Research on Parsing and Storage of BIM Information Based on IFC Standard

Ruoxiao Wang¹ and Yongjing Tang²,1,*

¹ College of Civil Engineering, Tongji University, Shanghai 200092, China.
² College of Civil Engineering, Tongji Zhejiang College, Jiaxing 314051, China.
*Corresponding author’s e-mail: ytang@tongji.edu.cn

Abstract. Industry Foundation Classes (IFC) defines the standard exchange data format of building information model. For making it possible to serve the whole life cycle of structure, it is necessary to parse and save BIM information based on IFC Standard into database. The research is implemented by Java language and MySQL database. The whole project is called IFCParser and consists of three modules: IFC Entity Module, IFC File Parser Module and Persistence Module. IFCFile Parser Module uses lexical analyzer and syntax analyzer to decompose IFC files based on EXPRESS language into entity classes, which is called the IFC entity. Persistence Module establishes the mapping relationship between Java class and MySQL database, and uses MyBatis framework to save information into relational database.

1. Introduction
With the development of computer technology and Internet, BIM has been fully applied in the process of project construction. Lightweight BIM for model display is also widely used[1]. However, the life span of the structure includes not only the design and construction periods, but also the performance stability periods and performance degradation periods. Therefore, only using BIM to display the structure is not enough. It is also necessary to store the data of BIM model persistently. At the same time, the storage of information should follow certain criteria to meet the requirements of different departments.

In order to store the information of BIM into database, it is necessary to extract the information from BIM. Most scholars use the secondary development method of Revit when collecting BIM information. Zhao[2] uses the secondary development of the Revit software to convert the geometric data of the model into OBJ format data. Shang[3] uses the Revit API to define a JSON data format, and proposes an algorithm for parsing the data format. This method is easy to develop, but Revit is only one of the development tools of BIM, but there are many BIM software, so the portability of this method is poor. IFC file is a general expression of information model[4], so this paper tries to collect information from IFC file instead of Revit. The whole standard of IFC refers to the object-oriented design pattern[5]. Some scholars use C++ to analyze IFC files[6], while C++ is often used to develop desktop applications and is not easy to develop in Internet applications. In order to better adapt to Internet projects, this paper uses Java language for parsing and persistence. Java is also an object-oriented static language, so it is more appropriate to use java language for IFC analysis.

When storing data, most scholar used to store BIM information in SQL Server. However, SQL Server has limited storage capacity and reading speed, which is not suitable for Internet transmission. At this stage, MySQL data is more suitable for Internet applications[7]. Zhou uses Hibernate, an open source
object relational mapping framework, to map all Java object data into MySQL database, so as to realize the persistent storage of IFC data in relational database[8]. While, Hibernate has been eliminated in Internet applications because it is not flexible enough to query database. Now the mainstream persistence layer framework is MyBatis[9], which will be used in this research.

The whole project is called IFCParse r and consists of three modules: IFC Entity Module, IFCFile Parser Module and Persistence Module.

2. IFC Entity Module

Every class in express language contains attribute and inheritance relation. In IFC Files, each row represents an instance. The representation of a beam in the IFC file is like:

\[
#181 = \text{IFCBEAM}('3Qhc8zUFf4wRXeI81394HG', '41', '300 \times 600\text{mm}', '5', '300 \times 600\text{mm}', '148', '177', '194956');
\]

Each comma separates an attribute, so IfcBeam has 8 attributes, including name, discription, shape and position. The inheritance path of IfcBeam is from IfcBuildingElement to IfcRoot.

The whole standard of EXPRESS refers to the object-oriented design pattern, and Java is also an object-oriented language. Therefore, the construction of Java entity is completely in accordance with the standard of EXPRESS language. Each Java entity in the module is named in the same with the IFC standard, and the Java inheritance relationship is also corresponding to the IFC standard. the inheritance relationship is as follows:

```java
class IfcBeam extends IfcBuildingElement
public abstract class IfcBuildingElement extends IfcElement
public abstract class IfcElement extends IfcProduct
public abstract class IfcProduct extends IfcObject
public abstract class IfcObject extends IfcObjectDefinition
```

Each Java entity includes not only the non inversion attribute inherited from its parent class, but also the StepLineNumber attribute representing the line number(make it easy to find the entity) and the String Array representing the attribute name(use to reverse generate IFC file). For making it easy to create new entity, it also provides the methods of validity test, initialization and cloning. Taking IfcBeam class as an example, the attributes and methods are shown in Figure 1.

