www.itcon.org - Journal of Information Technology in Construction - ISSN 1874-4753

INTERPRETED INFORMATION EXCHANGE: IMPLEMENTATION POINT OF VIEW

SUBMITTED: December 2017
REVISED: August 2018
PUBLISHED: February 2020 at https://www.itcon.org/2020/8
EDITOR: Turk Ž.
DOI: 10.36680/j.itcon.2020.008

Issa J. Ramaji, Ph.D., P.E., D.B.I.A., Assistant Professor
Director of Contruction Emerging Technology Lab
Roger Williams University
iramaji@rwu.edu

Ali M. Memari, Ph.D., P.E.,
Pennsylvania State University;
memari@engr.psu.edu

SUMMARY: *Engineering design is one of the most in-demand Building Information Modelling (BIM) uses. Due to efforts required for modifying and preparing an imported model for analysis, the difficulty-to-benefit ratio is low in this BIM use. These preparations are more geared toward modifying an imported model based on the designer's interpretation of the building information model and including additional engineering information. Automating the interpretation and model transformation process can significantly facilitate information exchanges. The Interpreted Information Exchange (IIE) concept is developed in this study for such automation during model exchanges. A platform is developed and presented in this paper for implementation of this concept. The platform contains procedures and functionalities required for inputting, processing, and exporting IFC information models through automated interpretation processes that implement IIE concept. The platform is especially formulated to be schema-independent to make it compatible with any standard or custom-defined version of IFC.*

KEYWORDS*: Building Information Modelling (BIM), Interpreted Information Exchange (IIE), Engineering Analysis, Model View Definition (MVD), Industry Foundation Classes (IFC).*

REFERENCE: *Issa J. Ramaji, Ali M. Memari (2020). Interpreted information exchange: implementation point of view. Journal of Information Technology in Construction (ITcon), Vol. 25, pg. 123-139, DOI: 10.36680/j.itcon.2020.008*

1. INTRODUCTION

Building information modelling (BIM) emerged almost two decades ago as a solution for information management of building construction projects. Over the past decade, BIM adoption in the industry has significantly increased as its numerous potential benefits have been achieved by improvement of the supporting technologies such as tools and platforms (Gallaher *et al.*, 2004; Fallon and Palmer, 2007; McGraw Hill Construction, 2012; AEC Magazine, 2013). The rich information contained in building information models can be used for several purposes (Lee, Eastman and Solihin, 2016). CIC Research Group (2013) identified 15 different BIM Uses from early stages of design to operation and maintenance phases. Automating the engineering modelling process for creation of high quality and accurate analytical simulation is one of the benefits of engineering design BIM Use that can eliminate design errors and reduce design cost and time (Sudarsan *et al.*, 2005; Gane and Haymaker, 2007; Sacks and Barak, 2008; Alwisy, Al-Hussein and Al-Jibouri, 2012; Liu *et al.*, 2013). Structural design/analysis and energy performance analysis are two examples for such uses that are being widely practiced in the industry. Although some BIM authoring software are capable of performing engineering analysis to some extent, they are not usually as sophisticated as stand-alone engineering analysis software products. Hence, many efforts are made to develop links between different BIM software products and engineering analysis tools (Becerik-Gerber and Rice, 2010; Guzmán and Zhu, 2014). These efforts are mostly focused on information translation from BIM physical models to analytical models as the key process for incorporating this BIM Use. Usually, these translators and information extractors can be found in three different forms: 1) entirely separate from base and self-sustaining, 2) a sub-routine within existing BIM authoring or engineering discipline-specific software such as Revit or SAP2000, or 3) using a server-based platform like BIMserver for data management along with query functions such as BIM Query Language (BIMQL) for extraction and manipulation of the required information. An example of such engineering analysis-related information translators is the web-based integration tool developed by Zhang et al. (2014) for bi-directional information exchanges between IFC BIM files and several common proprietary structural design file formats. Another example is a tool developed by Guzmán and Zhu (2014) to generate energy performance analysis models directly from building information models. Still another example is the use of Java programming language for partial information extraction from IFC files by Zhang and Issa (2013). Won et al. (2013) developed a tool for similar purpose while it uses an algorithm independent from the IFC schema data structure. Ramaji et al. (2016) created an add-on for the BIMserver platform to import architectural information model in IFC format and export the energy analysis model in native format of Energy Plus software. Other examples are the tools and platforms developed by Hassanien Serror et al. (2008), Wang et al. (2013), Nepal et al. (2012), Polter et al. (2014), Qin et al. (2011), and Liu et al. (2010) for different information transformation purposes.

According to Bazjanac (2008), to achieve an efficient information exchange for engineering analysis BIM Use, the information should be transformed and enriched before being used. Although Industry Foundation Classes (IFC) is the open file format for BIM information exchange, the current way of using this language for engineering analysis BIM Use is error-prone and ad hoc. Some of the current common issues in this BIM Use include the following: discontinuity between connected elements, lack of required information related to material and engineering properties of elements, inaccurate representation of elements, and missed information due to interoperability errors. Model transformations required for engineering design BIM Use include filtering, translating, and adding new information to the model (Bazjanac and Kiviniemi, 2007). Manual information transformation is tedious and in most cases inefficient, so many designers prefer to create the model from scratch instead of using the building information model. To address this issue, this study has developed a mechanism for documentation and automation of the interpretation process called Interpreted Information Exchange (IIE). In this process, an information model in IFC format is converted to an interpreted equivalent analytical model again in the IFC language, while having the enriching information added. In such workflow, where both input and output information are represented in IFC format, an open interpretation system would result, which can interact with a large variety of BIM authoring and engineering tools.

An IFC file can be easily opened in a text editor tool and the needed lines simply copied and pasted into another file. Although the files are somehow human readable and there is even room for instance manipulation, this is still not enough to make the resultant file useful. The file needs to consider interconnectivity between instances and the schema's constrains/rule sets. Because of such requisites, extraction, manipulation, and interpretation of BIM information need to be carried out in a controlled environment. To do so, a platform is developed and presented in this paper for extraction/Interpretation of information from building information models and implementation of the IIE concept. This platform offers functionalities required for reading an IFC file, extracting required

information, interpreting desired information, and generating an output IFC file for semantic exchange of the interpreted information. In other words, the platform provides several procedures and programing functions essentially required by a tool developer for programming an automated IIE process to be used for an engineering purpose. In such framework, the platform is the core while these tools would be extensions or add-ons, which are called interpretation engines. Such extensions/add-ons input information from the parsed instances of the input file and generate instances related to the interpreted information. The C++ language is used for development of this platform as it is one of the most powerful programming languages and is compatible with different computer operating systems.

In the following sections, first, the IFC format along with the structure of related files and schemas are explained as they are essential in the proposed platform. Next, the IIE concept and its documentation methodology are briefly discussed, followed by introduction of the overall workflow and individual modules of the developed platform, each devoted to a certain functionality. This paper also discusses the communication framework and the integration among different modules of the platform.

2. INDUSTRY FOUNDATION CLASSES

With improvements in manufacturing industry, the products have become more complex with a clear need for more organized product information model. Product modelling is a key element for implementation of advanced technologies in manufacturing (Zhao and Liu, 2008). As products employ such technologies, more experts and engineers need to be involved in the projects to address the need for resilient and robust information exchanges. In the 1980s, International Standard Organization (ISO) started an effort to address this need by developing a standardized product data model supplemented with a Standard for the Exchange of the Product model data (STEP) (Spiby, 1992). This product information model is developed in Express language (Pratt, 2001; Zhao and Liu, 2008). By introducing STEP in 1992, it became the main approach for information modelling in a wide range of industries such as mechanical and electrical systems, ship building process, and furniture manufacturing to name few (Pratt, 2005). In 1994, Industry Alliance for Interoperability (IAI) that later changed the name to International Alliance for Interoperability and is now called buildingSMART International (bSI) started to use STEP and Express language to develop a standard non-proprietary information model for the whole lifecycle of building construction (Laakso and Kiviniemi, 2012). The result of this efforts is Industry Foundation Classes (IFC) data model.

