TECHNISCHE UNIVERSITÄT DRESDEN

MASTER THESIS

Automated Simulation of Visitors Streams from IFC Files

Author:
Yunyi FU

Supervisor:
Prof. Dr.-Ing. R. J. SCHERER
Dr.-Ing. P.KATRANUSCHKOV
Dr. Angelika KNEIDL
M.Sc. Fangzheng LIN

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in

Institue for Construction Informatics
Faculty of Civil Engineering

15. May 2018

Abstract

The simulation visitor streams in buildings is based on the information of the relevant visitors in the buildings and the geometric input data. At present, Industry Foundation Classes (IFC) format brings a solution to the difficult manual work. The semantic information like doors, walls, rooms, etc. could directly be assigned to the geometric objects. It would be also possible to execute the simulations automatically, and reflect the simulation result in the building information modeling (BIM) models semantically and visually. STEP physical file (SPF) is settled as the official file format for IFC files. Since the extensible markup language (XML) files are widely used in engineering, BIM generates the ifcXML file according to XML document structure for ease of implementation and interoperability.

In this study, the main direction and task is to extract data in the model files and analyze input file to implement into the simulation software *crowd:it* provided by *accu:rate*. As the input file is written in XML format, the progress will be based on ifcXML files, and build out a filter and converter for extraction.

Contents

List of Figures

List of Tables

List of Abbreviations

ABS	**AB**stract **S**upertype
AEC	**A**rchitecture **E**ngineering **C**onstruction
AEC/FM	**A**rchitecture **E**ngineering **C**onstruction, and **F**acilities **M**anagement
API	**A**pplication **P**rogramming **I**nterface
BIM	**B**uilding **I**nformation **M**odeling
DOM	**D**ocument **O**bject **M**odel
GUI	**G**raphic **U**ser **I**nterface
GUID	**G**lobally **U**nique **ID**entifier
IDM	**I**nformation **D**elivery **M**anual
IFC	**I**ndustry **F**oundation **C**lasses
IFD	**I**ndustry **F**ramework **D**ictionaries
MVD	**M**odel **V**iew **D**efinition
SAX	**S**imple **A**PI for **X**ML
SPF	**S**TEP **P**hysical **F**ile
StAX	**St**reaming **A**PI for **X**ML
STEP	**St**andard for the **E**xchange of **P**roduct model data
SVG	**S**calable **V**ector **G**raphic
URI	**U**niform **R**esource **I**dentifier
UUID	**U**niversally **U**nique **ID**entifier
XML	E**X**tensible **M**arkup **L**anguage

Chapter 1

Introduction

The study of the thesis is based on data transformation among areas of architecture, construction and simulation. With a wide application of BIM, it is common to save data into BIM model files, which makes a great contribution to combine planning, design and construction. Relevant softwares also provide users to create 3D models. Because of standardized by the international organizations, these files all provide us a well performance in communication and exchanging.

1.1 Building Information Modeling

In the 1970s the concept of BIM has been founded [1]. Since "Building Information Modeling" or BIM was first put forward in 1992 [2], it has been employed in the Architecture Engineering Construction (AEC) industry. BIM is an advanced intelligent 3D model-based process and encompasses all aspects of the design, construction, and operation of a building with high efficiency in plans, constructions and even in management which made AEC stood in leadership.

The National Building Information Model Standard Project Committee defines BIM as: "Building Information Modeling is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition." [3]

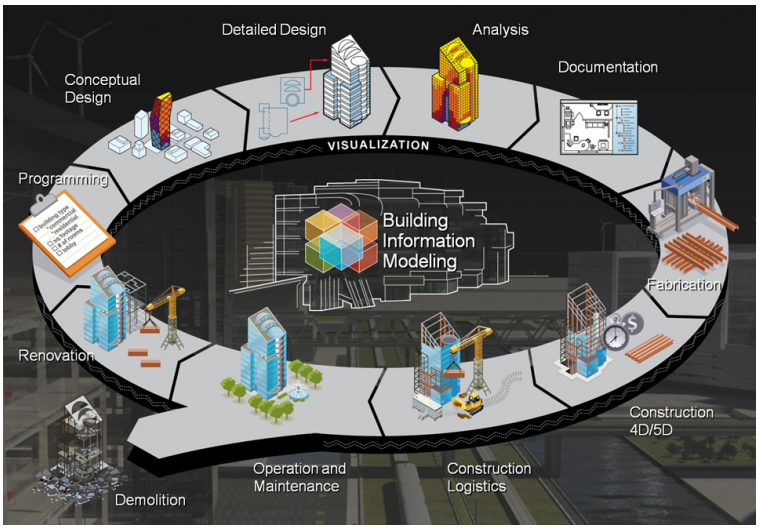

FIGURE 1.1: The process of BIM [4]

As a 3D model-based process in Figure 1.1, it helps engineers cast off relying on two-dimensional technical drawings in traditional building design. Furthermore, BIM is widely used by most of the world's leading architecture, engineering and also constructions firms, and becoming the advanced solution of the AEC projects. Now BIM extends itself beyond 3D, augmenting the three primary spatial dimensions (width, height and depth) with time as the fourth dimension (4D) [5] and further more the fifth (5D) [6]. So that, BIM covers not only just geometry and spatial relationships, but also light analysis, geographic information, and quantities etc.

There are four different types of BIM models: architectural model, mechanical model, electrical model and structural model. As our case is related with building models, it would be more focused on architectural and structural model.

The BIM models are saved in Industry Foundation Classes (IFC) files which was standardized by International Alliance for Interoperability [7]. There are two main file formats for IFC files: SPF and ifcXML files. Between them, the ifcXML would be focused on in this study. However, the BIM information embedded in IFC files as well as in ifcXML is complicated. The problem raised that it would be difficult to extract information from a model.

By research of literatures and using existing software tools, the deep relationships among building elements connection and property reference should be found. However, such information is not directly provided in the file. In this study, we need to fulfill the demand of the simulation from the industry partner, and find the solution for information conversion from IFC files to crowd simulation.

Thus, first step should be precisely find out the location of interested information in the ifcXML. Second step, re-right them in a way that it could be easier understood by the users. And last step, create input files for serving software: *crowd-it*.

Softwares for working with BIM on a building project have been developed by several companies, and could automatically gather all the relevant information into the project. Further architectural tools allows additional information such as time, cost, construction schedule and physical energy etc. Some of widely used BIM applications are *Autodesk Revit*, *Bentley Architecture* and so on. In this study the *Autodesk Revit* is used to create models and files.

Revit is a design and documentation platform that supports the design, drawings, and schedules required for BIM. During the building of models, *Revit* collects information about the building project and coordinates this information across all other representations of the project. It allows users to design a building and structure and its components in 3D, annotate the model with 2D drafting elements, and access building information from the building model's database. *Revit* is also 4D BIM capable with tools to plan and track various stages in the building's lifecycle, from concept to construction and later maintenance or demolition [8].

1.2 Crowd simulation

In the area of building design, crowd is an inconspicuous but a very important factor. The implementation of taking crowd into consideration would be crowd simulation,

which simulates the movement of a large number of people or entities.

Building is a kind of construction serving directly to human. Considering the area of movement is fixed, a large crowd of people not only could lead to a traffic jam but also would influence the practicability of the building. Especially in public areas, when confined spaces are crowded, how people move become critical [9].

The application areas of crowd simulation is extensive, which is also the reason why it is needed in the building design. There are four essential problems that crowd simulation is solving for irrespective of the crowd situation.

- Traffic and mobility of the crowd.
- Safety protection.
- Evacuation.
- Crowd capacity of the construction.

These four items also indicates the purpose of crowd simulation. Not only simulation attempt for an unobstructed crowd area, but also it actually concerns the safety of the people.

Obviously, human behavior is complicated. Unlike programme or robots, it is impossible to predict the movements of people. So that it is complicate to analyze the crowd behavior and very crowded places are characterized by complicated and dynamic systems.

Considering the places with relevantly high crowd flow rate and the places that mostly need to consider evacuation of the crowd. The simulation mainly considers below-listed situations.

- Public transportation and recreation areas.
- Public places with events.
- Other special places.

Firstly, the public transportation and recreation areas. These places usually have a certain regularity of crowd form. For example, there will be usually a large crowd of people during rush hours in the train stations and there will usually be a lot of people in the shopping mall during the weekends. But the movement of the people is hard to predict, especially in shopping areas, people are more likely to move randomly.

Secondly, the public places with events. This mainly refers to the places that hold temporary mega-events, for example, annual celebration, concert, football match and etc. These places only have crowds when events are held, and people usually have certain direction of movements.

And lastly, some other special places. This mostly refers to some tourist attractions or some under construction areas, that they usually have problem of the carrying capacity of pedestrians, or there might be some possibilities for the urgent evacuation. So that crowd simulation become necessary and crucial.

The crowd simulation enables designer to check in advance both the normal process and the evacuation ways in case of an emergency. That means it could help the designers to build not only a comfortable public atmosphere for the customers, but also with more consideration about safety by avoiding stampede and other similar emergency situation to the greatest extent. Also, when facing emergency situation, the dynamics that can occur at events with thousands of people cannot always be grasped just with human intuition, so that a proper evacuation needed to be conducted.

1.2.1 Combination of BIM and crowd simulation

The combination of BIM and crowd simulation is an advanced and efficient solution, and benefits in both sides.

One hand, BIM models could provide precise geometry and also plan graphs of the building, and help process of crowd simulations in a great extend. Taking advantage of the detailed and precise layout of the buildings, a particular estimate of the crowd can be simulated. Furthermore, semantic information is coded into BIM models, which differs from 2D CAD drawings with only geometry information.

On the other hand, crowd simulation also contributes to the integrity of building design especially in area of utilization. This helps to improve model practicability. During the construction or reconstruction of buildings and infrastructure, taking pedestrian into consideration would help the constructors to obtain a safer construction and minimize risk as early as in the planning phase.

Furthermore, with the growth of application in crossing field, BIM would gain a wider range of development. Therefore, it is a quite prospective area for combining BIM and crowd simulation.

1.3 Study plan

In this study, the main task is the implementation of geometry data conversion from BIM models to crowd simulation. So that, the emphases lays on solving the compatibility problem of IFC file especially ifcXML in simulation, and find solution to enables the crowd simulation software to get geometry data from IFC models that would be used in simulation of the pedestrian flow. According to this, the plan of the study is listed as follow.

According to Figure 1.2, there are three main steps in this work. First is the study of ifcXML files, and provide a general view of the ifcXML structure, including the information in ifcXML, how the data is recorded, and how the elements are connected with each other.

Second step is the design of data conversion based on ifcXML, and it includes five parts.

1. Data retrieval from ifcXML including about which elements are identified as relevant to crowd sim, and how they can be extracted.

2. Convertion of 3D to 2D models.

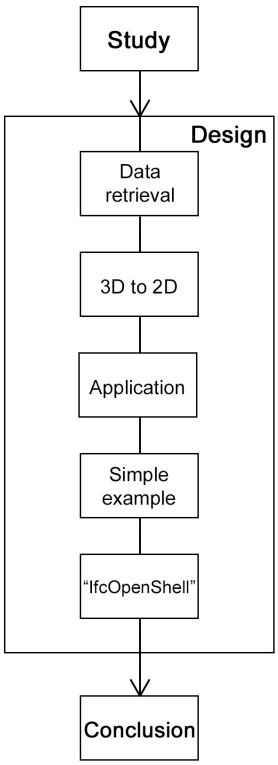

FIGURE 1.2: The flow of study plan.

3. A simple programme that applied for ifcWalls.
4. Application in simulation software (*crowd:it*).
5. A short study of *IfcOpenShell*.

IfcOpenShell is an open programme library that provides an application converting the implicit geometry in IFC files into explicit geometry[10]. It was found during the preparing of theoretical study. It enables a file format conversion of IFC files, which could output 2D floor plans in SVG format. This is a very attractive function for crowd simulation, although SVG may not the desiring file format. So that a short study of this library would be practicable and useful. The study of the *IfcOpenShell* would focus more on the usage, and analyze the output files to find the advantages and disadvantages.

Last step is the conclusion, basically to conclude the above-mentioned work respectively in previous two steps, and analyze the applicatability for the design work in step two.

1.3.1 Study challenges

In this whole study, there would be some limitations and challenges on account of the feasibility and practicability. Only part of the building elements would be analyzed as an example of all the IFC objects. So that, in this process, there are four main issues needed to be solved

- Relatively full-scale of exploring and understanding ifcXML data;

- Java parsing of the ifcXML files;
- Get the routine of the concerning information of the elements;
- Convert the 3D models into 2D models.

Combining to the listed process plan, firstly a comprehensive introduction of IFC schema as well as ifcXML files would be provided. And then the relationship between IFC elements would be found in order to put forward a method to retrieve the data that is needed. Based on the retrieved data, a further conversion of 3D model to 2D model is required to apply in crowd simulation software. Finally a summary or conclusion would be presented for the whole work in this study.

Chapter 2

Study of IFC and ifcXML

This chapter focuses on the main part IFC and ifcXML. As a basic, it is necessary to fully understand the structure and framework of the IFC schema. By using the schema, it would be easier to understand ifcXML file, and also to get the interested information.

The chapter consists of four parts: IFC fundamental study, IFC-SPF file format, ifcXML file format and their comparison. From the comparison, there would be a clear explanation of why the study is focusing on ifcXML file.

2.1 Industry Foundation Classes

In the beginning, it is necessary to have general comprehension of IFC, basic of ifcXML file. There are several non-proprietary building data models currently available, and one of these is the IFC. Initiated from 1994 it is an open standard data model developed by the International Alliance for Interoperability (IAI) [11], which was renamed as buildingSMART in 2006. IFC is a schema developed to define an extensible set of consistent data representations of building information for facilitating interoperability in the architecture, engineering and construction (AEC) industries [12].

International Organization for Standardization (ISO) later covered the IFC standard and applied in AEC/FM. It replied on the ISO-STEP EXPRESS language and definition. Actually IFC was designed as an extensible model framework, which supposed to address all building information, including whole building life-cycle, design, planning and construction.

IFC describes how to represent buildings and civil infrastructure in a digital format. It is actually an object-based file format, and is a commonly used as collaboration format in BIM based projects [13]. Through the digital models IFC describes not only what it looks like, but also how a facility is used and constructed. Based on BIM, IFC is a solution for information sharing. The users and software providers would decide what they want to share with IFC [14], building up a circulating building information environment.

2.1.1 IFC data schema architecture

Four conceptual layers are defined by data schema architecture of IFC, each individual schema is assigned to one of the definite conceptual layers, according to the connection and inheritance of objects. The structure of IFC data schema is easy for maintenance and expansion. At the same time differentiate the each area of AEC/FM [15].

It successfully reduces the difficulties of data exchanging automatically and directly between diverse softwares [16]. The IFC data schema architecture is shown in Figure 2.1.

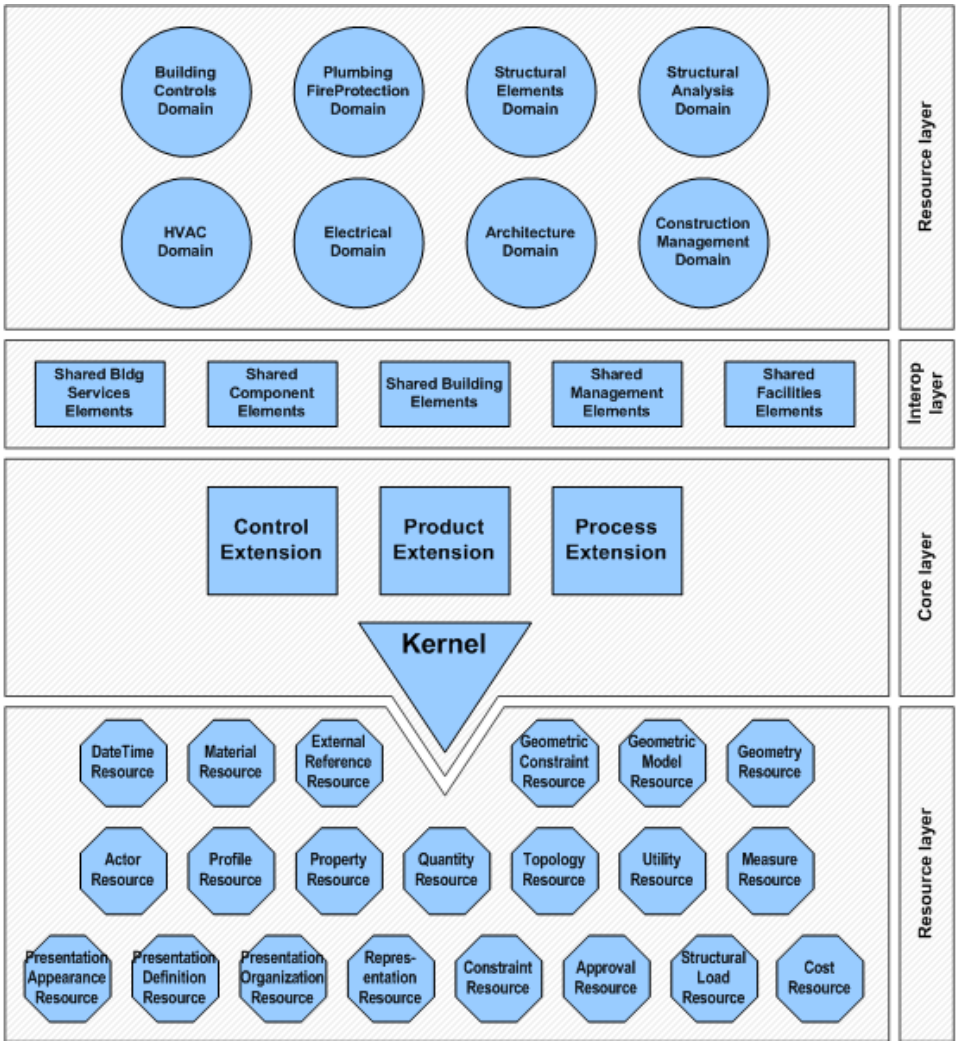

FIGURE 2.1: IFC data schema architecture [17]

The four conceptual layers follow hierarchical relationship. Each layer can only refer the data from itself and the lower layers, but can not refer to the upper layers [18]. This protects the data from disturbing by lower layers.

- Resource layer — the lowest layer includes all individual schemas containing resource definitions, which are lower class of concepts and entities [17]. And these definitions do not include a globally unique identifier and shall not be used independently of a definition declared at a higher layer.
- Core layer — this layer includes the kernel schema and the core extenstion schemas, containing the most general entity definitions. All entities defined at the core layer and above layers carry a globally unique id and optionally owner and history information [17].

This layer provides the basic structure of IFC models, and it can be subdivide into "Kernel" and "Upper Kernel" [17]. "Kernel" contains the most general entity definitions, for example *IfcPropertySet*, *IfcRoot*. "Upper Kernel" including "Control Extension", "Product Extension" and "Process Extension", which contains a relevantly higher entity definitions, such as *IfcBuilding* and *IfcElement* [19].

- Interoperability layer — this layer includes schemas containing entity definitions that are specific to a general product, process or resource specialization used across several disciplines. Those definitions are typicly utilized for inter-domain exchange and sharing of construction information [17]. The entities defined in this layer are common entities fulfilling the information exchange in AEC/FM, such as *IfcBeam* and *IfcWall*.
- Domain layer — the highest layer includes schemas containing entity definitions that are specializations of products, processes or resources specific to a certain discipline [17], those definitions are typically utilized for intra-domain exchange and sharing of information, such as *IfcStructuralAction* and *IfcReinforcingElement*.

EXPRESS is a data modeling language defined in ISO 10303-11 [20]. It is the essence of IFC, which officially used for defining the IFC schema. BuildingSMART has also published an XML schema definition (XSD) file. By using a language binding, XSD is initially the conversion from primary EXPRESS data definition. Since the release of the second edition of IFC2x the language binding is governed by ISO10303-28 [21].

The correlation between the schema definition that controls the models or document files can be seen in the Figure 2.2. Basically, EXPRESS plays the foundation role of STEP physical file and XSD builds up the XML file. Both files would be introduced in the further section in detail.

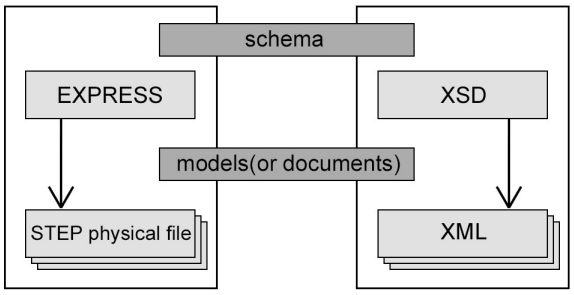

FIGURE 2.2: Schema and model differentiation for EXPRESS and XML [21]

EXPRESS data is defined textually and graphically. Textual form of an EXPRESS model schema is a plain text file [22] and the language elements are formed into a stream of text within an ASCII file, which is important to formal verification and application input.

The graphical representation would be more of perceptual intuition and more suitable for explanation and tutorials. EXPRESS-G is a standard graphical notation for

information models [23]. It is common used to represent data model structure by EXPRESS-G notation for displaying entity and type definitions, relationships and cardinality. The EXPRESS-G relationship graph is helpful to understand the structure in the files of the output in the further sections. With the orientated labels in EXPRESS-G, the target elements and properties can be easily found.

2.1.2 IFC model—Spatial structure and space element

The integrity of an IFC model usually contains some essential parts which are indispensable, such as project container element, base setting, spatial structure, space element, building element, building service element and etc. In this study, the key point is geometry of structural element, so that the emphasis would be laid on spatial structure, space element and building element, and other related elements.

Project container element and base setting

The project container is required in any IFC files named as *IfcProject*. Note that within an IFC file or an IFC database model there is only one instance of *IfcProject*. The *IfcProject* holds the global definitions for the presentation context and the units within the global context, and the information can only be provided once within the IFC file [21].

Before uniquely identify an object and preserve information about its ownership, a root class *IfcRoot* is needed for all subtypes to define the basic properties [24]. *IfcRoot* derives three basic abstract, which are *IfcObject*, *IfcPropertyDefinition* and *IfcRelationship*.

In the IFC schema, which measure types and related units should be used also need a definition and it is provided in the *IfcMeasureResource*. It had been based on the specification of the measure_schema as defined in ISO 10303-41. An IFC project file always has units defined for its shape representations. IFC project file without a unit definitions is not allowed [21].

