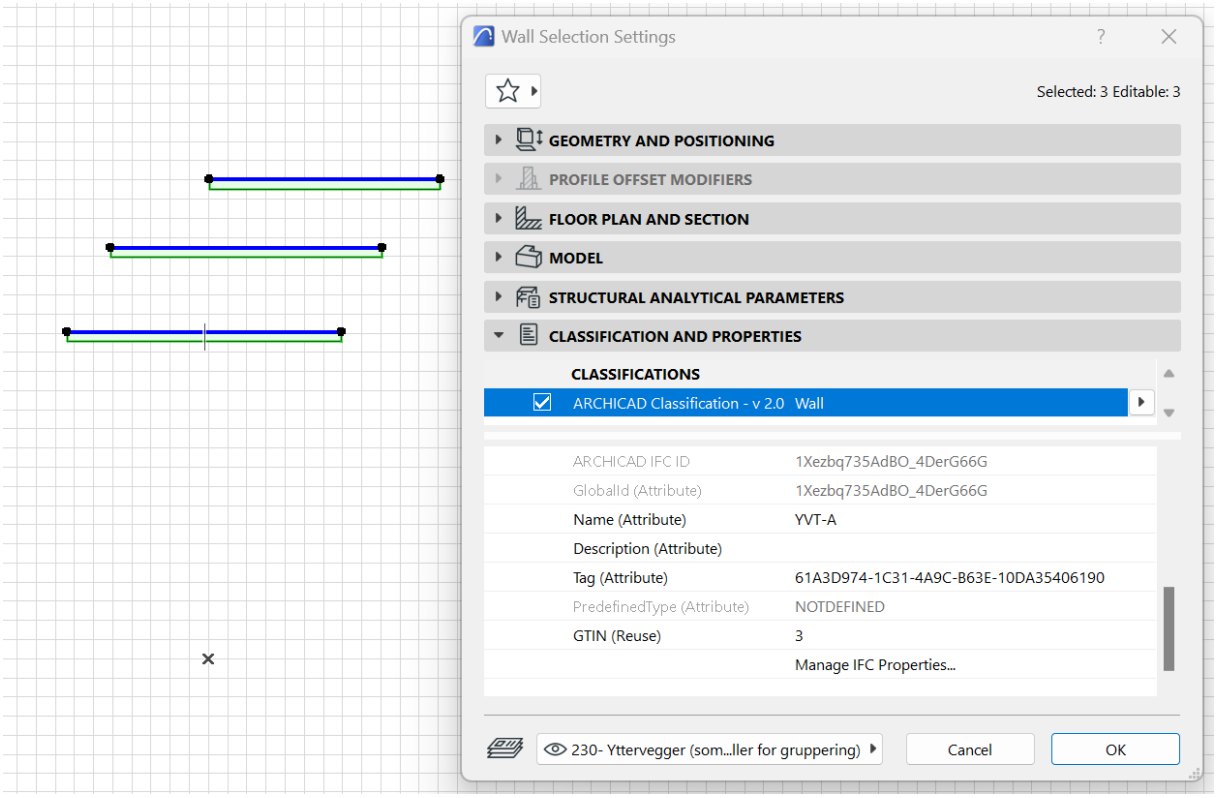
Using the custom web file upload service together with the Documents API web interface used by Solibri Office to select and download files, it was possible to upload STEP files with IFC data to the BCF API server. By using this combination of services on a single web interface, the STEP file can directly after upload be selected and downloaded to Solibri Office using the Documents API.

Implementing IFC-graph

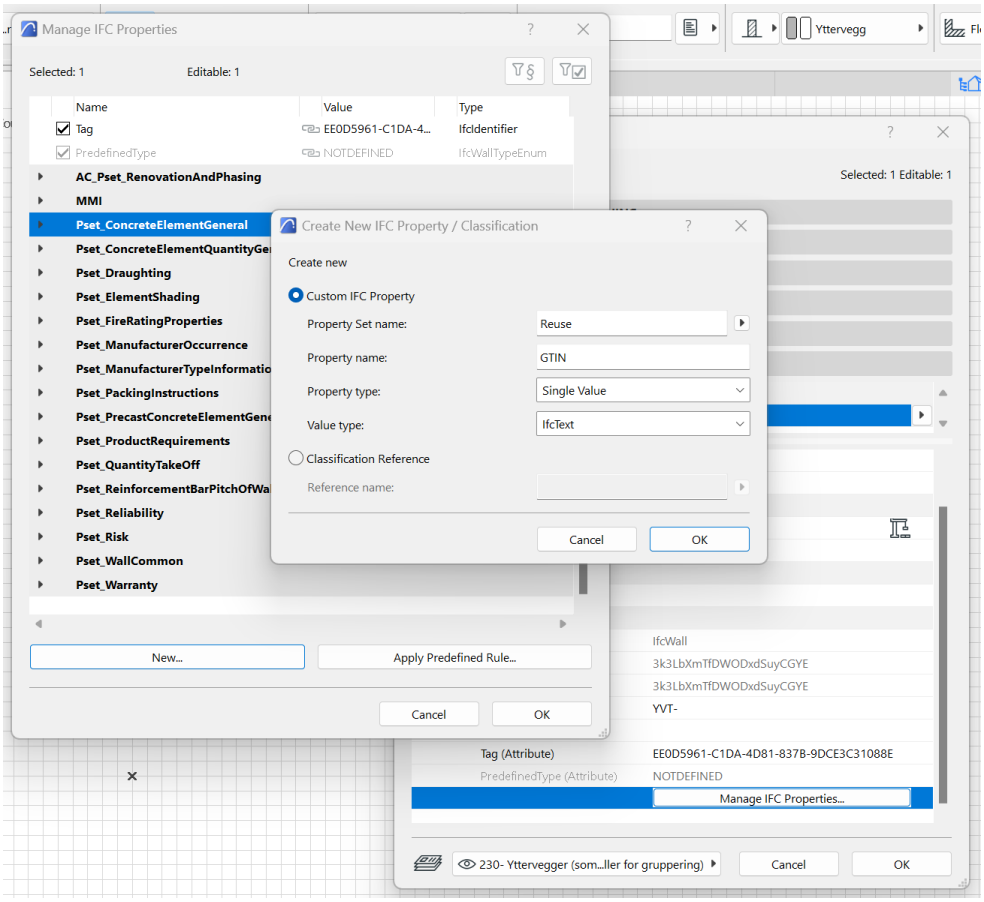
To achieve the second aim of this study, the IFC file should be converted to IFC-graph after upload using the algorithm by (Junziang, Peng, & Xiang, 2023). This algorithm itself uses the IfcOpenShell library for python to parse the STEP file into data structures (IfcOpenShell, 2023). The algorithm also uses the py2neo library for Neo4j to merge every node into the LPG. The other option would be to use the native official python driver for Neo4j. Using py2neo to interact with Neo4j is a good and practical choice for doing research and data science on Neo4j using python because it is an OGM that can easily integrate with the Python Data Analysis Library (pandas). But py2neo could be a problematic choice for running a production enterprise python API against a Neo4j server (Singer, 2021). Py2neo does not implement all features of the native official python driver for Neo4j and can also be significantly slower compared to the native official python driver for Neo4j (Medium, 2017).

