Code compliance checking for electrical systems within BIM models

Mart van Saane

2003219

A thesis submitted for the degree of Master of Science in Computer Games

Supervisor: Prof. Stuart D. Walker
Industrial Supervisor: Dr. Anasol Peña-Rios
School of Computer Science and Electronic Engineering
University of Essex

August 2021

Abstract

Manual code compliance checking is prone to errors, which can lead to big costs in the construction sector. Automatic code compliance checking algorithms within BIM have yet to see a good adaptation within the BIM workflow, but it could greatly benefit from it. As part of this research a proof of concept application has been developed that provides tools to assist in code compliance checking of electrical systems within an IFC file. Basing the application on IFC has made it easily adoptable with existing BIM applications. The tool allows the user to easily examine and analyse the electrical network in a building by providing 2D network graphs. And allowing the user find the shortest path between two elements in the electrical network.

Keywords: Building Information Modeling (BIM), Industry Foundation Classes (IFC), Mechanical, Electrical and Plumbing (MEP), code compliance checking

Table of Contents

Introduction

Building Information Modelling (BIM) has been revolutionizing the construction sector for the last two decades. The UK government was one of the first advocators of the integration of BIM. Which has already resulted in 73% of UK companies adopting the technology in some of their work, and only 1% not knowing about the technique (NBS, 2020).

BIM aims to centralize all the data for a construction project between all its shareholders. This dynamic data model starts at the design phase, but also plays a vital part in the construction phase. And can even be transferred after construction to the client for maintenance use.

A model containing a large amount of data like BIM is prone to contain errors. Therefore all the data in the model must be checked to see if it satisfies certain standards. This process of checking is also referred to as code compliance checking. Traditionally errors have to be discovered and fixed by manual code compliance checking. This is very time-consuming process and leads to errors in itself as well (Ismail et al, 2017).

This research has been done in collaboration with BT. As part of their Network Digital Twins project, they are looking to build applications to leverage the capabilities of BIM data (Leon-Garza et al, 2020). Where they are mostly interested in the electrical components within a BIM model, as that ties in with BT’s work.

So in this research a code compliance checking approach will proposed, suitable for analysing the electrical network in a BIM model. It was planned for it to be an automatic system, but in the end, it has been developed in a tool for assisting the user in the manual process. How the tool could be extended to an automatic approach will however be discussed as well.

1 Background knowledge

1.1 BIM

Building Information Modelling, from this point referred to as BIM, is a process for creating and managing information in construction shared by all stakeholders. BIM differs from conventional approaches, like the use of CAD, by centralizing all the information in one data form. Previously if one floorplan changed, this would have to be manually updated in all other models and sketches. Leading to a lot of mistakes. BIM prevents this by automatically updating all information in the data, when it gets changed in one place.

BIM models are generally split between three major domains, illustrated in figure 1. First of all, there is an architectural layer, responsible for the architecture of the building. So this layer determines how the final building will look on the surface. Secondly a structural layer is made. Which determines the geometry and materials used for the structure. Generally this layer is covered by the architectural layer, and therefore not visible from the outside. Lastly, the most important layer for this research which is the MEP layer.

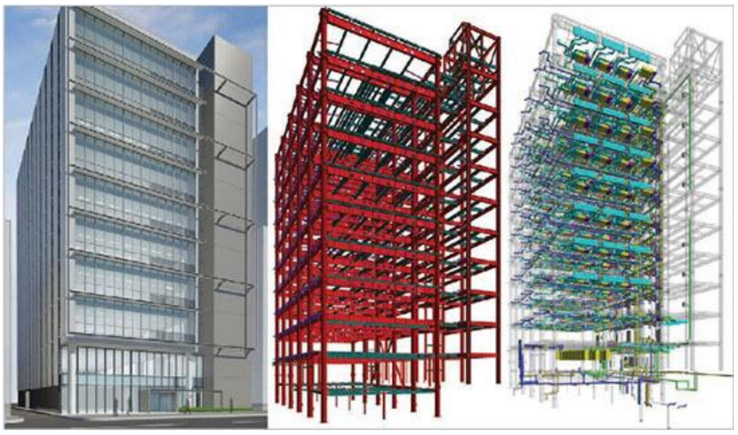


Figure 1. Example different layers in BIM. From left to right: architectural, structural, MEP (Innovation In Building, 2010)

The Mechanical, Electrical and Plumbing (MEP) layer mainly contains components the following three categories: ducts for Heating, Ventilation and Air conditioning (HVAC), piping for water and gas, and lastly cable trays and control boxes for electrical and communication systems. Within this layer the cable trays are mainly the point of interested. Where the specific cabling is often flexible, the cable trays are fixed. Therefore the routing of electric and network cables must be based upon this (Eastman et al, 2008). This makes it essential to check if these cable trays are placed and connected correctly before construction. For which this research will propose an application to aid in this process.