An IFC model file has to include representation context. This is represented as *IfcRepresentation* in IFC and is established by *IfcProject*. It references a single or multiple instances of *IfcRepresentationContext*.

Spatial structure

First of all, some of the definition and common concepts would be introduced. Spatial structure is, as the name implies, the subsets of the project model according to spatial arrangement, and it is common in most disciplines and design tasks. The entity of spatial structure in IFC is known as *IfcSpatialStructureElement*.

There are four subtypes subsumed under *IfcSpatialStructureElement* entity, which are *IfcSite*, *IfcSpace*, *IfcBuilding* and *IfcBuildingStorey* as shown in Figure 2.3. They are used to represent the levels of the spatial structure.

The spatial structure of a project is based on the fundamental decomposition relationship. Among these four subtypes *IfcBuilding* and *IfcBuildingStorey* are mandatory levels for exchanging the project data, the *IfcSite* and *IfcSpace* are optional levels. As

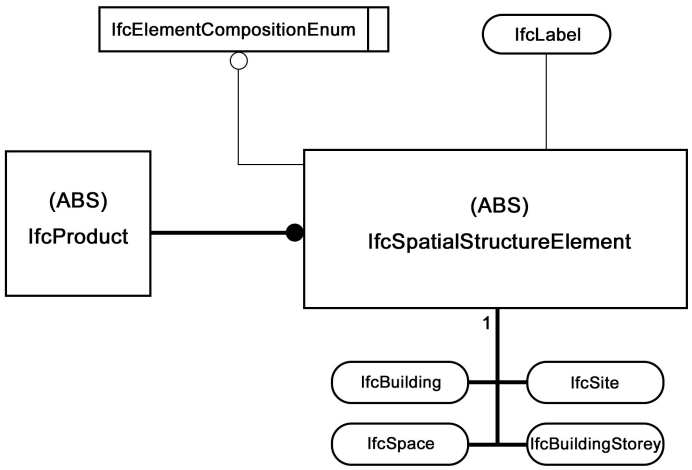

FIGURE 2.3: Definition of spatial structure elements [21]

mentioned before, only one *IfcProject* is allowed, but more than one *IfcBuilding* and *IfcBuildingStorey* in one building project could exist. For the optional levels, the number of the generated *IfcSite* and *IfcSpace* is not strict limited.

The above mentioned four layers could be linked by the *IfcRelAggregates* to establish an hierarchical structure and the instances of *IfcProject*.

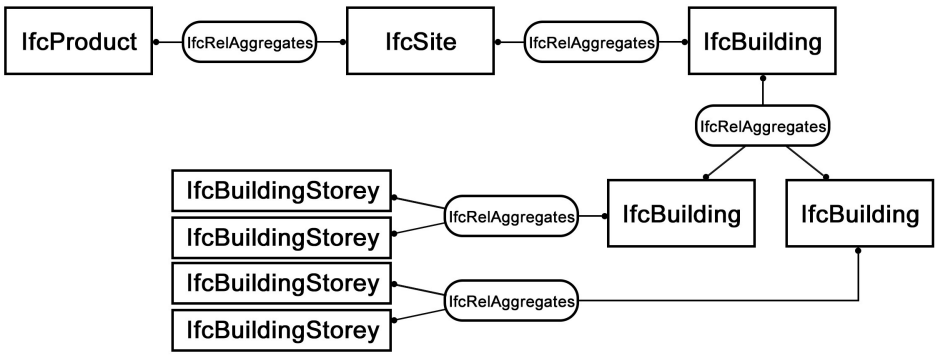

FIGURE 2.4: Decomposition of a spatial structure [21]

Figure 2.4 shows the connection between the instance of *IfcProject*, *IfcSite*, *IfcBuilding*, *IfcBuildingStorey*, *IfcSpace* to other instances of the spatial structure by *IfcRelAggregates*. Therefore, the vertical and horizontal division of the building also can be found.

Connecting elements

As the *IfcRelAggregates* in Figure 2.4, in IFC files there are also essential linking elements that maintain the integrity of the model. Besides the *IfcRelAggregates*, there is also a important element connecting specific building structure elements with *IfcBuildingElement* or *IfcBuildingStorey*, which is named *IfcRelContainedInSpatialStructure*

In our case, there would be some certain building elements that is mostly concerned. For example, *IfcWall* together with *IfcWallStandardCase*, *IfcColumn*, *IfcDoor*, *IfcStair*, and etc. These are actually subtypes defined by *IfcBuildingElement* which would be provided in details in further chapter.

2.2 IFC file formats and SPF file

There are several IFC file formats that is being used in different softwares and tools, supporting various encodings of the same underlying data [25].

- IFC-SPF (STEP physical file) is a text format defined by ISO 10303-21 ("STEP-File") with file extension "*.ifc". In this format, each line typically consists of a single object together with its attributes. It is the most widely used IFC format, with an advantage of supporting visualization for users.
- IFC-ifcXML is an XML format defined by ISO 10303-28 ("STEP-XML") with file extension "*.ifcXML". As XML, this format is suitable for interoperability with XML tools and exchanging partial building models. But the size of the file for building models is quite large, as a typical weakness of this format.
- IFC-ifcZIP is a ZIP compressed format consisting of an embedded IFC-SPF file or IFC-XML file with file extension "*.ifcZIP".

Currently the IFC-SPF is still being the most common method of data exchange between IFC compliant software applications. IFC-SPF is defined in ISO 10303-21:2002 with a text format containing human readable ASCII characters [20] in Table 2.1. The size of a SPF is usually relative large. For example, a simple small house test model could contain thousands lines of data.

```
 1. ISO-10303-21;
 2. HEADER;
 3. FILE_DESCRIPTION(('ViewDefinition [ReferenceView_V1.0]'),'2;1');
 4. FILE_NAME('001-00','2018-02-20T12:10:13',(''),(''),'The EXPRESS Data
    Manager Version 5.02.0100.07 : 28 Aug 2013','20160225_1515(x64) -
    Exporter 17.0.416.0 - Alternate UI 17.12.14.0','');
 5. FILE_SCHEMA(('IFC4'));
 6. ENDSEC;
 7. DATA;
 8. #1= IFCORGANIZATION($,'Autodesk Revit 2017 (ENU)',$,$,$);
 9. #5= IFCAPPLICATION(#1,'2017','Autodesk Revit 2017 (ENU)','Revit');
10. #6= IFCCARTESIANPOINT((0.,0.,0.));
11. #10= IFCCARTESIANPOINT((0.,0.));
12. #12= IFCDIRECTION((1.,0.,0.));
13. #32= IFCAXIS2PLACEMENT3D(#6,$,$);
14. #33= IFCLOCALPLACEMENT(#16541,#32);
15. #36= IFCPERSON($,'Macalister','Samuel',$,$,$,$,$);
16. #38= IFCORGANIZATION($,'Autodesk','',$,$);
......
```

TABLE 2.1: An extract from a STEP physical file [11]

As mentioned before, IFC-SPF files are able to be visualized by several stand-alone softwares. Object model servers and software components are also available to navigate SPF programmatically for use in application development. So that IFC-SPF is more user friendly.

2.3 IfcXML file

Another widely used file format is ifcXML: IFC data is exchanged by using the ifcXML representation (ISO 10303-28:2003) [7], which was made available to take advantage of the XML technology [26]. Most ifcXML data files are ended with extension ".xml" or occasionally "*.ifcxml" or "*.ifx".

```
 1. <?xml version="1.0" encoding="utf-8"?>
 2. <doc:iso_10303_28 xmlns:exp="urn:oid:1.0.10303.28.2.1.1"
    xmlns:doc="urn:oid:1.0.10303.28.2.1.3"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:oid:1.0.10303.28.2.1.1 ex.xsd"
    xmlns:ifc="http://www.iai-tech.org/ifcXML/IFC2x2/FINAL" version="2.0">
 3.   <exp:iso_10303_28_header>
 4.     <exp:name>Autodesk Revit 2017 (ENU)</exp:name>
 5.     <exp:time_stamp>2018-02-03T15:48:33</exp:time_stamp>
 6.     <exp:author></exp:author>
 7.     <exp:organization></exp:organization>
 8.     <exp:authorization></exp:authorization>
 9.     <exp:originating_system>20160225_1515(x64) - Exporter 17.0.416.0 -
        Alternate UI 17.12.14.0</exp:originating_system>
10.     <exp:preprocessor_version>Express Data Manager Version 5.02.0100.07
        Aug 28 2013</exp:preprocessor_version>
11.     <exp:documentation>ViewDefinition [ReferenceView_V1.0]</exp:documentation>
12.   </exp:iso_10303_28_header>
13.   <doc:express id="exp_1" external="" schema_name="IFC4">
14.   <doc:schema_population governing_schema="exp_1"
      determination_method="SECTION_BOUNDARY" governed_sections="uos_1">
15.   <ifc:uos id="uos_1" description="" schema="exp_1"
      configuration="IFCXML_Official" edo=""
      xmlns="http://www.iai-tech.org/ifcXML/IFC2x2/FINAL"
      xsi:schemaLocation="http://www.iai-tech.org/ifcXML/IFC2x2/FINAL
      http://www.iai-tech.org/ifcXML/IFC2x2/FINAL/ifc2x3g_alpha.xsd">
16.     <IfcOrganization id="i1903">
17.       <Name>Autodesk Revit 2017 (CHS)</Name>
18.     </IfcOrganization>
19.     <IfcApplication id="i1907">
20.       <ApplicationDeveloper>
21.         <IfcOrganization xsi:nil="true" ref="i1903"/>
22.       </ApplicationDeveloper>
23.       <Version>2017</Version>
24.       <ApplicationFullName>Autodesk Revit 2017 (CHS)</ApplicationFullName>
25.       <ApplicationIdentifier>Revit</ApplicationIdentifier>
26.     </IfcApplication>
27.     <IfcCartesianPoint id="i1908">
28.       <Coordinates exp:cType="list">
29.         <IfcLengthMeasure exp:pos="0">0.</IfcLengthMeasure>
30.         <IfcLengthMeasure exp:pos="1">0.</IfcLengthMeasure>
31.         <IfcLengthMeasure exp:pos="2">0.</IfcLengthMeasure>
32.       </Coordinates>
33.     </IfcCartesianPoint>
......
```

```
98.   </ifc:uos>
99. </doc:iso_10303_28>
```

TABLE 2.2: An extract from a IFC-XML file.

From Table 2.2, it can be seen that, same labels in EXPRESS language are used in IFC-XML with format of XML, for example *IfcProject*. And also because of the XML the information of each element are extended and more understandable. The names of the items' properties are written clearly below the elements, and the values are also explained.

IfcXML is currently used not as common as other methods. One reason is that the file size is significantly larger than SPF, usually two till even ten times larger [26], which might cause a great effort in reading the file data. However, ifcXML is more convenient for communication, thus it is mainly used to cover partial exchange, data and etc.

Many XML tools still do not support ifcXML, but ifcXML is a special XML file that is based on IFC standard. That means tools need to support IFC standards and need to be able to distinguish the hierarchical structure between elements, which is still not fully developed. Some tools are also less capable when dealing with very large models and files, where performance is critical. IFC models should be accessed from a model server or database [26]. This is also a challenge in this study that small simple examples are chosen to be the testers and would be discussed in details.

2.3.1 Extensible Markup Language

After the brief introduction of ifc file formats, a preliminary understanding of ifcXML form and advantages were given. In this section, it will concentrate on ifcXML to introduce more about ifcXML in details and provide an idea of how to parse the ifcXML. To start with the study, it is useful to first have a basic concept about Extensible Markup Language.

As it has been roughly introduced in the previous chapter, ifcXML is actually a subset of XML. XML was defined by the World Wide Web Consortium (W3C), is a simple and flexible text format for exchange of a wide variety of data [27][28]. The XML is actually a metalanguage, that means, it is a language for describing other languages. It was designed to store and transport data, which can be easily read by human and machine [29].

```
1. <?xml version = "1.0"?>
2. <class>
3.   <student rollno = "393">
4.     <firstname>Denis</firstname>
5.     <lastname>Smith</lastname>
6.     <nickname>Denny</nickname>
7.   </student>
8. </class>
```

TABLE 2.3: An example of an XML file.

Same as normal XML files, which can be seen in Table 2.3, ifcXML file consists of version specification in the first line. Schema specification and elements which can be found in the rest lines.

Every element again consists of tags, attributes and characters. There are two types of elements. One is full form with both start and end tags, while the other is simplified form that has only start tag with a slash in the end that omit the end tag.

It is necessary to specify that attribute has two meanings in this study. In XML files, attribute is the a markup construct consisting of a name–value pair that exists within a start-tag or empty-element tag. But in IFC schema, attribute is the unit of information within an entity, which would be explained in the further section.

Among these elements, some of them are coordinating relation for example "firstname", "lastname" and "nickname". Some of them are affiliation for example "class" and "student". And in affiliations, the elements that have sub-elements are called parents, and those sub-elements are called children.

Characters only appear in full form of the elements and are always embedded between start and end tags, while the start tag will always have the same name as the end tag.

Attribute may appear in start tag as a supplement for extra informations, and it usually stays after the title of the start element. Together with tags and characters these three parts build up an element which contains a complete statement with subject, predicate and object [30]. Attributes can be found both in full and simplified forms.

Every tag has its own markup and content. The content performs as subject and usually URL named resource, which describes what is the purport of the element, while the character as object shows the property value.

2.3.2 Elements relations in ifcXML

It is important to understand the relations between different elements especially in simple structures. The element itself is the primary type of data, but how they are being relating to each other is relying on two ways. One is that they could be related by being nested as sub-elements, which is named here as full sub-element nesting expression. The other way is to connected through references to identifiers [31]. Such as adding the referring id numbers in attributes to the referred element would also has its own unique id number, which establishes the id-ref pairs expression.

Actually, every property and part of the structure or ifc element (e.g. *IfcApplication*, *IfcSIUnit*, *IfcWallStandardCase* etc.) would always has an id number no matter which type of expressions. These two expressions can both represent different relationships between elements. Followings are examples to explain.

Full sub-element nesting expression

The full sub-element nesting expression is basic expression in ifcXML for a complete ifc element. It starts with the name and its id, and nesting its children elements

showing the specific parameters. The children could also be parents for other elements, all the children are nesting together. That means all the information will be shown together in the parent element.

An example of *IfcSIUnit* for this basic expression can be seen in the Table 2.4. Its id number can be found as an attribute in the start tag. It has two children: *UnitType* and *Name*, and both have their data in characters staying between start and end tag. Here the file basically shows that the element *IfcSIUnit* has two properties. So that characters are enough to describe them.

```
1. <IfcSIUnit id="i1955">
2.   <UnitType>timeunit</UnitType>
3.   <Name>second</Name>
4. </IfcSIUnit>
```

TABLE 2.4: An example of *IfcSIUnit*.

The second example in Table 2.5 shows another relative complex case with full sub-elements nesting expression. In this situation there are several children elements. It can be seen that *IfcAxis2Placement3D* has a child in the category of location, named *IfcCartesianPoint*, it also has its own id, and also has its child *Coordinates* inside it. The final data can be found inside *Coordinates* with three children representing coordinates values.

```
 1. <IfcAxis2Placement3D id="i2044">
 2.   <Location>
 3.     <IfcCartesianPoint id="i2042">
 4.       <Coordinates exp:cType="list">
 5.         <IfcLengthMeasure exp:pos="0">-15237.17153</IfcLengthMeasure>
 6.         <IfcLengthMeasure exp:pos="1">7449.831949</IfcLengthMeasure>
 7.         <IfcLengthMeasure exp:pos="2">0.</IfcLengthMeasure>
 8.       </Coordinates>
 9.     </IfcCartesianPoint>
10.   </Location>
11. </IfcAxis2Placement3D>
```

TABLE 2.5: An example of *IfcAxis2Placement3D* in full sub-element nesting expression.

This is a case that there is only one child from the parent, and the child has some grandchildren. There are also a lot of cases where the parent has several children and the children could also have several their own children elements etc.

Obviously, in full sub-element nesting expression all the information are gathered together, so one advantage of this case is that it is clear and easy to find the children and the relationships between these elements. Besides, it would save lines for the XML files, and decrease the element number in the file. Also it is fast to export from the model in the software.

But when there is a large amount of children and grandchildren, it would be a large

number of lines for one element, it could also be complicated for an abstract class while it usually includes a lot of layers of sub-classes.

To be mentioned is that, the parent-children expression is only suitable when the children elements has only one parent. While the children element has more than one parent or they are needed to be declared in several places, the following expression is more often used.

Id-ref pairs expression

Id-ref pairs are actually used for shortening the length of element definition, instead of nesting the whole content of children inside the parent, only the name and the id numbers as a reference would be nested. And the children themselves will become separate elements alongside.

Again taking the item *IfcAxis2Placement3D* as an example, the file content is shown in Table 2.6, but with id-ref pairs. It can be seen that, instead of nesting inside the *Location*, the *IfcCartesianPoint* is written separately outside the parent element, but still have the same relationship as the Table 2.5.

```
 1. <IfcCartesianPoint id="i1908">
 2.   <Coordinates exp:cType="list">
 3.     <IfcLengthMeasure exp:pos="0">0.</IfcLengthMeasure>
 4.     <IfcLengthMeasure exp:pos="1">0.</IfcLengthMeasure>
 5.     <IfcLengthMeasure exp:pos="2">0.</IfcLengthMeasure>
 6.   </Coordinates>
 7. </IfcCartesianPoint>
 8. <IfcAxis2Placement3D id="i1994">
 9.   <Location>
10.     <IfcCartesianPoint xsi:nil="true" ref="i1908"/>
11.   </Location>
12. </IfcAxis2Placement3D>
```

TABLE 2.6: An example of *IfcAxis2Placement3D* in id-ref pairs relationship.

Similar to full sub-element nesting expression, the parent could also have several children and several grandchildren. Each pair of elements with one affiliated by the other can be represented in id-ref pairs expression.

Obviously, in id-ref pairs expression, each element has less lines of descriptions, that makes the element easy to be caught by readers. This is one of the advantages of this expression. Another advantage comes when one ifc object is referred by more than one other objects, which means it is being a children element of more than one parents. In this case, the content of the object does not need to be repeated because instead of nesting to each parent it would be a independent element, and the parents just need to add its id number.

But on the contrary to the full sub-element nesting expression, there would be much more format paratactic elements needed and the length and size of the file would be increased. It would be very hard for readers to get a full structure of the elements, and a huge work would also lies on the searching of the elements.

Composed expression

Actually in the real ifcXML files, these two expression are used together in a composed expression. So that, there are usually both two types of expression in one element. Thus the expression is not too complicated and still systematic.

In order to have better comparison, *IfcAxis2Placement3D* is again taken as an example. From Table 2.7, it can be seen that *IfcAxis2Placement3D* has three children. *IfcCartesianPoint* is shown directly in the element but two *IfcDirection* belonging to *Axis* and *RefDirection* are written only with ref-numbers.

```
 1. <IfcAxis2Placement3D id="i2158">
 2.   <Location>
 3.     <IfcCartesianPoint id="i2156">
 4.       <Coordinates exp:cType="list">
 5.         <IfcLengthMeasure exp:pos="0">11442.22792</IfcLengthMeasure>
 6.         <IfcLengthMeasure exp:pos="1">2527.308404</IfcLengthMeasure>
 7.         <IfcLengthMeasure exp:pos="2">-3000.</IfcLengthMeasure>
 8.       </Coordinates>
 9.     </IfcCartesianPoint>
10.   </Location>
11.   <Axis>
12.     <IfcDirection xsi:nil="true" ref="i1922"/>
13.   </Axis>
14.   <RefDirection>
15.     <IfcDirection xsi:nil="true" ref="i1920"/>
16.   </RefDirection>
17. </IfcAxis2Placement3D>
```

TABLE 2.7: An example of *IfcAxis2Placement3D* in composed relationship.

Actually the *IfcAxis2Placement3D* is a grandchild of the *IfcWallStandardCase*, which is is defined at the supertype *IfcWall* [14], and the relationship between these entities can be found in Figure 2.5. As it can be seen in this figure, *IfcAxis2Placement3D* belongs to *RelativePlacement*, one of the attribute of *IfcLocalPlacement*, and *IfcLocalPlacement* represents *ObjectPlacement* of *IfcWallStandardCase*.

In IFC schema, the attribute is a very important concept. Basically it refers to the properties that attached to the entity. Such as it is shown in in Figure 2.5, there are a list of attributes that are written under the entity *IfcWallStandardCase*. In this study, it is necessary to get basic understanding of those attributes.

Among the attributes in each entities, some of them are in black and others are in gray color. Those are in black are explicit attributes. They can be found directly in the entities in IFC files, and are similar to the term "field" in common programming languages. Explicit attributes may be defined as scalar values or collections including Set (unordered, unique), List (ordered), or Array (ordered, sparse) as defined in ISO 10303-11 [20]. Those gray attributes are inverse attributes and derived attributes. Inverse attributes do not list out directly in the entity in the IFC files, but define queries for obtaining related data and enforcing referential integrity [14]. Derived attributes can be found being computed in some way from other attributes. Similar to inverse attributes they are a kind of redundant data and thus are excluded from data sets delivered file based. The expression for computing the value of a derived attribute is defined in the schema [14].

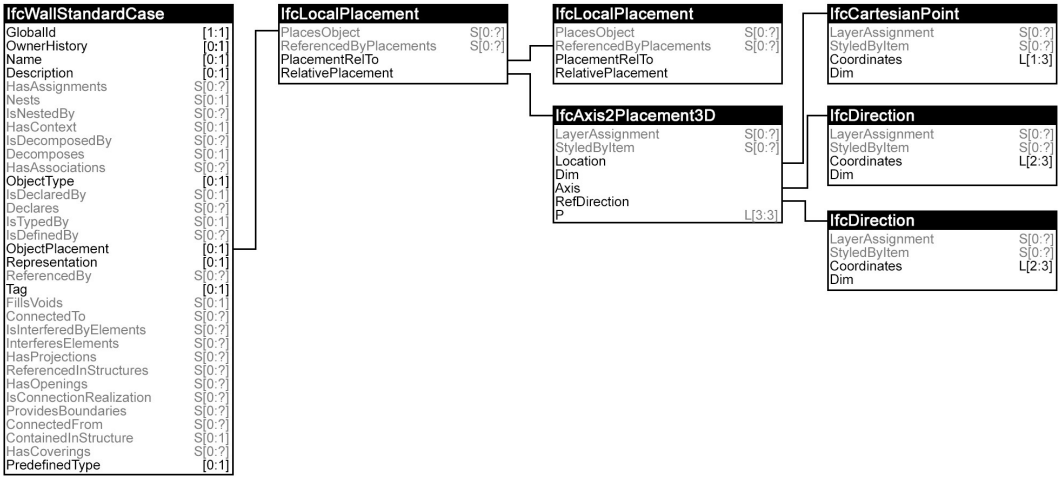

FIGURE 2.5: *IfcWallStandardCase* in graph expression.

