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

AUTOMATIC BUILDING INFORMATION MODEL QUERY GENERATION

SUBMITTED: April 2015
REVISED: October 2015
PUBLISHED: December 2015 at http://www.itcon.org/2015/30
EDITOR: Amor R.

Yufei Jiang, Ph.D. Candidate,
College of Information Sciences and Technology, The Pennsylvania State University;
yzj107@psu.edu

Nan Yu, Software Engineer,
IBM Watson;
nyu@us.ibm.com

Jiang Ming, Ph.D. Candidate,
College of Information Sciences and Technology, The Pennsylvania State University;
jum310@ist.psu.edu

Sanghoon Lee, Assistant Professor,
Civil Engineering Department, University of Hong Kong;
sanghoon.lee@hku.hk

Jason DeGraw, Mechanical Engineer,
National Renewable Energy Laboratory;
Jason.DeGraw@nrel.gov

John Yen, Professor,
College of Information Sciences and Technology, The Pennsylvania State University;
jyen@ist.psu.edu

John I. Messner, Professor,
Department of Architectural Engineering, The Pennsylvania State University;
jmessner@engr.psu.edu

Dinghao Wu, Assistant Professor,
College of Information Sciences and Technology, The Pennsylvania State University;
dwu@ist.psu.edu

SUMMARY*: Energy efficient building design and construction calls for extensive collaboration between different subfields of the Architecture, Engineering and Construction (AEC) community. Performing building design and construction engineering raises challenges on data integration and software interoperability. Using Building Information Modeling (BIM) data hub to host and integrate building models is a promising solution to address those challenges, which can ease building design information management. However, the partial model query mechanism of current BIM data hub collaboration model has several limitations, which prevents designers and engineers to take advantage of BIM. To address this problem, we propose a general and effective approach to generate query code based on a Model View Definition (MVD). This approach is demonstrated through a software prototype called QueryGenerator. By demonstrating a case study using multi-zone air flow analysis, we show how our approach and tool can help domain experts to use BIM to drive building design with less labour and lower overhead cost.*

KEYWORDS*: BIM, data integration, query generation, case study.*

REFERENCE: *Yufei Jiang, Nan Yu, Jiang Ming, Sanghoon Lee, Jason DeGraw, John Yen, John I. Messner, Dinghao Wu (2015). Automatic building information model query generation, Journal of Information Technology in Construction (ITcon), Vol. 20, pg. 518-535, http://www.itcon.org/2015/30*

1. INTRODUCTION

An inherent challenge for building design and construction is the complex collaboration between different disciplines (Lee et al. 2012). Energy Efficient Building (EEB) design and construction, as an emerging area in the Architecture, Engineering and Construction (AEC) community, involves even more collaborators than traditional building, which calls for more extensive information sharing. To make a building energy efficient, teams from different domains need to perform a series of energy simulations and analyses. Different energy simulations and analyses depend on various simulation tools whose input formats and data structures are often heterogeneous. Most EEB designers and analysts spend a significant amount of time on developing an analysis model. According to Bazjanac (2001), they spend more than 50% of their total time on creating input data, and out of that time, they spend up to 80% generating geometric models, which is laborious and time consuming.

Building Information Modelling (BIM) is a potential solution to the information-sharing problem in the EEB design and construction area. BIM is "a digital representation of physical and functional characteristics of a facility" (National Institute of Building Sciences 2014). In many scenarios, EEB designers and analysts might need a small piece of data from many different teams. Such a fine-grained Object-Oriented (OO) representation means the building models can be integrated and divided according to user requirements. This feature allows EEB designers and analysts to conduct partial model query from a master BIM data hub that integrates all BIM data from different teams. Therefore, an effective approach to performing partial model query is the key of adoption BIM to EEB design and construction workflows.

There are two approaches to conducting BIM partial model query. Both of them are limited. The first category is to take advantage of existing general programing or query languages such as Java or SQL. However, for most users from the AEC community, mastering one or more programming languages sharply increases the cost of performing partial model query on BIM; meanwhile, programmers do not have domain workflow knowledge and lack comprehensive understanding of the requirements of the building design. In addition, to compose a piece of query, users must understand the internal data structure of BIM, which violates the loose-coupling principle.