1.2 IFC

Industry Foundation Classes (IFC) is an open international specification to share BIM data between various actors in building construction. The application developed for this research is exclusively build upon IFC. An added benefit of having the application based on IFC files, is that it can be used regardless of the software in which the BIM model has been produced.

IFC is defined in four conceptual layers (Bazjanac & Crawley, 1997), with each layer containing an associated data scheme (buildingSmart, 2020), as shown in figure 2.

The first layer is the **resource layer**, this layer defines all low-level features unrelated to specific industries. Therefore a resource in this layer can't exist on its own, and is only to be used as a reference for the higher layers. All the elements in this layer are part of the resource definition data schemes. This layer for example contains the IfcDateTime data scheme, which in itself is not part of a building, but it can by referenced to by components in the building.

The second layer in the IFC architecture is the **core layer**, connected to the core data schemes. This layer is used to define the features of the resource layers into products, processes and controls. From this layer onwards, all data schemes derive from IfcRoot, meaning they contain an unique identifier, name and description. Elements within this layer are mostly used to define relations within a BIM model.

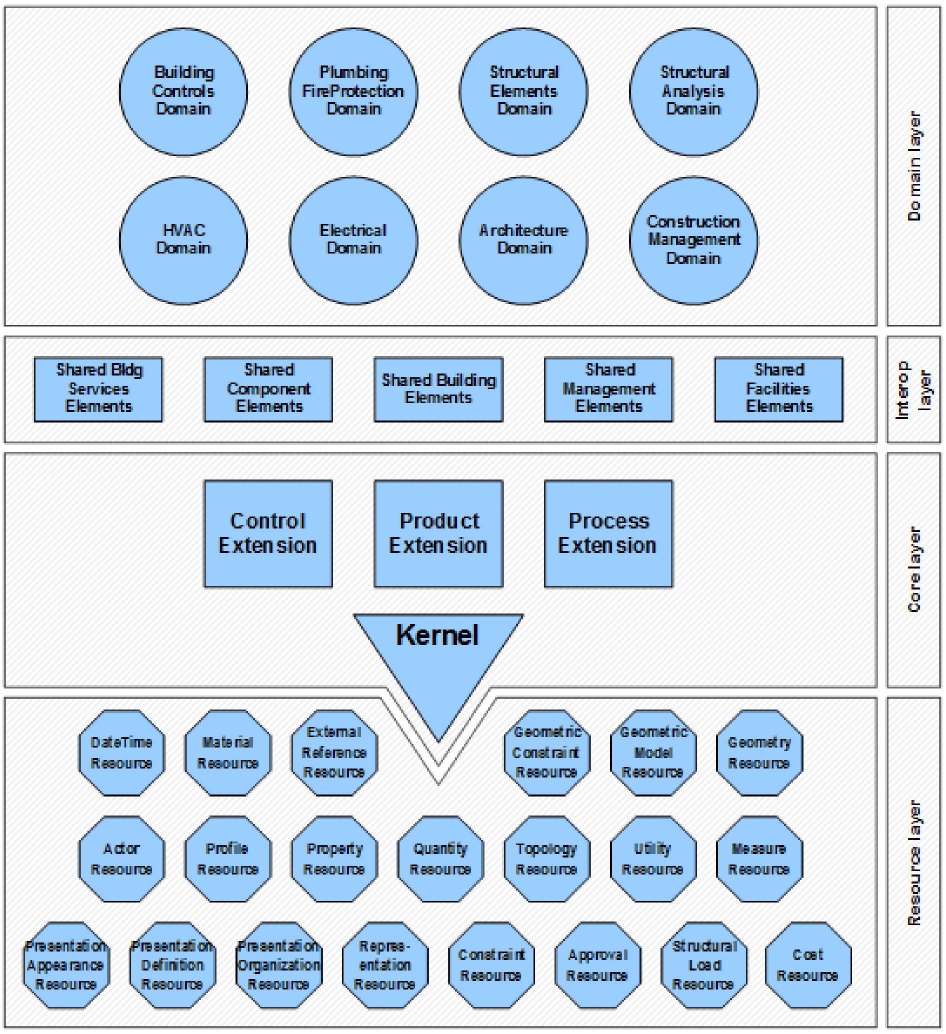


Figure 1. Data architecture of IFC with conceptual layers (buildingSmart, 2020)

The **interop layer** contains models to serve the domain layer. It contains definitions for specific to general products like wall, column and door. It’s mostly used as a means to share common information between different industries, therefore these elements are contained in the shared element data schemas. Interesting for this research are the IfcFlowSegment and IfcFlowFitting elements in this layer, which will be further elaborated on in section 2.4.

Lastly, the **domain layer** contains definitions for industry specific objects. Here a differentiation between the architectural, structural and MEP domains can once again be noticed. Where the MEP layer has been split into the domains of: HVAC (heating, ventilating and air condition), Electrical, Plumbing FireProtection and Building Controls. The elements in this layer are part of the domain specific data schemas. None of the elements in this layer are directly used in this research.