For example, the *ObjectPlacement* in Figure 2.5 links to *PlaceObject* which is in gray in *IfcLocalPlacement*, that means the *PlaceObject* can not be found in the entity *IfcLocalPlacement*.

For a better understanding, the example of *IfcLocalPlacement* is shown in Table 2.8. It can be seen that the atrribute *PlaceObject* can not be found in the element *IfcLocalPlacement*. Since *IfcLocalPlacement* is related to *IfcWallStandardCase*, the id number of *IfcLocalPlacement* can be found in element *IfcWallStandardCase* as a referring element in its child *ObjectPlacement*. So that, without including *PlaceObject*, the connection can still be found in *IfcWallStandardCase*.

```
 1. <IfcLocalPlacement id="i2045">
 2.   <PlacementRelTo>
 3.     <IfcLocalPlacement xsi:nil="true" ref="i2032"/>
 4.   </PlacementRelTo>
 5.   <RelativePlacement>
 6.     <IfcAxis2Placement3D xsi:nil="true" ref="i1994"/>
 7.   </RelativePlacement>
 8. <IfcLocalPlacement id="i2045">
 9. <IfcWallStandardCase id="i2084">
10.   <GlobalId>2qDQkwZlH2yuxdvqIaccMR</GlobalId>
...
18.   <ObjectPlacement>
19.     <IfcLocalPlacement xsi:nil="true" ref="i2045"/>
20.   </ObjectPlacement>
...
24. </IfcWallStandardCase>
```

TABLE 2.8: An inverse attributes *IfcLocalPlacement* in ifcXML file.

Same case can also be seen in IFC files. Table 2.9 shows a part of the IFC file, which including *IfcLocalPlacement* and *IfcWallStandardCase*. Obviously, with simplicity and concise expression, the IFC file is much shorter than ifcXML file. But the content in blanket actually includes all the explicit attributes.

In line #182 of *IfcWallStandardCase*, there are nine partitions of descriptions separated by commas. These exactly follows the attributes shows in Figure 2.5, and also in the same order. Same case can be found in *IfcLocalPlacement*. As it can be seen that the inverse attributes still do not appear in the file, but the referring can be found in the *IfcWallStandardCase*. Same location in Figure 2.5, the sixth place provides a referring number which exactly referring to the *IfcLocalPlacement*, implicitly indicates the inverse attribute related data and enforcing referential integrity.

```
#143= IFCLOCALPLACEMENT(#130,#142);
...
#182= IFCWALLSTANDARDCASE('2qDQkwZlH2yuxdvqIaccMR',#42,'\X2\57FA672C5899\X0\:
   \X2\5E3889C4\X0\ - 200mm:391703',$,'\X2\57FA672C5899\X0\:\X2\5E3889C4\X0\ -
   200mm:1044',#143,#176,'391703',.NOTDEFINED.);
```

TABLE 2.9: An inverse attributes *IfcLocalPlacement* in IFC file.

From the *IfcWallStandardCase* shown in Table 2.9, there are three cases of data expression. Besides referring number that has been discussed before, the data would be directly written when possible, but when an explicit attribute is defined as optional in express and an object (an entity instance) does not provide a value for such an attribute, then the attribute will be marked in dollar sign ($).

However in ifcXML files, for the optional explicit attributes, there will not be a symbol to indicate its information absence. The Table 2.10 is an example of full *IfcWallStandardCase* element in ifcXML. As it can be seen, the *IfcWallStandardCase* has eight paratactic children instead of nine. One *Description* is missing, so that the optional explicit attributes who do not provide values would not be appear in the ifcXML files.

```
 1. <IfcWallStandardCase id="i2084">
 2.   <GlobalId>2qDQkwZlH2yuxdvqIaccMR</GlobalId>
 3.   <OwnerHistory>
 4.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
 5.   </OwnerHistory>
 6.   <Name>Basic wall:general - 200mm:392314</Name>
 7.   <ObjectType>Basic wall:general - 200mm:1044</ObjectType>
 8.   <ObjectPlacement>
 9.     <IfcLocalPlacement xsi:nil="true" ref="i2045"/>
10.   </ObjectPlacement>
11.   <Representation>
12.     <IfcProductDefinitionShape xsi:nil="true" ref="i2078"/>
13.   </Representation>
14.   <Tag>391703</Tag>
15.   <PredefinedType>notdefined</PredefinedType>
16. </IfcWallStandardCase>
```

TABLE 2.10: An example of *IfcWallStandardCase* in ifcXML file.

Also it is also necessary to compare about the data record between IFC and ifcXML files. Primarily, the id numbers that used to markup the entities are different. In

IFC files the line numbers are directly used, while a specific number with letter "i" in front of line number is used in ifcXML files. For the attributes, some data of them is exactly the same in both files such as "GlobalId", "Tag" and etc. Apart these, there are also some of them that is quite different from each other, such as "Name" and "ObjectType". It is very easy to understand the name of the wall that is written in ifcXML, but the characters in IFC files rather indicates unreadable. This characteristic of ifcXML also shows its superiority in some extent, which is much more readable and understandable than IFC files, that could be easier for implement in data retrieval.

The discarding of the blank attributes in ifcXML files could help saving lines, but it could also leads to disarray of the entities. This may cause some difficulties in reading and understanding. Further chapters would focus on discussing the data expression of ifcXML files.

2.4 Comparison of the SPF and ifcXML

Both IFC-SPF and IFC-ifcXML are created under IFC schema and delivering the same data information, so that there is no difference with regard of contents. Between them, the XML file would be more user familiar, because XML files has a broader range of supporting utilities and database implementations. Another important area is web services, where XML is the basic for the development, even some web browser is supporting XML.

In general, the exchange of complete building models, particularly within a CAD environment is primarily based on SPF format. The SPF format is still remaining an ASCII format, but it allows for more compact file sizes [26].

Comparing to SPF files, ifcXML files validate against the ifcXML XSD. We could list out the main characteristics of ifcXML.

- IfcXML files are usually much larger than the equivalent "*.ifc" (SPF) files.
- IfcXML can be considerable widely read, transformed and written. There are a great number of tools and toolkits providing the supporting of XML, which are freely available.
- It has been widely supported by many enterprise and desktop system, which can handle XML.

Therefore ifcXML has been added as a valid representation of the IFC specification. And it is quite useful to implement the post-processing of ifcXML.

It could be partitioned that the exchange scenarios sharing large information sets (often including geometric models) are focusing on using the SPF file format. Other scenarios that are sharing partial models, reports, schedules or set of manufacturer information, would not be influence much by the file size. They would benefit much more from the XML format for IFC interoperability [26].

In this study, the processing of IFC is served for simulation of visitor streams, which is based on the information of the relevant visitors in the buildings and geometric input data. Since the final input file of the simulation should be in the format of

XML and the development is based on Java language, XML has large advantages in external supporting.

Chapter 3

Design of data conversion from ifcXML

Since increasing number of researches and companies have made efforts on developing BIM for production and construction usage, BIM and IFC have permeated into more and more industries and studies. The breadth and depth of the IFC models are tangible reflections of the ideas of the designers and clearly indicate the broad, diverse extent of their intended use [32]. The models not only provide 3D geometry for building structure and manage information but also provide data structures, which are supported by many applications and softwares for the exchange of building geometry and all types of AEC project information. A new expansion into crowd simulation is concerned in this study. Thus the geometry of corresponding building structures is necessary to simulate the routine of visitors and staffs in them.

In this chapter, we would concentrate on the data conversion from the existing IFC models in ifcXML files into the geometry files for the crowd simulation. The software *crowd:it*, developed by the company *accu:rate*, is used in this study for crowd simulation. This software simulates the pedestrian walking path basing on the 2D plan graph of buildings. Therefore, a 2D floor plan is the required. So that the main task in this chapter is to develop of a method to retrieve the 2D geometry from the ifcXML file of the corresponding buildings. The implementation design of data conversion is shown in Figure 3.1. It mainly consist of four parts as shown in the middle of the figure, which are ifcXML filter, geometry data retrieval, 3D to 2D conversion and output the files for simulation softwares.

The ifcXML filter is usually required to pick out the necessary information for the 3D to 2D data retrieval. Since ifcXML files are always with large size and the parsing directly with the original ifcXML file may have unexpected data overflow. The theoretical basic of filtering ifcXML file would be the combination of XML structure and IFC schema.

The most important and challenging steps are the geometry data retrieval and 3D to 2D conversion. During the data retrieval, all the necessary data instances should be linked together. Thus an analysis of the connection among data instances in ifcXML file is interesting. Based on the previous study of IFC and ifcXML in Chapter 2, ifcXML files are a special type of XML files that are generated from BIM models. Data instances in the ifcXML file are defined and identified through unique markers. These instances are also able to link each other. To find out how one data instance is connected to another one, the user needs to search and follow the link, which is defined in ifcXML file. This would also be explained in this chapter. After the dimension conversion, the 2D geometry information is generated. In the final step,

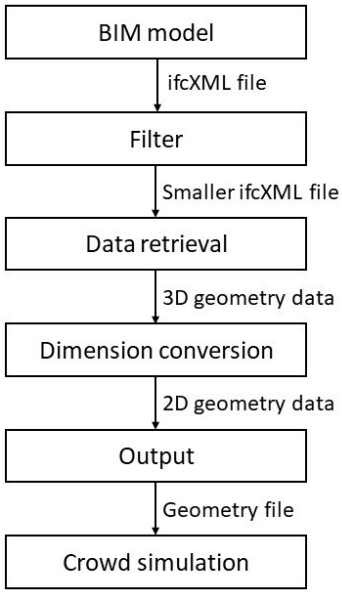

FIGURE 3.1: Design of data conversion from ifcXML to crowd simulation

the new data would be rewritten into readable file format for *crowd:it* to the crowd simulation.

3.1 Information retrieval from ifcXML

From the chapter 2, the meaning of each tags and the relations between elements are introduced. The fundamental concept and schema structure of industry foundation classes are also rough clarified. However information retrieval from ifcXML is more refined and more complex than the rough structure study.

In this section, we would focus on the specific building elements such as *IfcWall*, *IfcStair*, *IfcDoor* and etc.. How the information about those elements are recorded and the link between each other in the ifcXML files will also be talked.

3.1.1 Frame of IFC building element

Building elements defined from ISO 6707-1 are the major functional parts of a house, which comprise all elements that are primary part of the construction of a building, i.e.: the structural and space separating system. Building elements are all physically existent and tangible things [17].

In the IFC schema, the *IfcBuildingElement* is a generalization of all elements that participate in a building system. Typical examples of *IfcBuildingElement* are:

- building elements within a space separation systems
- building elements within an enclosure system (such as a facade)
- building elements within a fenestration system
- building elements within a load bearing system

- building elements within a foundation system

In EXPRESS graph, *IfcBuildingElement* is an abstract class which defines specific building structure elements, including 21 items of building elements, for example walls, curtain walls, doors, columns, piles, and etc. The *IfcBuildingElement* itself is an abstract entity that cannot be instantiated. So that, for other arbitrary building elements, which are not subtypes of *IfcBuildingElement*, can be added to *IfcBuildingElementProxy*.

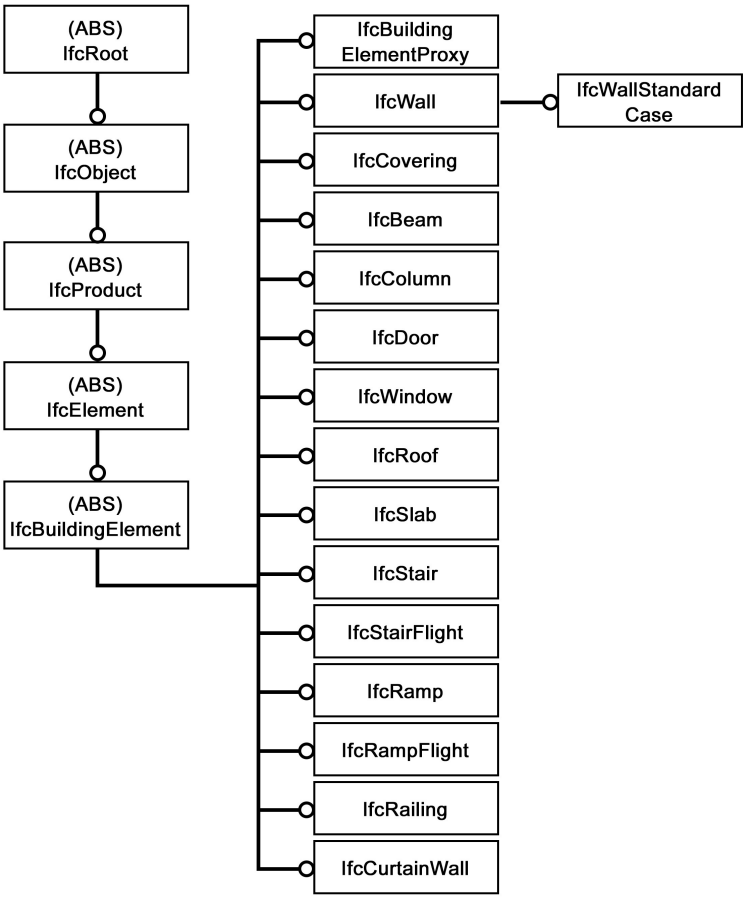

FIGURE 3.2: Entity inheritance chart of building elements.[21]

As can be seen in the Figure 3.2, some of the specific building elements are listed as subclasses from *IfcBuildingElement*. And above *IfcBuildingElement*, it can be seen that the abstracts are all derived from *IfcRoot*.

The various subtypes of the *IfcBuildingElement* would define further attributes. In general, each of the subtypes inherits the following attributes and inverse relationships from *IfcElement*: Elements also have the following additional attributes in Table 3.1.

```
ENTITY IfcElement
  GlobalId        : IfcGloballyUniqueId;
  OwnerHistory    : IfcOwnerHistory;
  Name            : OPTIONAL IfcLabel;
  Description     : OPTIONAL IfcText;
  ObjectType      : OPTIONAL IfcLabel;
```

```
  IsTypedBy       : OPTIONAL IfcRelDefinesByType;
  ObjectPlacement : OPTIONAL IfcObjectPlacement;
  Representation  : OPTIONAL IfcProductRepresentation;
  Tag             : OPTIONAL IfcIdentifier;
INVERSE
  HasAssignments  : SET OF IfcRelAssigns;
  Nests           : SET [0:1] OF IfcRelNests;
  IsNestedBy      : SET OF IfcRelNests;
  HasContext      : SET [0:1] OF IfcRelDeclares;
  IsDecomposedBy  : SET OF IfcRelDecomposes;
  Decomposes      : SET [0:1] OF IfcRelDecomposes;
  HasAssociations : SET OF IfcRelAssociates;
  IsDeclairedBy   : SET [0:1] OF IfcRelDefinesByObject;
  Declares        : SET OF IfcRelDefinesByObject;
  IsDefinedBy     : SET OF IfcRelDefines;
  ReferencedBy    : SET OF IfcRelAssignsToProduct;
  FillsVoids      : SET [0:1] OF IfcRelFillsElement;
  ConnectedTo     : SET OF IfcRelConnectsElements;
  IsInterferedByElements  : SET OF IfcRelInterferesElements;
  InterferesElements      : SET OF IfcRelInterferesElements;
  HasProjections          : SET OF IfcRelProjectsElement;
  ReferencedInStructures  : SET OF IfcRelReferencedInSpatialStructure;
  HasOpenings             : SET OF IfcRelVoidsElement;
  IsConnectionRealization : SET OF IfcRelConnectsWithRealizingElements;
  ProvidesBoundaries      : SET OF IfcRelSpaceBoundary;
  ConnectedFrom           : SET OF IfcRelConnectsElements;
  ContainedInStructure    : SET [0:1] OF IfcRelContainedInSpatialStructure;
  HasCoverings            : SET OF IfcRelCoversBldgElements;
END_ENTITY;
```

TABLE 3.1: Attributes of *IfcElement*

As a subtype of *IfcElement*, *IfcBuildingElement* has the same attribute as *IfcElement*. That means there are no extra attribute added by *IfcBuildingElement*, and all the attributes are inherited from the entities of supertypes.

In the Figure 3.3, the instance diagram of attribute inheritance is displayed. In inheritance extend, the attributes can be divided into *IfcRoot*, *IfcObjectDefinition*, *IfcObject*, *IfcProduct* and *IfcElement*. It can be seen, the order of the attributes are also listed as same as the inherited order, which is shown in Figure 3.2.

Also in the figure it includes explicit and inverse attributes, where explicit attributes are in black and others are in gray. From chapter 2 we can know that only explicit attributes would be seen in the file content. That means there would be eight attributes for *IfcBuildingElement* in IFC and also IFCXML files.

3.1.2 Wall elements

First of all, in IFC schema, two concepts need to be specified according to the wall elements, which are *IfcWall* and *IfcWallStandardCase*. Generally speaking, *IfcWallStandardCase* is a subset of *IfcWall*. *IfcWallStandardCase* refers to the walls that have a constant thickness along all the wall path. And *IfcWall* handles the rest of the cases

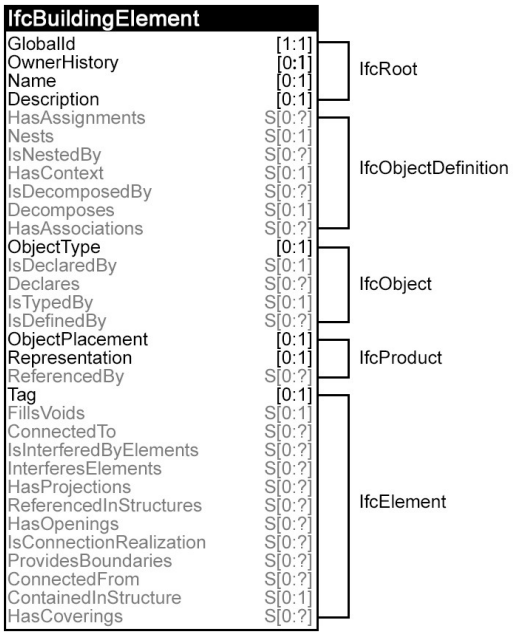

FIGURE 3.3: Attribute inheritance chart of building elements

of the walls.

From the explanation above, the *IfcWallStandardCase* should satisfy the following conditions[21].

- having one (material) thickness along the wall path;
- having a wall path being either a straight line or a circular arc;
- may have various offsets from the wall path;
- may have a single or multiple material layers;
- may have a constant height or a varying height along the path.

Meanwhile, for the other case in *IfcWall*, it would including the following cases, which are not covered by the standard wall.

- Do not have a single (material) thickness along the wall path;
  either have a foot print with non-parallel sides, i.e. a varying material thickness along the wall path (such as a cone as foot print),
  or have a cross section, not being rectangular, like a shear wall having an L-shape cross section,
  or have openings such as windows and doors.
- Do not have a wall path being either a straight line or a circular arc;
  either have an elliptic arc,
  or a spline curve,
  or any other irregular path geometry.

However, the attribute inheritance of both entities are exactly the same, and the attributes can be seen in Figure 3.6. Also it can be seen that, there is a new attribute,

which is *PredefinedType*. This is a new attribute provided in IFC4. This refers to a predefined generic type for a wall that is specified in an enumeration. Note that the *PredefinedType* shall only be used if no *IfcWallType* is assigned, providing its own *IfcWallType*.

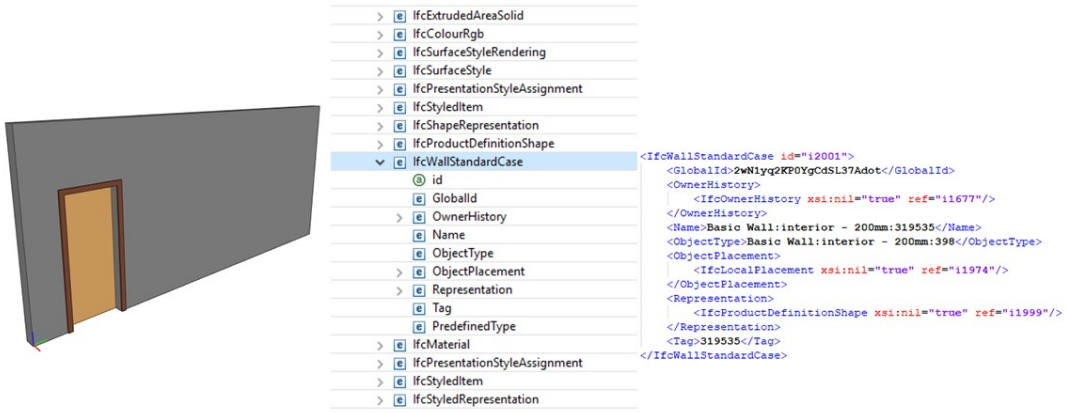

FIGURE 3.4: IfcXML representation of the 3D wall

The Figure 3.4 shows the representation in ifcXML files of the 3D wall model. It can be seen that all the empty attributes are concealed and not presented in the ifcXML file.

For the information retrieval, we would go a deeper study with standard walls. For each standard wall, the *IfcWallStandardCase* element provided in Figure 3.6 includes all the geometry information. The detailed introduction would be presented in three part:

1. Location of the wall;
2. Axis of the wall;
3. Body shape of the wall;

The location of the wall can be found in *ObjectPlacement*, where *IfcLocalPlacement* is nested. Inside *IfcLocalPlacement*, *IfcAxis2Placement3D* provides further coordinates of the starting point of the wall, and then an axis would be provided to show the path or the center line of the wall from top view. Both axis and body shape of the wall can be found in attribute of *representation*, where *IfcProductRepresentationShape* provides both axis and body *IfcShapeRepresentation*.