All definitions of information units developed for building industry along with relevant relationships are written in the IFC schema file using Express language. According to Eastman (1999), schema is a unit that defines Universe of Discourse (UoD) and declares objects associated with their dependencies and purposes. UoD is defined by Halpin and Morgan (2010) as the application area from which an information model is being developed. IFC files use the entities defined in schema for declaring project objects and related information. In other words, IFC files represent project-specific information in the data structure defined in the IFC schema. Both IFC files and schemas are stored in text-based file formats, respectively, with “ifc” and “exp” file extensions. IFC schema gives meaning to the IFC file, without which IFC files will be some meaningless lines of text.

Two types of entities are defined in the IFC schema: Object and Links. Objects can be either physical or non-physical components or properties such as beams, walls, project’s owner, time, and dimensions used in the building industry. Since Express is not a fully object-oriented language, STEP lacks links that connects objects together. This problem is solved in Express by defining some entities that play the role of links (also known as Relations), which are the entities that connect Object entities together. An example is IfcRelAggregates that is used for defining an object composed of many other objects such as ramp that is made of railings, flight, and slab. Another example is IfcRelAssigns that is used for assigning two objects together like assigning a client to a project.

IFC schema is made up of several blocks, each defining one entity. In an entity definition block, four important types of definitions could be found including super-type, sub-types, attributes, and rules. As mentioned before, STEP uses the object-oriented concept, thereby each entity can be a sub-type of another entity, and in this case inherits all attributes of its super-type. At the beginning of the block, super-type and all sub-types of the entity, if there is any, are declared. Then, immediately after sub-type and super-type declarations, specific attributes of the entity that are not inherited from its super-type are defined. The attributes of an entity in STEP can be classified in three categories that are introduced in the following:

- **Explicit:** These attributes are information units that characterize the properties of the entity. An explicit attribute can be a single value like an integer number, a set of values with the same type, or a reference to other Object entities.
- **Derived:** This type of attribute is not loaded as data like Explicit attributes, but derived from Explicit attributes of that entity.
- **Inverse:** These attributes are like connectors that can be used to relate two or more Object entities together using Link entities.

The last part in the block is the rule definition that is used in the schema to define constrains. Rules are defined in the form of equations and are shown after the "WHERE" line in the entity definition block. The value of the rules can be true, which means it is following the rule; false, that means it is violating the constraint; and unknown, which means some attributes are missing.

IFC files are written in a digital text-based format for modelling building information. Files in this format are made of many instances, each containing an information unit or defining a relationship between units. Each information unit or relationship is defined to be an object in the IFC data structure, and each instance constructs one of these objects, which are defined in the IFC schema as entities.

3. INTERPRETED INFORMATION EXCHANGE

Unless the required interpretation has been automated in the process of information exchange, incorporating BIM for engineering modelling purposes is very tedious and time consuming. A methodology is developed in this study for documentation and standardization of these interpretations using IFC language (Ramaji and Memari, 2016). In this methodology named Interpreted Information Exchange (IIE), firstly the whole interpreted information exchange is divided in several exchange Units (eUs). Two types of eUs were identified to be included in each automated IIE process: directly-exchanged Units (dUs), and interpreted Units (iUs). The dUs are responsible for transferring information from the input model to the output model without any changes, whereas iUs transform a set of information units to another set. Three types of information interpretation were identified to be required for transformation of physical models to analytical models: enrichment, representation transformation, and simplification. In the enrichment process, additional information required for engineering simulation of the building are included in the model; adding mechanical properties of structural materials to the model based on the materials' names is an example of this type of transformation. Representation transformation may be carried out in an IIE to deliver information in a more desirable format; an example is transforming the extruded area representation of a linear element to its equivalent topological representation. Simplification that is generally the most complicated type of interpretation consists of reducing a complex building component or any other complex information unit to its equivalent simplified analytical from; interpretation of the equivalent structural shell area element from a cold-formed steel wall in a building information model is an example of simplifying transformation. According to the developed methodology, the following five steps are needed in order to design an IIE for automated generation of an engineering analytical model directly from building information models:

- **Categorizing the information units into direct-exchange and interpretation units:** The first step recognizes the required information in the analytical model that could be exchanged directly from a building information model, as well as the ones that are not included in the building information model and need to be interpreted. In other words, this step divides the information units contained in the engineering model in two categories: directly-exchanged and interpreted information.
- **Defining eUs and mapping them to the related units in the building information model:** This step classifies all the recognized information units from the previous step into several eUs and maps the interpreted information units to the related information in the building information model. In other words, the information in the building information model upon which the interpreted information could be extracted are specified. For better management of the IIE, the eUs should either transfer interpreted (iUs) or directly-exchanged (dUs) information units. It is preferred to have only one dU in each IIE, whereas several iUs may be considered in an IIE, each carrying out one specific interpretation process. The result of this step is specification of inputs and outputs for each eU.
- **Binding the iUs to the IFC file format data structure:** Binding the input and output of eUs to the IFC schema is the focus of this step. The outcome of this step specifies the IFC file instances

that should be queried from the input file, the instances that should be directly passed to the output file, and the instances that should be created based on representing the interpreted information. Binding these eUs to IFC data model makes it compatible with large variety of BIM tools as IFC is an open standard file format for building information models.

- **Design a translation mechanism for interpretation:** This step designs the algorithm for interpreting the outputs of iUs from their inputs. In other words, this step specifies how the input and output of each iU are connected. As interpretation may need input from external references other than the input file, the source of these references should also be specified in this step. An example is the thermal resistance of a layer of insulation that is required for energy performance evaluation analysis of a building; such information can be obtained from the product specification table published by the manufacturer.
- **Testing the designed IIE:** The last step is testing the designed IIE by means of case studies to make sure that the defined interpretation units along with the interpretation algorithm are designed properly. To investigate the interoperability of the platform, it is preferred to run the case studies for information exchange between different combinations of BIM authoring and engineering analysis tools, as each export IFC file is slightly different than others.

Once the first four steps of the IIE are designed, the platform presented in this paper can be used for testing and implementation of the automated interpretation process. Since IFC is the sole open standard file format for building information models, an interoperable interpretation platform needs to import IFC files as input information; it should also be able to parse the input file IFC instances to the related IFC objects. Since export file format is also IFC, the platform should be capable of transforming interpreting objects into IFC instances to be written in the output file. As the workflow of the interpretation platform in FIG. 1 illustrates, in addition to the functionalities required for reading and writing IFC file, the platform should also be capable of querying information from the objects generated from input IFC files as well as generating IFC objects related to the interpreted information. The querying functions of the platform feed the required information to both iUs and dU. In brief, reading IFC file to equivalent IFC objects, querying required information from the read IFC objects, generating new IFC objects, and writing the IFC objects into the output file are the four main needed functionalities for an interoperable platform. These four functionalities could be coupled with any interpretation engine for automating an interpretation process. Interpretation engines use information provided by the query functions along with several internally defined rule-sets to generate interpreting information. This is followed by passing the information through the IFC object generator functions to create the interpreted IFC objects. The last step would be generation of the output file by writing the directly-passed and interpreted IFC objects in the form of IFC instances. As in any process, quality of output models is relied upon the quality of input models. Therefore, an additional module is also added to these four functions in order to check the quality of input models based on the imported schema. In the following sections, each of these platform components is discussed in a separate section.