1.3 IfcOpenShell

Within the development of the application IfcOpenShell has extensively been used. IfcOpenShell is an open-source software library designed to assist with the development of applications using IFC (IfcOpenShell, 2021). Part of this toolset is a python library that allows for easy parsing of IFC files. For example, the library can be used to lookup all the components belonging to a certain IFC tag. A technique that will be further elaborated in section 2.4.

Additionally the toolkit also contains an application for converting IFC files into several file formats, named IfcConvert. This application allows IFC geometry to be converted into traditional geometry file types like: WaveFront OBJ, Collada, STEP and IGES. And also allows the metadata in the IFC files to be converted to XML. Which allows for much easier parsing.

1.4 Shortest path algorithm

The goal of a shortest path algorithm is to find a path from a start point to an end point in a network. There are many algorithms to achieve this, a lot of these are based on Dijkstra’s algorithm. Seeing as it is one of the simpler algorithms, that is the one we will focus on in this project.

Dijkstra’s algorithm can be broken down in a few steps (Candra et al, 2020):

1) *Set a distance value to every node. This will be zero for the start node, and infinite for any other node.*
2) *Put all the nodes in an open set, which identifies them as unvisited.*
3) *Get the node with the lowest distance value from the open set, and remove it.*
4) *Calculate distance to all the neighbours of this node. Compare the current distance assigned to a neighbour to this newly calculated distance, and assign the smaller one.*
5) *Repeat from step 3 until the node defined as end point is found. Or until the open set only contains nodes with a distance value of infinity, in which case no path exists between the given start and end point.*

1.5 Related work

Studies exploring automated design and code compliance checking have been conducted since the 1960s, this research has especially been fuelled be the recent surgency of BIM. Designers and local authorities traditionally have to do the code compliance checking manually. Which introduces a lot of room for error, and it is a very time-consuming task. While it is essential for these errors to be spotted during the design phase, as to avoid causing costly reworks of the building at a later date.

Automatic code compliance is currently mostly implemented with rule-based approaches, where it generally only focuses on specific regulations (Ismail et al, 2017). This also applies to this research, because it only focuses on the domain of the electrical network.

Research has been done into code compliance checking software for water distribution systems. For this application the LicA database has been developed (Martins and Abrantes, 2010). Information in the domain of water distribution systems in Portugal is included in this database. After extracting all the relevant data from a BIM model, with the use of an automatic converter. The data can then be put through the database to perform the automatic code compliance checking. After which the user can use a separate application (LiCAD) to see the result of the checks. These results are indicted with a colour coding system, telling the user the amount of compliance a specific element has to the domestic regulations.

In the initial study of the LicA application on specific BIM filetypes. But in later studies the algorithm was also adopted to work on IFC files (Martins and Monteiro, 2013). Although they did not succeed to get the exact same results as with the BIM files. As the database turned out to be only compatible with IFC for 90%. So this should note that an application based on IFC is slightly more limited than an application based on BIM files. But the advantage is that the application gets a lot more versatile, because it will be able to interact with any other BIM software.

2 Design and Implementation

2.1 Requirements

In this section the requirements for the application developed for this research will be discussed. These requirements are a mix of the requirements determined at the project proposal, and requirements determined during development.

Functional requirements

- 🟩 The application shall have a GUI
- 🟩 The GUI shall have a hierarchy, showing the structure in the BIM model
- 🟩 The GUI shall be able to enable and disable certain BIM objects
- 🟩 The GUI shall show the user the metadata connected to a BIM object
- 🟩 The user should be able to search in the GUI hierarchy
- 🟩 The user should be able to save the conversion from IFC to OBJ and XML
- 🟨 The application should have a VR controller
- 🟥 The application should have an algorithm to highlight missing information
- 🟩 The GUI may contain a network view showing the electric network
- 🟩 The user may be able to find the shortest path between the nodes in the network view

External interface requirements

- 🟩 The application shall run on Windows
- 🟥 The application should run on Unity WebGL

The colours show the status of the requirements. Where green requirements have been completed successfully, yellow requirement have partly been developed and the red requirements have not been implemented. Why some of these requirements have failed will be elaborated now.

Initially the application was envisioned to have a VR controller. The application does contain a camera controller, to allow the user to navigate around the BIM model. However, adapting this to a VR controller or headset was not deemed necessary during development.

Similarly the application was planned to contain or interact with a machine learning algorithm to highlight missing information in the electric components of the BIM model. Because there was not enough time to implement this in this research. It was instead opted to develop an algorithm to find the shortest path between two nodes in the electrical network.

The application has been developed in Unity, which is a cross-platform game engine. One of the platforms Unity can export to is WebGL, that allows the application to run in a browser, making it much more accessible to people. Therefore the application was supposed to be web-based, with the use of the built-in WebGL build option in Unity. However it is much more difficult within WebGL to access files on the user's computer. Which is vital, as the user has to upload an IFC file. It is possible to interact with the file system if the WebGL client interacts with a local PHP server. However problems with the installation of PHP occurred during the development stage. Which is why the web client was not deemed necessary at this point in the application. And instead the project has been exported as a windows application.