Exporting IFC from Archicad

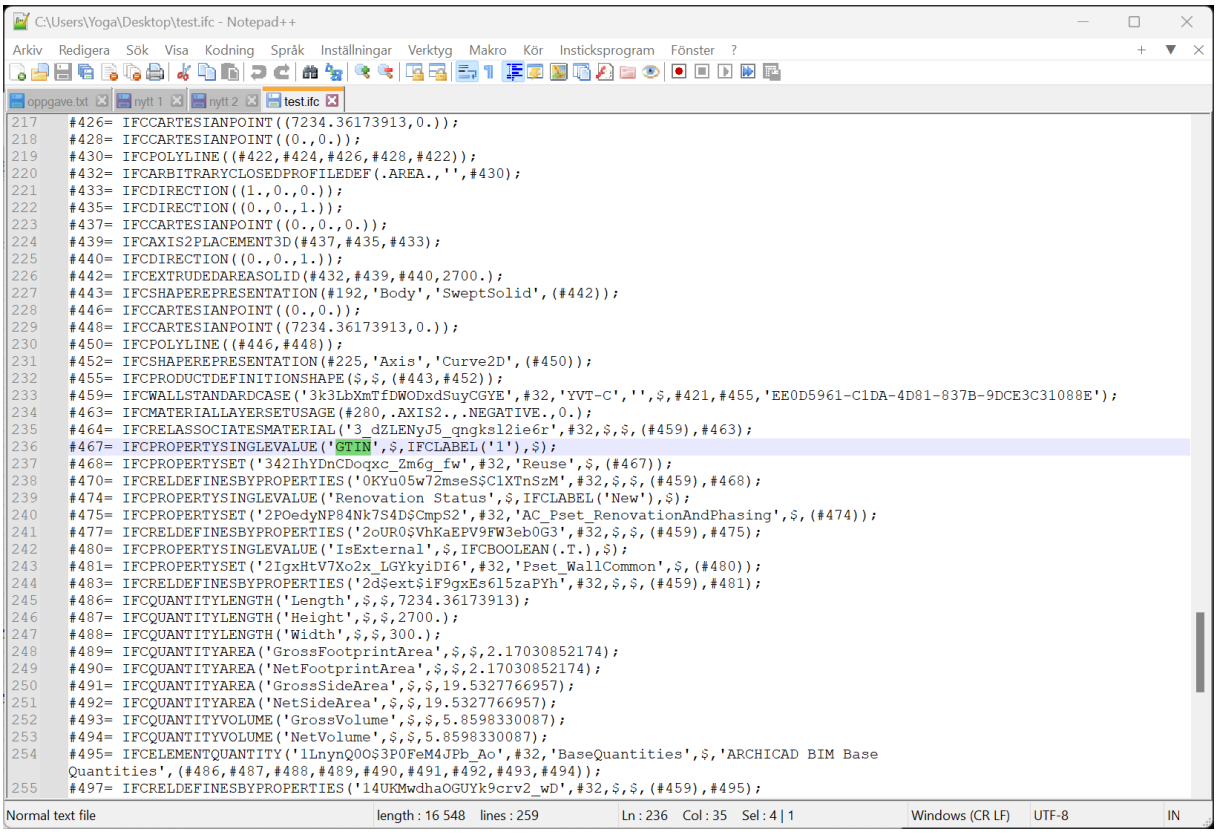
To get the test data in the example above, a simple STEP file with IFC data was exported from Archicad. The building model in this STEP file only consisted of three walls.



An IFC property set called “Reuse” with a property “GTIN” was created for all three walls. The values of this property were set to 1, 2 and 3 respectively.



The model was exported to a STEP file with IFC data using Archicad. The simple STEP file generated by Archicad consisted of only 259 lines of STEP code.



Converting STEP files with IFC data to IFC-graph

The screenshot below shows the file upload user interface side by side with a server terminal window logging the parallel IFC-graph generation (after upload).

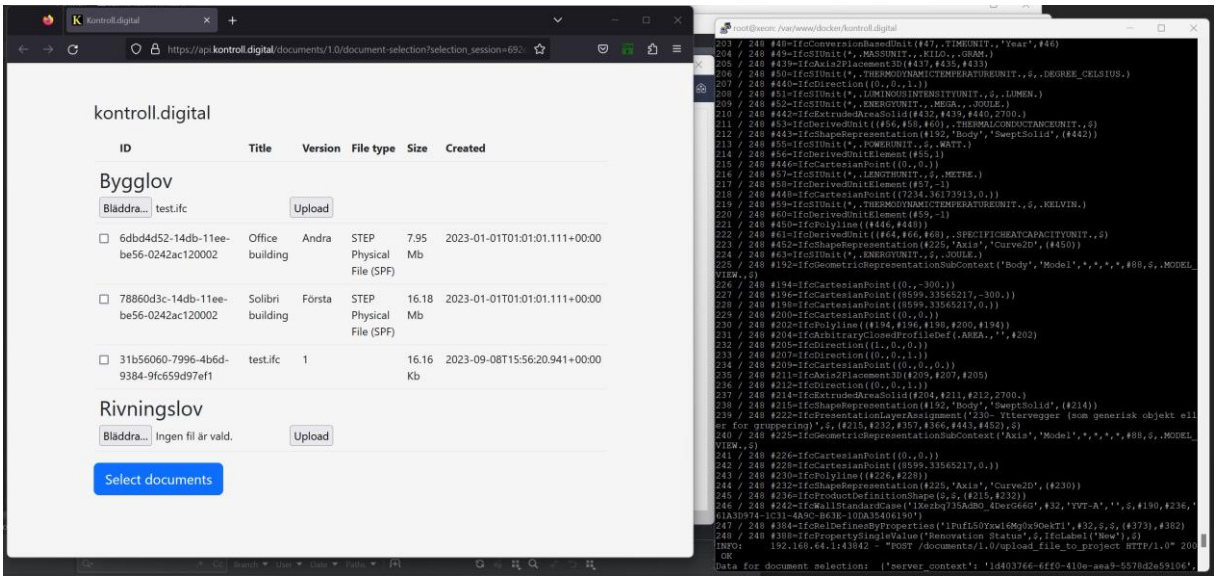


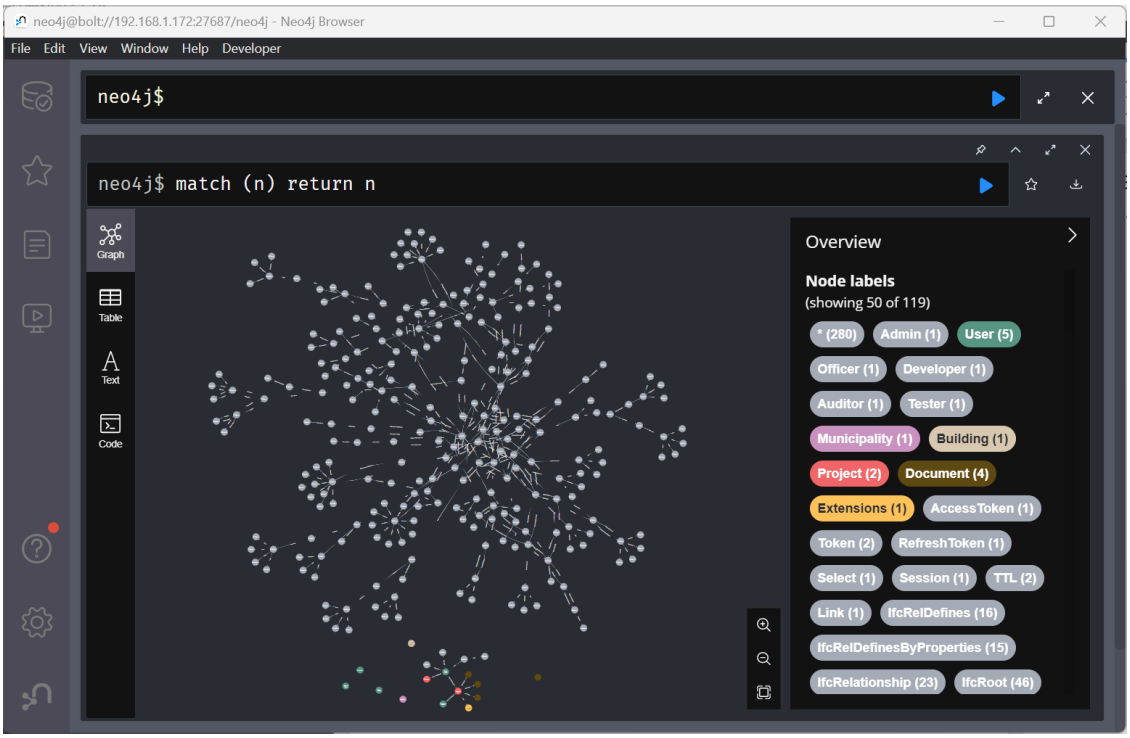
Figure 6 Uploaded STEP file with IFC data (left window) was converted into an IFC-graph (as shown in the logs in the right server terminal window).