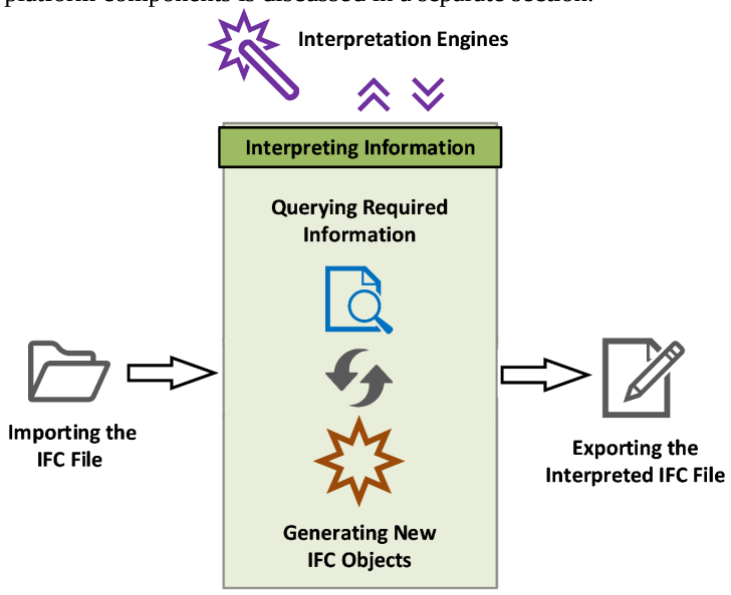

FIG. 1: Workflow of the interpretation platform

4. IMPORTING THE IFC FILE

Reading IFC files involves converting text-based instances included in an IFC file to the related digital IFC objects. An example is converting an "IfcBeam" instance shown in FIG. 2 to an object that contains several attributes as shown at the bottom of this figure. IFC files do not include any information about the meaning of the values listed in the contained instances. However, the meaning of the values in an IFC file can be interpreted based on the IFC schema being referenced in the header of the file. Therefore, parsing an IFC file has two steps: 1) reading instances from the IFC file and 2) assigning meaning to the read values. In the following, each of these steps is discussed in more detail.

```
#162= IFCBEAM('0YfVDXovP6ixekvUyl$0s1',#41,'W-Wide-
Flange:W12X26:508639',$,'W-Wide Flange:W12X26:164481',#129,#158,'508639');
```

#162 - IFCBEAM

Items	Values
GlobalId	'0YfVDXovP6ixekvUyl$0s1'
OwnerHistory	#41
Name	'W-Wide Flange:W12X26:508639'
Description	$
ObjectType	'W-Wide Flange:W12X26:164481'
ObjectPlacement	#129
Representation	#158
Tag	'508639'

FIG. 2: An example for the parsed IFC object

4.1 Reading IFC File Instances

As shown in FIG. 3, each instance contains three main parts including index, concept name, and item list. Each instance starts with an index, followed by a concept name and ends with a list of items. All indexes start with the "#" symbol and indicate the unique identification number of each instance in the IFC file; the concept name is the name of the entity from IFC schema that the instance is constructing; and the items are the attributes contained in the instance. Items included in an instance are separated by the "," symbol in the list, which include only the values of items while the name and meaning of these items could be inferred from the related schema.

FIG. 3: Anatomy of IFC instances

In majority of available platforms for reading IFC files, a class is defined for each of the entities defined in the IFC schema. These classes are predefined in the library of these platforms and a copy of each is constructed per related instances contained in the imported IFC file. This approach makes the platform highly dependent on the schema it is designed for, and makes maintenance of the platform difficult for capturing schema's version changes. A new approach is taken in this research for development of the IFC parsing module. In this approach, one class is defined to support all instances with different concept names instead of defining one class for each.

As depicted in FIG. 4, the class defined for modelling IFC instances includes six variables and two functions. The "index" variable stores the index of the instance. The "indexNum" stores the index without the "#" symbol in a numerical format to be used for sorting the instance list. Sorting instance list based on a numerical variable significantly speeds up the querying process compared to searching based on the string-type index. The "concept" variable stores the concept name of the entity being constructed in the instance. The "items" variable stores the

value of the instance's items, which may be a single value or a list of sub-items. For example, the list of building elements contained in a building story is written as one item in an "IfcRelContainedInSpatialStructure" instance. This type of item is named list-item in this paper. Since each of these sub-items may be independently queried, the "itemList" variable is defined to store the sub-items individually. The "raw" is the last variable that keeps the raw un-parsed text-based value of the instance. This variable is used in the first and last steps of reading and writing stages of the platform's workflow. The two functions included in this class are defined for parsing information to the variables. The "Instance_Analysis" function parses information from the raw form of the input instance, while the "listItem_Analysis" function breaks down the list-item variables into several sub-items and stores them in the "listItems" array.

```
class Instance {

public:
    string index;
    unsigned long int indexNum;
    string concept;
    vecString items;
    vector<vecString> listItem;
    string raw;

    //------------------------------------------------------

    void Instance_Analysis(string raw) { ... }

    //------------------------------------------------------

    vecString listItem_Analysis(string listItem) { ... }

    //------------------------------------------------------

};
```

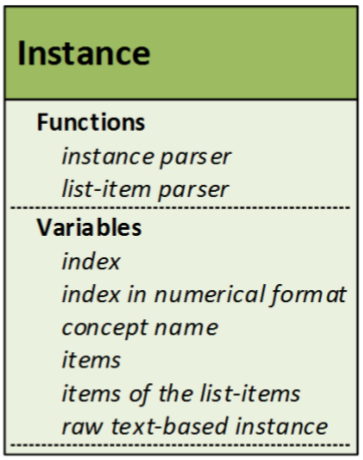

FIG. 4: Class definition for modelling IFC instances

4.2 Reading IFC Schema Entities

The IFC schema defines several entities related to the concepts being used in the building construction industry. In the data schema of IFC files, both relationship and concepts are considered to be entities, which are separately defined in entity blocks in the IFC schema along with their inheritance relationships. Reading an IFC schema mainly consist of parsing information contained in these blocks, with anatomy of an entity block shown in FIG. 5.

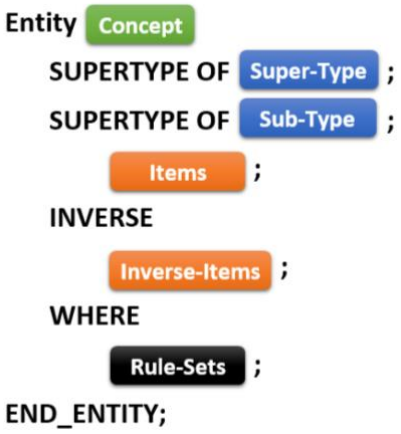

FIG. 5: Anatomy of IFC entity blocks

Similar to the instance parsing module, the same single-class approach is taken for modelling the schema's entities. As shown in FIG. 6, this class includes seven variables. The "concept" variable stores the name of the concept, while the "supertype" and the "subtype" variables keep information related to the concept name of the parent (super-type) and children (subtypes) of the entity. The "abstract" variable is a logical variable that can be assigned true or false, and specifies whether or not the concept is an abstract object. The "items" variable stores entity's list of attributes, which contains both the specific attributes of the concept and the attributes inherited from its super-type. These items are the ones whose values appear in IFC instances. Next, we have the variable "invItems" that stores Inverse attributes of the entity. These attributes specify the types of relationships that the concept may have with other concepts and are used for checking the import file against schema. The last variable is the "block" that is read in the first step of the schema importing process and keeps the raw text-based definition of the schema. The "SchEntity_Analysis" function is used for parsing information from the block variable to other entity's variables.

```
class SchemaEntity {
public:
    string concept;
    bool abstract;
    string superType;
    vector<string> subType;
    vector<Item> Items;
    vector<InvItem> InvItems;
    vector<string> block;

    //-----------------------------------------------------

    void SchEntity_Analysis(vector<string> block) { ... }

    //-----------------------------------------------------

};
```