The establish of these geometry information of a wall is not simply through coordinates or line equations. Firstly it starts with a local axis on an endpoint of the wall, and this endpoint would be the original point of the axis. Then there would be a path line along the floor. The thickness as well as the depth of the wall would be provided separately. The two attributes *ObjectPlacement* and *representation* would be introduced afterwards.

The *ObjectPlacement* includes *IfcLocalPlacement* which belongs to resource definition data schemas, and it defines the relative placement of an object referenced to another

object or the absolute placement of an object within the geometric representation context of the project. It has two attributes which are *PlacementRelTo* and *RelativePlacement*, and they are corresponding to *IfcObjectPlacement* and *IfcAxis2Placement2D*(or *IfcAxis2Placement3D* in three-dimensional axis case).

The *PlacementRelTo* actually signs the coordinate system of the object. When the *IfcLocalPlacement* is inserted with a *PlacementRelTo* attribute value, it defines the relative placement of an object in relation to another object and finally through the intermediate referenced placements within the geometric representation context of the project. But when the *IfcLocalPlacement* is not set with any value, it defines the absolute placement of a product in relation to the global coordinate system as established by the assigned geometric representation context of the project.

So that from attribute *PlacementRelTo* whether the coordinate system is global coordinate should be confirmed. Note that when the walls are required to be contained within a spatial structure element for complete building model exchange, then the *PlacementRelTo* shall point to that spatial structure element.

The *RelativePlacement* refers to *IfcAxis2Placement2D* or *IfcAxis2Placement3D*. It directly provides the coordinate of the object. Each local placement is given by an coordinate, which can be either in a 2D or a 3D coordinate system. Either *IfcAxis2Placement2D* or *IfcAxis2Placement3D* has three attributes which are *Location*, *Direction* and *RefDirection*, which are given by[21]:

- *Location*: the location of start point of the wall;
- *Direction*: the direction of the local Z-axis (in case of 3D) – if omitted always [0, 0, 1];
- *RefDirection*: the direction within the positive XZ plane (in case of 3D) or the direction of the X-axis (in case of 2D) – if omitted always [1, 0, 0] – or [1, 0] in 2D.

According to the rules showing above, the *Location* would never be omitted where a coordinate is given to locate the original point of the axis. The *Direction* and *RefDirection* together provide the direction to establish the axis.

```
 1. <IfcCartesianPoint id="i2042">
 2.   <Coordinates exp:cType="list">
 3.     <IfcLengthMeasure exp:pos="0">-8938.26803</IfcLengthMeasure>
 4.     <IfcLengthMeasure exp:pos="1">10130.1963</IfcLengthMeasure>
 5.     <IfcLengthMeasure exp:pos="2">0.</IfcLengthMeasure>
 6.   </Coordinates>
 7. </IfcCartesianPoint>

 8. <IfcAxis2Placement3D id="i2044">
 9.   <Location>
10.     <IfcCartesianPoint xsi:nil="true" ref="i2042"/>
11.   </Location>
12. </IfcAxis2Placement3D>

13. <IfcLocalPlacement id="i2045">
14.   <PlacementRelTo>
15.     <IfcLocalPlacement xsi:nil="true" ref="i2032"/>
```

```
16.   </PlacementRelTo>
17.   <RelativePlacement>
18.     <IfcAxis2Placement3D xsi:nil="true" ref="i2044"/>
19.   </RelativePlacement>
20. </IfcLocalPlacement>

21. <IfcWallStandardCase id="i2084">
22.   <GlobalId>2qDQkwZlH2yuxdvqIaccMR</GlobalId>
23.   <OwnerHistory>
24.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
25.   </OwnerHistory>
26.   <Name>Basic wall:interior - 200mm:392314</Name>
27.   <ObjectType>Basic wall:interior - 200mm:1044</ObjectType>
28.   <ObjectPlacement>
29.     <IfcLocalPlacement xsi:nil="true" ref="i2045"/>
30.   </ObjectPlacement>
31.   <Representation>
32.     <IfcProductDefinitionShape xsi:nil="true" ref="i2078"/>
33.   </Representation>
34.   <Tag>391703</Tag>
35.   <PredefinedType>notdefined</PredefinedType>
36. </IfcWallStandardCase>
```

TABLE 3.2: Example of attribute *ObjectPlacement* in IfcXML

From the example in Table 3.2, *IfcLocalPlacement* is an example drew from *IfcWallStandardCase*, and the *IfcAxis2Placement3D* can be found in its attributes. In this case only *Location* is presented. It means the axis is in default direction which is same as global axis. From the *PlacementRelTo*, another *IfcLocalPlacement* can be found. In this case the *IfcLocalPlacement* with i2032 is one spatial structure element where the wall is contained. The coordinate is global.

The *IfcProductDefinitionShape* is included in *representation* and refers to at least two *IfcShapeRepresentation* according to the geometry information, where the axis and body shape of the wall can be found separately. Different *IfcShapeRepresentation* can represent axis, surface, body and etc. Usually for standard walls, there will be two *IfcShapeRepresentation* and one represents the wall axis, the other represents the wall body.

For the wall axis, the *IfcShapeRepresentation* is given with the following conventions [21]:

- *RepresentationIdentifier* would be given as "Axis";
- *RepresentationType* would be given as "Curve2D";
- *Items*: The *IfcBoundedCurve* should be used. For multi-segmented path, the *IfcCompositeCurve* shall be used. For a single line segment, it can be represented using an *IfcTrimmedCurve* with *BasisCurve* being an *IfcLine*, or (recommended) an *IfcPolyline* with exactly two *IfcCartesianPoint*.

Wall axis directly shows the wall path starting from the original point in the above-mentioned coordinate system. It presents a line from top view. Besides that, the wall

axis also provides the reference line to relate offset between the material layer set and the wall. The implicit offset line of the material layer set is the right edge of the first material layer. Some examples of straight standard walls can be seen in Figure 3.5.

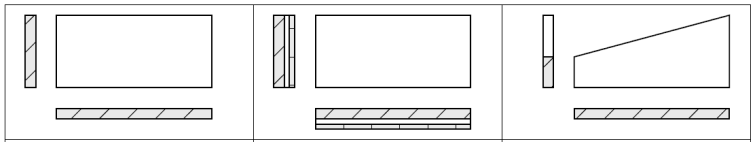

FIGURE 3.5: Examples for standard walls (ground view, cross section and elevation) [21]

Usually in wall structures, there is only one tag segment in attribute *Items*. For straight walls, either the *IfcTrimmedCurve* or *IfcPolyline* can be found in this attribute. For curved walls, the *Items* would only be an *IfcTrimmedCurve* with *BasisCurve* being an *IfcLine*.

In the recommended *IfcPolyline*, the line would always along the XY direction. Two *IfcCartesianPoint* would be provided in the *Point*, which is the only attribute of the *IfcPolyline*. And finally the path of the wall can be get from lining up the two points.

For the wall body, the *IfcShapeRepresentation* would follow the listed conventions [21]:

- *RepresentationIdentifier* would be given as "Body";
- *RepresentationType* would be given as "SweptSolid" (for walls with equal height) or "Clipping" (for walls with varying height);
- *Items*: The *IfcSweptAreaSolid* should be used with two different types of sweeping operations.

For the *IfcSweptAreaSolid*, the sweeping operations could be linear extrusion or revolution referring to *IfcExtrudedAreaSolid* and *IfcRevolvedAreaSolid* separately. In standard walls, usually the *IfcExtrudedAreaSolid* will be used. In case of clipped standard wall bodies, the use of *IfcBooleanClippingResult* is required.

The wall body is the part that really represents the 3D shape of the wall. Not only the depth and thickness of the wall, but also a 2D arbitrary closed curve of the foot print of the wall is given. The depth is given directly from the attribute *Depth* in *IfcExtrudedAreaSolid*, while the thickness and the length can be found in attribute *SweptArea*. As for the foot print of the wall, it can be found in the attribute *Position* with also datatype *IfcAxis2Placement2D*.

Because the extrusion is in the vertical direction, it would provide a same body geometry for the straight walls and the curved walls in terms of the sweep operation, which are represented by *IfcExtrudedAreaSolid* with *SweptArea* of datatype *IfcArbitraryClosedProfileDef* or *IfcRectangleProfileDef* (for walls with regular rectangle shapes). That means the attribute *Depth* should be equal to the wall height that could be found in the data of the *SweptArea*.

Therefore, the extrusion coordinate system which is in the *Position* of *IfcSweptAreaSolid* could be aligned with the object coordinate system in *RelativePlacement* of

IfcLocalPlacement. And a same value of the length of the wall could be found in these two *IfcShapeRepresentation*.

```
 1. <IfcPolyline id="i2049">
 2.   <Points exp:cType="list">
 3.     <IfcCartesianPoint exp:pos="0" xsi:nil="true" ref="i1912"/>
 4.     <IfcCartesianPoint exp:pos="1" xsi:nil="true" ref="i2047"/>
 5.   </Points>
 6. </IfcPolyline>
 7. <IfcShapeRepresentation id="i2051">
 8.   <ContextOfItems>
 9.     <IfcGeometricRepresentationSubContext xsi:nil="true" ref="i2001"/>
10.   </ContextOfItems>
11.   <RepresentationIdentifier>Axis</RepresentationIdentifier>
12.   <RepresentationType>Curve2D</RepresentationType>
13.   <Items exp:cType="set">
14.     <IfcPolyline xsi:nil="true" ref="i2049"/>
15.   </Items>
16. </IfcShapeRepresentation>
17. <IfcExtrudedAreaSolid id="i2065">
18.   <SweptArea>
19.     <IfcRectangleProfileDef xsi:nil="true" ref="i2061"/>
20.   </SweptArea>
21.   <Position>
22.     <IfcAxis2Placement3D xsi:nil="true" ref="i2064"/>
23.   </Position>
24.   <ExtrudedDirection>
25.     <IfcDirection xsi:nil="true" ref="i1922"/>
26.   </ExtrudedDirection>
27.   <Depth>10000.</Depth>
28. </IfcExtrudedAreaSolid>
29. <IfcShapeRepresentation id="i2075">
30.   <ContextOfItems>
31.     <IfcGeometricRepresentationSubContext xsi:nil="true" ref="i2003"/>
32.   </ContextOfItems>
33.   <RepresentationIdentifier>Body</RepresentationIdentifier>
34.   <RepresentationType>SweptSolid</RepresentationType>>
35.   <Items exp:cType="set">
36.     <IfcExtrudedAreaSolid xsi:nil="true" ref="i2065"/>
37.   </Items>
38. </IfcShapeRepresentation>
39. <IfcProductDefinitionShape id="i2078">
40.   <Representations exp:cType="list">
41.     <IfcShapeRepresentation exp:pos="0" xsi:nil="true" ref="i2051"/>
42.     <IfcShapeRepresentation exp:pos="1" xsi:nil="true" ref="i2075"/>
43.   </Representations>
44. </IfcProductDefinitionShape>
45. <IfcWallStandardCase id="i2084">
46.   <GlobalId>2qDQkwZlH2yuxdvqIaccMR</GlobalId>
47.   <OwnerHistory>
48.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
49.   </OwnerHistory>
50.   <Name>Basic wall:interior - 200mm:392314</Name>
51.   <ObjectType>Basic wall:interior - 200mm:1044</ObjectType>
52.   <ObjectPlacement>
53.     <IfcLocalPlacement xsi:nil="true" ref="i2045"/>
54.   </ObjectPlacement>
```

```
55.   <Representation>
56.     <IfcProductDefinitionShape xsi:nil="true" ref="i2078"/>
57.   </Representation>
58.   <Tag>391703</Tag>
59.   <PredefinedType>notdefined</PredefinedType>
60. </IfcWallStandardCase>
```

TABLE 3.3: Example of attribute *Representation* in ifcXML

The Table 3.3 shows an example of ifcXML data file regarding to the attribute *Representation*. Also in the same standard wall case, there are two *IfcShapeRepresentation* representing axis and body shape of the wall. According to the above-mentioned introduction of the conventions, we would focus on the attribute *Item* in *IfcShapeRepresentation*. The *IfcPolyline* and *IfcExtrudedAreaSolid* can be found in this case. Thus the geometry information would be easily obtained.

Based on all of the information introduced above, a short summary of data retrieval would be achieved. Figure 3.6 is an example chart of geometry retrieval. The whole process is divided into two main parts starting from *ObjectPlacement* and *Representation*. The Cartesian point can be found in the attribute *ObjectPlacement*. The path and the body of the wall would be found in the attribute *Representation*. The figure shows a simple 3D standard wall case and indicates the relationship of these data types. It also provides a certain path to locate the geometry information in the ifcXML file.

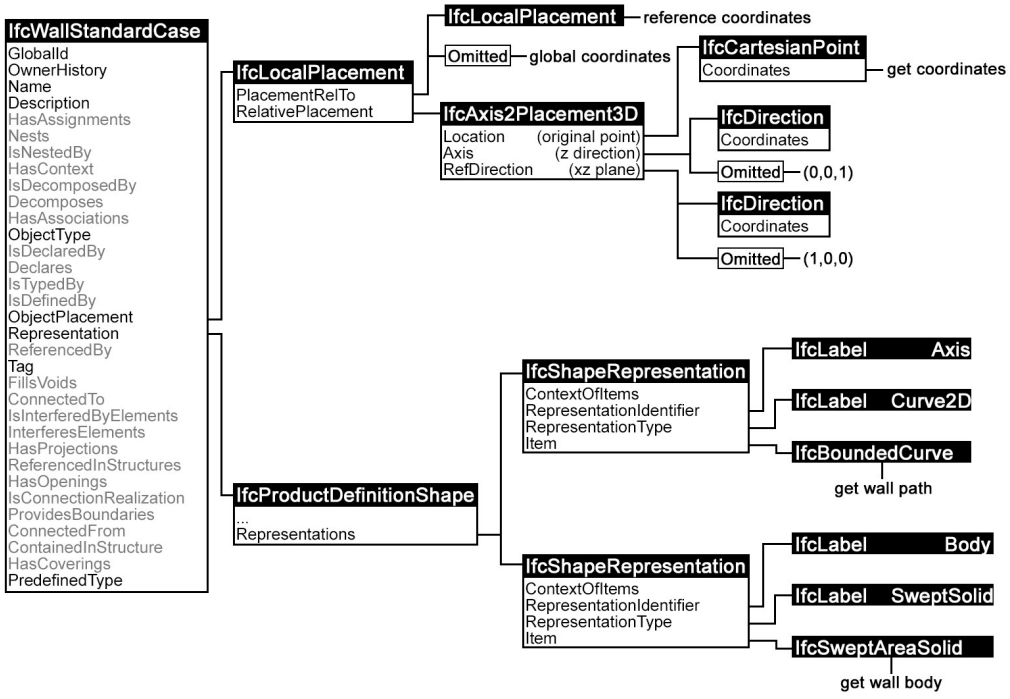

FIGURE 3.6: Geometry retrieval chart of *IfcWallStandardCase*

3.1.3 Stair elements

Stairs are passageways that are used for walking through floors, which have different elevations. As it has been shown in chapter 2, *IfcStair* is defined to represent this building element in IFC schema.

IfcStair has not only the same entity inheritance relationship as other building elements like *IfcWall*, but also the same attributes as *IfcWall* and *IfcWallStandardCase*. So basically *IfcStair* should have a similar way of geometry representation as walls.

```
 1. <IfcRelAggregates id="i7080">
 2.   <GlobalId>1SsQGlds52BAv6sNzbFrzP</GlobalId>
 3.   <OwnerHistory>
 4.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
 5.   </OwnerHistory>
 6.   <RelatingObject>
 7.     <IfcStair xsi:nil="true" ref="i2197"/>
 8.   </RelatingObject>
 9.   <RelatedObjects exp:cType="set">
10.     <IfcStairFlight xsi:nil="true" ref="i3132"/>
11.     <IfcMember xsi:nil="true" ref="i3161"/>
12.     <IfcMember xsi:nil="true" ref="i3189"/>
13.     <IfcRailing xsi:nil="true" ref="i5122"/>
14.     <IfcRailing xsi:nil="true" ref="i7066"/>
15.   </RelatedObjects>
16. </IfcRelAggregates>
```

TABLE 3.4: Example of *IfcRelAggregates* connecting *IfcStair* and *IfcStairFlight*

Since stairs are composed of stair flights. Stair flight can be separated into risers and treads. Every rise and tread may have different shapes. The geometry information of stair is written in *IfcStairFlight*, which can be found being connected with *IfcStair* in *IfcRelAggregates*. The attaching side edge beams is written in *IfcMember* and handrails are defined in *IfcRailing*. Table 3.4 shows an example of the connection. Based on this, the information of stairs, which is necessary for data retrieval is listed as following.

- The location of the stair;
- The number of total risers and treads;
- The length and width of each riser and tread;
- If needed, the geometry of the platform and handrail etc.

IfcStairFlight is also a building element, but with a little difference from *IfcStair*. It has several its own attributes, which can be seen in figure 3.7. The number of risers and treads in one stair flight can be found in the attribute *NumberOfrisers* and *NumberOfTreads*.

The geometry information therefore is quite similar to *IfcWall* that can be found in the attribute *representation*. Also it contains *IfcProductDefinitionShape*, which includes the *IfcShapeRepresentation* representing the shape information. In the case of stairs,

IfcShapeRepresentation follows the conventions, which is similar to wall body, and the important data is listed in *IfcExtrudedAreaSolid* of the *Items*. User would find the number of *IfcExtrudedAreaSolid*, which is the same as the sum of risers and treads. So that there are two kinds of body representation for risers and treads. By comparing the value of attribute *depth*, which represents the waist thickness of tread or the riser height, they can be distinguished.

The attribute *RiserHeight* and *TreadLength* of *IfcStairFlight* have been deprecated, so that the values are no longer reliable. These two geometry values can be found in *Pset_StairFlightCommon* instead. *Pset_StairFlightCommon* is an *IfcPropertySet* that is connected to *IfcStairFlight* by *IfcRelDefinesByProperties*. Besides the two properties mentioned above, it contains all other geometry information that is needed for the model.

Both the *IfcExtrudedAreaSolid* of risers and treads provide coordinates. The *IfcExtrudedAreaSolid* of risers provide coordinates of middle point both on bottom and top edges of each riser, while *IfcExtrudedAreaSolid* of treads provide the endpoints' coordinates. According to the author, the coordinates provided by risers would be easier to locate.

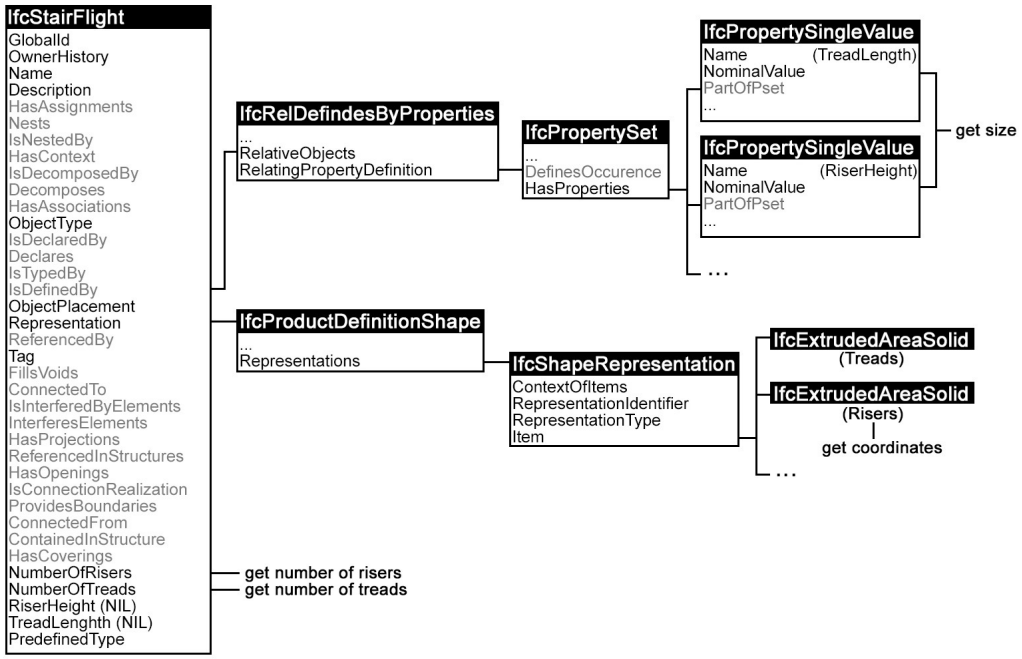

FIGURE 3.7: Geometry retrieval chart of *IfcStairFlight*

Figure 3.7 shows the flow chart of the data retrieval of *IfcStairFlight*. There are all together three parts, namely: the location of the stair, the number of risers and treads and the size of each riser and tread. Before this, it also needs to confirm the corresponding between stairs and stair flights.

Stairs usually contains several attachments such as side edge beams, handrail, platform and etc. As introduced, the beams and handrail can be found in *IfcMember* and *IfcRailing*. When a stair has a platform that makes a turning between two stair flight,

then the user might find a lot more *IfcMember* representing the side edge beams, and *IfcSlab* may also be found in the *RelatedObjects* that represents the platform. So that the geometry data of these elements can be found by tracing these tags.

3.1.4 Door elements

Doors are entrance/exit and access between two spaces. In the crowd simulation, door is essential for providing the path into a certain floor of the building. The gate of a building is also the entrance and exit of the pedestrians. In IFC schema, it is defined as *IfcDoor*, which is also a building element.

Similar to *IfcStairFlight*, *IfcDoor* also have several exclusive attributes which can be seen in Figure 3.8. Since the doors are attached to the walls between spaces, so that the connection between these two building elements are also important.