The process of merging nodes into LPG using py2neo is time consuming and depends on the size of the graph. The algorithm needs longer time to merge each node on a larger graph. This is why the algorithm gets slower and slower as the size of the graph increases.

When trying this algorithm on the server, converting a normal sized STEP file with IFC data into an IFC-graph with approximately half a million nodes took several hours. This is a lot of time compared to the short time it takes for IfcOpenShell to parse a STEP file with IFC data into data structures in computer memory. A process that can be counted in seconds.

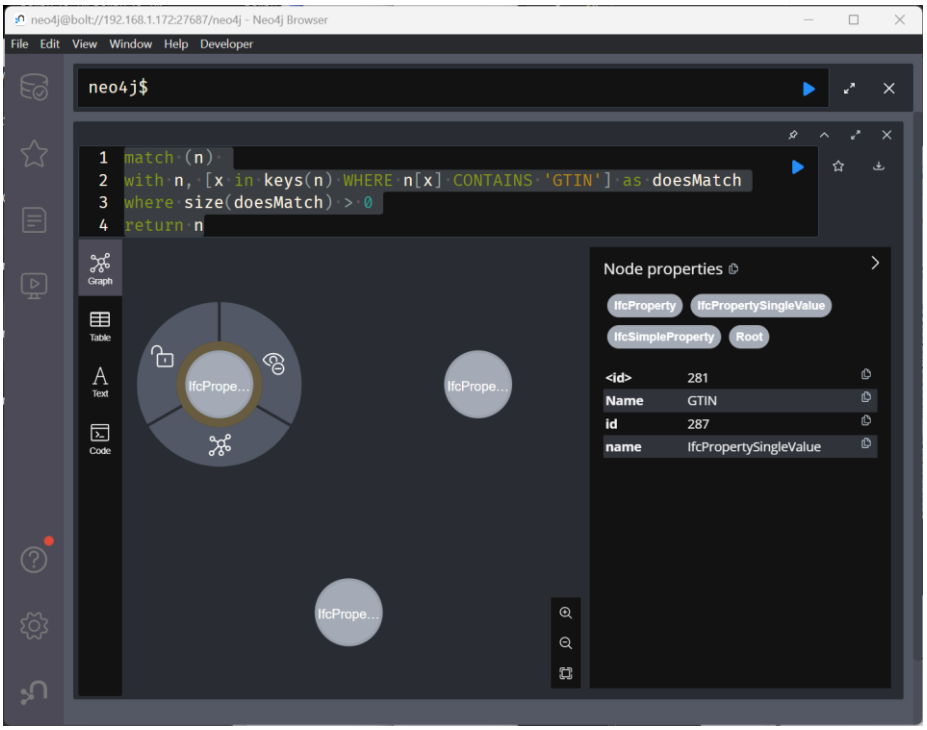
For a faster algorithm (compared to using the py2neo library and its graph.merge function) one could try to utilize the official native python driver for Neo4j and insert the nodes and their relationships using Cypher CREATE statements in batch mode instead of using MERGE statements. Coding such an algorithm is not needed for the purpose of this study. Instead, a very short and simple STEP file with IFC data was used to make the process go faster. The building model contained in this STEP file only consisted of three walls, i.e., three IfcWall entities.

The resulting IFC-graph was visualized in the Neo4j browser. The generated IFC-graph consisted of a little bit more than one hundred nodes. The colored nodes at the bottom of the graph are the nodes used by the BCF API to save information about projects, users, topics etc. These nodes are called BCF data in this study. The grey nodes at the top of the graph are the data of the IFC-graph. It seems like all grey nodes have relationships to each other in the IFC-graph.

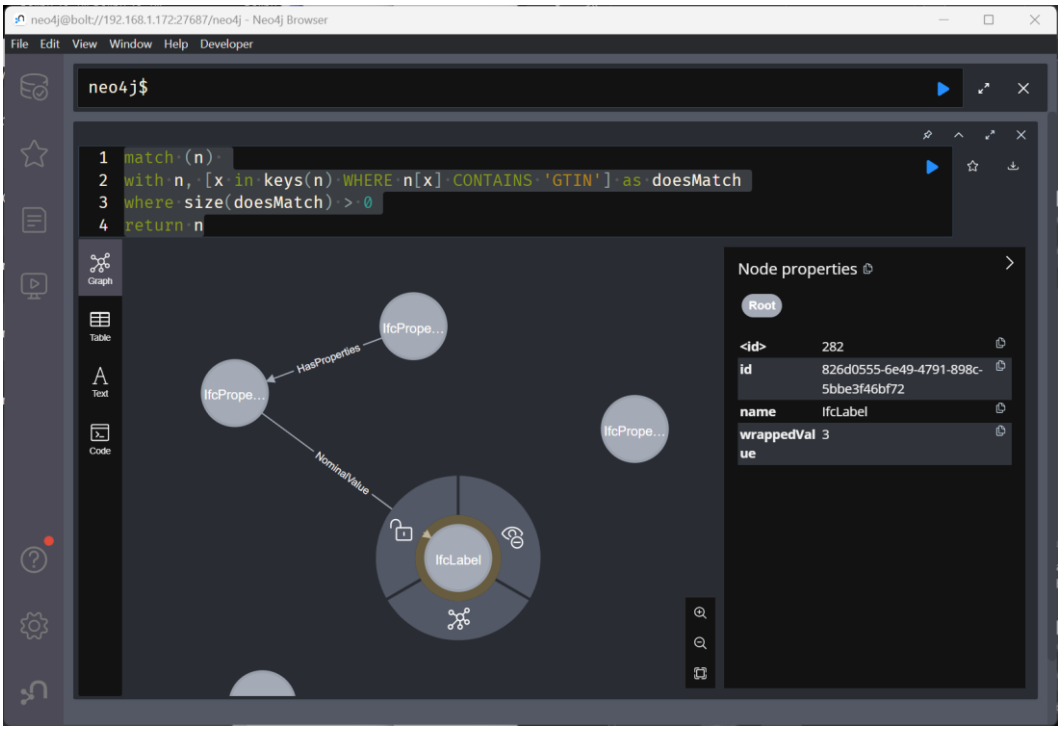


Analyzing IFC-graph

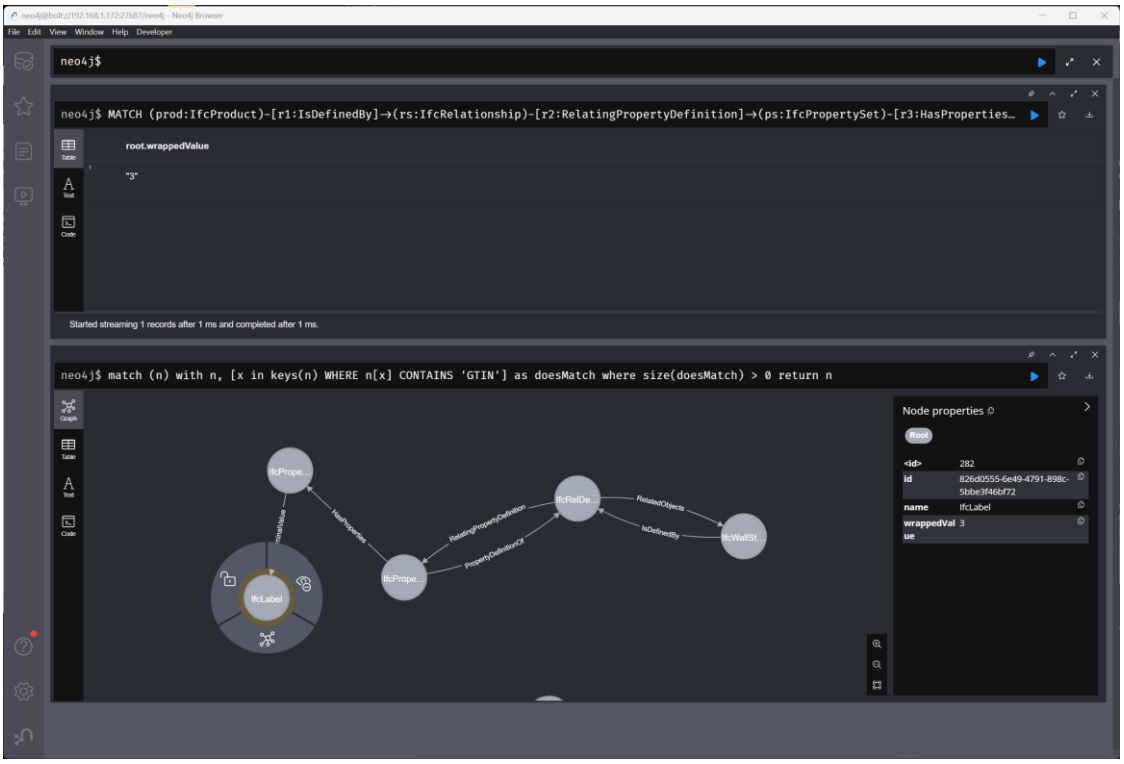
The next task was to find out how to find the added GTIN property of a wall in the IFC graph using Cypher. A query to find any node containing a property equal to “GTIN” was used. This query returns three nodes of type IfcPropertySingleValue.