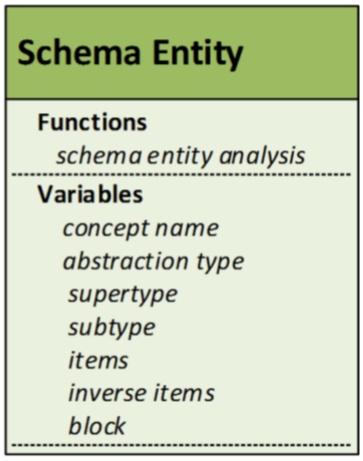

FIG. 6: Class definition for modelling IFC schema entities

Concept names are the connections between the schema entities and the IFC file instances. In other words, each instance is an ally of the IFC entities that has the same concept name. As a result, their "items" variables are allies of one another. Hence, the name and meaning of each attribute of the instance could be inferred from the items of related concept entity. This method of parsing information makes the platform independent from the version of the schema and the IFC file. As far as the version of the IFC file is consistent with the imported schema and the IFC file does not violate the schema's constraints, the platform can parse the information.

5. CHECKING THE INPUT FILE

Reading, writing and querying functions of the proposed platform are independent from the schema, whereas instance generator functions and interpretation engines are closely dependent on the IFC schema. On the other hand, the output IFC file is a mix of instances generated based on the interpreted information and the instances passed directly from the input file. Since the interpreted instances are generated based on the schema, instances of the input file should observe the schema to have a homogeneous output file that is consistence with the schema.

Malfunctioning of interpretation engines is another problem that can be caused due to inconsistency of the input file with the schema. Places where certain information could be found in the input instances are based on a series of assumption in the interpretation engine for the. For example, the length of a beam element in an "IfcBeam" instance could be extracted from the "Representation" item that is the seventh item in the item list. Based on the schema, "IfcProductRepresentation" is the concept that should be cited in this item. Hence, an interpretation engine that needs to extract the length of an "IfcBeam" object may be designed to find the information in an "IfcProductRepresentation" instance or any of its subtypes. In this case, if an IFC concept that is not of the same type or any of its sub-types appears in this item, the engine cannot find the desired information, and as a result, it may misinterpret the targeted information.

To guarantee uniformity in output files generated by this platform, a module is included in this platform to check the input file against schema before running interpretation engines. Several possible violations that the input IFC file may have are identified, and a check function is accordingly designed for each. In the following, these potential inconsistencies are discussed along with the workflow of checking functions:

- **Validity of the concept:** This checks the existence of the instance's concept in the version of the IFC schema to which the file is related. This check can be done simply by searching for the concept name in the entity list of the imported schema. The check passes if an entity with the same concept name exists in the schema.
- **Length of the item list:** This check is needed to see if the number of items included in the instance is consistent with the schema. This check counts the number of attributes specified in the related schema entity and compares it with the length of the instance's item list.
- **Item availability:** The purpose of this check is to see if all items that are not optional have values in the instance. To do so, the check function goes through all items mentioned in the related schema entity to determine if it is optional; if it is not, the function checks the value of the related item in the instance. If the item value is not equal to "$", it means that the item is not blank and the check passes.
- **Item cardinality:** This inspects all items of the instance, finds its list-items, and checks the cardinality of their sub-items. In other words, this function identifies the item-lists of the instance based on the schema and checks if their values are enclosed in prentices as it is a requirement for IFC files. This function then tests the number of the sub-items contained is checked against schema.
- **Referred instance:** This check is intended to see if the cited instance in an item observes the constraints of the schema. To do so, the check function goes over the instance item to determine if the cited instances match the IFC concept that is specified in the IFC schema for that specific item. If there is no match, the query function finds all sub-types of the referred concept in the schema and checks the referred instance's concept with this sub-type list. If the instance concept is available in this list, it means that the check passes and the item is following the schema's constraints in referring to other instances. If the item is a list-item, this check function would be executed for each of the sub-items separately.

These are the essential checks all IFC files should pass. The designed checking module in this platform queries the constraints directly from the schema being used for generation of the input IFC file. Hence, this module is independent from the version of the input IFC file. In addition to the constraints defined in the schema, there are some constraints that are specific to the Model View Definition (MVDs) followed by the input file. These constrains are more related to the representation of elements and relationships between different instances. Hence, the interpretation engines should be designed to test if the constraints of the used MVDs are observed in the input file, or at least filter the instances that violate these constraints.

6. INFORMATION QUERYING

Information querying in the imported file is one of the fundamental needs of any information platform. Query functions of the developed platform that are the links between input files and interpretation engines search the parsed information instead of the raw instances of the IFC file, and this significantly speeds up the querying process. Generally, the simplest form of information query in an IFC platform would be searching for a specific instance, where sorting and searching take place based on the numerical format of the instances' index. The reading functions are designed to sort the imported instances at the time of parsing input file. Since the objects are read sequentially, the Insertion Sort method is used for sorting the parsed instances, where combining reading and sorting processes by insertion of the read instances at their right sorted places speed up the sorting process. Searching for certain types of instances is another type of query. Some of the interpretation engines need to access a certain type of object. For example, all instances declaring columns contained in a certain story may be queried by an interpretation engine. In this type of query, the "concept" variable of the instances are inspected to see whether or not they match the searched instance type.

In majority of the instances, definition of the concepts is carried out by referring to other instances. Therefore, the relationships among different instances are key elements for extraction of the required information. Generally, the

relationships among IFC objects could be classified in two categories: essential dependency and relative dependency as discussed in the following.

- **Essential dependency:** Essential dependents of an instance are part of the information required for declaring an instance. These instances are referred to at the item list part of an instance, where referring takes place by mentioning the index of the referred instance. For example, in the IFC object shown in FIG. 7, the instance refers to instance "#158" for specifying representation of element. In this example, instance "#158" is an essential dependent for the "#162". Essential dependents of an instance can be categorized in two groups: immediate essential dependents and indirect essential dependents. Immediate dependents of an instance are the ones that are cited in its item list. In some cases the immediate essential dependents of an instance refer to other instances for their definition. Referring to other instances continues until none of the cited instances contains any essential dependent. All of these instances that are essential dependents of the instance and do not appear in the instance are called indirect essential dependents of the instance.
- **Relative** dependency: As mentioned in the previous section, there are some concepts in the IFC schema called Link that are to relate several objects to one another. In fact, they are the relationships in the object-oriented data structure of the building information model. There are different types of Link IFC concept defining different types of relationships, but they are all sub-types of the "IfcRelationship" concept. The names of all these concepts start with "IfcRel", thereby, instances related to these concepts could be queried by checking the first six characters of the instance "concept" variable. Assigning a property to an object, assigning several elements to the assembly they are aggregating, and assigning a containment relationship between the elements located in a space to the containing space are a few examples for application of these concepts. These relationships are the attributes that are referred to as the Inverse attributes of the IFC concepts in the schema. The instances that are of this kind and relate an instance to other instances are called the relative dependents of that instance. These dependents do not appear in the item list of an instance, but they cite the instance in their item list. In the example shown in FIG. 7, instance "#194" is a relative dependent for the instances "#162" and "#189", while they are both essential dependents of the instance "#194".