In IFC schema, another element *IfcOpeningElement* is used in this case. Basically, *IfcOpeningElement* refers to the opening on the wall for doors or windows, and *IfcDoor* refers to the door itself. So that only *IfcOpeningElement* can be found connecting with certain walls by *IfcRelVoidsElements*. The Table 3.5 shows an example of these connection in ifcXML file.

```
 1. <IfcRelVoidsElement id="i2299">
 2.   <GlobalId>3z8vncq$z8g8wy05_Meex8</GlobalId>
 3.   <OwnerHistory>
 4.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
 5.   </OwnerHistory>
 6.   <RelatingBuildingElement>
 7.     <IfcWall xsi:nil="true" ref="i1825"/>
 8.   </RelatingBuildingElement>
 9.   <RelatedOpeningElement>
10.     <IfcOpeningElement xsi:nil="true" ref="i2294"/>
11.   </RelatedOpeningElement>
12. </IfcRelVoidsElement>

13. <IfcRelFillsElement id="i2302">
14.   <GlobalId>1un7pZVQH6hg1SonpwFnlx</GlobalId>
15.   <OwnerHistory>
16.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
17.   </OwnerHistory>
18.   <RelatingOpeningElemen>
19.     <IfcOpeningElement xsi:nil="true" ref="i2294"/>
20.   </RelatingOpeningElemen>
21.   <RelatedBuildingElement>
22.     <IfcDoor xsi:nil="true" ref="i2062"/>
23.   </RelatedBuildingElement>
24. </IfcRelFillsElement>
```

TABLE 3.5: Example of connection between *IfcDoor*, *IfcOpeningElement* and *IfcWall*

By comparing the *IfcDoor* and *IfcOpeningElement* tags from the file, both of them would provide sizes and coordinates. The coordinates from *IfcOpeningElement* is the same as the one from *IfcDoor*, and both are relevant to *IfcWall*. Here the size consists

of the height , width and thickness of the door. So that in our case, either through *IfcDoor* or through *IfcOpeningElement* would be enough to retrieval all the geometry information. Since only *IfcOpeningElement* is connecting to walls, so that it would be practical to use *IfcOpeningElement* to locate the doors.

For the doors or opening elements, it is embedded into the wall, so that the thickness is no loner important. And all the other aspects of information retrieval are listed bellow.

- The location of the door or opening;
- The height and width of the door or opening;

Because the coordinates are relevant to walls, so that finding the location of door or opening has two steps, firstly is to find the attached wall, and then is to locate the door or opening on this wall. As it has been mentioned before, finding the attached wall would rely on the *IfcOpeningElement*. Similar to any other building elements, the relevant coordinates can be found in the attribute *ObjectPlacement* where the *IfcLocalPlacement* is provided, and the origin point of the coordinates is same as the origin point that is used in wall axis. The coordinates they provided is the middle point of the door or opening.

As it has been mentioned, the height and width can both be found in *IfcDoor* and *IfcOpeningElement*. In *IfcDoor*, it has two direct attributes which are *OverallHeight* and *Overallwidth*. And in *IfcOpeningElement*, they can be found in attribute *Representation* similar to *IfcWall*.

```
 1. <IfcDoor id="i3035">
 2.   <GlobalId>1sx2gZimz46g6iHbaMd6bg</GlobalId>
 3.   <OwnerHistory>
 4.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
 5.   </OwnerHistory>
 6.   <Name>M_single:0915 x 2134mm:322650</Name>
 7.   <ObjectType>0915 x 2134mm</ObjectType>
 8.   <ObjectPlacement>
 9.     <IfcLocalPlacement xsi:nil="true" ref="i9188"/>
10.   </ObjectPlacement>
11.   <Representation>
12.     <IfcProductDefinitionShape xsi:nil="true" ref="i3028"/>
13.   </Representation>
14.   <Tag>322650</Tag>
15.   <OverallHeight>2134.</OverallHeight>
16.   <OverallWidth>915.</OverallWidth>
17.   <PredefinedType>door</PredefinedType>
18.   <OperationType>single_swing_right</OperationType>
19. </IfcDoor>
```

TABLE 3.6: Example of *IfcDoor* representing room in ifcXML file

Table 3.6 provides an example in ifcXML file. As it can be seen that the height and width are provided, and the coordinates can be found in attribute *ObjectPlacement* through a routine similar to *IfcWall*.

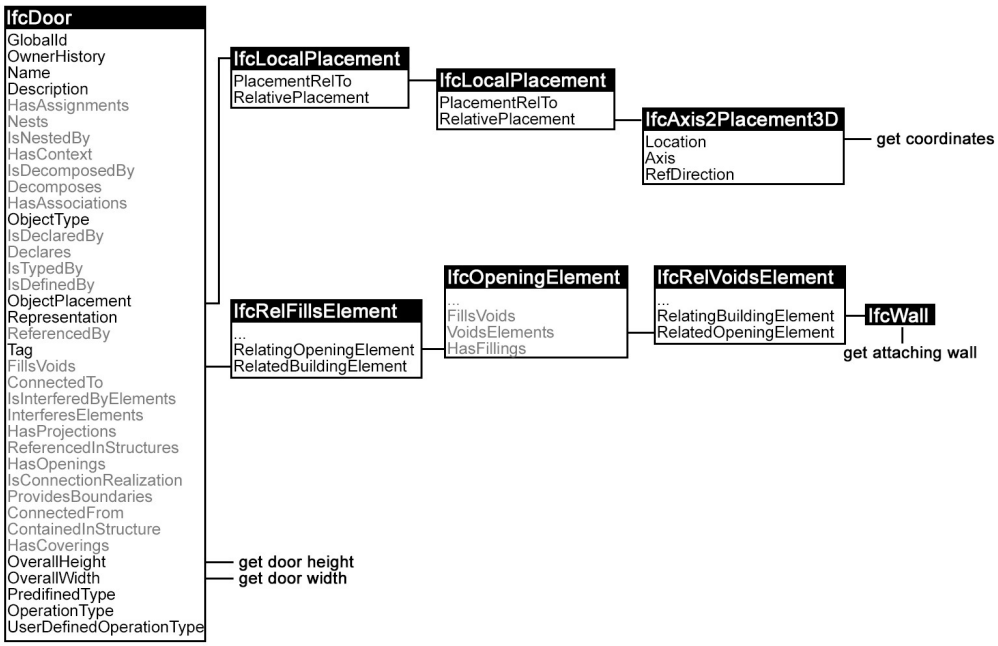

FIGURE 3.8: Geometry retrieval chart of *IfcDoor*

Figure 3.8 shows the flow chart of the data retrieval based on *IfcDoor*. As it can be seen, besides the getting of height and width, there are still two steps. One is to get the coordinates and the other is to find the attached wall.

3.1.5 Space elements for rooms

Rooms are closed areas for human activities, where people spend most of their time in this building. So that they are usually destinations or starting points of the pedestrians in the crowd simulation.

Actually room is an abstract concept for a certain area, there isn't any building element that could match room, or to say, room is not a building element. But in IFC schema, there is a way to describe rooms in the building model by using *IfcSpace*. When the rooms are added in the application when creating building model, in the outputted IFC files, specific *IfcSpace* would be written with its *LongName* called "room".

```
 1. <IfcSpace id="i1780">
 2.   <GlobalId>3eVqApzBbF4uxz$rr62H5j</GlobalId>
 3.   <OwnerHistory>
 4.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
 5.   </OwnerHistory>
 6.   <Name>1</Name>
 7.   <ObjectPlacement>
 8.     <IfcLocalPlacement xsi:nil="true" ref="i1763"/>
 9.   </ObjectPlacement>
10.   <Representation>
11.     <IfcProductDefinitionShape xsi:nil="true" ref="i1776"/>
12.   </Representation>
13.   <LongName>room</LongName>
14.   <CompositionType>element</CompositionType>
```

```
15.   <InteriorOrExteriorSpace>internal</InteriorOrExteriorSpace>
16. </IfcSpace>
```

TABLE 3.7: Example of *IfcSpace* representing room in ifcXML file

Table 3.7 shows an example of the room representation in ifcXML file. Different from building elements such as *IfcWall* or *IfcDoor*, *IfcSpace* is not inherited from *IfcBuildingElement* but they all inherited from *IfcProduct* which means they all have attributes *ObjectPlacement* and *Representation* for geometry information. Following data are needed to analyze rooms.

- The location of the space;
- The height, length and width of the space;

In the *IfcSpace* which is used to represent a room, different as normal space, the location coordinates are not provided in attributes *ObjectPlacement* but in *Representation* together with sizes. It locates the central point of the space.

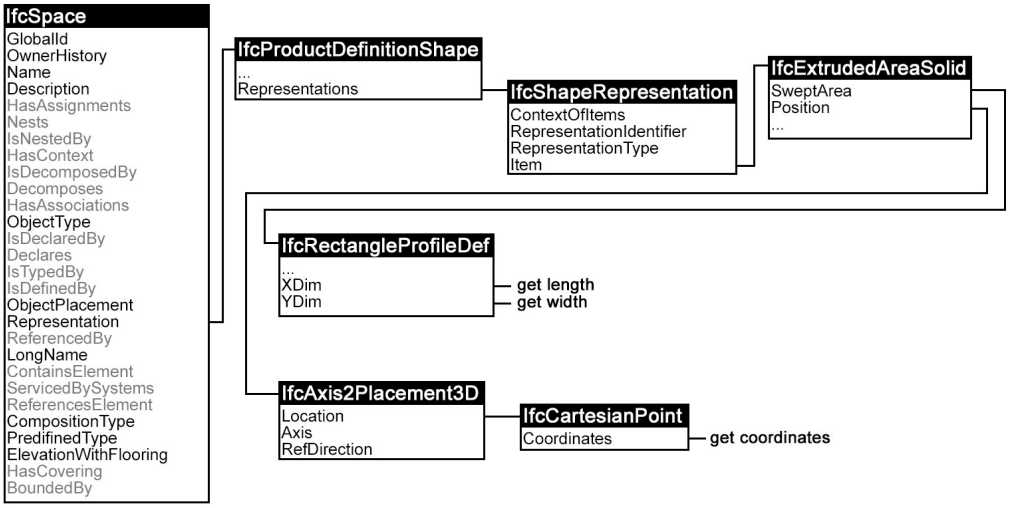

FIGURE 3.9: Geometry retrieval chart of *IfcSpace* for rooms

The Figure 3.9 shows a flow chart of the geometry retrieval from *IfcSpace*. And the two parts of data retrieval are both done in attribute *Representation*, which makes the procedure a lot easier.

3.2 Solutions for 3D to 2D conversion

Original BIM models are built in three-dimensional, so all the geometry information would also be in 3D. But in our case for the crowd simulation, only 2D floor plans of building are interested. In order to build the 2D floor plane of each floor, the conversion of data from 3D to 2D is needed.

Usually there are two approaches of the dimension conversion.

- Based on original file;
- Based on output file.

In the first way, mainly the original file need to be modified in order to get a 2D building model. So that it is necessary to fully understand the structure of the file and the relationship between different data. For that a programme for the conversion is necessary. Obviously it would be a rather huge work to build such a programme. But once it is done, it could keep the integrity of structure and data in a large extend. And it would be easier to build the further procedure. Also it would decrease the amount of work in upgrading or modifying this project.

The other way, which is based on the output file would hold more pertinence to the implementation. The idea is to focus on the certain elements in the original file, and to pick out the relative data. By dispose of the data of z axis, only two-dimension data would be collected in the output file. Comparing to the first way, the design and time costs are significantly lower. The output files would only contain the concerning elements and information. But therefore it would have limitations that might can only be applied in some cases, and further upgrade and modification would be very hard. The above-mentioned two approaches is presented in Figure 3.10. In

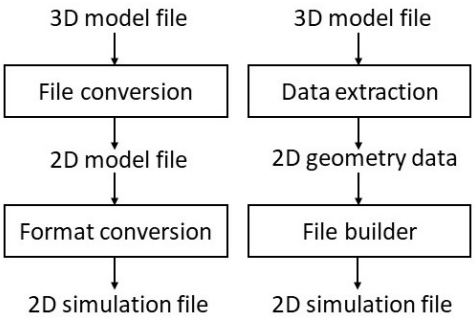

FIGURE 3.10: Dimention conversion based on original file(left) and output file(right)

this study, because of time limitation, the second method is chosen for a attempting study. The process according to this approach is to locate the data positions in the input file firstly, and then write out all the geometry data. In the retrieved data, the z direction data would be disposed, and the rest two-dimensional data would be used to build the input file for crowd simulation.

From the previous section, the data retrieval of standard walls are introduced. As it has been clarified that, the ifcXML file would first provide an endpoint with its coordinate of the wall as the original point and build a new coordinate system. Then the path and body of the wall would be provided separately. This is actually reduce the difficulty in disposing z direction.

Based on standard wall case, the modification would only needed for dispose the coordinate in z direction. Because the wall path itself would definitely a two-dimension line, by adding the thickness of the wall, all the data for plane graph is gathered. An extra work in this process would be confirming the storey of each wall. In the case of stairs, doors and spaces, they do not include path axis, so that it is more directly to

extract the 2D information from the element attribute by disposing z direction. But process of confirming the storey of each element is still needed.

3.2.1 Building storey information

The building storey represents a horizontal aggregation of spaces and elements that are vertically bounded [17]. *IfcBuildingStorey* is used to mark the levels of the building and contains corresponding building elements in each level. It is always associated to one *IfcBuilding*, either *IfcBuilding* represents a building or a section in that building.

All spaces and elements that are in the same building storey would be referenced to the that storey. Thus the storey information could be represented. A building storey with several partial storeys can also be decomposed into parts.

IfcBuildingStorey is defined as a subclass of *IfcSpatialStructureElement*, like *IfcSite*, *IfcBuilding* and *IfcSpace*. By inheriting and receiving from its own class, *IfcBuildingStorey* has ten explicit attributes. Among them, the information that might needed in this study are *Name* and *Elevation*. From the *Name* we could easily mark the connecting elements. All the connecting elements should have the same elevation and stored in *Elevation*.

The assigning of building elements to the *IfcBuildingStorey* is handled by the *IfcRelConnects*. In our case, the relating elements and storey can be found in specified one *IfcRelContainsInSpatialStructure*.

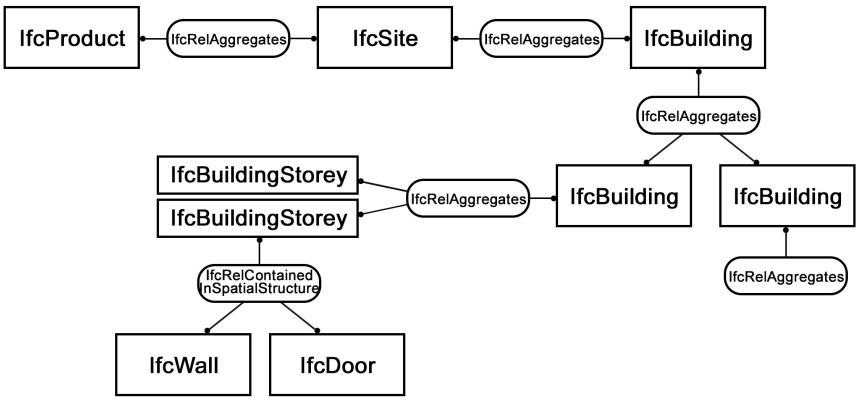

FIGURE 3.11: Relation between spatial structure and building elements[17].

As it can be seen in Figure 3.11, the building elements are connected to the certain building storeys where they belong. The *IfcSpace* however does not inherit from *IfcBuildingElement*, but directly from *IfcSpatialStructureElement*. There is also a relation class provided between *IfcSpace* and *IfcBuildingStorey*, marked as *IfcRelAggregates*.

The *IfcRelContainsInSpatialStructure* have six attributes, the first four are inherited from *IfcRoot*, including id, name and basic information. The other two are from *IfcRelContainsInSpatialStructure* itself, which are *RelatedElements* and *RelatingStructure*. *RelatedElements* has several assigned referencing tags marking the building elements that belong to the same building storey. The *RelatingStructure* provides the certain building storey, only one storey item could be assigned in this attribute.

Similar to the *IfcRelContainsInSpatialStructure*, *IfcRelAggregates* also have six attributes, but the relating objects are represented by *RelatedObject* and *RelatingObject*. *RelatedObject* marks the spaces which are rooms in this case, and *RelatingObject* will provide the certain building storey.

```
1. <IfcBuildingStorey id="i2034">
2.   <GlobalId>3AkbiG8v18Uhj0zSSdrr1d</GlobalId>
3.   <OwnerHistory>
4.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
5.   </OwnerHistory>
6.   <Name>Level 1</Name>
7.   <ObjectPlacement>
8.     <IfcLocalPlacement xsi:nil="true" ref="i2032"/>
9.   </ObjectPlacement>
10.   <LongName>Level 1</LongName>
11.   <CompositionType>element</CompositionType>
12.   <Elevation>0.</Elevation>
13. </IfcBuildingStorey>

14. <IfcRelContainedInSpatialStructure id="i2486">
15.   <GlobalId>3Zu5Bv0LOHrPC10066FoQQ</GlobalId>
16.   <OwnerHistory>
17.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
18.   </OwnerHistory>
19.   <RelatedElements exp:cType="set">
20.      <IfcWallStandardCase xsi:nil="true" ref="i2084"/>
21.      <IfcWallStandardCase xsi:nil="true" ref="i2181"/>
22.      <IfcWallStandardCase xsi:nil="true" ref="i2216"/>
23.      <IfcWallStandardCase xsi:nil="true" ref="i2251"/>
24.   </RelatedElements>
25.   <RelatingStructure>
26.     <IfcBuildingStorey xsi:nil="true" ref="i2034"/>
27.   </RelatingStructure>
28. </IfcRelContainedInSpatialStructure>

29. <IfcRelAggregates id="i2225">
30.   <GlobalId>2JF4e6axWHqu3u0C9FZlmi</GlobalId>
31.   <OwnerHistory>
32.     <IfcOwnerHistory xsi:nil="true" ref="i1677"/>
33.   </OwnerHistory>
34.   <RelatingObject>
35.     <IfcBuildingStorey xsi:nil="true" ref="i1754"/>
36.   </RelatingObject>
37.   <RelatedObjects exp:cType="set">
38.     <IfcSpace xsi:nil="true" ref="i1780"/>
39.   </RelatedObjects>
40. </IfcRelAggregates>
```

TABLE 3.8: Example of *IfcBuildingStorey*, *IfcRelAggregates* and *IfcRelContainsInSpatialStructure* in ifcXML

In Table 3.8, the attributes and assigned elements in *IfcRelContainsInSpatialStructure* are presented. It can be observed that, there are four standard walls lying in level one, and they are all identified by using id numbers. The referenced *IfcBuildingStorey*

is also presented, including the name and elevation that are needed for the crowd simulation.

From the above-mentioned study, the information of building storeys and connections to other building elements are based on *IfcRelAggregates* and *IfcRelContainsInSpatialStructure*. Some important features of these two items would be listed.

- In *IfcBuildingStorey* the content of *Name* is same as that of *LongName*;
- *IfcBuildingStorey* does not include any other structure elements in its tag;
- *IfcRelContainsInSpatialStructure* would provide information of not only building storey but also other spatial buildings and structures.
- Some certain *IfcRelAggregates* would provide information of the building storey which spaces belong to.

So that in order to find out the belonging level of each building elements, the searching path should directly point to the corresponding *IfcRelContainsInSpatialStructure* and *IfcRelAggregates*. And the corresponding items can be filtered out by matching its referencing id numbers with *IfcBuildingStorey* id. The referencing element ids in *RelatedStructure* and the *RelatingObject* of the matched items will be saved. Through that, all the building elements that have been collected before can be categorized.

3.2.2 Summary

In the above sections, several building elements and building storey are introduced. how they could be converted to the data which is necessary for the crowd simulation are analyzed. A whole procedure of geometry data retrieval and dimension conversion of each would be presented.

According to the necessity of crowd simulation, the most important IFC building elements are *IfcWall*, *IfcStair*, *IfcDoor* and *IfcSpace*. Among them, *IfcWall* and *IfcStair* are most important and complicated. For then several different geometry cases need to be considered. *IfcDoor* is always related with *IfcOpening* associated with *IfcWall*. The rooms are represented by *IfcSpace*.

As it has been introduced before, the building storey is connected to building elements through *IfcRelContainsInSpatialStructure*. So that we would build a new path for storey information retrieval, and finally integrate together with geometry information. The first step is to locate the *IfcRelContainsInSpatialStructure*. Secondly to check the *RelatingStructure* whether it contains *IfcBuildingStorey*. If yes, then write down the related elements and their id numbers. Together with the geometry retrieval, the procedure flow can be found in the following figure.

In the Figure 3.12, it can be seen that in the example of standard walls, the total data retrieval can be provided into two main parts for geometry and storey information. In the retrieval of geometry information there would be three steps, which are coordinates of the origin point, wall path and wall body shape. The information of the storey which the wall belongs to can be directly gathered from the class *IfcRelContainsInSpatialStructure* through the matching of the wall id numbers. Finally, we could list out all the important data for each wall and extract into the output file.