Expanded relationships from/to these three nodes show a NominalValue relationship to an IfcLabel node with a WrappedValue of “3”. This node should somehow also be connected to the Wall with the GTIN value of number “3”.



The graph was traversed to find a path of relationships between the node representing the wall and a node containing the WrappedValue-property, containing the value of “3” which is the value of the “GTIN” property of the “Reuse” property set in the IFC data from the STEP file. The following path was found:



This path was possible to find using the following Cypher query:

```
MATCH (prod:IfcProduct)
-[r1:IsDefinedBy]->(rs:IfcRelationship)
-[r2:RelatingPropertyDefinition]->(ps:IfcPropertySet)
-[r3:HasProperties]->(prop:IfcProperty)
-[r4:NominalValue]->(root:Root)
WHERE prod.GlobalId = '1Xezbq735AdBO_4DerG66G'
AND ps.Name = 'Reuse'
AND prop.Name = 'GTIN'
RETURN root.wrappedValue
```

When sending a UUID of a wall as a selected component in a viewpoint service of the BCF API, the server will now be able to find the GTIN of this wall in the IFC-graph using the Cypher query above. The server just have to replace the value of the prod.GlobalId-property with the UUID of the component sent from the client to the server.

Combining IFC-graph and BCF data

A function that could connect each component in the BCF-data with its corresponding IfcWall-node in the IFC-graph, using matching UUID was developed. This function could also read the GTIN value of the corresponding wall-nodes in the IFC-graph and set the GTIN property of the matching component node to this value. This function would also create a [SAME_AS]-relationship between the component wall node in the BCF data and the corresponding IfcWall-node in the IFC-graph.

The same STEP file was opened in Solibri Office as a model, and the three walls were communicated to the BCF API server as components using the Viewpoint service.

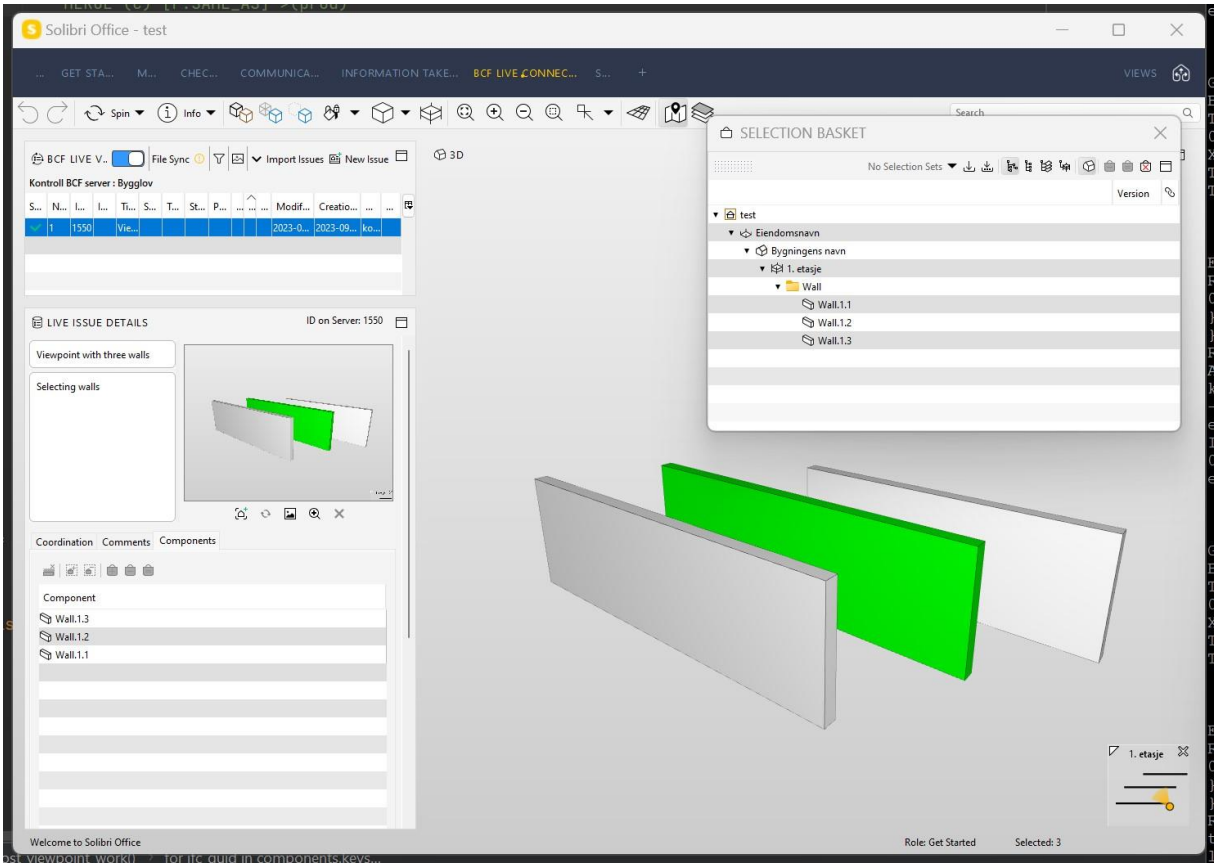


Figure 7 Three walls have been selected as "selected components" in a viewpoint, and the viewpoint have been successfully synchronized to the BCF server (the arrow is green).

In the previous screenshot we can see that three walls have been sent to the BCF API server and saved as component nodes in the LPG. In the following screenshot we can see that [SAME_AS]-relationships have been created between the wall nodes in the BCF data and IfcWall nodes in the IFC-graph. This relationship was created by using the Cypher below that matches the UUID of BCF data component nodes with the same UUID (prod.GlobalId of IfcProduct) in the IFC-graph.

```
MATCH (prod:IfcProduct) WHERE prod.GlobalId = $uuid
MATCH (c:Component) WHERE c.ifc_guid = $uuid
MERGE (c)-[r:SAME_AS]->(prod)
```

Another Cypher query was used to fetch the GTIN value from the tree IfcWall-nodes in the IFC-graph and insert their GTIN values in the three corresponding component wall nodes of the BCF data.

```
MATCH (prod:IfcProduct)
    -[r1:IsDefinedBy]->(rs:IfcRelationship)
    -[r2:RelatingPropertyDefinition]->(ps:IfcPropertySet)
    -[r3:HasProperties]->(prop:IfcProperty)
    -[r4:NominalValue]->(root:Root)
    WHERE prod.GlobalId = $uuid
    AND ps.Name = 'Reuse'
    AND prop.Name = 'GTIN'
MATCH (c:Component) WHERE c.ifc_guid = $uuid
SET c.GTIN = root.wrappedValue
```

The relationships between BCF data nodes and IFC-graph nodes are shown in the following screenshot using the Neo4j browser:

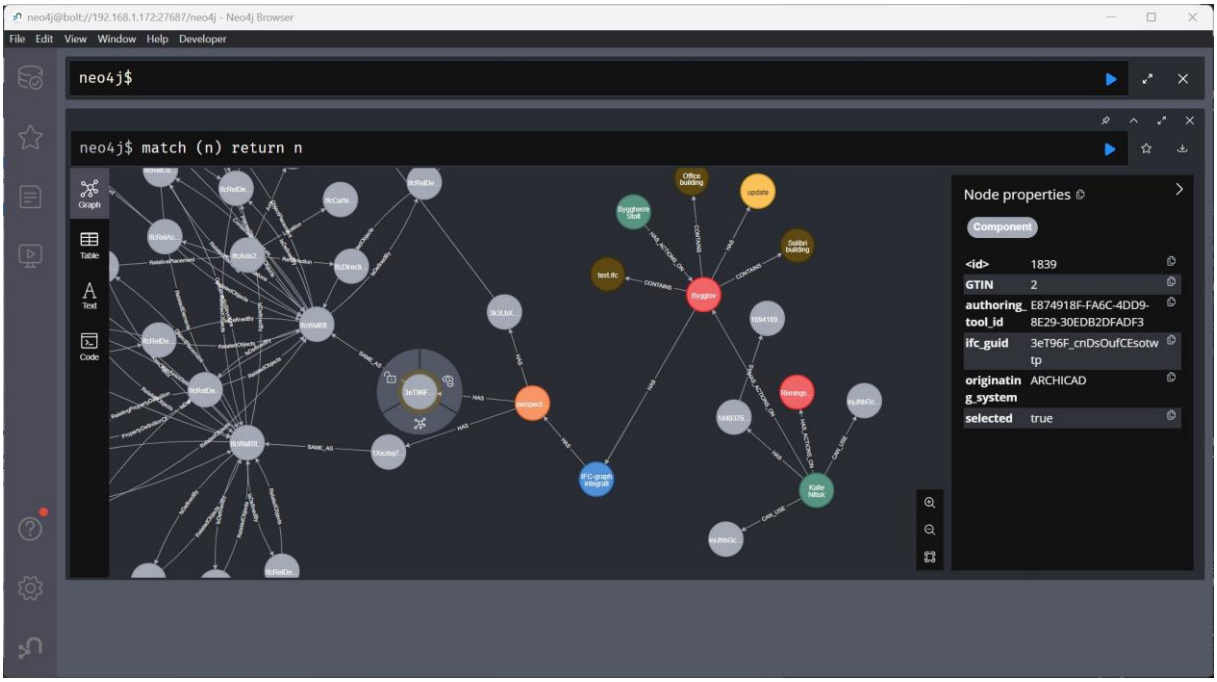
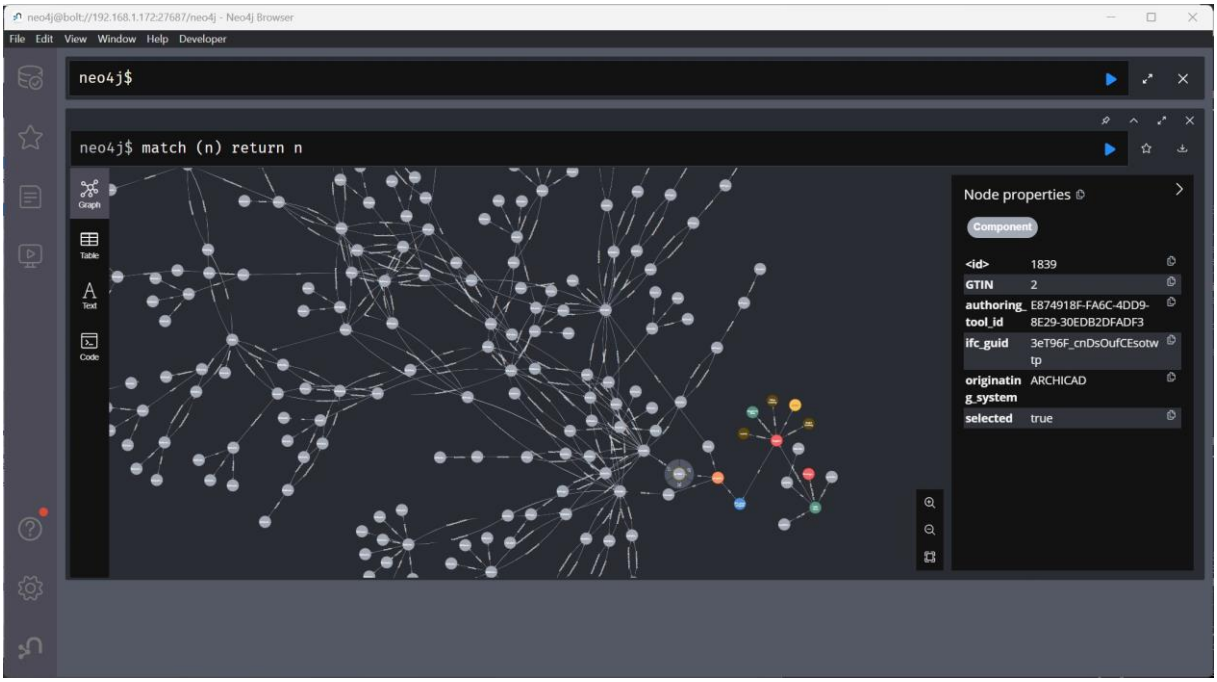


Figure 8 The orange node is a viewpoint-node connected to three component nodes that represent the three BCF-walls. The three BCF-walls are connected using [SAME_AS]-relationships to the corresponding node (with the same UUID) in the IFC-graph.

In the table on the right side on the previous screenshot, we can see that the GTIN value from IFC-graph has been added as a value to the GTIN property of the selected gray BCF data component node. If we zoom out, we get a better view of the relationships created between the BCF data (in colors and three gray wall component nodes) and the IFC-graph data (in gray color).



It follows that Cypher can be used to find GTIN-values of BCF-components in an IFC-graph using their common UUID. The GTIN can in turn be used to find lots of additional interesting building information useful for decisions on reuse, recycling or waste management of building products and materials. This information can be retrieved from the same system or from another system/API.

Discussion and results

Not being able to use only BCF API version 3.0

The first aim of this study was to see if the BCF API can be used to communicate information about reusable building products and management of waste, required by national legislation on inspection plans. If such a possibility existed, it was also interesting to know what kind of such information the BCF API can communicate, and how.

It was not possible to design a BCF API server in a way that made Solibri Office usable for communicating the information necessary for developing a system for digital inspection plans. By using Solibri Office as a client and the BCF API it will not be possible to meet relevant OIR suitable for such a system. At least not using the present version of the BCF API specification, version 3.0.

Before development of the BCF API server, the project extensions service seemed like an interesting method to extend the functionality of Solibri Office to make it possible to send additional "metadata" for topics. However, Solibri Office did not let project extensions add extra labels/fields to topics for allowing additional data for topics. The feature of adding "custom fields" to topics using project extensions might be added to the next version of the BCF API specification, version 4.0 (buildingSMART, 2020). And then this feature will probably also be added to Solibri Office in the future. However, this feature is not implemented today. The other methods of communicating building information could not be tested using Solibri Office. BIM snippets are not used by any client. Document service are not used by Solibri Office, etc.

The conclusion is that the BCF API could not be used to communicate the information necessary, at least not using the present version 3.0 of the BCF API specification together with Solibri Office as a BCF client. Alternative options could be to develop a custom client that makes use of BCF API version 3.0 services in other ways, or to develop a new API dedicated to communicating the specific information required for digital building permits and digital inspection plans. It is also possible to wait until the next version of the BCF API specification, version 4.0, gets implemented in BCF clients.

Alternatives to using only BCF API

In order to get a national system of digital inspection plans working (including digital building permits) a dedicated client could be developed. This dedicated client could be a web application and it could use a custom API tailored for the specific OIR of such a system, i.e., a "Digital inspection plan API". This "Digital inspection plan API" will be able to communicate all kinds of data that should be included in a digital inspection plan.