2.2 Application workflow

This section will discuss the workflow of the application accompanied with figure 3.

The user starts with importing an IFC file in the application. This IFC file will be converted to three different file types by the IfcConvert application belonging to the IfcOpenShell library (section 1.3). The resulting files are an OBJ, MTL and XML file.

The OBJ file includes all the geometry included in the BIM model, separated in individual BIM objects. This file gets parsed on a separate thread to retrieve all the vertices, uvs, normals and faces, as to not block the main Unity thread, because Unity only runs on one thread. Every face in the geometry is parsed as having double sided normals. The generation of GameObjects, that allow to geometry to be rendered by Unity, can only be done on the main thread however. So after parsing the OBJ file, the data will be included in GameObjects on the main thread.

To parse the OBJ file correctly the MTL file is needed, to apply the correct materials to all the objects. This importer can run alongside the OBJ importer, where the materials can be applied during the generation of GameObjects. The MTL importer has been designed to be able to parse the diffuse and specular colours of materials, as well as the alpha and glossiness value of the material. In reality, a MTL file can contain more variables, however these variables did not seem to be generated by the algorithm of IfcConvert. So they got left out to be able to easily convert to Unity's material system.

One more limitation of the materials is that only one material is applied to a single object. While some objects may contain multiple. It's impossible within Unity to define which faces use which material, therefore only the first material encountered in the OBJ file will be used. It would be possible to convert a single object into multiple objects to allow for multiple materials. But because of a time constrained, this didn't get implemented as part of this application.

After the OBJ file has been parsed, the application will have a list of GameObjects named after their GUIDs. To attach meaningful data to the geometry, the XML file generated by IfcConvert also needs to be parsed. This file contains all the metadata connected to the IFC file, including the hierarchy of all the objects.

From the XML file all the IFC elements are generated. If a GameObject with the GUID of the IFC element exist, it will be connected to that element. An IFC element doesn't necessarily own any geometry, often times it does contain geometry in its children. All the GameObjects are put in the same hierarchy as found in the XML file, this makes easy to disable or enable the geometry, while making sure the children also do the same.

After having defined all the IFC elements. The hierarchy view and the network view in the GUI is also be generated. More information on this can be found in sections 2.3 and 2.4.

Because the conversion of IFC by IfcConvert is time-consuming, the application also allows the user to save the conversion in case they want to check the BIM model at a future date as well. Normally the OBJ, MTL and XML files will only be stored temporarily. But when the user decides to store the conversion, they can define a location in their file system too store these three files. Which can then be used at a later date to import the BIM model directly from the OBJ file. For this to work the application will check to see if an XML file with the same name exists in the same folder. Because they are both required for the application to successfully import the BIM model.