![UML graph of IfcBeam Entity.](image)

3. IFCFile Parser Module

Making the computer understand a language, a compiler is necessary. It includes tokenizer and syntactic parser. Tokenizer is also called lexical analysis. It is the process of converting character sequence into word(Token) sequence in computer science. Syntactic parser is used to check grammar and construct data structure composed of input words(Tokens). After the work of tokenizer and syntactic parser, the
character sequence becomes a data structure that computer can understand, and a tool is used to turn the data structure into a real entity model. In the project, it is called the Entity Factory.

Therefore, the whole parser can be divided into three core parts: **Tokenizer, Syntactic Parser and Entity Factory**. Tokenizer and Syntactic Parser execute in single thread, and Entity Factory execute in thread pool to speed up.

### 3.1. Tokenizer
The lexical analyzer can divide a character sequence into subunits. In this paper, these subunits are named Token, which means that the lexical analyzer can split a STEP format IFC file into many tokens. Each Token corresponds to a type. According to the type of the Token, the parser will do different logical parsing. Several types of Token can be obtained by analyzing IFC file rules. The corresponding relations of Token type, meaning and expression are shown in Table 1.

| TokenType   | Annotation   | Expression | TokenType   | Annotation   | Expression |
|-------------|--------------|------------|-------------|--------------|------------|
| EOF         | end of the file | 0x03       | STAR        | asterisk     | "*"       |
| SPACE       | black        | " "       | COMMA       | comma        | ","       |
| EOL         | new line     | "\n"      | DOT         | full stop    | "."       |
| DOLLAR      | null attribute | "\$" | SLASH       | slash        | "/"       |
| LPAREN      | left parentheses | "(" | COLON       | colon        | ":"       |
| RPAREN      | right parentheses | ")" | SEMICOLON   | semicolon    | ":"       |
| STAR        | asterisk     | "\*"     | EQ          | equals       | "\="     |
| LBRACKET    | left bracket | "["       | LINE_NUMBER | line number  | "\n"     |
| RBRACKET    | right bracket | "]" | ENUM        | enum         | "\="     |
| DEFAULT     | default      | string    |             |              |            |

Some tokens are special token, such as "#" means the start of a line and the line number is behind it. Some letters such as letter "c" can not be a token by itself, it must be combined with subsequent letters to become a word like "concrete". Taking line 181 as an example, the line that shows a beam will be separated into many tokens corresponding to Table 1, such as the entity name, the line number and many special tokens. Each of the token has a type. The form is shown in Table 2.

**Table 2. Separate line 181 into Tokens.**

Original format: 

```
#181= IFCBEAM('3Qhc8zUFf4wRXe181394HG',#41,'300 x 600mm','$','300 x 600mm',#148,#177,'194956');
```

Token Sequence: 

```
{"#181": '="', 'IFCBEAM': '"', '3Qhc8zUFf4wRXe181394HG': '"', '"', '"#41', '"', '"300 x 600mm"', '"', '"$"', '"', '"300 x 600mm"', '"', '"#148', '"', '"#177', '"', '"194956', '"'}
```

The procedure of getting token sequence by Tokenizer is outlined in Algorithm 1. The logic is that when encountering a special characters shown in Table 1, the information at the current byte stream is intercepted, and the Token type and content are assigned. If the current byte stream is a normal character, the byte stream will read the information backward normally. The class Token has two main attributes, type and image. Type is the TokenType shown in Table 1 and image means the value of the token. Each time the method is called, a new token is obtained.