Such a dedicated "Digital inspection plan API" could also be integrated with an implementation of the existing BCF API so that some of the resources on the server could be accessed also using BCF API. The purpose of such an integration could be to allow users of the system to view details in building models using BCF clients. Or to allow applicants to download topics that must be corrected in a digital building model used for a building permit application. However, such an integration with BCF API would not be considered as the core functionality of a system for digital inspection plans. And users would not be able to use their BCF clients to carry out work on digital inspection plans.

A dedicated “Digital inspection plan API” could also integrate with implementations of the Documents API to facilitate the transfer of files/documents used in digital building permit applications and digital inspection plans. And the “Digital inspection plan API” could integrate with the Foundation API to enable user authentication and service discovery. It is also possible to develop a BCF client that implements the BCF API specification version 4.0 before it is released. Then there would be less difference between the “Digital inspection plan API” and the BCF API only.

Being able to use IFC-graph

The second aim of this study was to see if IFC-graph could facilitate access and query of building information useful for decisions on reuse, recycling or waste management of building products and materials in a system for digital inspection plans.

It was possible to export STEP files with IFC data to IFC-graph and it was possible to relate information in IFC-graph to other building information imported into the LPG using the BCF API. IFC-graph can potentially be used to facilitate access and query of any kind of building information from files containing IFC data. This is why IFC-graph can facilitate access and query of building information useful for decisions on reuse, recycling or waste management of building products and materials. But there are some problems related to IFC-graph that must be addressed before usage of IFC-graph can be practical in a production system.

Problems of IFC-graph

The current algorithm for export of STEP files with IFC data to IFC-graph is way too time consuming. Many solutions to this problem are discussed in the section on future work. The result is that IFC-graph is not suitable for systems where the contents of files with IFC data frequently change and thus frequently must be converted to IFC-graph.

IFC-graph should also probably only be used to analyze LPG data, not to manipulate LPG data, because Neo4j is a schemeless graph database. A solution to this problem might be validation against a scheme using the API, but then you might need to use an OGM.

If other API-related data is not stored as LPG, for example if other building information (such as BCF-data) is stored in a regular relational database, then the relations between IFC-graph and other data must be done by the API, not by the database itself. A solution to that problem would be to use a native multi model database such as ArangoDB. ArangoDB uses AQL, a completely different query language. This means that a different export algorithm (from IFC STEP file to IFC-graph) must be developed. The export algorithm has to be optimized anyway.

Future work

Suggested limitations of IFC-graph

The previously mentioned paper on IFC-graph (Junziang, Peng, & Xiang, 2023) gives a description of some limitations on the use of the proposed IFC-graph format and some aspects that are interesting to explore in future work. Two limitations are mentioned in this paper: Efficiency of graph generation and handling of geometric information.

Efficiency of graph generation

“Despite the promising outcomes of this study, limitations have been noticed during experiment, mainly regarding the efficiency of graph generation. This study adopted a conservative algorithm to generate LPG. This algorithm can assure that no duplicated nodes or relationships are generated, but this results in long processing time, because the existing graph must be traversed to ensure the to-be-

created nodes or relationships have not been created. … It can be noticed that the processing time grows quickly when the number of nodes increases. IFC-Graph is, therefore, currently more suitable to be used in stable situations, such as the operation and maintenance phase of the building lifecycle. To improve the efficiency of graph generation, two possible solutions can be considered."

"The first is to use the CREATE function of Cypher. This function directly generates nodes and relationships without traversing the graph to check duplicates, which can improve the graph generation efficiency but requires an additional mechanism to avoid generating duplicated nodes and relationships."

"The second is to use parallel computing, in which multiple processes can be simultaneously carried out. The current graph generation algorithm did not take parallel computing into consideration. If a proper parallel algorithm can be specifically designed for this purpose, the graph generation efficiency can be greatly improved."

Handling of geometric information

"The current method converts all geometric information into LPG, which might not be a good idea. (a) Frist, these explicit points in boundary representation (B-rep) can greatly increase the size of graphs, which adversely impact graph-based information query. (b) Second, graphs, if no additional labels (attributes) are used, cannot present the sequence information of ordered lists (Pauwels & Roxin, SimpleBIM: From full ifcOWL graphs to simplified building graphs, 2016)*, such as the ordered points in IfcPolyLoop instances. These two problems should be properly handled in the future."*

Comments on limitations

It took several hours on a slow Ubuntu server, to generate an IFC-graph from a normal size IFC STEP file containing about one hundred thousand lines of STEP code, using the python algorithm provided by (Junziang, Peng, & Xiang, 2023). The process of adding nodes got slower and slower as the graph increased in size. This means that the time to generate IFC-graph will get slower and slower as the database grows.

So long time to generate an IFC-graph is not practical and makes frequent conversions to IFC-graph impossible. As mentioned in the paper (Junziang, Peng, & Xiang, 2023) there are ways to speed up the algorithm. Using the Cypher CREATE statement to create nodes and relationships instead of merging them into the graph is one way. There can also be other ways to make the algorithm faster. Here are some ideas:

- Use the official python driver for Neo4j instead of the py2neo library.
- Use iterated precompiled Cypher queries with variable parameters (Allen, 2022).
- Use python to perform the merging on data structures, so that CREATE statements can be used instead of MERGE statements in Cypher. Then use some of the batch import or load features of the APOC library for Neo4j (Neo4j, 2023). NetworkX might be suitable for this purpose. NetworkX graph will be generated from IFC STEP file. NetworkX can dump a graph as graphml which can be imported into Neo4j.
- Use a compiled programming language for IFC-graph generation such as C#, instead of python.
- The most practical solution would probably be to write a custom STEP file import library for Neo4j in Java. This library would be loaded into the Neo4j database server and would provide functions for STEP file loading and IFC-graph generation that can be called within Cypher queries regardless of programming language.

Suggested future work on IFC-graph

Four aspects to explore in future work are mentioned by the paper on IFC-graph generation (Junziang, Peng, & Xiang, 2023): Handling multiple IFC datasets in LPG, tailoring Cypher for building information queries, proper handling of geometric information in graphs, and industry support for the generation of LPG.

Handling multiple IFC datasets in LPG

"This study stores one IFC model in one graph database, which has no problem for information querying from one model but not for situations where multiple buildings are to be involved. Ismail et al. (Ismail, Nahar, & Scherer) managed to store multiple IFC datasets in one graph database, where models are distinguished by using a unique label. However, this workaround would likely lead to over-sized graphs if the model-driven approach is used, which may adversely influence the efficiency of information querying. Considering that information query from multiple buildings might be required in the context of smart city and digital twins, a better way for handling multiple IFC datasets using graph database should be investigated."

Comments on handling multiple IFC datasets in LPG

There are probably only a few distinct methods to distinguish between multiple IFC datasets using Neo4j, besides the solution of using unique labels for identification nodes of different buildings, as already suggested by (Ismail, Nahar, & Scherer, 2023).

Neo4j enterprise edition can manage multiple graphs in multiple databases on the same server. Each building can then be saved into a separate graph in a separate database on the same server. Queries across multiple databases on the same server can be created in Neo4j by using a composite database that merges several databases into one functional unit. A composite database makes it possible to access all the graphs in all the databases on the server using a query across multiple shards (Neo4j, 2023).

- RedisGraph natively support queries across multiple graphs on the same database server, as well as a subset of Cypher, but unfortunately RedisGraph will be discontinued (Redis, 2023).
- Make a distinction between separate buildings by using separate subgraphs indicated by some unique relationship. This solution is only possible if all nodes that are representing a building are connected to each other by known relationships. Each Cypher query of any path of a building can relate to a node that identifies the building. IfcProject could be that node. IfcProject inherits IfcRoot which means that IfcProject could be referenced by its GlobalId/UUID.
  The problem with this solution is that a path to IfcProject could consist of relationships with many different labels which can make it difficult to write short, valid and future proof Cypher queries.
- A solution to that problem could be to let all connected nodes connect using an extra relationship with a single known label that always leads to or from the IfcProject node. Then it would be easy to traverse the graph from any node back to IfcProject using variable length relationship pattern matching (such as [r:CONNECTED_TO*]->(IfcProject)) (Neo4j, 2023). The IfcProject node could in turn have a relationship to a document reference node of the BCF data that points to a file with IFC data on the file system. This will make it easier to write shorter and simpler Cypher queries that relate to IfcProject by its GlobalId/UUID.
- A third way is to make use of the fact that all IFC entities that inherit IfcRoot have a GlobalId which is a UUID. This means for example, that a component with a UUID that has been sent to the server using Solibri and BCF API will match with the same component in the right building in the IFC-graph, even if the query would cover all components of all building. This is

true simply because there will most likely never be any other IFC entity with the same UUID in any other building. UUID is universally unique regardless of building.