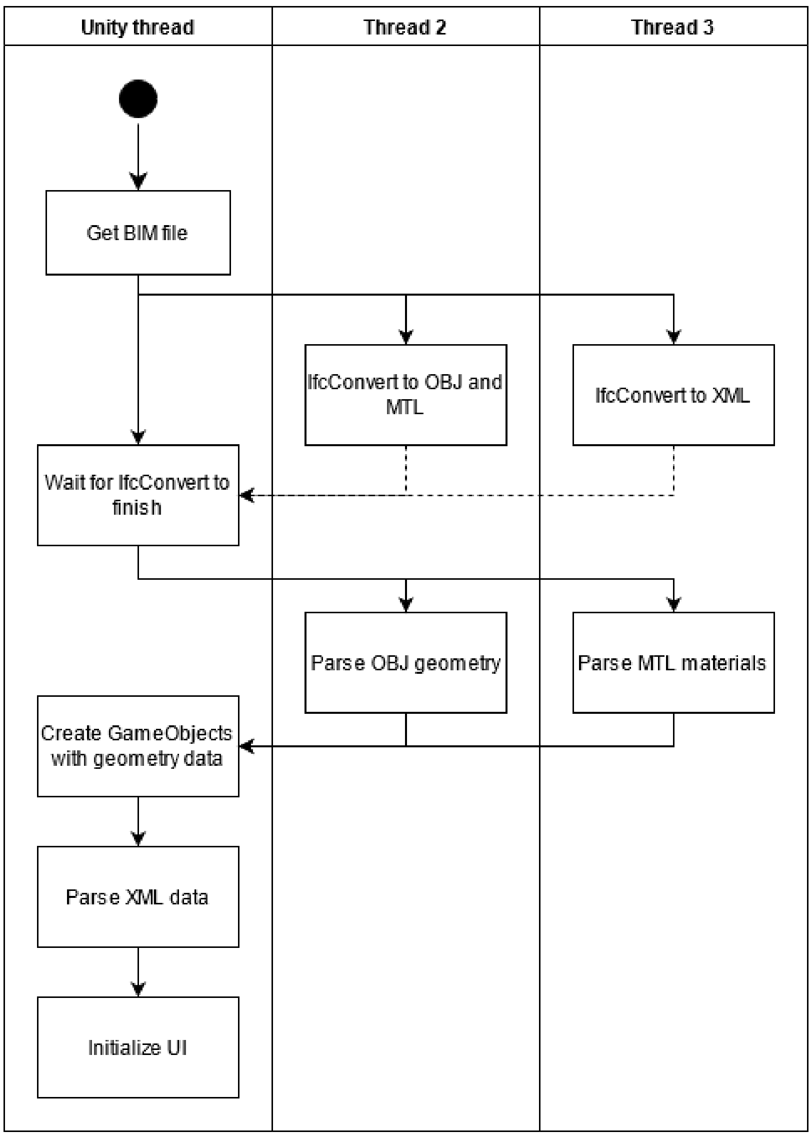
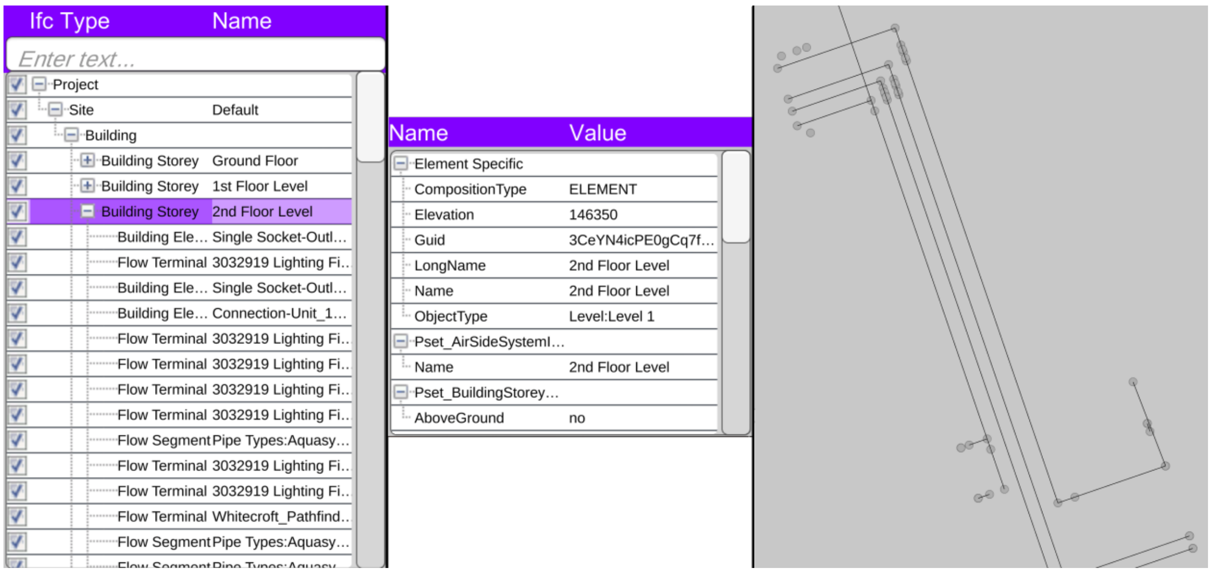


Figure 3. Activity diagram of the application

2.3 GUI

A major part of the work while developing the application went into the design of the graphical user interface (GUI). The design can be identified as three major parts. First of all the application has a tree view, containing all the separate elements and the respective hierarchy as contained in the BIM model. The user can use this to find specific BIM elements. Connected to this part is a property view, which shows the user all the metadata connected to the currently selected element. Lastly the GUI contains a network view, allowing the user to easily check the electrical components in the BIM file, this will be further elaborated in section 2.4. These three major parts are depicted in figure 4 .



Enter text...

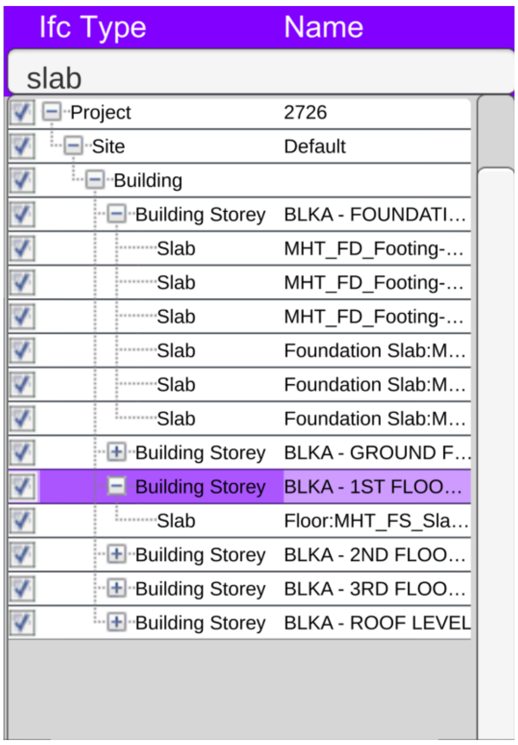
Ifc Type	Name
Project	
Site	Default
Building	
Building Storey	Ground Floor
Building Storey	1st Floor Level
Building Storey	2nd Floor Level
Building Ele…	Single Socket-Outl…
Flow Terminal	3032919 Lighting Fi..
Building Ele…	Single Socket-Outl…
Building Ele…	Connection-Unit_1…
Flow Terminal	3032919 Lighting Fi..
Flow Terminal	3032919 Lighting Fi..
Flow Terminal	3032919 Lighting Fi..
Flow Terminal	3032919 Lighting Fi..
Flow Segment	Pipe Types:Aquasy…
Flow Terminal	3032919 Lighting Fi..
Flow Terminal	3032919 Lighting Fi..
Flow Terminal	Whitecroft_Pathfind..
Flow Segment	Pipe Types:Aquasy…