**Algorithm 1 getNextToken**

**Input**: NULL

**Output**: Next Token

1. Create nextT as new empty Token
2. while true do
3.   let curChar as OutputStreamReader.nextChar

```
4. while curChar < 'r' do //invalid character
5. curChar = FileStreamReader.nextChar
6. switch curChar
7. case 0x03 : nextT.type = EOF
8. case ' ' : nextT.type = SPACE
9. case '$' : nextT.type = DOLLAR
10. case '
' : nextT.type = EOL
11. case '(' : nextT.type = LPAREN
12. case ')' : nextT.type = RPAREN
13. case '*' : nextT.type = STAR
14. case ',' : nextT.type = COMMA
15. case '!' : nextT.type = DOT
16. more case in upper table
17. default :
18. while OutStreamReader.nextChar is not Special characters do
19. nextT.image.append(OutputStreamReader.nextChar)
20. nextT.type = STRING
21. end switch
22. fill nextT by OutputStreamReader
23. return nextT

The Token sequence splited by lexical analysis will be passed to the Syntactic Parser for syntax judgment and logical handling.

3.2. Syntactic Parser
Different types of Tokens decomposed by Tokenizer have different meanings. Different tokens are analyzed by Syntactic Parser, and new IFC entity will be created and attribute assignment will be performed. The procedure of creating new Entity by Syntactic Parser is outlined in Algorithm 2.

Each line can generate an entity, so "(" indicates the beginning of a line, a token after that indicates the line number, and the EOF character indicates the end of a line. Then, the parsing of the next line is started, and the whole parsing process is a cycle. The type of entity is determined by a token after ",=" and its type is set to the image of this token.

Algorithm2 parseFile
Input: FilePath
Output: IfcModel
1. Create OutputStreamReader from FilePath
2. while Token var == GetNextToken() && var.type != EOF do
3. Create new NodeObject node
4. if var.word is start with "#" then
5. node.lineNumber = var.word[1::]
6. getNextToken() // skip Token "="
7. if tmp = getNextToken().type == LPAREN then
8. node.ClassName = tmp.image
9. getNextToken() // skip Token "("
10. createNodeObject (node)
11. else throw FileFormatException
12. end if
13. buildModel(IfcModel, node)
14. end if
15. return IfcModel
Attribute resolution starts with token "(" and end with token ")". Each "," separates an attribute. For EXPRESS language, there are four types of attributes: basic type, enumeration type, entity type and selection type. For the basic type and enumeration type, the value of the attribute can be extracted directly. For the entity type, since the entity type referenced by this line may have not been created, only save the line number at this step, and then processed in the third step (Entity Factory). For the selected type, if "+" or "+" appears, it will be given a null value. If a collection appears in an attribute, continue to loop through the attributes and add them into the list. The procedure of attribute assignment by Syntacti Parser is outlined in Algorithm 3.

The resolved entity type and attribute do not directly generate the target entity and assign values, but are added to a `NodeObject`. NodeObject is the top-level parent class of each IFC entity class. It has only two attributes: line number and a list contains storage attribute, each resolved attribute value is added to this list but not do the assignment. The purpose to delay the process of creating entity to the third step so as to carry out multi thread for entity creation and assignment, and accelerate the parsing speed. In this step, files are parsed from the beginning to the end, so it can only be used with a single thread. Program only need to extract the attributes from the file now.