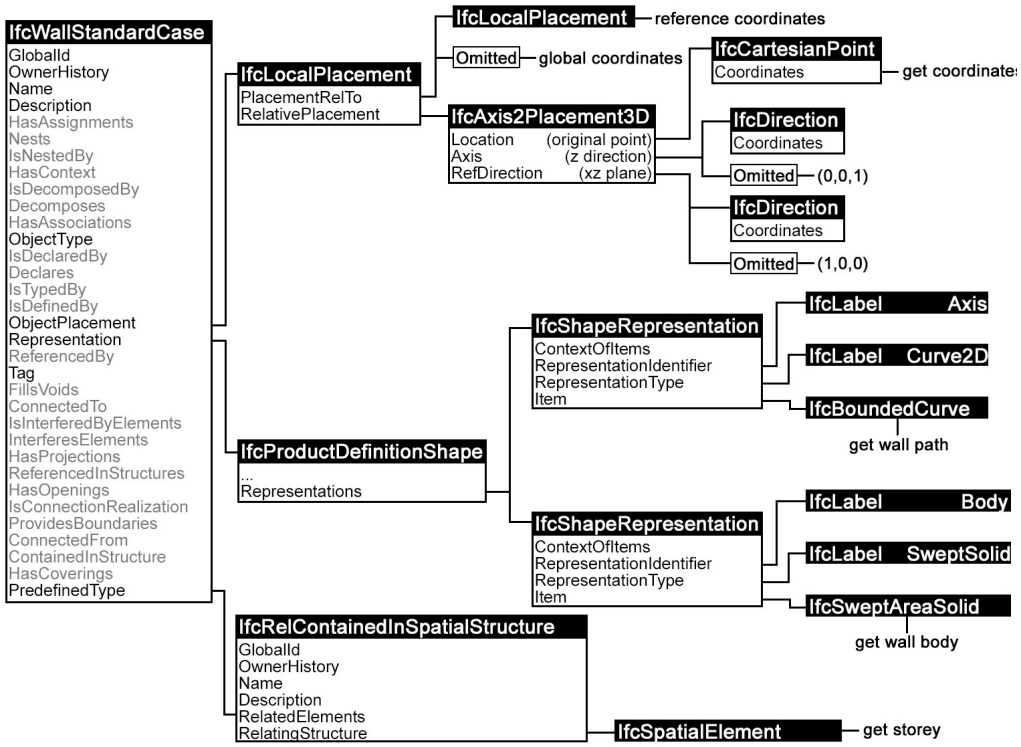

FIGURE 3.12: Whole geometry retrieval chart of *IfcStandardWall*

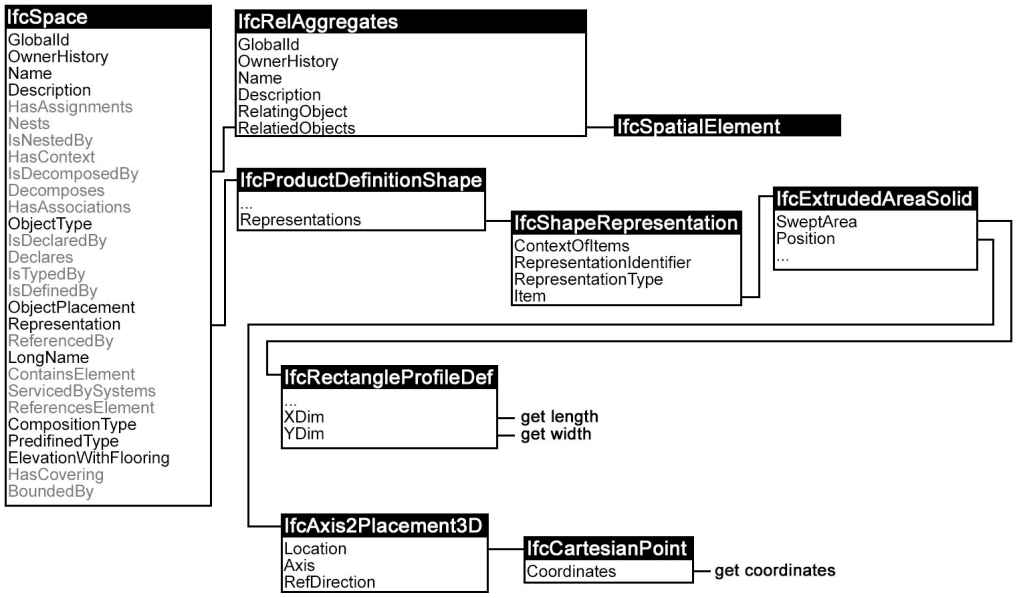

FIGURE 3.13: Whole geometry retrieval chart of *IfcSpace*

As for the *IfcSpace*, it doesn't inherit from *IfcBuildingElement*, so the storey information would receive from *IfcRelAggregates*. The procedure for getting access to the relating storey is similar to the procedure of *IfcStandardWall*. The final total geometry retrieval chart for *IfcSpace* could be found in Figure3.13.

3.3 Application in simulation softwares

The final step of this design is the application in simulation softwares: *crowd:it*. It is a software package for microscopic, agent-based crowd simulation. It has been tested in various projects and at the mean time passed all of the tests for simulators of RiMEA. Not only the application range of *crowd:it* is quite wide and large, but also the operation is very simple and quick. It provides realistic movement patterns and precise visualizations. Crowd:it is a very versatile software for crowd simulation.

The user interface can be seen in Figure 3.14. As the figure shows, the 2D geometry plan graph is used in the middle area for displaying pedestrian movement. On the right side the parameters are captured and calculated. The simulation proceed together with all the building floors at the same time. So that the precise 2D floor plan is essential.

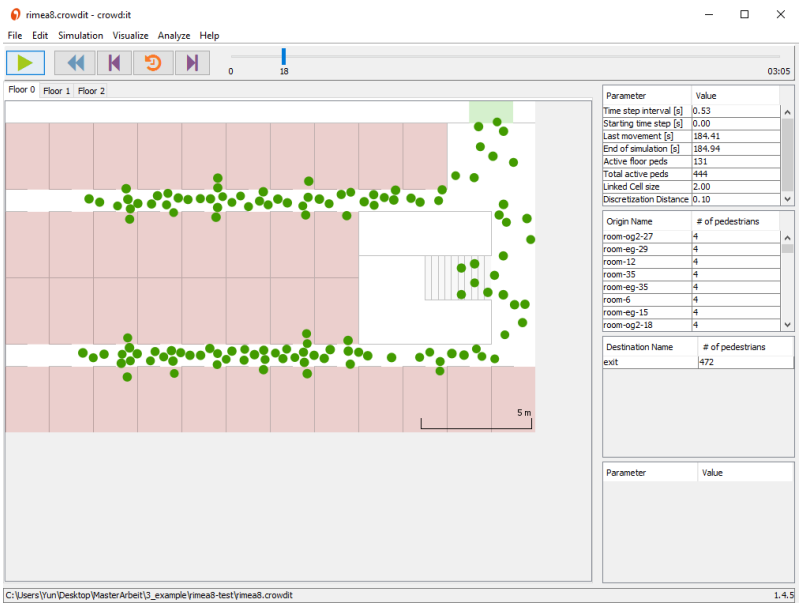

FIGURE 3.14: Crowd simulation with crowd:it

The output based dimension conversion requires a new simulation input file would be written based on the geometry information that has been retrieved. Therefore, the most important work in this part would be analysis of the input file of the software. In this section a basic introduction of the input file would be provided. Based on this, the basic idea of how to write the input file would be introduced. And finally a full flow of the data conversion would be summarized and presented.

3.3.1 Input of *crowd:it*

The input file of *crowd:it* is also XML file format but with extension as "*.floor". It records the coordinates of points from the building elements, and these points line up the outline of the elements. And each file only provides the geometry information for a certain floor. All floor files together build up the building geometry which is used for simulation. The following table shows the example elements representation in a sample floor file.

```
 1. <?xml version="1.0" encoding="UTF-8"?>
 2. <floor xmlFormat="0.7.1" isoDate="2017-08-22T23:01:46.544Z">
 3.   <layer id="geometry-of-floor0">
 4.     <wall id="ow2" closed="false">
 5.       <point x="2" y="0"/>
 6.       <point x="2" y="3"/>
 7.     </wall>
 8.   </layer>
 9.   <layer id="trafficObjects-of-floor0">
10.     <wunderZone id="room-eg-40-WZ">
11.       <point x="0" y="0"/>
12.       <point x="2" y="0"/>
13.       <point x="2" y="3"/>
14.       <point x="0" y="3"/>
15.     </wunderZone>
16.     <wunderZone id="stair-1-eg-WZ">
17.       <point x="22.0" y="8.0"/>
18.       <point x="22.0" y="6.0"/>
19.       <point x="19.0" y="6.0"/>
20.       <point x="19.0" y="8.0"/>
21.     </wunderZone>
22.     <wunderZone id="exit-WZ">
23.       <point x="21" y="14"/>
24.       <point x="23" y="14"/>
25.       <point x="23" y="15"/>
26.       <point x="21" y="15"/>
27.     </wunderZone>
28.   </layer>
29. </floor>
```

TABLE 3.9: Fragment example of a part of floor file

In the Table 3.9, it is obvious that the file is written in XML. There are four elements provided in the fragment, which are wall, room, exit and stair. Among these, wall, room and stair can be provided in the IFC building models. Exit however, is settled manually.

- Wall: In wall element, only two points are nested as children in the file. These two points represent the two endpoint of the wall and could be lined up to provide a straight wall. The attribute of the start tag provides the id number of the wall and also the information of whether the wall is closed.
- Exit: The exit element is the gate of the building. In this case four points are used to provide a closed area representing the exit. The start tag attribute only provides an id number.
- Stair: As for the stair, the single detailed steps are not included, and only the outer contour is presented also by four nodes. From the attribute of the start tag, it can be seen that the corresponding floor is provided.
- Room: Similar to the exit and stair, the room element is also presented by four nodes surrounded area. The area are usually enclosed by walls.

Among these elements, the wall is different from room, stair and exit, and it is settled in the geometry layer. The rest three are settled in traffic objects layer.

From the IFC files, the geometry information can be obtained, which means we could definitely build the geometry layer based on *IfcWall* and *IfcWallStandardCase*. In traffic objects layer, stairs can be build by geometry information based on *IfcStair*. Exit can be built through the door element *IfcDoor*, and the room can be obtained from *IfcSpace*. But the door element can either for rooms or as a gate for building, so that the manual selection for the gate is needed. In the proposing of the programme in chapter 4, this selection may not included.

The writing of the input floor file can be done by printing out the coordinates of the bounding points. The coordinates can be calculated from the geometry data retrieved before. So that the basic idea of the creating input file has two steps. First step is the calculation of the geometry data to get coordinates of necessary points. The step two is to create the routing of XML writing which includes all the elements and points from the step one.

3.3.2 Summary

From the previous sections, it can be seen that, the input file of the *crowd:it* is not complicated. With the enough geometry information, the file can be easily created. According to the content in this section, the main procedure of the next programming work could be enumerated.

- Data retrieval from the ifcXML file of building model;
- Data process into coordinates for *crowd:it* input files;
- Output "*.floor" XML file.

The programme would consist of these three parts, firstly is the data retrieval that has been introduced in this chapter. Second would be some mathematical calculations for the geometry data, and get final coordinates that could be written. The last step would be output these coordinates into the XML file format with the identifiable postfix.

Furthermore, the retrieval data would mainly includes the outline shape of the element especially for stairs. The geometry of each tread and riser would not be included. So that further refinements of traffic objects could be done in the main *crowd:it* file.

Chapter 4

Result and Analysis

According to previous chapters, a programme for data conversion to crowd simulation is designed in this chapter. Firstly, a short introduction of XML file parsing in Java would be presented. Then a filter and a converter programme are realized in Java. through both programmes the "*.floor" input file for crowd simulation would be generated from ifcXML file. Besides that, the tool *IfcOpenShell* will also be shortly introduced. The result from *IfcOpenShell* will be compared with the result from the developed programme.

4.1 Parse in Java

Java is a programming language which is concurrent, class-based, object-oriented. It is one of the most popular language particularly for client-server web applications. By using Java byte-code, software components can be reused across heterogeneous computing platforms [33]. A close relationship between XML and Java has existed since the early days of the XML effort [34].

To process the data in XML documents, an programme is to be developed. The Java programming language contains several methods for XML processing. Making use of the methods, the programme could mark and retrieve data from XML documents. These methods are the application programming interface (API) softwares that sits between the application and the XML documents to help the developer from the intricate XML syntax. The parser reads a raw XML document, first ensures that it is well-formed, and then validate the document against a data type definition (DTD) or schema [35].

The API offers programmers a set of functions that they can directly use to request information from the parser which processes the document. Some parsers are able to check (or validate) an instance in an XML document against the DTD that is used to describe the vocabulary, and to check whether the actual markup conforms to the rules of the markup language [33].

4.1.1 API tools for XML parsing in Java

There are two basic approaches to build an XML programming, and both are easily available through Java. They refer to the document object model (DOM) and the Simple API for XML (SAX) which are the two most common used APIs. These approaches are tree-walking and event-based.

- Tree-walking: The document is parsed into a tree, and the application walks around the tree looking for interesting or target elements.

- Event-based: Each time the parser detects an interesting event, it calls some method on one or more application objects. This can be further subdivided into APIs using simple callbacks and APIs using event objects [34].

The representative tree-walking API DOM is developed by W3C. The SAX developed by David Megginson is most often used event-based API. Although SAX is not sanctioned by any standards body, it is supported by most of the available XML processors [36]. From Figure 4.1, it is clear to find the different approaches of paring the XML file.

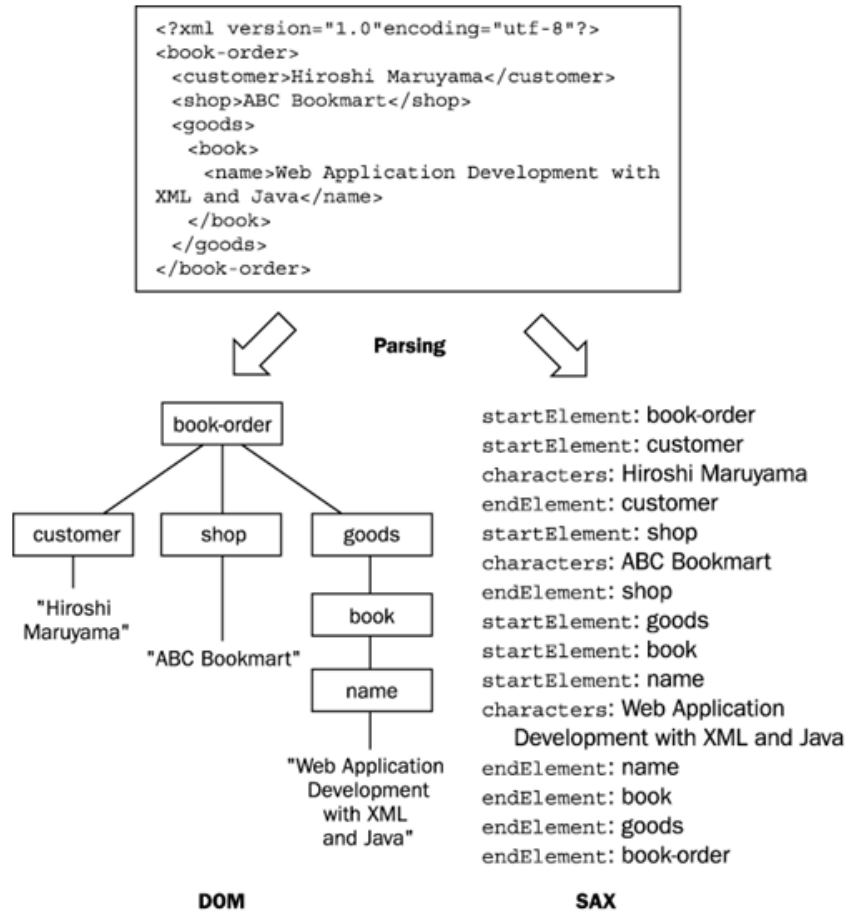

FIGURE 4.1: Comparison between DOM and SAX [36].

In DOM, an XML document is parsed as a tree with nodes representing elements, text, and so on. The tree would be generated by an XML processor and hands it over to an application. The tree has only one single root node, and all other nodes in this tree except for the root have a single parent node. Furthermore, each node could has a list of child nodes. In some cases, there could be a leaf node when the list of children of the node is empty [37]. A DOM based XML processor creates the entire structure of an XML document in memory, so that during the process of the DOM, the whole XML input file has to be loaded entirely, which may cost a large portion of internal storage.

SAX is designed to be quite lightweight that does not generate a tree structure of an input document, so that it is a simple API for XML. Based on event driver, SAX

has several event handler interfaces and also provides the default implementation class [36]. As a Java native, parser-independent, event-based API for processing XML documents, it is also suitable to use with very large documents and streaming data and is organized mostly around interfaces [37].

DOM and SAX are two original XML processing APIs. After the development of DOM and SAX, there are also several new APIs work on XML files. Most of them inherit the advantages of DOM and SAX but are extended with new functions and features. For example, JDOM and DOM4i both create XML trees during parsing process, and StAX walks only one way through XML files similar as SAX.

Based on these Java APIs, after parsing the ifcXML file, a small routine will be developed, which can help user to find a certain information in different models. Then, another programme for data conversion would be developed. This would help user to have a routine of full function to get goal file for simulation directly from ifcXML file.

4.2 Case study of ifcXML filter

This ifcXML filter example is written in the StAX iterator API. The programme is developed to pick out the corresponding elements in the building. At the same time, the ifcXML remains its integrity in a large extend.

StAX, with the full name streaming API for XML, is an API for reading and writing XML documents. The StAX API exposes methods for iterative, event-based processing of XML documents. Similar as SAX, XML documents are treated as a filtered series of events, and info-set states can be stored in a procedural style [38].

However, StAX has its own features comparing to SAX. StAX is a Pull-Parsing model while SAX pushes data to the application. In StAX, application can take the control over parsing the XML documents by pulling the events from the parser [39]. Furthermore, the StAX API is bidirectional, enabling both reading and writing of XML documents [38].

Inherited the advantages of SAX, the StaX is an economy and light API that can be generally used in different size of XML, and also quite efficient to pull out the elements that the user are really interested.

The StAX API has two distinct API sets:

- Cursor API.
- Iterator API

The StAX cursor API represents a cursor with which the user can go through an XML document one way from beginning to end. This cursor can point to each construct like characters, tags etc. during the way, and the user can set one construct once it is pointed [38]. Note that, the cursor would always move forward, and never backward.

The StAX iterator API regards an XML document stream as a set of discrete event objects [38], and actually it is based more on the file stream. User could pull out the

interested events which are read in from the source XML document.

Comparing to the tree-walking and event-based API, StAX shows a great advantage in parsing XML. It not only avoids the collapse of the system by enormous XML document tree, but also concentrates more on pulling out the elements which are required by users. By comparing cursor and iterator StAX APIs, iterator based API would be more user friendly. Because of the event based feature, there would not be much differences and difficulties in ifcXML file parsing comparing to normal XML file. The most important is to distinguish goal tag name through IFC schema in order to locate it correctly in the file.

Based on the comparison with other API, the filter programme would be developed by using the StAX to parse a ifcXML file, and the iterator API is chosen in this case.

4.2.1 Introduction and programme test

A crowd simulation of a whole building consists of part simulations for each floor. According to the procedure, only elements that obstruct passengers in the space should be drawn into the plane graph. So that the interested elements to be extracted would be mainly the building elements such as walls, doors, stairs and etc.

The main features of the ifcXML that is concerned in this programming are listed below. After the study of IFC and structure of ifcXML file, we would come up with a proper algorithm to applicate on the real model.

- Every entity would be given with a unique id number.
- There are two types of expression for parent and children referring chapter 2 section 3.2.
- The explicit attributes would not appear in the ifcXML while they are empty.
- Every IFC element would have its unique name, while its attributes and properties could be same as other IFC elements'.

As a filter of building elements being applied to crowd simulation, there would be some requirements and expectations.

- Integrity of the building elements.
- Access to the selected building elements.
- Output file in XML form.
- High efficiency to decrease the file size.
- Appropriate running time.

As listed above, the aim of the programme is actually to extract entities related to the selected building elements and their explaining instances. So that, it would be necessary to find out the relationship between entities and building elements.

By the research on different IFC and ifcXML files, it is not difficult to find out that the attributes of an element would be related either by full sub-element nesting expression or id-ref pairs expression. So that, a basic algorithm of the filter would be to make full use of those attributes to relate a selected element in an ifcXML file [16].

```
1. <IfcOwnerHistory id="i1944">
2.   <OwningUser>
3.     <IfcPersonAndOrganization xsi:nil="true" ref="i1941"/>
4.   </OwningUser>
5.   <OwningApplication>
6.     <IfcApplication xsi:nil="true" ref="i1907"/>
7.   </OwningApplication>
8.   <ChangeAction>nochange</ChangeAction>
9.   <CreationDate>1517668138</CreationDate>
10. </IfcOwnerHistory>
11. ......
12. <IfcLocalPlacement id="i2045">
13.   <PlacementRelTo>
14.     <IfcLocalPlacement xsi:nil="true" ref="i2032"/>
15.   </PlacementRelTo>
16.   <RelativePlacement>
17.     <IfcAxis2Placement3D xsi:nil="true" ref="i2044"/>
18.   </RelativePlacement>
19. </IfcLocalPlacement>
20. ....
21. <IfcProductDefinitionShape id="i2078">
22.   <Representations exp:cType="list">
23.     <IfcShapeRepresentation exp:pos="0" xsi:nil="true" ref="i2051"/>
24.     <IfcShapeRepresentation exp:pos="1" xsi:nil="true" ref="i2075"/>
25.   </Representations>
26. </IfcProductDefinitionShape>
27. ......
28. <IfcWallStandardCase id="i2084">
29.   <GlobalId>2qDQkwZlH2yuxdvqIaccMR</GlobalId>
30.   <OwnerHistory>
31.     <IfcOwnerHistory xsi:nil="true" ref="i1944"/>
32.   </OwnerHistory>
33.   <Name>Basic wall:general - 200mm:392314</Name>
34.   <ObjectType>Basic wall:general - 200mm:1044</ObjectType>
35.   <ObjectPlacement>
36.     <IfcLocalPlacement xsi:nil="true" ref="i2045"/>
37.   </ObjectPlacement>
38.   <Representation>
39.     <IfcProductDefinitionShape xsi:nil="true" ref="i2078"/>
40.   </Representation>
41.   <Tag>391703</Tag>
42.   <PredefinedType>notdefined</PredefinedType>
43. </IfcWallStandardCase>
```

TABLE 4.1: An example of *IfcWallStandardCase* and its attributes in ifcXML file.

From Table 4.1, it can be seen that all the attributes of the *IfcWallStandardCase* that appears in the file are nested to the parent *IfcWallStandardCase*. Some of them are written with their full content, but others who have more content or further attributes are written with referencing numbers. Such as "i1944", "i2045" and "i2078", they can be found in the file before the *IfcWallStandardCase*.

This algorithm could be explained as follow. Firstly, directly search the unique name

of the element that is concerned and record the id numbers in a list. Secondly, get the referencing numbers mentioned in this element and also add them to that list. Repeat this step until no more referencing numbers can be found. At last, delete the elements whose id numbers are not appeared in this list and output the new file.

Figure 4.3 shows an example flow of the programming algorithm, which is actually a iterative algorithm. Set the target element as name_0, and its id number id_0 could be found. Then there might be several referencing element ids, and for example there are two here, which are id_1, id_2. After that, both of them can also be located in the ifcXML file, and their referencing element ids can also be found, for example there are together five. If there is no more referencing id can be found in those elements and the elements which are referenced here, then the iteration would stop. Finally, the total id list can be obtained from id_0 to id_7. Back to the ifcXML file, only these eight elements would be remained in the output XML file.