The solution already suggested and used by (Ismail, Nahar, & Scherer, 2023) will probably also work.

Tailoring Cypher for building information query

"Cypher is designed for general graph information query, not specifically for building information. Even though this study has provided Cypher commands for querying some common building information, such as spatial structure, building element connectivity, and space accessibility, these commands have a limited coverage and have not been finely tuned for high performance. Poorly designed queries can lead to long querying time. Therefore, to make IFC-Graph more practical, tailoring Cypher for efficient building information query should be further investigated."

Comments on tailoring Cypher queries

The paper on IFC-graph (Junziang, Peng, & Xiang, 2023) gives some examples of Cypher queries building information retrieval from IFC-graph. For examples finding all the spaces that are separated by a door:

```
MATCH (n1:IfcSpace)
 -(r1:IfcRelSpaceBoundary)
 -(n2:IfcDoor)
 -(r2:IfcRelSpaceBoundary)
 -(n3:IfcSpace)
```

The following Cypher query can be used to find the value of a property (for example "GTIN") of a property set (for example "Reuse") of any object that inherits IfcProduct and is identified by its GlobalId/UUID.

```
MATCH (prod:IfcProduct)
 -[r1:IsDefinedBy]->(rs:IfcRelationship)
 -[r2:RelatingPropertyDefinition]->(ps:IfcPropertySet)
 -[r3:HasProperties]->(prop:IfcProperty)
 -[r4:NominalValue]->(root:Root)
WHERE prod.GlobalId = '1Xezbq735AdBO_4DerG66G'
 AND ps.Name = 'Reuse'
 AND prop.Name = 'GTIN'
RETURN root.wrappedValue
```

Maybe the IFC-graph can (and should or have to) be optimized for enabling shorter and more efficient Cypher queries. For example, maybe instances of some classes can be imported as relationships instead of nodes. Maybe multiple instances of some classes can be imported as one node instead of several nodes.

If IfcRelSpaceBoundary is imported as a relationship instead of a node, then the query for finding all the spaces separated by doors could be rewritten as:

```
```

```
MATCH (n1:IfcSpace)
 -[r1:IfcRelSpaceBoundary]-(n2:IfcDoor)
 -[r2:IfcRelSpaceBoundary]-(n3:IfcSpace)

MATCH (n1:IfcSpace)-[r1:IfcRelSpaceBoundary*2..2]-(n3:IfcSpace)

…or simply:

MATCH (n1:IfcSpace)-(n2:IfcDoor)-(n3:IfcSpace)
```

This query would then produce the same result but is easier to read and understand and will execute faster. However, optimizing the graph for this purpose could lead to information loss. This practice would also require that every Cypher developer knows how the IFC-graph import algorithm transforms the IFC EXPRESS schemes into schemeless LPG in Neo4j. More compact graphs requiring less memory and enabling more compact queries could potentially also be achieved by using the more complex modelling principles of TypeDB (TypeDB, 2022).

Properly handling geometric information in graph

"In the area of IFC-to-RDF conversion, Pauwels et al. (Pauwels & Terkaj, EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology, 2016) *managed to develop a method to preserve the ordered lists in RDF, which are mainly used for geometric information, such as boundary representation. However, this would lead to large graph size. The way geometric information should be managed in graphs should be further investigated. It is even possible to exclude geometric information from graphs, as according to Pauwels* (Pauwels & Roxin, SimpleBIM: From full ifcOWL graphs to simplified building graphs, 2016)*, the release of geometrical and representation data can result in a reduction of 31.9% of triples."*

Comments on properly handling geometric information in graph

Neo4j natively supports ordered lists as a datatype for properties. It would be practical if ordered lists could be exported into IFC-graph as a property with the "list" datatype. This would decrease the size of the graph. Neo4j should also be able to assign a list of coordinates in space to a property (Neo4j, 2023). This is a list with POINT as inner type, i.e., a list of points.

Industry support for the generation of LPG

"Another limitation of the LPG application is the gap between practitioners who create and use building models and the tools developed for accessing and developing LPG. When RDF was introduced into the BIM community, it was expected to see that some applications or extensions would be developed, e.g., in Tekla or Revit, to easily export building models into RDF, but it never happened. This problem might happen to LPG as well and should be considered in the future, but before that, more research is required to show the full potential of LPG."

Comments on industry support for IFC-graph

IFC-graph will probably not be used by applications such as Tekla, Revit, BlenderBIM or Archicad directly, because such applications are normally used to design one building (or a few buildings) at a time. Such software needs to access data internally in computer memory and there is probably no benefit of importing an IFC-graph into computer memory, compared to importing a STEP file with IFC data into computer memory. There is no way to export an IFC-graph into computer memory at all now. There is not even a way to round trip an IFC-graph back into a STEP file with IFC data.

Suggestions on industry use of IFC-graph

IFC-graph probably makes most sense to use in systems where you frequently need to access data from very many buildings at the same time, while the data of building models do not change very often. There is no way to manipulate IFC-graph using Cypher while validating the instance graph against a database scheme. This is why Cypher probably should not be used to manipulate IFC graph. Every time a change is made to a model in proprietary software, a new expert to a STEP file with IFC data must thus be generated and then this file must be exported into IFC-graph. The process of exporting STEP files with IFC data into IFC-graph is time consuming using the existing algorithm. So, this export cannot be done very frequently.

The problem with STEP files with IFC data on the other hand, is that its content must be completely parsed and loaded into computer memory before data can be accessed by a service or software. STEP files with IFC data are relatively large. If a software or service must access data from many buildings frequently and at the same time, then many STEP files must be parsed and loaded into computer memory frequently and at the same time. A database solution could be a more efficient solution in this scenario, and since STEP files stores graph data, a graph database would probably often be the best option.

But if the IFC-graph must change frequently, then the extra time needed for frequently exporting STEP files with IFC data into IFC-graph will be too large, compared to the time savings earned from being able to use queries to read data directly from the database, instead of having to parse STEP files to read IFC data. This means that IFC-graph will probably not be used by software such as Tekla, Revit or Archicad, but rather by systems that frequently might need to access data from many models of buildings at the same time.

If a BCF client uses native IFC in a version control system, then it might be possible to only update the part of IFC graph that has been changed in a commit of IFC data. Then importing the whole IFC graph again might not be necessary. This is why it is interesting to combine IFC graph with software capable of working in native IFC.

A system for digital inspection plans

In order to get a functional system of digital inspection plans working a dedicated client could be developed. This dedicated client could be a web application and it could use a custom API tailored for the specific OIR of such a system, i.e., a “Digital inspection plan API”.

This “Digital inspection plan API” will be able to communicate all kinds of data that should be included in a digital inspection plan. This means that such a system will be able to meet all OIR according to the legislation. And then all possibilities of such a system to support a circular construction sector described in the introduction part of this paper will be enabled.

Such a dedicated “Digital inspection plan API” could also integrate with an implementation of the BCF API, version 3.0. The purpose of such an integration could be to allow users of the system to 3D-view details in building models using BCF clients. However, such an integration with BCF API would not be considered as the core functionality of a system for digital inspection plans.

Such a dedicated “Digital inspection plan API” could also integrate with an implementation of the next BCF API version, version 4.0. Then the difference between the “Digital inspection plan API” and the BCF API version 4.0 does not need to be large.