Name	Value
Element Specific	
CompositionType	ELEMENT
Elevation	146350
Guid	3CeYN4icPE0gCq7f…
LongName	2nd Floor Level
Name	2nd Floor Level
ObjectType	Level:Level 1
Pset_AirSideSystemI…	
Name	2nd Floor Level
Pset_BuildingStorey…	
AboveGround	no

Figure 4. Picture displaying the GUI, with from left to right the tree view, property view and network view.

The tree view took the most amount of work. This was caused by the BIM file structure having a hierarchical structure, which also needed to be reflected in the GUI. Therefore the design was based of two tree view design guidelines made by Adobe (Adobe 2021) and Rivers (Rivers 2020).

Most importantly from this design is the search function. While the user searches the tree for something, the tree needs to filter out all the elements which don’t satisfy the search query. But if an element does satisfy the search query, then all its parents also still need to be included in the GUI. Otherwise the hierarchical structure is lost. Additionally the tree view should also be consistent, so the tree should remain in the current state of which parents have been expanded. Otherwise the user will get lost in which part of the hierarchy they were working in. This is demonstrated in figure 5.



Ifc Type	Name
slab	
Project	2726
Site	Default
Building	
Building Storey	BLKA - FOUNDATI…
Slab	MHT_FD_Footing-…
Slab	MHT_FD_Footing-…
Slab	MHT_FD_Footing-…
Slab	Foundation Slab:M…
Slab	Foundation Slab:M…
Slab	Foundation Slab:M…
Building Storey	BLKA - GROUND F…
Building Storey	BLKA - 1ST FLOO…
Slab	Floor:MHT_FS_Sla…
Building Storey	BLKA - 2ND FLOO…
Building Storey	BLKA - 3RD FLOO…
Building Storey	BLKA - ROOF LEVEL

Figure 5. Search in GUI retaining the tree structure

2.4 Network view

To be able to interact with the network view, this network first has to be generated. This is done with an algorithm created by Hugo Leon-Garza from BT as part of his Ph.D. research. Pseudo code of the algorithm can be found in algorithm 1.

```
ifcFile = ifcopenshell.open(file)

cabletrays = ifcFile.by_type('IfcFlowFitting' or 'IfcFlowSegment`)

foreach tray in cabletrays:
    if tray does not contain keyword "cable":
        cabletrays.remove(tray)

json = new json
foreach floor in floors:
    add new floor to json
    foreach element in  floor:
        if element in cabletrays:
            floor.nodes.add(element)
        floor.edges.add(element.hasPorts)
```

Algorithm 1. Algorithm to generate the network view

So the algorithm looks for elements of type IfcFlowSegment or IfcFlowFitting. This will however also contain elements from the water installation for example. Therefore any element that does not contain the word cable will be filtered out. Leaving only the elements concerning the cable tray system of a building.

From these elements the element ID, position, building storey and label are extracted, giving all the nodes of the network. After which the algorithm also needs to gather the data for the edges of the network. IfcFlowSegment and IfcFlowFitting contain a property HasPorts, containing a set of IfcPort elements. Which represent either a logical or physical connection in the network. The relation between two elements is extracted from the "connected from" and "connected to" properties in the IfcPort.

The result of the algorithm is a JSON containing the floors of the building, with the respective nodes and edges found of the floors. The structure of the JSON is as follows:

```
"Floor name 1":{
   "nodes":[
      {  "id":"1uq87nwqT6IhktBe0A8PD6",
         "label":"Cable Tray with Fittings",
         "x":119315.55287068,
         "y":152695.047543581  },
        ......
        ],
   "edges":[
      {  "from":"1uq87nwqT6IhktBe0A8PD6",
         "to":"1uq87nwqT6IhktBe0A8P5H"  },
        ......
        ]
}
"Floor name 2": {
      .....
```

Currently this specific this JSON has been hardcoded into the application. So the network graph only works for the BIM model which was used throughout the development of the application. It was intended that the application would send the IFC file to a cloud based application, for it to parse the file and return a processed JSON. The cloud based service was however not able to be finished in time by BT for this research. Nevertheless, with the pre-generated JSON file, it was possible to fully implement and test the application.