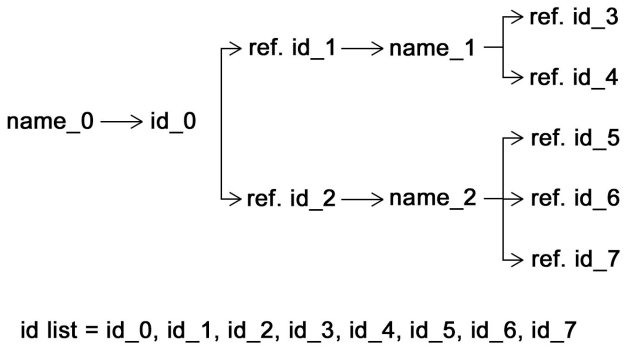

FIGURE 4.2: Example flow of the iteration algorithm in the filter

The main idea of this algorithm is based on the target elements and guarantees the integrity and attributes, which covers the geometry data, would be included in the output file. However, as a result, all the branches that are stretched out from the target IFC building element are still not complete enough for a full spatial structure. At present, there is no programme or application could build 3D model directly from ifcXML files. So that our judging criteria would base on section 2.1.3, where the compulsory settings and elements are listed.

So, with the help of this iterative algorithm, we could make up the final process of the programme. There are basically four steps in the filter programme.

1. Create an id list of entities based on the input ifcXML file;
2. Iteration work on the id list based on input demanded entities;
3. A new id list would be created with only interested entities;
4. Use the StAX XML output packages to generate the final ifcXML file.

In Figure 4.3, it shows the above-mentioned procedure. Block filter shows the brief introduction of this example programme.

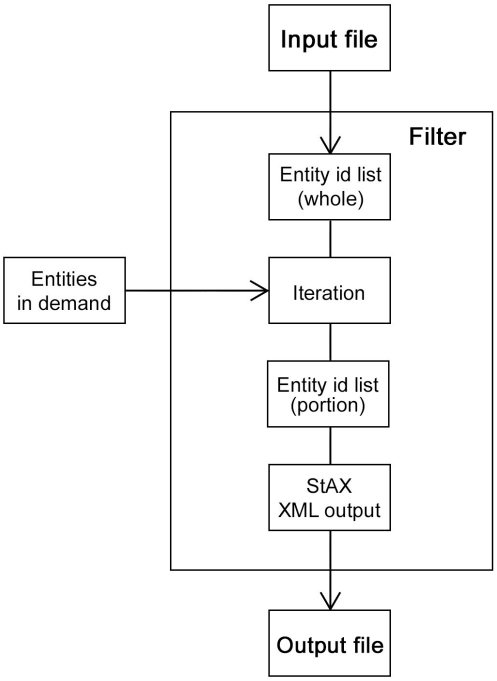

FIGURE 4.3: Flow chart of the the file filter

After the introduction of the programme, we would focus on the integrity of the ifcXML file. From the section 2.1.3, it is clear that some of the elements are essential to keep the integrity of the ifcXML file. And those following elements must be contained in the output file.

- *IfcProject*
- *IfcRoot*
- *IfcMeasureResource*
- *IfcRepresentation*
- *IfcPropertySet*

Some of them are not directly linked to the building elements. That means after the filtering, they are still not included in the id list. In this case, a multiple iteration is used. It is based on the first iteration result, which would be compared to the list above. Then the names that are not included would be added. And then, the programme would gather all the id numbers from the iteration, and delete the repeated id numbers, list out the final ids. So that all the elements in the final list are contained in the output XML file.

In the Figure 4.4, it shows the FilterWork class which contains all the filter algorithms and output. The two containers below are used in the FilterWork for recording the id numbers as well as referencing id numbers in order to make the calculation and iteration.

All the control would be done in the FilterMain class which is provided in Figure

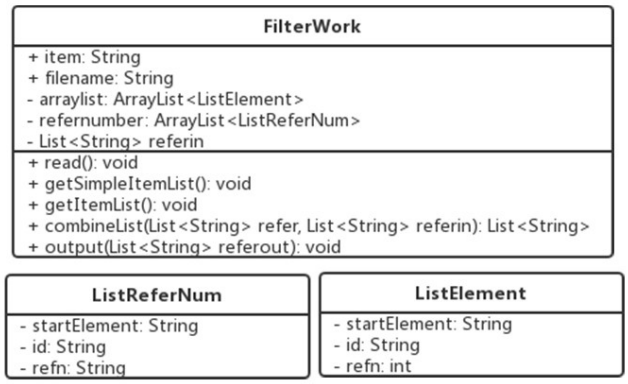

FIGURE 4.4: Classes for FilterWork

4.5. This class not only invokes the FilterWork class, but also creates the graphical user interface (GUI) for user. The GUI is developed for providing the convenience. User would quickly know how to operate when facing the GUI, even if one doesn't know the correct name or spelling of the IFC entities.

FIGURE 4.5: FilterMain class

The GUI is provided in Figure 4.6. User is demanded to input the name of the input file, and meanwhile, the input file has to be saved in the same folder as the one of the programming code. Then four options is provided, which are *IfcWall*, *IfcDoor*, *IfcSpace* and *IfcStair*. Also the programme itself provides a further choice of the building elements.

All the target building elements has been extracted and saved in the output file of this programme. After checking the integrity of the output file, the essential elements are all included.

Comparing the output file with the input file, it can also be found that the size of the file has been significantly decreased, which can be up to 80%. This also depends on the complexity of the original model. The time consuming of this programme is evaluated. After trying with different models which include large complex models, it might be a bit slow, but based on the user experience, it is still acceptable. Also, there are still some improvements that could be made in this programme. About the GUI, a upload of input file can be provided to the user to be more convenient.

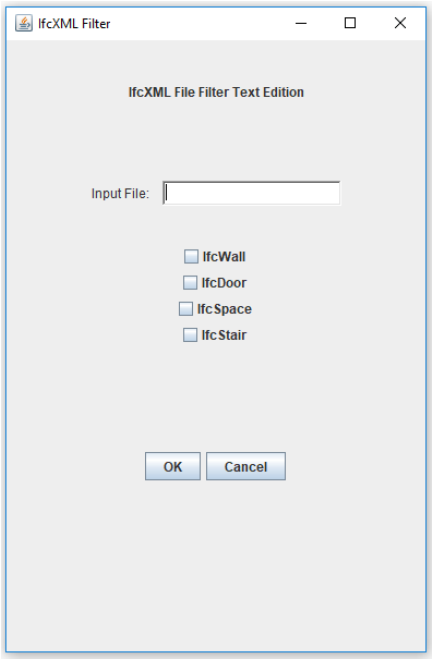

FIGURE 4.6: IfcXML file filter test edition GUI.

Regarding to the function of the programme, more options of the building element could be provided. And based on the algorithm of the iteration, there might be a faster algorithm that is waiting to be developed.

4.3 Case study of file converter

The previous example of ifcXML filter implements the first step of the total data conversion design, which can be seen in Figure 3.1. In this section, the rest three steps which are geometry data retrieval, 3D to 2D conversion and file output would be discussed and implemented. After comparison and analysis, the JDOM API is chosen for the XML parsing and output in this converter programme.

JDOM is an open source library that can be used to read, write, create and modify XML Documents. JDOM is lightweight, fast, Java optimized and uses Java collections. Although it's similar to the DOM, it's an alternative document object model that was not based on DOM or modeled after DOM. JDOM is an alternative to DOM and SAX parser, but it also works well with DOM and SAX APIs [40].

JDOM integrates not only with DOM but also SAX, and it also supports XPath and XSLT. Based on the filter programme, some requirements have been well satisfied. The access to selected building elements and XML file output are achieved. Because of the better performance with walking through the XML file of the JDOM API, the locating of specific element with it id number would be quite easier. From the previous chapter, it clearly introduces the routines of each interested data. During the dimension conversion, the geometry data judging would be done with the help of IFC schema. Therefore, all the issues of this programme will be all discussed and solved.

4.3.1 Introduction

By the use of Java, above-mentioned can be programmed into one integrated project, which could build up a integral file converter, and this is actually a extension of the filter programme. There are three preconditions of the example programme, which are parsing on ifcXML file, dimension conversion based on output file and "*.floor" format XML output.

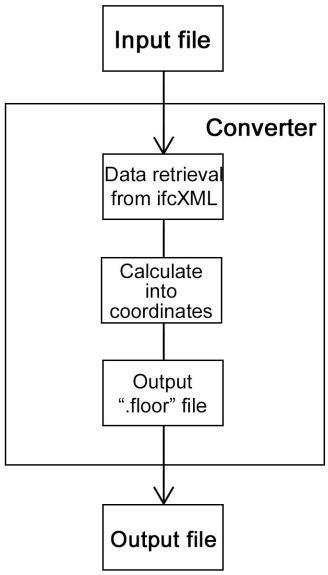

FIGURE 4.7: Flow chart of the file converter.

To design the programme structure, those three preconditions must be included, after that the efficiency and extendability are also needed to be considered. So that in this example programme, the fundamental functions listed.

- Locate the specific element by its id number.
- Obtain all elements with the same name as the located one.
- Obtain the specific attribute information in those elements.
- Judge the geometry data and filter out or calculate 2D data.
- Output the XML file.

The programme is mainly consists of three parts, which are actually corresponding to the three steps in Figure4.7.

Figure 4.8 presents the programme flow and procedures are presented. As can be seen, the file would be converted and outputs by stories loops. There would be a whole procedure from data retrieval to output for each storey. The three main functions are marked on the left side in orange colour, among them the most complicated would be the data retrieval. Not only because there is a large amount of elements to handle, but also since data retrieval is element depended for different elements or geometry cased, different data retrieval methods are needed. According to the mainly concerned four building elements, which are *IfcWall*, *IfcStair*, *IfcDoor* and *IfcSpace*, there are four classes corresponding them. An extra class for combing doors

with their attached walls is needed, this would provides coordinates of new walls divided by the doors. So that in the programme, all these five classes contain not only the geometry data retrieval but also the coordinates calculation.

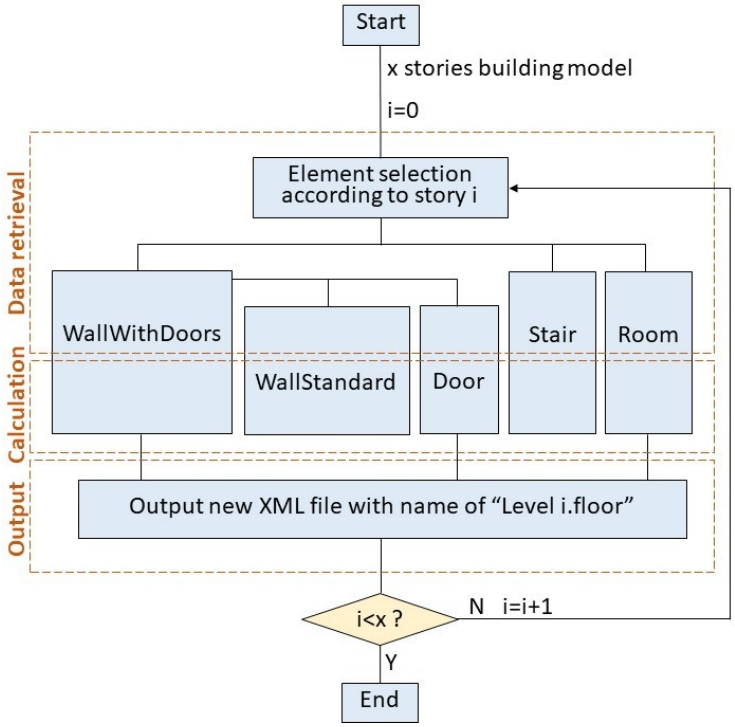

FIGURE 4.8: Overview of programme structure.

Figure 4.9 shows three classes for standard wall and door. In class WallStandard and Door, there would be firstly a function to locate the goal element in the file and to detect the geometry. Then another function to get the coordinates through calculation of geometry values is also needed. The class WallsWithDoor is for combining the walls with attached doors. It would detect all the doors on a certain wall and through a calculation to get coordinated of fragment walls.

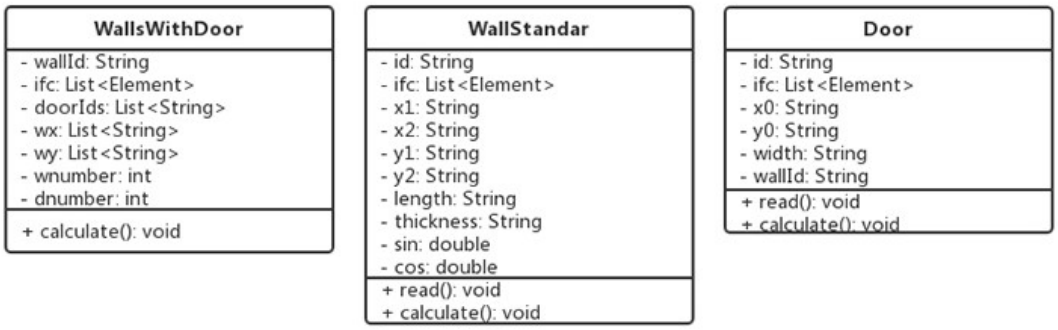

FIGURE 4.9: Classes for standard wall and door

Figure 4.10 shows two classes for stair and room. Similar to class WallStandard and Door, this two classes also have two functions for geometry reading and coordinates

calculation. Since for straight stairs and normal rooms with four walls, they are both in shape of rectangle so that there would be four nodes of coordinates to calculate.

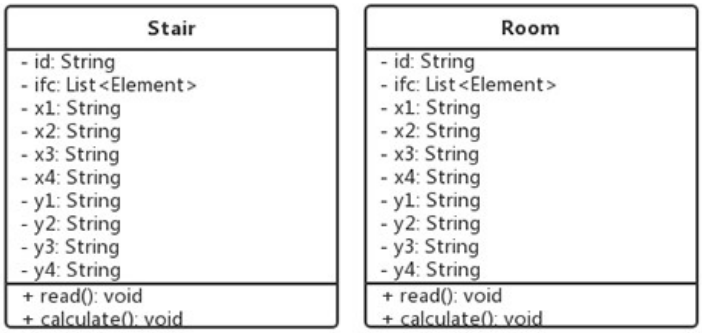

FIGURE 4.10: Classes for stair and room

Despite these classes for building elements, there are three more classes which are main class including the element selection according to story number i, output class for writing the "*.floor" file and the fundamental function class including static methods of detecting attributes and values in certain elements. All the instance methods in this class are shown in Figure 4.11.

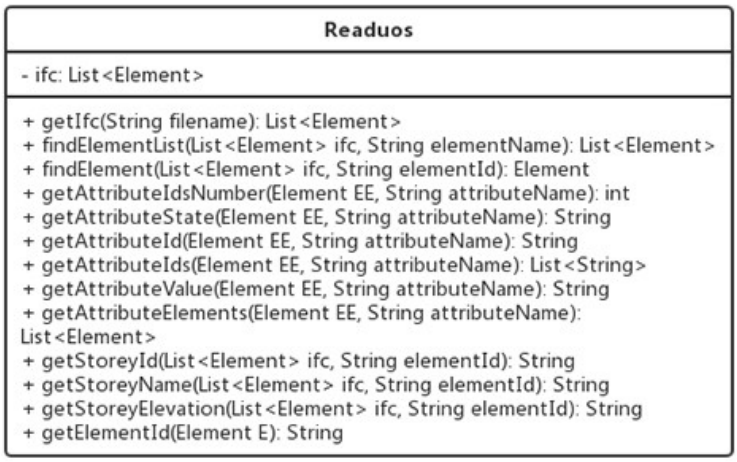

FIGURE 4.11: Classes for fundamental function Readuos

Therefore, through the whole programme, user would get "*.floor" output files from ifcXML input file. All the geometry information of walls, doors, stairs and room would be intructed and saved in "*.floor" file for crowd simulation. But as an initial attempt and JDOM based programme, it still has several limitations. The advantages, disadvantages and limitations are listed below.

- Advantages:
  1. Easy and simple controlling;
  2. Realize the data conversion from IFC to crowd simulation.
- Disadvantages and limitations:
  1. Relatively slow operation;
  2. Only walls, doors, stairs and rooms are under operation;
  3. Only standard walls are considered;
  4. Only straight stairs with single stair flight are considered;
  5. Only rectangle rooms are considered..

In this programme, user couldn't select elements, and only the mentioned four entities are considered and would be operated. Furthermore, the programme could only deal with standard walls which should be straight and equal height, thickness along the length. Also the stairs should be straight and could only have one stair flight without platforms. Because of using JDOM API, the running progress would be rather slow, especially when dealing with large and complicate model files. Therefore, the combination with filter programme is necessary when the file size is too large. While combining the filter and converter programme, user could only control through the filter GUI to provide ifcXML file and selected concerned elements. Since in the filter programme, only the four elements can be handled, user needs to select corresponding options in filter GUI.

4.3.2 Programme test and result

To test the developed programme, a small house model has been built through *Autodesk Revit*.

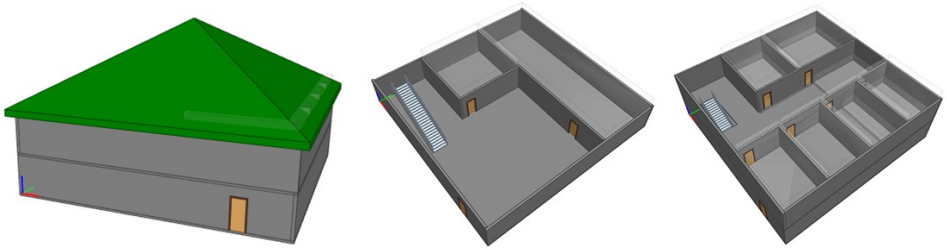

FIGURE 4.12: Simple house 3D model graphs in BIM Vision

As can be seen from the Figure 4.12, It is a small house with only two floors with walls, slabs, doors, stairs and rooms. From the 2D floor plans, the arrangements of two floors are settled differently. The model is saved in the ifcXML file format and imported into the developed programme. Since it is a simple small building model, the operating is quite fast.

From the original building model, two floor files with walls, stairs and room would be exported. As the top level is only with roof slabs, an empty output file for top level is expected.

As a result, the file for level three is empty, and the elements in other two level are all exported in the other two output files. After imported into the *crowd:it*, the following floor plan would be received.

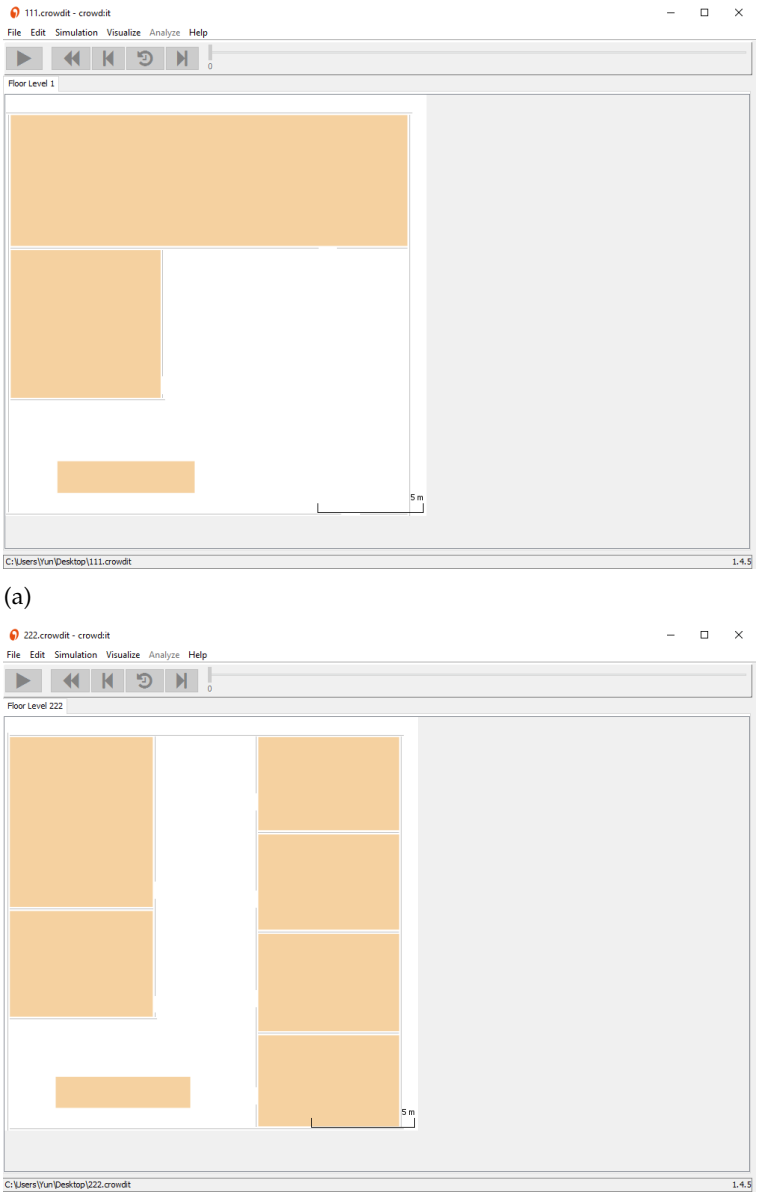

FIGURE 4.13: Geometry display in *crowd:it* for level 1 (a) and level 2 (b)

Figure 4.13 shows the converted file from the developed programme with *crowd:it*. Comparing to Figure 4.12, the complex geometric data has been correctly converted by developed programme and shown in *crowd:it*.

4.4 A short study of *IfcOpenShell*

During the on-line study on the buildingsmart website, they also provide a collection of available open resources to support software development for IFC based interoperability. In these open sources there is a software library that especially works with IFC file format, which enables IFC files being converted into other file formats. Among these output file format, the SVG file is the most interested one. The procedure is quite similar to this study, that it creates 2D SVG floor plans from 3D model IFC files.

After a short introduction of the library and usage, the sample structure would be tested and its performance would also be analyzed and compared with the one from developed programme.

4.4.1 Introduction and usage

IfcOpenShell is an open source software library that helps users and software developers to work with the IFC file format. It uses *Open CASCADE* internally to convert the implicit geometry from an IFC files into explicit geometry supported by any CAD software or modeling package. *IfcOpenShell*'s permissive LGPL license encourages it to be used freely in both proprietary and open source software [10].

IfcOpenShell has many mesh-based viewers. It has a significant advantage that it is backed by a powerful modeling kernel which is the *Open Cascade*. This has already proven its high performance in the ease of creating 2d floor plans from 3d elements in the SVG exporter [41].

At present, *IfcOpenShell* is still under development, the current version supports only files with the "*.ifc" extension and it can not fully understand all geometry information that is presented in an IFC file.