A dedicated “Digital inspection plan API” could also integrate with implementations of the Documents API to facilitate the transfer of files used in applications and inspection plans. And the “Digital inspection plan API” could integrate with the Foundation API to enable API version discovery and user authentication etc.

References

Allen, D. (2022, February 16). *Neo4j driver best practises*. Retrieved from Neo4j: https://neo4j.com/developer-blog/neo4j-driver-best-practices/

BIMformation AB. (2020). *Automatiserade standardiserade kontroller.* Stockholm: SBUF.

Boverket. (2021, March 31). *Building process*. Retrieved from Boverket: https://www.boverket.se/en/start/building-in-sweden/developer/production/building-process/

Boverket. (2023). *Digitalisering av kontrollplaner - Förstudie.* Karlskrona: Boverket.

Boverket. (2023, January 9). *Utsläpp av växthusgaser från bygg- och fastighetssektorn*. Retrieved from Boverket: https://www.boverket.se/sv/byggande/hallbart-byggande-och-forvaltning/miljoindikatorer---aktuell-status/vaxthusgaser/

buildingSMART. (2020, September 16). *Allow Clients to show custom Topic Properties*. Retrieved from Github: https://github.com/buildingSMART/BCF-API/issues/219

buildingSMART. (2021, June 18). *BCF REST API 3.0*. Retrieved from Github: https://github.com/buildingSMART/BCF-API

buildingSMART. (2021, June 25). *OpenCDE Foundation API 1.0*. Retrieved from Github: https://github.com/BuildingSMART/foundation-API/tree/v1.0

buildingSMART. (2021, March 3). *Remove BIM snippet*. Retrieved from Github: https://github.com/buildingSMART/BCF-XML/issues/314

buildingSMART. (2023, February 17). *BCF API*. Retrieved from SwaggerHub: https://app.swaggerhub.com/apis/buildingSMART/BCF/3.0

buildingSMART. (n.d.). *The EXPRESS Definition Language for IFC Development.* Retrieved from buildingSMART: https://standards.buildingsmart.org/documents/Implementation/The_EXPRESS_Definition_Language_for_IFC_Development.pdf

Dangl, G. (2023, September 13). *Dangl.OpenCDE*. Retrieved from Github: https://github.com/Dangl-IT/Dangl.OpenCDE

Electronic frontier foundation. (2023, September 13). *Certbot*. Retrieved from Electronic frontier foundation: https://certbot.eff.org/

Environmental Protection Agency. (2019). *Establishin effective markets for secondary building materials.* Copenhagen: Ministry of Environment and Food of Denmark.

European Comission. (2023, September 13). *Buildings and construction*. Retrieved from European Comission: https://single-market-economy.ec.europa.eu/industry/sustainability/buildings-and-construction_en

Fager, E., & Johansson, F. (2022). *På väg mot återanvändning och återvinning av hög kvalitet - En studie öve rhinder och lösningar för icrkulära materialflöden i bygg- och rivningsbranschen samt hur detta pvåerkas av implementeringen av PBL 10 kap. 6, 11 och 19 §§.* Lund: Lunds tekniska högskola.

FastAPI. (2023, September 13). *FastAPI*. Retrieved from FastAPI: https://fastapi.tiangolo.com/

Finansdepartementet. (2022, February 2). *Uppdrag att utveckla arbetet med omställningen till en cirkulär ekonomi i byggsektorn.* Retrieved from Regeringen: https://www.regeringen.se/regeringsuppdrag/2022/02/uppdrag-att-utveckla-arbetet-med-omstallningen-till-en-cirkular-ekonomi-i-byggsektorn/

Finansdepartementet. (2022, april 28). *Uppdrag om fler lösningar som främjar en enhetlig tillämpning av plan- och bygglagen (2010:900) i en digital miljö.* Retrieved from Regeringen: https://www.regeringen.se/regeringsuppdrag/2022/04/uppdrag-om-fler-losningar-som-framjar-en-enhetlig-tillampning-av-plan--och-bygglagen-2010900-i-en-digital-miljo/

IfcOpenShell. (2023, September 13). *IfcOpenShell The open source IFC tookit and geometry engine*. Retrieved from IfcOpenShell: https://ifcopenshell.org/

Ismail, A., Nahar, A., & Scherer, R. (2023, September 13). *Application of graph databases and graph theory concepts for advanced analysing of BIM models based on IFC standard.* Retrieved from IFCwebserver.org: https://ifcwebserver.org/doc/ifc2gdb_eg-ice2017_ismail.pdf

Junziang, Z., Peng, W., & Xiang, L. (2023, April). IFC-graph for facilitating building information access and query. *Automation in Construction*.

Knoth, K., Seilskjaer, E., & Mamo Fufa, S. (2022). Barriers, success factors, and perspectives for the reuse of construction products in Norway. *Journal of Cleaner Production*.

Medium. (2017, May 5). *Comparing Neo4j driver, py2neo and neo4jrestclient with some basic commands using the Panama Papers Data*. Retrieved from Medium: https://medium.com/@hisanor714/comparing-py2neo-and-neo4jrestclient-with-some-basic-commands-using-the-panama-papers-data-223a67bbe533

Neo4j. (2023, September 13). *Import*. Retrieved from Neo4j: https://neo4j.com/docs/apoc/current/import/

Neo4j. (2023, September 13). *Set up and use a Composite database.* Retrieved from Neo4j: https://neo4j.com/docs/operations-manual/current/composite-databases/

Neo4j. (2023, September 13). *Spatial values*. Retrieved from Neo4j: https://neo4j.com/docs/Cypher-manual/current/values-and-types/spatial/

Neo4j. (2023, September 13). *Syntax and semantics.* Retrieved from Neo4j: https://neo4j.com/docs/Cypher-manual/current/patterns/reference/

Neo4j. (2023, September 13). *Time To Live (TTL) - Expire Nodes*. Retrieved from Neo4j: https://neo4j.com/labs/apoc/4.3/graph-updates/ttl/

Norsk standard. (2019, March 1). Organisering og digitalisering av informasjon om byggverk, inkludert bygningsinformasjonsmodellering (BIM) Informasjonsforvaltning med BIM Del 1: Begreper og prinsipper. *NS-EN ISO 19650-1:2018* .

Pauwels, P., & Roxin, A. (2016). SimpleBIM: From full ifcOWL graphs to simplified building graphs. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2016*.

Pauwels, P., & Terkaj, W. (2016). EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology. *Automation in Construction*, 100-133.

Pei-Yu, W., Sandels, C., Mjörnell, K., Mangold, M., & Johansson, T. (2022). Predicting the presence of hazardous materials in buildings using machine learning. *Building and Environment*.

Redis. (2023, July 5). *RedisGraph End-of-Life Announcement.* Retrieved from Redis: https://redis.com/blog/redisgraph-eol/

Regeringen. (2020). *Proposition 2019/20:156 Genomförande av EU-direktiv på avfallsområdet.* Stockholm: Regeringen.

Riksdagen. (2010, July 1). Plan- och bygglag (2010:900). *SFS*.

Schulz, O., Oraskari, J., & Beetz, J. (2021). bcfOWL: A BIM collaboration ontology. *Linked Data in Architecture and Construction.* Department of Design Computation, RWTH Aachen University, Aachen, Germany.

Singer, J. (2021, September 11). *Which Python Driver Should I Use To Access Neo4j?* Retrieved from SingerLinks: https://singerlinks.com/2020/10/should-i-use-py2neo-or-the-native-python-driver-to-access-neo4j/

Solibri. (2023, September 13). *Using the Documents API Integration*. Retrieved from Solibri: https://help.solibri.com/hc/en-us/articles/10233885447447-Using-the-Documents-API-Integration

Sundqvist, J.-O. (2022). *Kort om byggavfallsstatistik*. Retrieved from Svenska miljöemissionsdata (SMED): https://www.diva-portal.org/smash/get/diva2:1647853/FULLTEXT01.pdf

TypeDB. (2022, April 6). *TypeDB Academy: Modelling principles.* Retrieved from Youtube: https://www.youtube.com/watch?v=vfSAgFayLnk