The second category relies on GUI-based filters to fetch data from BIM. Although it is easy to use, GUI-based approach can only support a limited number of basic use cases. EEB simulations and analyses are highly specific. The number and types of arguments need to be tuned case by case. So this line of work is also insufficient to support all partial model query requirements in EEB workflows.

In this paper, we propose a new method to address the limitations of existing approaches. Our purpose is to offer an approach to support all EEB partial model query requirements with flat learning curve. Our approach leverages Model View Definition (MVD) to semi-automatically generate partial model query code. This new technique hides the details of BIM internal data structure and programming languages from end users. This approach relies on MVD as its input. More specifically, starting from MVD, the data schema and constraints are parsed at the front end. Next, an Intermediate Representation (IR) is generated based on template-based pattern matching. At last, the IR will be transformed into query code of a specific programming language based on the implementation of the BIM data hub to query from.

We have developed a prototype called *QueryGenerator* based on open source BIMserver (Beetz et al. 2010). We evaluate our prototype by applying it to a real-world multi-zone airflow analysis. We use the code generated from QueryGenerator to query against the BIMserver. The query results help the airflow analysts finish their tasks correctly and efficiently.

The scientific contributions of this paper are summarized as follows.

- We propose a semi-automatic query generation approach, which makes the programming language details and BIM data hub internal data structure implementation hidden from the end users.
- We define a customized and simplified IR to drive the code generation process.
- We build a prototype called QueryGenerator, which implements our approach, to perform partial model query on the BIMserver. Our case study on a real-world multi-zone airflow analysis shows that QueryGenerator can save time and make the process less laborious. In summary, our approach and tool can help domain experts use BIM to drive EEB design with less labour and lower overhead.

The remainder of this paper is organized as follows. We review background information of BIM, BIMserver, Industry Foundation Classes (IFC) and other related works in section 2. Then, section 3 discusses the proposed approach to generating BIMserver query code semi-automatically, followed by the details of the QueryGenerator implementation in section 4. We then describe our case study and evaluation in section 5, followed by the limitations and future work in section 6. We conclude in section 7.

2. RELATED WORK

In this section, we first generally review some research and development efforts that took advantage of BIM and BIM data hub to facilitate information exchange among different stakeholders in a building design and construction process. Then we investigate the works that focus on partial model extraction and query from BIM.

2.1 BIM: Data Schema and Applications

2.1.1 Data Schema

BIM data could be described, modelled and transferred by some existing open standards or modelling languages. EXPRESS data modelling language is defined by ISO 10303 (Pratt 2005). EXPRESS language specifies the aspects of product data that can be defined. Industry Foundation Classes (IFC) is an "international, open standard, and platform neutral" data model that describes physical and abstract building-related objects (buildingSMART International Ltd. 2014a). It offers a complete set of data schema that covers the data required by different disciplines through the whole building life cycle. It could be used as a universal data model specification for all participants of BIM-enabled building lifecycle. It also could be used as an intermediate representation to facilitate the exchange of several proprietary data formats. Model View Definition (MVD) defines a subset of IFC to specify exchange requirements for some specific scenarios in AEC industry (buildingSMART International Ltd. 2014b). MVDs can be encoded in MVDXML, a format that defines "allowable values at particular attributes of particular data types" in a MVD (buildingSMART International Ltd. 2014b). The Green Building XML (gbXML) open schema is designed to streamline the data transfer from BIM to EEB analysis tools (gbXML.org, 2014). gbXML has been integrated into various open source and commercial BIM platform products and analysis tools.