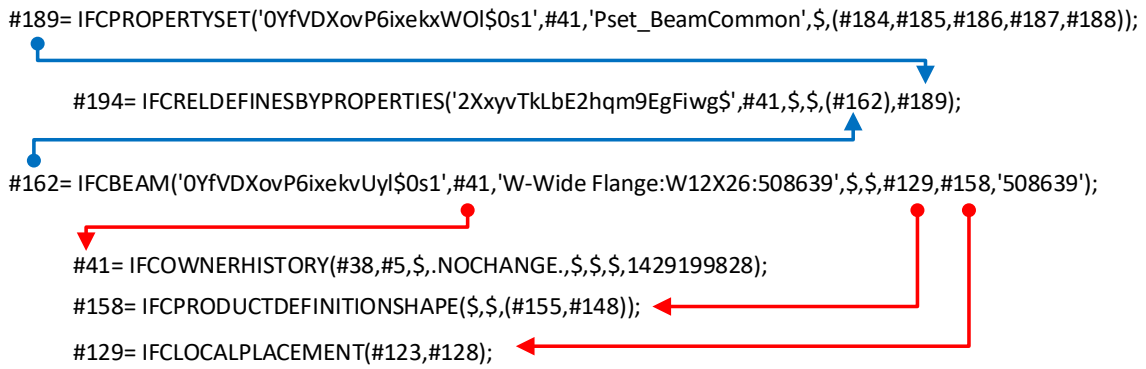

FIG. 7: Relative and essential dependents of an instance.

According to the above explanation, the five main types of query functions designed in this platform are as follows:

- **Finding an instance based on the index:** This function reads the index of an instance, transforms the index to a numerical format and searches the transformed index in the sorted instance object list.
- **Finding immediate essential dependent of an instance:** This function starts with an instance index, finds the instance in the list using the instance finder function, and then makes a list of all its immediate dependents by going through the instance's item list.
- **Finding essential dependent of an instance:** This function too starts with the index of an instance and runs the immediate dependent finder function recursively on the instance and its immediate essential dependents to find all its immediate and indirect essential dependents.

- **Finding relative dependent of an instance:** This function searches for relative dependents of an instance by using the instance's index. It goes through the item list of all IFC relationship objects and checks for any relations of the input instance to other objects. This function returns a list of relative dependents of the instance.
- **Finding instances with a certain concept name:** This function starts with a concept name and a list of IFC objects and searches all the objects to see if any of their "concept" variable is equal to the input concept name. This concept can be combined with any of the dependent finder functions to find instances of a certain type in the dependent list of an object. For example, it can be used to check if a certain piece of information such as a certain property set of representation type exist for an object.

In addition to the query functions required for data extraction from the input IFC file, some query functions are also defined for querying information from the IFC schema. In the following, their main four functions are discussed.

- **Finding schema entity related to an instance:** This function starts with the concept name of an instance and finds the related entity in the IFC schema. This is carried out by comparing the input concept name with the value of the "concept" variable of the schema entities.
- **Finding sub-types of a schema concept:** This function starts with a concept name and finds all its sub-types and sub-types of the found sub-types. The process continues to the point where none of the sub-types has any lower sub-type. This function is used to check whether a concept is sub-type of another concept at any level.
- **Finding attributes of a concept:** Each schema block defines only the attributes that are specific to the entity being defined, while the sub-types of the entity inherit these attributes. This function is designed to make a list of attributes that are either specific to the input concept or inherited form the concept's super-type.
- **Finding location of an attribute in a concept:** Since the instance class defined in this platform stores the instances' items regardless of their meaning, the capability of searching for a certain attribute is added to the platform. This function uses the instance's concept name and the name of the attribute that is considering; it then returns the location of the item in the instance. The locations are addressed by order of the item in the array of the item list.

7. GENERATING NEW IFC OBJECTS

Information interpreted by the interpretation engines should be written in the IFC language to make the output file importable by a wide range of analytical tools. Although the data structure of the interpretation engine may be different from the IFC schema, it should be transformed into the IFC data structure before being written as the instances of the IFC file. As interoperation engines feed interpreted information to the IFC instance generator functions that are designed for certain MVDs, the information should be transformed to the data structure of the MVDs before calling these functions. Usually all the information about an object cannot be written in only one instance, and as a result, a group of instances are required to write an information unit in the output file. Therefore, the process of generating instances for representation of an information unit could be divided into the following two steps: 1) generating individual raw instances, and 2) assigning values to the generated instances and linking them together. These two steps are discussed in the following and their connection is illustrated in Fig. 8.

7.1 Generating Raw Instances

The first step in generating an instance is constructing an instance object. An index should be assigned to the newly constructed instance to give it an identity and make it possible to refer to the instance. As the index of the object should be unique, and considering the fact that some instances of the input file would pass directly to be part of the output file, we cannot start the indexes from "#1". This problem is addressed in this platform by finding the largest index of the input file and starting from there for generating an index for the instance. Assigning the concept name to the instance is the next step. Since the input of the instance generator function is the concept name, the function just has to copy the input concept name into the "concept" variable.

The next step would be generating an array of blank variables to be filled by the items of the instance. The number of items may be different for each concept and could be found from the schema. To find the length of the array,

the instance generator function goes through the schema and counts the attributes the concept has or inherits from its super-type concept. These item variables would stay blank until being filled based on the interpreted information and connections of the instance with other instances. Since the blank items are shown by the "$" symbol in the IFC files, this symbol would be inserted for all these blank variables. This is for the case where no information is generated for an optional item during the interpretation process.

All the concepts that are sub-types of the "IfcRoot" concept need to have a Globally Unique Identifier (GUID). The GUIDs that IFC files use are not the standard 32 hexadecimal digits. Since development of the IFC language started in 1990s when the computer's small memory was a real issue for computational tasks, IFC language was designed to use a shorter GUID. To assign GUID to the related item variable, this platform first generates a standard 32-character standard GUID and then encodes it to a base-64 22-character using an algorithm developed by the buildingSMART (buildingSMART International, 2015).

Since the instances generated in this step refer to the schema for acquiring the required information for generation of instance, this function does not need to be maintained for the IFC schema changes. In addition, since in this step no relationships to other instances are assigned to the generated instances, this step is also independent from the MVDs that are used in interpretation engines for structuring the export information. The process of generating instances in this step along with its connection with the next step is depicted in FIG. 8.

7.2 Linking and Value Assignment

In this step, the values of the items would be assigned based on the relationships of the instances or results of the interpretations. The relationships are presented by citing instance index of the related instances in the item variables. Since IFC is a redundant file format, instances can be related to one another in many different ways. Therefore, the standardized structure of the MVD that is targeted by the interpretation units should be used for relating the generated instances. As a result, unlike the first step, this step is not model view-independent and the functions that are defined in this step are specific to the MVD they are designed for. In addition, since the meaning and order of the items used for defining connection are important in these functions, these are not independent from the version of the IFC and should be maintained for the version changes of the schema.

Since the essential dependents of each instance are required for its definition, the dependent instances should be constructed prior to the definition of the instance. Therefore, instance generation related to an information unit should start from the lowest level of the essential dependents and continue to the upper levels. The values of the items listed in these instances may be specified by referring to other instances or direct insertion of the value obtained from the related interpretation engine. Since interpretation engines may not follow the data model of the IFC schema for modelling the extracted information, the imported information may need to be processed before being inserted into the item variables. Furthermore, such information should be consistent with the structure of the observed MVD and provide enough information for generating instances and assigning values to their non-optional items.