The JSON file is parsed by the application. Where all the nodes get initialized as circled on the position provided in the file. The edge information is used to draw the connecting lines between the nodes. The full network graph is then moved and resized to fit on the user's screen. Additionally, every node also gets connected to the GUI element of the same id. Allowing for all the interaction between the network graph and GUI.

3 Evaluation

3.1 Network

The application allows the user to check the electric network, by checking the path between two electric components in the BIM model, as illustrated in figure 6. To do this the user selects a start and end point, which are coloured green and red respectively. If path between these nodes is found, the nodes found along the shortest path will be coloured orange. Depicted in the network view, GUI and 3D model. Which allows the user to easily inspect the elements in the path.

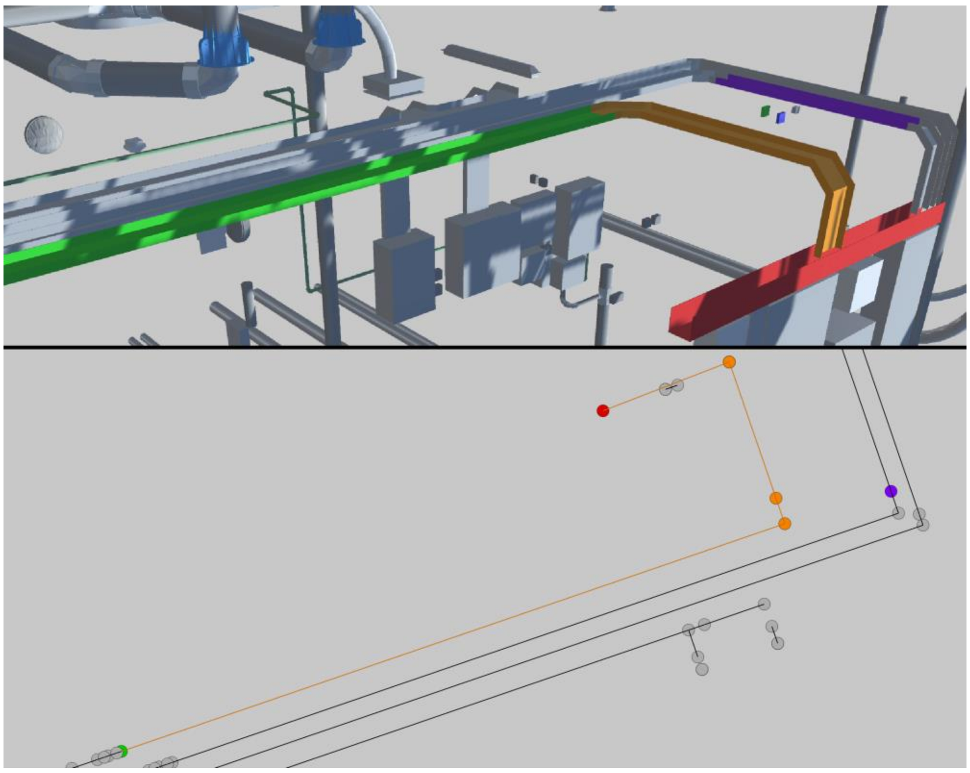


Figure 6. Illustration of the path finding tool to check code compliance
(Top: BIM model, bottom: network graph)

Because of the limited data BT was able to provide for this research. No studies were able to be conducted on the speed of different path finding algorithms. As there was only access to one BIM model, which is relatively small. With the biggest graph only containing 54 nodes, and where most of the graph is disconnected from the rest of the graph

As mentioned before Dijkstra was implemented as the path finding algorithm. By adding a heuristic to the algorithm the path finding algorithm could benefit from an increase in speed.

One thing that should be noted in figure 6, is that some of the spatial properties can lost in the network graph. In the network graph the electric network gets flattened, causing elements positioned directly above each other to be at the same location in the graph. The application does already provide a solution for that, as the BIM elements included in the graph can also be set as start and end points directly from the 3D model.

But as mentioned in section 2.4, the graph is currently generated separately per floor. So it is not possible to calculate the path between two nodes on different floors. To achieve this the algorithm made by Leon-Garza (algorithm 1) could be adapted to generate a 3D graph instead. Which would already be able to be integrated with the application, as the path finding algorithm also works in three dimensions as long as the correct distance are applied.

It does bring the problem of displaying the 3D graph, for this two solutions are proposed. First of all the application can be kept the same as the current state. Where the 3D graph is still generated separately for different floors, but where the nodes will also contain the ‘invisible’ connections to nodes on other floors. Essentially creating a 2.5D graph, with slices of the floors, for the user to interact with.

Alternatively the graph could be displayed fully in 3D. For this the application could allow to filter out all the elements which aren’t connected to the network graph. So that user has the possibility to examine the whole graph in one view. This 3D graph could either be displayed as in a minimalistic manner, with lines and spheres. Or still using the BIM geometry, which might make it the code compliance process much clearer.

However it would also be possible to implement a hybrid of the two proposed solutions. Where the user will both be able to check the graph in a 2D view, as well as easily get an overview in the 3D view.