2.1.2 Applications

Berlo and Laat (2010) investigated the integration of BIM and Geospatial Information Systems (GIS). The project facilitates the data transformation from IFC to GeoBIM. Chen (2011) reported a conceptual framework for a BIM-based collaboration platform supported by mobile computing. Kim et al. (2011) introduced a BIM-based visualization tool, which gives a better insight into the design process and easy access to the reasons behind decision-makings. Singh et al. (2011) developed a theoretical framework of technical requirements for using BIM data hub as a multi-disciplinary collaboration platform. Liu et al. (2013; 2014) suggested a solution to combine BIM and Knowledge Management (KM) to achieve a Building Knowledge Modeling (BKM) platform. Beetz et al. (2010) developed an IFC data server called BIMserver to host BIM data as a central data server. Jiang et al. (2012) discussed BIMserver requirements to support the energy efficient building lifecycle. Yu et al. (2013) proposed a unified approach to facilitate and automate the data exchange between the BIMserver and OpenStudio. BIM4GeoA combines the BIMserver to integrate BIM data in IFC into a 3D navigation and viewing environment (Hijazi et al. 2010). Qi et al. (2012) developed a fall hazard checking tool by using the data queried from a BIM data hub. The authors also discussed the difficulties they encountered during composing query code.

2.2 Partial Model Query

Mazairac and Beetz (2013) proposed Building Information Model Query Language (BIMQL), which is an open source, domain specific query language for BIM. BIMQL is designed for selecting, updating and deleting the data in IFC formats. Some features of BIMQL such as open source, domain specific, and platform independent makes it a practical approach to many BIM information extraction and manipulation problems. BIMQL has relatively simple syntax and grammar to ease the usage. Many lexical components of BIMQL are designed to be similar to the natural language. However, as a brand new query language with limited scope of usage compared with other generic programming languages like Java, it still causes the cost of learning to all end users.

Daum and Borrmann (2014) designed Query Language for Building Information Model (QL4BIM) to provide "metric, directional and topological operators for defining filter expressions with qualitative spatial semantics". According to the authors, the previous BIM query languages are limited on this feature. Most of them can only filter out the building components by their numeric relationship. However, the spatial relationship between different building components plays a significant role in a building design and construction process. QL4BIM makes the query like "are there any valves inside room 101" easy to perform.

Partial Model Query Language (PMQL) is an earlier research effort to facilitate partial model query from an IFC model server (Adachi 2003). Specifically, a query in PMQL is a piece of XML data. A XML file could be directly transported among different web service via Simple Object Access Protocol (SOAP). Users can use PMQL to select, update, and delete IFC partial models. The PMQL allows recursive and nested object structure and conditional expression. Koonce et al. (1998) designed and implemented EQL, an EXPRESS Query Language. It is an SQL-like query language, which could be used to query data from object-oriented EXPRESS modeling formatted files.

Some other researchers try to ease the process of data query by taking advantage of Data as a Service (DaaS) (Dan et al. 2007; Dwivedi and Kulkarni 2008; Truong and Dustdar 2009; Olson 2009), instead of designing new query languages. As a new architecture that promotes software interoperability at this data level, the DaaS model provides data based on existing data sources or allows data providers to create, retrieve, update and delete data (Truong and Dustdar 2009; Cagle 2010).

3. DESIGN

3.1 Architecture

FIG. 1 illustrates the architecture of our design. We have designed the query code generation framework based on MVD as input. The whole design consists of 4 conceptual components. They are the front end which is used to parse the input lexically, the intermediate representation generator, intermediate representation optimizer, and target query code generator.

There are four steps in this automatic query code generation process. In the first step, the front end of our framework first accepts MVD as input, which could be in EXPRESS, mvdXML, JSON, and other formats. It parses MVD and outputs a token stream. What kinds of format to be supported are subjected to implementation requirements and use cases. By changing the front end only, the whole framework could be extended without affecting rest parts of the framework (Section 4.2.2).

In the second step, the Intermediate Representation Generator is executed with the token stream as input. The output is Intermediate Representation (IR). This component does not depend on MVD format, generated query code language type, and the platform to query from (Section 4.2.3).