There are some instances that are referred to by many other instances related to different information units. An example is the "IfcOwnerHistory" instance along with all its relative and essential dependents that specify the authors' information. In this example, since many information units may be created by an author, making a separate "IfcOwnerHistory" instance for each of these information units significantly increases the size of the export file. Default "IfcLocalPlacement" and "IfcGeometricRepresenationContext" instances are some of the other instances that are widely used by different instances. These types of instances are called public instances in this research, while others are called private instances. Since public instances are required at the time of generating private instances related to the interpreted information units, they have to be generated before transformation of the interpreted information to IFC instances. "IfcUnit" along with all its essential dependents that define units of different numerical values mentioned in instance items are another example of public instances that should be generated before private instances. FIG. 8 illustrates the connection of this step with the raw instance generation step, and shows the process of structuring the information by linking instances.

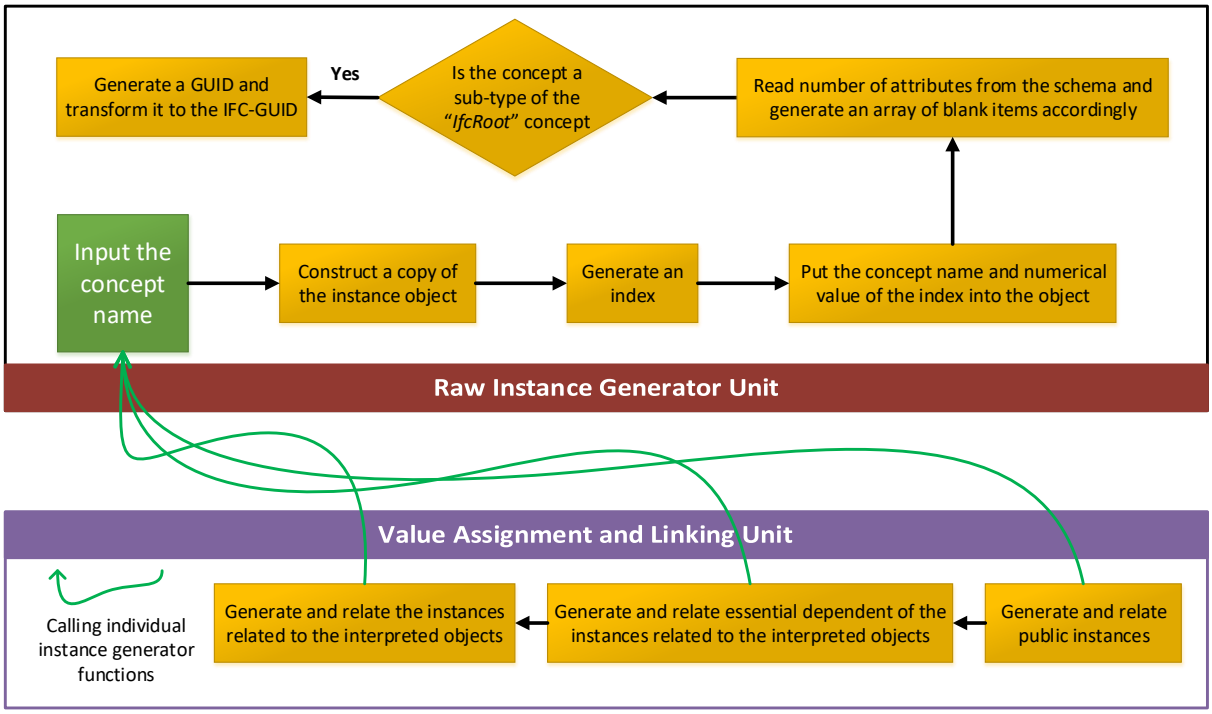

FIG. 8: Process breakdown of transforming an information unit into IFC instances

8. EXPORTING THE INTERPRETED IFC FILE

Writing IFC instance objects into the IFC file is the last step of the IIE process. Before writing the instances into the file, first the header of the file should be generated. The header block that is made up of nine lines specifies some general information about the file including the version of the used STEP language ISO standard, file name, file description, and the IFC schema version used for constructing instances. Seven out of these nine lines appear before instances, while the other two appear at the end of the file after the last instance. The last header line before the instance and the first line at the end of the file are, respectively, indicators of the beginning and end of the file instance section. These header lines are constant standard expressions and are, respectively, equal to "ENDSEC;" and "DATA;".

The next step in writing the IFC file is writing the instances that should pass from the input file directly to the output file. These instances are related to the directly-exchanged Unit (dU). Definition of an instance is impossible without defining its essential dependents. Therefore, the essential dependents finder query function could be used for finding instances in the input file that should pass to the output file along with instances pointed out in the dU. Since relative dependencies of dU's instances are not of concern, it is not required to pass the instances that are their relative dependents. Instances that pass from the input files along with the newly generated instances form the output file, hence the version of the input and output files should be identical. Otherwise, the instances related to dU should be parsed into the format of the new version and then be written into new instances accordingly.

The file writing process continues with writing the public instances, followed by writing the private instances. To write an instance, the instance objects should be transformed into the standard text based format. The transformation process is in reverse order of the instance parsing process. First, the sub-items should be attached to one another to make the equivalent list items enclosed between parentheses with commas between sub-items. Next, the items should be attached to one another to make the item part of the instance. By adding the concept name and instance's index to the beginning of the string, the raw text-based of the instance would be obtained. In the last step, these raw strings are written in the IFC file.

The process of generating the IFC file from the instance objects and input file is illustrated in FIG. 9.

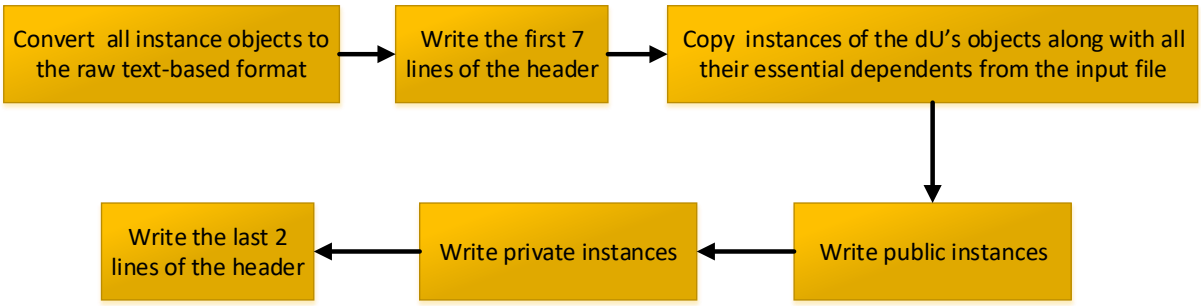

FIG. 9: Process breakdown of transforming an information unit into IFC instances

9. CASE STUDY

To validate the performance of the proposed platform and related algorithms, a case study is conducted by developing a tool for interpretation of analytical structural models directly from building information models (Ramaji and Memari, 2018). In this effort, an interpretation engine was developed on top of the platform to automate the transformation process. The engine inputs information from a building information model in IFC Design Transfer View (DTV) and creates a structural analysis model in IFC Structural View.

The engine contains one dU and four iUs. The dU exchanges basic building information such as project information, unit definitions, and height of building storeys. The four iUs considered in the engine are briefly described in the following:

- **iU-01, linear element translator:** This unit translates linear building elements such as beams, columns, and braces to their topological representation desired in structural analysis models.
- **iU-02, planar element translator:** This units translates planar elements such as slabs and walls to their equivalent shell elements with topological representation.
- **iU-03, material properties interpreter:** This unit extracts mechanical properties of structural materials from an external material property library and adds it to export models.
- **iU-04, coordinate modifier:** This module adjusts the coordinates of elements' end/corner points to integrate them at connection points. Geometry of building elements in a physical model is different from its topological representation in a structural analytical view. In structural analytical models, elements are connected at the intersection of their centreline/centre-surface, whereas elements are connected at their interface surfaces in physical models. Such a difference results in gaps between connected elements at their connection points. This interpretation unit integrates connected elements at their connection points by merging the coordinates of the end/corner points of the connected elements.