The *IfcConvert* as one of the tools provided by *IfcOpenShell* is used for format conversion. It could be installed in Windows Visual Studio or Linux. The format conversion of an IFC file supports to output following formats. Among these file format, SVG would be the one that will be used in this study.

- "*.obj": Wavefront OBJ;
- "*.dae": Collada, digital assets exchange;
- "*.stp": STEP, standard for the exchange of product data;
- "*.igs": IGES, initial graphics exchange specification;
- "*.xml": XML, property definitions and decomposition tree;
- "*.svg": SVG, scalable vector graphics (2D floor plan).

The operating of the programme is through command line in the terminal window. Then, the output file will be automatically created. The command line is basically consist of four parts: programme sign, input file, output file and further options.

User must type *IfcConverter* in the beginning of the command line for invoking the installed library. Second part is the full name of the input file, and note that the input

file should be saved under the same path where the terminal points. The name of output file can be fully customized by user, but must end with extension of "*.svg". The output file would be saved under the same path as the input file. Last part is to set the operation options including geometry and serialization options, for example the bounds, filtering entities and etc.

Here are some options that could be used in converting to SVG [10].

- –bounds arg
  Specifies the bounding rectangle, for example 512×512, to which the output will be scaled. Only used when converting to SVG.

- –section-height arg
  Specifies the cut section height for SVG 2D geometry.

- –use-element-names
  Use entity names instead of unique IDs for naming elements upon serialization.

- –use-element-guids
  Use entity GUIDs instead of unique IDs for naming elements upon serialization.

- –include arg
  Specifies that the entities that match a specific filtering criteria are to be included in the geometrical output:
  1) 'entities': the following list of types should be included. SVG output defaults to *IfcSpace* to be included. The entity names are handled case-insensitively.
  2) 'layers': the entities that are assigned to presentation layers of which names match the given values should be included.
  3) 'arg <ArgumentName>': the following list of values for that specific argument should be included. Currently supported arguments are *GlobalId*, *Name*, *Description*, and *Tag*.

- –include+ arg
  Same as –include but applies filtering also to the decomposition and/or containment (IsDecomposedBy, HasOpenings, FillsVoid, ContainedInStructure) of the filtered entity, e.g. –include+=arg Name "Level 1" includes entity with name "Level 1" and all of its children. See –include for more information.

- –exclude arg
  Specifies that the entities that match a specific filtering criteria are to be excluded in the geometrical output. See –include for syntax and more details.

- –exclude+ arg
  Same as –exclude but applies filtering also to the decomposition and/or containment of the filtered entity. See –include+ for more details.

- –filter-file arg
  Specifies a filter file that describes the used filtering criteria. Supported formats are '–include=arg GlobalId ...'and 'include arg GlobalId ...'. Spaces and tabs can be used as delimeters. Multiple filters of same type with different values can be inserted on their own lines. See –include, –include+, –exclude, and –exclude+ for more details.

- –plan
  Specifies whether to include curves in the output result. Typically these are representations of type Plan or Axis. Excluded by default.

- –model
  Specifies whether to include surfaces and solids in the output result. Typically these are representations of type Body or Facetation. Included by default.

- –convert-back-units
  Specifies whether to convert back geometrical output back to the unit of measure in which it is defined in the IFC file. Default is to use meters.

- –use-world-coords
  Specifies whether to apply the local placements of building elements directly to the coordinates of the representation mesh rather than to represent the local placement in the 4x3 matrix, which will in that case be the identity matrix.

4.4.2 Test and result

This open library would also be tested with the simple building model created in *Autodesk Revit*, which can be seen in Figure 4.12.

Since the entities are added respectively after the "–include" command, firstly we would try to output only with walls by following command.

```
~/Documents$ IfcConvert littlehouse.ifc house.svg --bounds 512x512 --include
entities IfcWall
```

In order to compare the performance of this converter with the one from developed programme in previous section, a contractive command would be created with further entities, which are *IfcDoor*, *IfcStair* and *IfcSpace*. They would be simply added after *IfcWall* in command line as below.

```
~/Documents$ IfcConvert littlehouse.ifc house.svg --bounds 512x512 --include
entities IfcWall IfcDoor IfcStair IfcSpace
```

Figure4.14 shows the first floor plan graphs from results under those two commands comparing to the model in Revit. From the figure, it can be seen that, the Figure (b) has only walls presented, but the Figure (c) has both walls and doors.

The stair is expected to be shown in Figure 4.14 (b), but however the output file shows that *IfcStair* can not be understood by the library. The *IfcSpace* is also unrecognized. We also tried command lines that only have these two entities, and it was resulted with an error that no geometrical entities found, which also proved that they are not recognized. And from the graph (b), the walls are automatically separated by the doors. Only with the command including doors, the doors symbol would be displayed. Then user could know which gaps are because of doors and which are initially different walls.

From the output file, it can also be seen that, all the walls that in this model would be drawn in the SVG format. It contains both floors and are presented all together by default. So that a SVG editor is needed for restore these two floors separately.

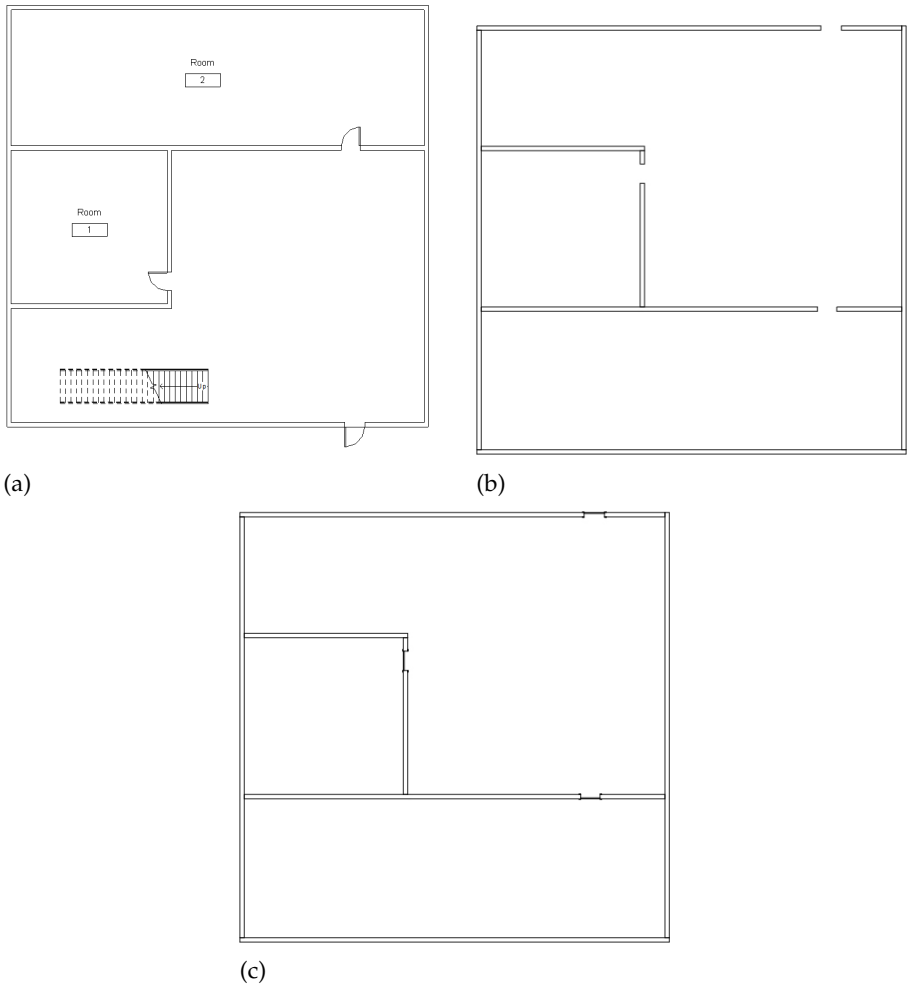

FIGURE 4.14: Plan graph of first floor from *Revit*(a) comparing with *IfcOpenShell* output with only wall command (b) and command to add door information(c)

In the plan graph from *Autodesk Revit*, the upward direction is north, which is opposite to the one from *IfcOpenShell*. So we could see in the Figure 4.14, the figures (b) and (c) are rotated with 180 deg. The same can be found in the second floor plan graphs in Figure 4.15.

Furthermore, the *IfcOpenShell* also has a filter function. The user can filter according to specific value of the certain attribute. For example, -include+ arg Name "Level 1" includes entities with name of "Level 1" and all of its children.

So that it is also possible to output only interested floors of the entire model. With following example command, the output SVG file would only contains all the walls in level 1.

```
~/Documents$ IfcConvert littlehouse.ifc house.svg --bounds 512x512 --include
entities IfcWall --include+ arg Name "Level 1"
```

Besides, there are some other options such as exclude and etc. But for 2D floor plan output, it would be enough by using introduced commands.

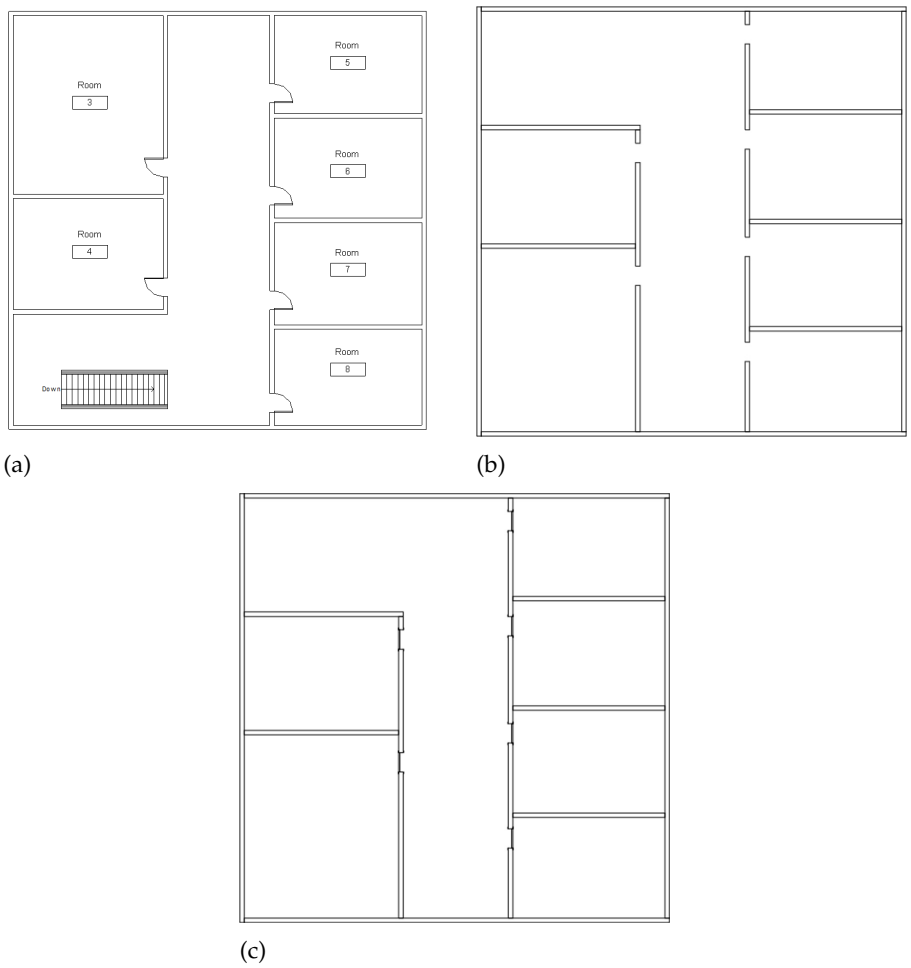

FIGURE 4.15: Plan graph of second floor from *Revit* (a) comparing with *IfcOpenShell* output with only wall command (b) and command to add door information (c)

In the end, for crowd simulation, the geometry accuracy and integrity would be critical. After the comparison and summary with further models, the advantages and disadvantages with *IfcOpenShell* would be listed below:

- Advantages:
  1. Fast and stable operation;
  2. Accurate and integral geometry output;
  3. The coverage of all stories.
- Disadvantages:
  1. Limited recognizability of the building elements;
  2. Further SVG supporting needed for file conversion into crowd simulation.

As a open source library working with the IFC file format, it is quite helpful for user to quickly obtain 2D floor plan from 3D BIM model. But for crowd simulation, the biggest problem of this library would be the limited recognizability of the building elements, especially *IfcStair*. So that further improvement is needed in this library.

Comparing with the developed programme in previous section, the *IfcOpenShell* approach has lower design cost and it can recognize the wall elements quite well. But it can not be used to handle some other building elements like stairs. And the output file is in SVG format. Extra data conversion tools are needed to match *crowd:it*. On the contrary, the developed programme is optimized to generate the proper file for crowd simulation. The output file is directly with extension "*.floor", which is exactly the supported file format of *crowd:it*. Above all, the designed programme can extract correctly all the important geometry information of doors, stairs, walls and rooms, which is necessary for crowd simulation. Even with longer run time, the designed programme shows better performance with more accuracy. It matches the requirement of the study.

Chapter 5

Conclusion

As a broad application of BIM model in building and construction, it is more practical to run a crowd simulation basing on BIM. In this case, a 2D floor plan of a building is necessary. The most efficient way to get the 2D geometry information is to directly convert from the existing 3D BIM model. Of course the converted 2D floor plan should have exactly the same information as the original 3D model. In this study, a functional Java based programme is designed. Its performance is also compared with the open source library *IfcOpenShell*. The comparison shows that the designed programme can satisfy the requirement of the study. This study basically consists of four main parts.

- Study of IFC files;
- Analysis of data conversion algorithm;
- Java implementation and test;
- Performance comparison with *IfcOpenShell*.

In the first part, fundamental knowledge of IFC schema and IFC model representation has been presented. Besides, the two IFC file formats: IFC-SPF and IFC-ifcXML have been introduced. IfcXML has been studied and analyzed in details, especially about the elements relations. After a comparison of these two file formats, the advantages of ifcXML file have been summarized.

The analysis of data conversion algorithm has been talked in three steps. Based on the detailed study of ifcXML, the idea of geometry information retrieval has been put forward. In the first step, paths have been presented for important building elements including walls, stairs, doors and spaces. After the introduction of building storey represented in ifcXML files, a solution of 3D to 2D file conversion has been introduced in the second step. In the final step, the input file of *crowd:it* has been analyzed.

For the Java implementation and test, a filter programme and converter programme have been developed and integrated together to convert the data from BIM ifcXML file into the one for crowd simulation. The developed programme not only handles the important building elements but also directly output with extension "*.floor", which directly suits the demands of crowd simulation software *crowd:it*. Thus no more extra converter is needed, to transfer other file types into "*.floor".

In the final part, an open library *IfcOpenShell* has been introduced and studied. In comparison to the former developed programme, *IfcOpenShell* shows its advantages

in applicability and run time, but has a limit number of recognizable building elements. Compared with it, the designed programme shows a higher feasibility, which can recognize almost all the essential building elements.

	Designed programme	*IfcOpenShell*
Pros	Optimized for *crowd:it*; easy operation	stable; fast
Cons	limit number of element geometry cases; slow	limit number of recognizable elements; SVG file supporting needed

TABLE 5.1: Comparison of designed programme with *IfcOpenShell*

As a matter of fact, the developed programme still has a list of disadvantages including slow operation and limited application cases, as mentioned in section 4.3.1, for example, only straight standard walls and only stairs are considered.

In the future study, more building elements that could be used in crowd simulation should be added. Improvements are needed in some functions for geometry data retrieval. Other geometry cases such as curved walls or stairs with platforms could be added into BIM models in order to find the paths of their geometry information in ifcXML files and include them into the designed programme. The efficiency of the programme is also needed to be increased.

As a primary study of combing BIM with crowd simulation, the work fulfills the expected requirements and provides a visualization of the IFC model in the *crowd:it*. Since the programming work is an attempt testing in this study, it still has some deficiency and limitations. Therefore, modifications and improvements are needed in the future.

Bibliography

[1] Charles Eastman et al. *An Outline of the Building Description System*. Institute of Physical Planning, Carnegie-Mellon University., 1974.

[2] G.A. van Nederveen and F.P. Tolman. "Modelling multiple views on buildings". In: *Automation in Construction* 1.3 (1992), pp. 215–224.

[3] *Frequently Asked Questions About the National BIM Standard-United States*. https://www.nationalbimstandard.org/faqs. 16.10.2014.

[4] Jon Williams. *The Three Worlds of BIM*. https://thebimhub.com/2015/04/14/the-three-worlds-of-bim/. 13.04.2015.

[5] Jim Jacobi. "4D BIM or Simulation-Based Modeling". In: *Structure Magazine* (2011), pp. 17–18.

[6] *ASHRAE Introduction to BIM, 4D and 5D*. https://www.cadsoft-consult.com. 29.05.2012.

[7] *ISO International Standard 10303-28, Industrial automation systems and integration — Product data representation andexchange — Part 28: Implementation methods: XML representations of EXPRESS schema and data*. Geneva, Switzerland (1994).

[8] Autodesk. *Introduction to Revit*. https://knowledge.autodesk.com/support/revit-products/learn-explore?sort=score. 04.2014.

[9] *Simulation of people flow*. https://www.accu-rate.de/en/simulation-of-people-flow/. Retrieved: 2017.

[10] *IfcOpenShell*. http://ifcopenshell.org/index.html. 03.2017.

[11] Johannes Dimyadi, Micheal Spearpoint, and Robert Amor. "Sharing Building Information using the IFC Data Model for FDS Fire Simulation". In: *FIRE SAFETY SCIENCE–PROCEEDINGS OF THE NINTH INTERNATIONAL SYMPOSIUM* (2008), pp. 1329–1340.

[12] Chuck Eastman et al. *BIM Handbook. A Guide To Building Information Modeling*. Hoboken, New Jersey: Wiley, 2011.

[13] *Building SMART*. https://www.buildingsmart.org/bim/. Retrieved: 19.03.2017.

[14] *IFC Introduction*. http://www.buildingsmart-tech.org/ifc/. Retrieved: 2017.

[15] Pingwang Shi, Liangfan Lin, and Xueyuan Deng. "Research on Representation and Management of IFC-Based Building Components". In: *JOURNAL OF GRAPHICS* (2016).

[16] Jongsung Won and Ghang Lee. "Algorithm for Efficiently Extracting IFC Building Elements from an IFC Building Model". In: *Computing in Civil Engineering* 416 (2011), 713–719.

[17] *IFC Version 4.1 Final Release*. http://www.buildingsmart-tech.org/ifc/IFC4x1/final/html/. Retrieved: 2017.

[18] Chih-Wei Tsai. "Retrieving information for Structural Analysis form IFC Building information". MA thesis. National Chiao Tung University, 2007.

[19] Ya-hsing Chen. "The Conversion of IFC Building Spatial Data to CityGML". MA thesis. National Chiao Tung University, 2010.

[20] *ISO International Standard 10303-11, Industrial automation systems and integration — Product data representation andexchange — Part 11: Description methods: The EXPRESS language reference manual, International Organization for Standardization*. Geneva, Switzerland (1994).

[21] G.A. Thomas Liebich and F.P. Tolman. "IFC 2x Edition 3 Model Implementation Guide". In: *buildingSMART International Modeling Support Group* (2009).

[22] *EXPRESS data modeling language*. `https://www.loc.gov/preservation/digital/formats/fdd/fdd000449.shtml`. Retrieved: 21.12.2016.

[23] *4 EXPRESS-G Language Overview*. `http://www.steptools.com/docs/devtools/devtools-7.html`. Retrieved: 9.11.2008.

[24] Yi-fan Dai and Liang Dong. "Brief Analysis on IFC Standard for Data Representation and Exchanging of Building Information". In: *BUILDING SCIENCE* 24 (2008).

[25] *IFC Overview summary*. `http://www.buildingsmart-tech.org/specifications/ifc-overviews`. Retrieved: 19.03.2017.

[26] N. Nisbet and T. Liebich. "ifcXML Implementation Guide. Modeling Support Group, International Alliance for Interoperability". In: (2007).

[27] P. De Meo et al. "Integration of xml schemas at various severity levels". In: *Information Systems* 31 (2006), 397–434.

[28] H. S. Thompson, D. Beech, and M. Maloney. *Xml schema part 1: Structures second edition*. `http://https://www.w3.org/TR/xmlschema-1/`. Retrieved: 28.10.2004.

[29] *XML 1.0 Specification*. `https://www.w3.org/TR/REC-xml/`. World Wide Web Consortium. Retrieved 2010-08-22.

[30] Liyang Yu. *A Developer's Guide to the Semantic Web*. Atlanta, USA: Springer, 2014.

[31] Nayantara Duttachoudhury. "IfcXMLExplorer: A Visualization Tool for Exploring and Understanding IfcXML Data". MA thesis. The University of British Columbia, 2015.

[32] E.F. Begley, M.E. Palmer, and K.A. Reed. *Semantic Mapping Between IAI ifcXML and FIATECH AEX Models for Centrifugal Pumps*. U.S. DEPARTMENT OF COMMERCE, 2005.

[33] Mark Austin. "Tutorial: XML and Java for Scientists/Engineers". In: (2001).

[34] Matthew Fuchs. "Why XML is Meant for Java? Exploring the XML/Java Connection". In: *Web Techniques* (1999).

[35] Hock-Chuan Chua. *Java Programming: Java and XML*. `https://www.ntu.edu.sg/home/ehchua/programming/java/J6d_xml.html`. Last modified: March, 2009.

[36] Hiroshi Maruyama. *XML and Java : developing Web applications*. Addison-Wesley, 2002.

[37] Elliotte Rusty Harold. *Processing XML with Java : a guide to SAX, DOM, JDOM, JAXP, and TrAX*. Addison-Wesley, 2003.

[38] *The Java™ Web Services Tutorial. For Java Web Services Developer Pack 1.6*. `https://docs.oracle.com/cd/E17802_01/webservices/webservices/docs/1.6/tutorial/doc/JavaWSTutorial.pdf`. 14.06.2005.

[39] Lars Vogel. *Java and XML - Tutorial*. `http://www.vogella.com/tutorials/JavaXML/article.html#javaxml`. 06.10.2016.

[40] Jason Hunter. "JDOM and XML Parsing". In: *Oracle Magazine* (2002).

[41] IfcOpenShell. *IfcOpenShell-Blog*. `http://blog.ifcopenshell.org/`. 10.2015.