```
Algorithm3 createNodeObject
Input: NodeObject
Output: NodeObject
1. flag = true
2. while flag do
3. var = getNextToken()
4. switch var.kind
5. case DOLLAR : NodeObject.addParameter(null)
6. case STAR : NodeObject.addParameter(null)
7. case LINE_NUMBER : NodeObject.addParameter(var.word[1::])
8. case STRING : NodeObject.addParameter(var.word[1:: len - 1]) //skip quotation
9. case SEMICOLON : flag = false //end of the line
10. case LPAREN :
11. create new empty List list, flagR = false
12. NodeObject.addParameter(list)
13. while !flagR do
14. switch nextT = getNextToken()
15. case RPAREN : flagR = true
16. case COMMA : break
17. default : NodeObject.addParameter(nextT)
18. end switch
19. case ENUMERATION : NodeObject.addParameter(var.word[1:: len - 1])
20. case RPAREN : break
21. case COMMA : break
22. default : break
23. end switch
24. return NodeObject
```

3.3. Entity Factory
Each line in the IFC file has been parsed and extracted into NodeObject. The specific IFC entity is generated by the construction factory called ObjectFactory (ObjectFactory inherits from HashMap, where key is the name and value is the factory class called IfcxxBuilder) using the factory pattern. There are all IfcxxBuilder factory classes of IFC entities in ObjectFactory. The design pattern of IfcxxBuilder is shown in Figure 2. IfcxxBuilder extracts the data from the List attribute in NodeObject, and calls the initialization method of IFC entity. In this way, the attribute of entity will be assigned and the entity will be formally created.
Because the number of IFC entities generated by a large BIM model reaches tens of thousands, if only one single thread processing is used, it will take too much time and can not make full use of the multi CPU and multi-core resources of modern computers. Therefore, the parser uses the ThreadPool to optimize, and multiple threads are used to generate the entity of the construction factory at the same time. Due to this program is a CPU intensive calculation, the number of thread pool is set to the CPU core number of the server + 1.

When parsing IFC files, if not handled properly, the problem of circular dependency will appear. Because in the IFC Standard, the attributes of a class may depend on another class. For example, IfcBeam(#181) depends on IfcObjectPlacement(#148). If the IfcObjectPlacement entity is not created when assigning the IfcBeam attribute, a null pointer exception will occur. In this research, in the second stage, the file is parsed by single thread, and the entity is created without attributes. In the third stage, the construction factory is used to formally assign values by multithreading. This method not only solves the efficiency problem of single thread, but also solves the problem of circular dependence.

Figure 2. design pattern of IfcxxBuilder.

4. Persistence Module
When persisting the BIM model data into the database, the most frequently used method is to use the export function of Revit to import the building model directly into SQL Server database. but this persistence method has many defects: first of all, SQL Server has limited storage capacity and reading speed, which is not conducive to the subsequent lightweight data transmission. At this stage, MySQL data is more suitable for Internet applications; secondly, there is data loss phenomenon in the persistent data through Revit, such as the size and shape of components[8]; more importantly, there are a variety of BIM application software, and Revit is only one of them, which has poor portability.

Mybatis framework is used for persisting IFC entity to MySQL database. MyBatis can map the original Java POJO (plain old Java objects) as records in the database through simple XML files or annotations. The main idea is the establishment of mapping relationship. The mapping relationship of data types between EXPRESS language and MySQL database is shown in Table 3.

| EXPRESS DATATYPE | MySQL DATATYPE          |
|------------------|-------------------------|
| String           | VARCHAR                 |
| Integer          | INT                     |
| Double           | DOUBLE                  |
| Boolean          | TINYINT                 |
| Enum             | VARCHAR                 |
| LIST/SET         | VARCHAR                 |
| Entity           | TableName + LineNumber  |
The design of the table structure is divided into two parts: inversion attribute and non inversion attribute. The non inversion attribute can be mapped one-to-one directly. The attribute name is the field name, and an entity may correspond to multiple inversion attributes. Therefore, the non inversion attribute is designed as a one to many table relationship.

Taking IfcBeam as an example, there are three entity type attributes in IfcBeam: OwnHistory(Represents history editing), ObjectPlacement(Represents the position), and Representation(Represents the shape). For a beam, there is only one shape and one position, so it is a one-to-one relationship. However, for reverse attributes, such as #378 is an attribute set, and the attribute name is material and decoration(UTF-8 encoding), and the corresponding single value attribute is #341 (structural material: concrete - cast in place concrete). 

A beam has a variety of properties, including material, reinforcement, strength and other information, so the table of beam and attribute set belong to one to many relationship. Table structure and table relationship of IfcBeam are shown in Figure 3.

Figure 3. table structure and table relationship of IfcBeam.

5. Conclusion
This Research utilize Java language to extract the information from IFC files directly, instead of the secondary development based on Revit, which is more applicable. The whole parser can be divided into three core parts: Tokenizer, Syntactic Parser and Entity Factory. It obtains the complete information in BIM and turn it into Java entity. In the syntactic parsing step, only create entities, but not assign attribute. Using multi thread, attribute assignment is completed by the entity factory. It will not only make full use of multi-core CPU to speed up the processing, but also solve the problem of circular dependence. In order to save the information into the database for subsequent work, the research designs the table structure and table relationship according to IFC standard, and uses MyBatis framework to store the entity parsed by Java into a powerful database MySQL.
The IFCParser makes it easier for engineers who don't know IFC standard to get BIM information and store it into their own server, subsequently, they can focus more on solving problems, and do not need to consider the underlying implementation.

Acknowledgments
This paper was supported by Projects of National Natural Science Foundation of China (Grant No.52078373).

References
[1] Liu X, Xie N and Tang K. (2016) Lightweighting for Web3D visualization of large-scale BIM scenes in real-time. Graphical Models, 88: 40-56.
[2] Zhao F. (2018) Lightweight Research and Implementation on BIM Models of Ancient Architecture Based on WebGL. Master's Theses of Xi’an University of Architecture and Technology.
[3] Shang D. (2019) Research and Implementation of Light Building Model on Web. Master's Theses of Hebei University of Science and Technology.
[4] He G. (2011) The Three Pillars of BIM. Journal of Information Technology in Civil Engineering and Architecture, 3: 108-116.
[5] Afsari K, Eastman C and Castro D. (2017) JavaScript Object Notation (JSON) data serialization for IFC schema in web-based BIM data exchange. Automation in Construction, 77: 24-51.
[6] Guo M. (2016) Research and Design of Data Collaboration Module of BIM System. Master's Theses of Shanghai Jiao Tong University.
[7] Lu N and Ma Z. (2012) Comparison of managing IFC data using object-oriented and relational databases. Journal of Tsinghua University, 52: 836-842.
[8] Zhou Y, Guo H and Luo Z. (2018) Research on automatic mapping method from IFC data to relational database. In: The 4th National BIM Conference. Hefei. pp. 311-316.
[9] Zhang D, Wei Z, Yang Y. (2014) Research on Lightweight MVC Framework Based on Spring MVC and Mybatis. In: Sixth International Symposium on Computational Intelligence & Design. NW Washington. pp. 350-353.