A simple building information model is transformed through the engine to test the engine and the algorithm it implements. *FIG. 10* demonstrates how the automated interpretation can be implemented for transformation of building information models to analytical models. The input information model is shown at the top of the figure. The test model contains beams, columns, a slab, a wall, and a bracing element; so, it includes all five types of the main structural building elements. The model at the bottom-left hand side of the figure shows the interpreted analytical model after performing all the direct and interpretation units except iU-04. As shown in this figure, end/corner points of the elements are not connected at the connection points and each element at the floor level has a different elevation. The model after connectivity adjustments is shown at the bottom-left hand side of this figure. As illustrated in this figure, elements are connected to one another. Comparing the building information model and the final interpreted analytical model, the number of elements increased from 10 to 11; breaking the beam element in two separate elements is the reason for the increase in the number of elements. The number of the points decreased from 25 to 7 as a result of merging end/corner points at the connection points. The case study validates the proposed approach by showing that targeted information including unit system, geometries, cross sections, material, and element connectivity are transformed seamlessly

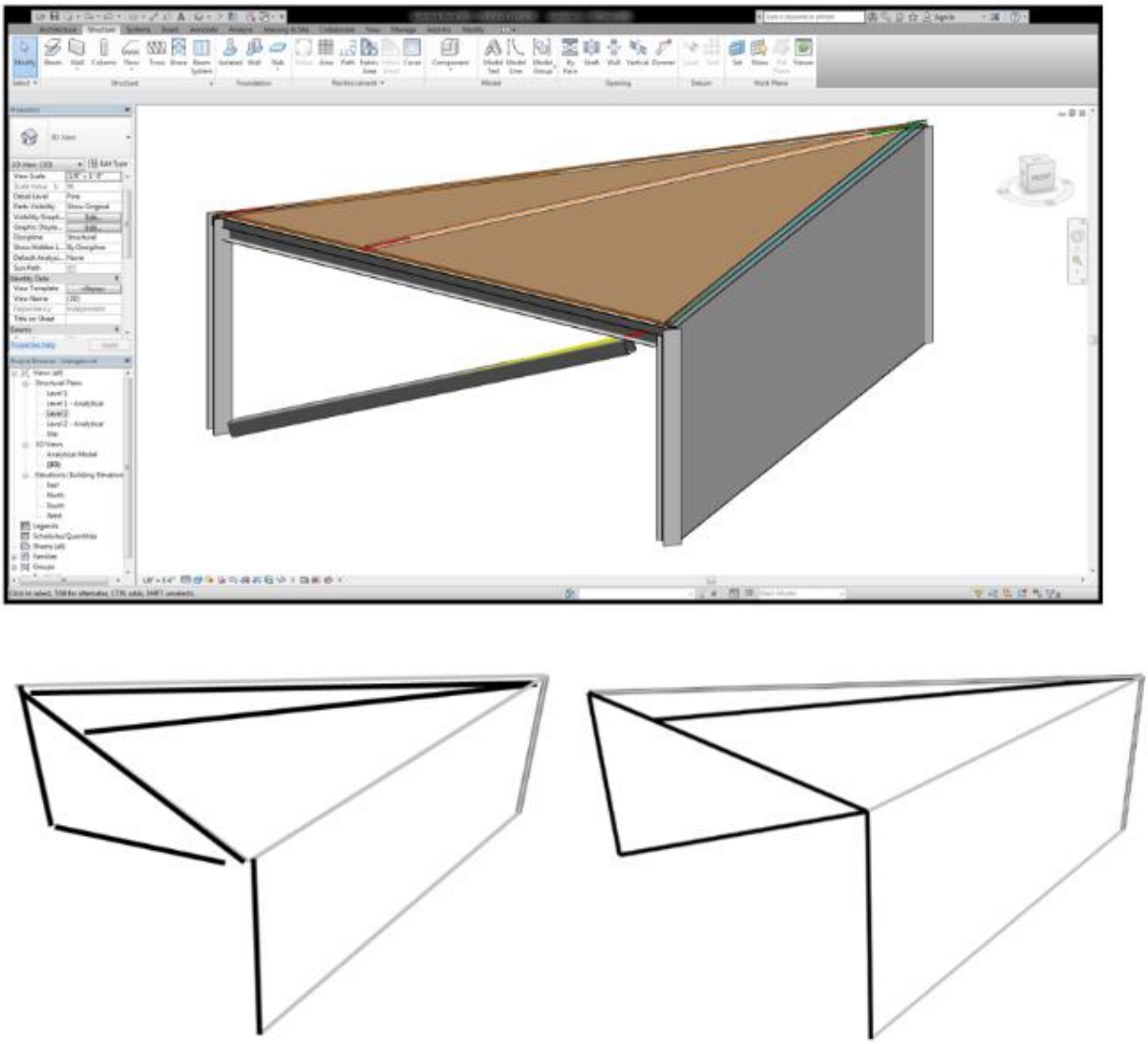

FIG. 10: The test model in Design Transfer View (top) and Structural View before (bottom-left) and after (bottom-right) connectivity adustments

10. SUMMARY AND CONCLUDING REMARKS

Automating the interpretations required for transforming the building information models to their equivalent analytical models can significantly enhance efficiency of this information exchange and as a result increase interests in using BIM for engineering design uses. Execution of the Interpreted information Exchange (IIE) needs supporting frameworks and technologies such as tools and platforms. To address this need, this paper presented a platform created for implementation of the IIE concept and made development of interpretation tools. This platform is made up of five modules, each responsible for performing a certain functionality: reading IFC file, checking the input file, information querying, generating new IFC objects, exporting IFC file. These five modules can provide enough tools for the developers to couple them with the interpretation engines to develop a tool for implementation of an automated IIE. The reading module imports the input IFC file and related schema, and then parses the read files into the native version-independent objects of the platform. Checking module compares the imported input file against the related schema to see if the file observes the constraints and rule-sets defined in the schema. Querying functions are responsible for searching and of extraction of the required information from the input file and related schema. Instance generator functions transform the interpreted information units to the IFC instance format, while the writing file functions write the newly generated instances to the output IFC file along

with the instances that should pass to the output file directly from input files. This platform could be coupled with any interpretation engine to automate an IIE process.

The new approach used for parsing the IFC instances and the schema's entities made this platform independent from the version of the input IFC file and the schema used. One of the challenges with IFC-based platforms is the high number of entities in the IFC schema, which is still rapidly growing as a result of multiple project going on for extension of the scope of the schema. This study showed that defining one generic object for IFC file instances and one generic object for IFC schema enteritis, and then mapping them together can address this challenge. It also illustrated that such data structure for IFC data management can be used for development of modules for reading IFC files, querying information from IFC instances, checking IFC files against the schema, and creating/writing IFC files. Since the approach inputs and exports models in IFC format, the version-independent nature of the platform makes is highly interpretable regardless of the capability of the BIM authoring and analytical tools for support of a specific version of IFC. In other words, the platform does not require the version of the input and output files be necessary the same.

REFERENCES

AEC Magazine (2013) *Executive guide to BIM: part 2*, *AEC Magazine website*. Available at: http://aecmag.com/technology-mainmenu-35/564-executive-guide-to-bim-part-2 (Accessed: 5 January 2015).

Alwisy, A., Al-Hussein, M. and Al-Jibouri, S. H. (2012) 'BIM approach for automated drafting and design for modular construction manufacturing', in *Proceedings of the Computing in Civil Engineering 2012*. June 17-20, Clearwater Beach, FL, USA: American Society of Civil Engineers, pp. 221–228. doi: 10.1061/9780784412343.0028.