3.2 Automatic code compliance checking

The original intend of this research was to create an application for automatic code compliance checking of the electric network in a BIM model. Where the application would highlight areas which were suspected to contain some missing information, for example if a cable tray isn’t connected to anything. In the end because of a lack of time it wasn’t possible to achieve this, and the research instead transformed to a more manual application.

An automatic code compliance algorithm could however work by analysing the electrical network, to check if no unconnected cable trays exist in the building. Because it would be unexpected for a cable tray to not be connected with the rest of the network. The algorithm could then flag these cable tray, and make it known to the user that it is suspected that these elements are missing information. After which the user can manually check if this is the case, or determine that the model is considered correct.

While the current application only works based on the IFC data schemas IfcFlowSegment and IfcFlowFitting. An automatic code compliance algorithm could likely benefit from using more of the domain specific data schemas. Where it also has access to IfcDuctSegment for example, eliminating the need to manually filter through the flow segments and fittings to only end up with the cable trays. Furthermore this domain also contains elements like power outlets, cable segments, junction boxes and communication appliances. Vastly broadening the scope of code compliance checking that could be done within the electrical network.

4 Conclusion

A newly developed application for assisting in manual code compliance checking on the electrical network of a BIM model was presented. The application allows the user to examine the standard contents of a BIM model, but also gives a new way to analyse the electric components in a building. Generated graphs of the electrical network and path finding within the network can be used to analyse the model for any missing components.

While the research in this project has been limited to a single building, because of a lack of more data, this application will be especially helpful in larger BIM models. While checking the code compliance of multiple electrically connected buildings for example. In which finding the shortest path between two electrical components might be a lot less trivial.

As the application mostly serves as a proof of concept, there are many areas of future work. First of all the application could benefit immensely from an automatic code compliance algorithm, to identify areas suspected of missing information, which can then be manually further analysed with the tools given in this project.

Furthermore the tools could also be implemented into other similar problem areas, mainly the ones in the domain of MEP, for example checking the code compliance of water and gas pipes in a building.

Acknowledgements

Thank you to my industrial supervisor, Dr Anasol Peña-Rios, for providing assistance in every stage of the project and for her help in shaping the application to what it is now. Thanks also to Hugo Leon-Garza for his work in the generation of the network view, and helping me explain how it works. Lastly thanks to my supervisor Prof Stuart D Walker for his guidance in writing this dissertation.

References

Adobe. (2021, March 23). *Tree view*. Retrieved 8 30, 2021, from Spectrum: https://spectrum.adobe.com/page/tree-view/

Bazjanac, V., & Crawley, D. B. (1997). *The Implementation of Industry Foundation Classes in Simulation Tools for the Building Industry.*

buildingSmart. (2020). *Industry Foundation Classes 4.0.2.1: Introduction*. Retrieved 4 16, 2021, from buildingSMART Internation Standards Server: https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/link/introduction.htm

Candra, A., Budiman, M. A., & Hartanto, K. (2020). Dijkstra's and A-Star in Finding the Shortest Path: a Tutorial. *2020 International Conference on Data Science, Artificial Intelligence, and Business Analytics (DATABIA)*, (S. 28-32).

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2008). *BIM Handbook.* John Wiley & Sons.

IfcOpenShell. (2021). *The open source ifc toolkit and geometry engine*. Retrieved 8 25, 2021, from http:/ifcopenshell.org/

Innovation In Building. (2010). Retrieved 8 25, 2021, from https://www.yearininfrastructure-digital.com/yearininfrastructure/2010/MobilePagedArticle.action?articleId=1456730#articleId1456730

Ismail, A. S., Ali, K. N., & Iahad, N. A. (2017). A Review on BIM-Based Automated Code. *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*, (S. 1-6). doi:10.1109/ICRIIS.2017.8002486

Leon-Garza, H., Peña-Rios, A., & Conway, A. (2020). *Using BIM and Augmented Reality in Field Service Operations.* BT.

Lomio, F., Farinha, R., Laasonen, M., & Huttunen, H. (2018). Classification of Building Information Model(BIM) Structures with Deep Learning. In *2018 7th European Workshop on Visual Information Processing (EUVIP)* (S. 1-6). Tampere, Finland.

Martins, J. P., & Abrantes, V. (2010). Automated code-checking as a driver of BIM adoption. *International Journal for Housing Science and Its Applications, 34*(4), 287-295.

Martins, J. P., & Monteiro, A. (2013). LicA: A BIM based automated code-checking application for water distribution systems. *Automation in Construction, 29*(23), 12-23.

NBS. (2020). *10th Annual Bim Report.* Retrieved August 25, 2021, from https://www.thenbs.com/knowledge/national-bim-report-2020

Rivers, H. (2020, June 23). *Interaction Design for Trees*. Retrieved 8 30, 2021, from Medium: https://medium.com/@hagan.rivers/interaction-design-for-trees-5e915b408ed2