Bazjanac, V. (2008) 'IFC BIM-based methodology for semi-automated building energy performance simulation', *Lawrence Berkeley National Laboratory*. CA, USA: University of California, Berkeley.

Bazjanac, V. and Kiviniemi, A. (2007) 'Reduction, simplification, translation and interpretation in the exchange of model data', in *Proceeding of the CIB W78 Conference*. June 28, Maribor, Slovenia, pp. 163–168.

Becerik-Gerber, B. and Rice, S. (2010) 'The perceived value of building information modeling in the US building industry', *Journal of information technology in Construction*, 15(2), pp. 185–201.

buildingSMART International (2015) *IFC GUID Summary*. Available at: http://www.buildingsmart-tech.org/implementation/get-started/ifc-guid (Accessed: 14 July 2015).

CIC (Computer Integrated Construction) Research Group (2012) *BIM project execution planning guide version 2.0*, *Computer Integrated Construction Research Group*. University Park, PA, USA: The Pennsylvania State Univesity, bim.psu.edu website. Available at: bim.psu.edu (Accessed: 2 February 2018).

Eastman, C. M. (1999) *Building product models: computer environments, supporting design and construction*. Florida, USA: 1st Edition, CRC press.

Fallon, K. K. and Palmer, M. E. (2007) *General buildings information handover guide*. National Institute of Standards and Technology (NIST), Report No. NISTIR 7417.

Gallaher, M. P. *et al.* (2004) 'Cost analysis of inadequate interoperability in the US capital facilities industry', *National Institute of Standards and Technology (NIST)*. Gaithersburg, Maryland, USA: Report No. NIST GCR 04-867.

Gane, V. and Haymaker, J. (2007) 'Conceptual design of high-rises with parametric methods', in *Predicting the Future, 25th eCAADe Conference Proceedings*. Stanford University, USA, pp. 970–978.

Guzmán, E. and Zhu, Z. (2014) 'Interoperability between building design and building energy analysis', in *Computing in Civil and Building Engineering (2014)*. American Society of Civil Engineers, pp. 17–24. doi: doi:10.1061/9780784413616.003.

Halpin, T. and Morgan, T. (2010) *Information modeling and relational databases*. Burlington, MA, USA: Second Edition, Morgan Kaufmann.

Hassanien Serror, M. *et al.* (2008) 'Shared computer-aided structural design model for construction industry (infrastructure)', *Computer-Aided Design*. Elsevier, 40(7), pp. 778–788. doi: 10.1016/j.cad.2007.07.003.

Laakso, M. and Kiviniemi, A. (2012) 'The IFC standard - a review of history, development, and standardization', *Journal of Information Technology in Construction (ITcon)*. International Council for Research and Innovation in Building and Construction, 17, pp. 134–161.

Lee, Y.-C., Eastman, C. M. and Solihin, W. (2016) 'An ontology-based approach for developing data exchange requirements and model views of building information modeling', *Advanced Engineering Informatics*. Elsevier, 30(3), pp. 354–367. doi: 10.1016/j.aei.2016.04.008.

Liu, X. *et al.* (2013) 'Extending the information delivery manual approach to identify information requirements for performance analysis of HVAC systems', *Advanced Engineering Informatics*. Elsevier, 27(4), pp. 496–505.

Liu, Z., Li, Y. and Zhang, H. (2010) 'IFC-based integration tool for supporting information exchange from architectural model to structural model', *Journal of Central South University of Technology*. Springer, 17, pp. 1344–1350. doi: 10.1007/s11771−010−0640−z.

McGraw Hill Construction (2012) 'The business value of BIM in North America: multi-year trend analysis and user ratings (2007-2012)', *Smart Market Report, McGraw Hill Construction*. Bedford, MA, USA.

Nepal, M. P. *et al.* (2013) 'Ontology-based feature modeling for construction information extraction from a building information model', *Journal of Computing in Civil Engineering*. American Society of Civil Engineers, 27(5), pp. 555–569.

Polter, M., Ismail, A. and Scherer, R. J. (2014) 'Towards an integrated grid-and cloud-based structural analysis platform', in *Proceedings of the Computing in Civil and Building Engineering Conference 2014*. June 23-25, Orlando, Florida, USA: ASCE, pp. 431–438. doi: 10.1061/9780784413616.054.

Pratt, M. J. (2001) 'Introduction to ISO 10303—the STEP standard for product data exchange', *Journal of Computing and Information Science in Engineering*. American Society of Mechanical Engineers, 1(1), pp. 102–103.

Pratt, M. J. (2005) 'ISO 10303, the STEP standard for product data exchange, and its PLM capabilities', *International Journal of Product Lifecycle Management*. Inderscience, 1(1), pp. 86–94.

Qin, L., Deng, X. and Liu, X. (2011) 'Industry foundation classes based integration of architectural design and structural analysis', *Journal of Shanghai Jiaotong University (Science)*. Springer, 16, pp. 83–90. doi: 10.1007/s12204-011-1099-2.

Ramaji, I. J. and Memari, A. M. (2018) 'Interpretation of structural analytical models from the coordination view in building information models', *Automation in Construction*, 90, pp. 117–133. doi: 10.1016/j.autcon.2018.02.025.

Ramaji, I. J., Messner, J. I. and Leicht, R. M. (2016) 'Leveraging Building Information Models in IFC to Perform Energy Analysis in OpenStudio', in *ASHRAE and IBPSA-USA SimBuild 2016*. Salt Lake City, Utah, pp. 251–258.

Ramaji, I. and Memari, A. (2016) 'Interpreted Information Exchange: Systematic Approach for BIM to Engineering Analysis Information Transformations', *Journal of Computing in Civil Engineering*. American Society of Civil Engineers, p. 4016028. doi: 10.1061/(ASCE)CP.1943-5487.0000591.

Sacks, R. and Barak, R. (2008) 'Impact of three-dimensional parametric modeling of buildings on productivity in structural engineering practice', *Automation in Construction*. Elsevier, 17(4), pp. 439–449. doi: 10.1016/j.autcon.2007.08.003.

Spiby, P. (1992) 'ISO 10303 industrial automation systems–product data representation and exchange–part 11: Description methods: The express language reference manual', *ISO DIS*. Switzerland, pp. 10303–10311.

Sudarsan, R. *et al.* (2005) 'A product information modeling framework for product lifecycle management', *Computer-aided design*. Elsevier, 37(13), pp. 1399–1411.

Wang, X. *et al.* (2013) 'Creating Structural Analysis Model from IFC-Based Structural Model', *Advanced Materials Research*. Trans Tech Publ, 712, pp. 901–904. doi: 10.4028/www.scientific.net/AMR.712-715.901.

Won, J., Lee, G. and Cho, C. (2013) 'No-Schema Algorithm for Extracting a Partial Model from an IFC Instance Model', *Journal of Computing in Civil Engineering*. American Society of Civil Engineers, 27(6), pp. 585–592. doi: 10.1061/(ASCE)CP.1943-5487.0000320.

Zhang, L. and Issa, R. R. A. (2013) 'Ontology-based partial building information model extraction', *Journal of Computing in Civil Engineering*. ASCE, 27(4), pp. 576–584. doi: 10.1061/(ASCE)CP.1943-5487.0000277.

Zhang, X. *et al.* (2014) 'An Industry Foundation Classes (IFC) web-based approach and Platform for Bi-Drectional conversion of structural analysis models', in *Computing in Civil and Building Engineering (2014)*. American Society of Civil Engineers, pp. 390–397. doi: 10.1061/9780784413616.049.

Zhao, W. and Liu, J. K. (2008) 'OWL/SWRL representation methodology for EXPRESS-driven product information model: part i. implementation methodology', *Computers in Industry*. Elsevier, 59(6), pp. 580–589.