At last, Target Query Code Generator then uses the optimized IR to construct query code. The generated query code could be directly sent to a specific BIM data hub implementation to fetch data of interest. Similar to the front end, this component is also subject to implementation requirements and use cases. At this phase, IR could be transformed into different target languages including Java, BIMQL or QL4BIM according to user requirements. Therefore our approach is orthogonal to those research efforts. To change the language of generated code, a developer only needs to re-implement Target Query Code Generator without changing the rest part of the framework (Section 4.2.4).

Before we specifically present our prototype implantation details of our framework in next Section. We discuss several key insights behind our design first.

3.2 MVD as Input

MVDs are designed to explicitly define the model information for a specific use case. buildingSMART international defines an Information Delivery Manual (IDM) as a set of user requirements document mainly written in natural language. The second step is to produce the MVD by developing concepts from the information requirements of IDM and mapping the concepts to a BIM that can be implemented in software data exchange. According to buildingSMART International (2014b), "each MVD Concept provides unambiguous implementation guidance to software vendors."

Since MVDs establish the connection between domain descriptions and BIM data schema, they can be used as an unambiguous information source to compose query code. Specifically, domain experts will review and interpret the MVD first and identify the MVD concepts of interests. Some well-defined MVDs exist in the AEC community. Users can reuse them according to their needs. The process of parsing will be elaborated in more detail in Section 4.

An important advantage of using MVD as input is that such an approach takes advantage of existing assets of the AEC community. According to buildingSMART International (2014b), many MVDs have been developed and released. For example, there are IFC4 Reference View, IFC4 Design Transfer View, IFC2X3 Coordination View, Space Boundary Addon View, Basic FM Handover View, Structural Analysis View, etc.

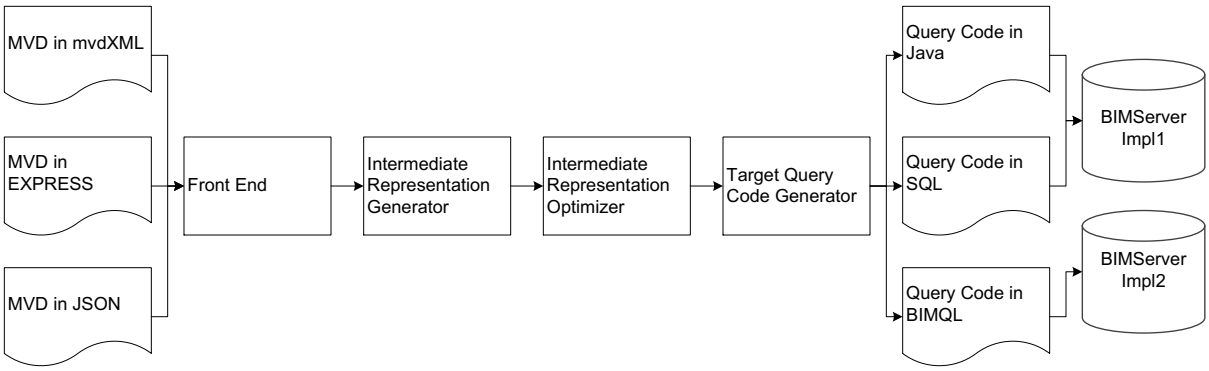

FIG. 1. Automatic Partial Model Query Generator Framework Design

3.3 Specification Translator

In this subsection, we analyze automatic query code generation from a programming language (PL) point of view to justify our approach.

A specification translator translates a specification S into a program P. FIG. 2 illustrates the model of a specification translator. From a theoretical point of view, our framework belongs to this model. Using specification as input has several merits in the context of BIM partial model query. First, compared with programming, specifying is easier to most end users. Specification usually is written in a description language, and many programming languages are imperative-based. By using a specification as input, end users just need to specify "what", rather than programming "how". On the other hand, compared with natural language, a description language usually has no ambiguity. By applying certain rules, a specification could be parsed and translated by a machine.

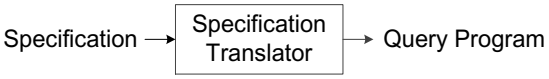

FIG. 2. The Model of a Specification Translator

In our context, the program that we will generate is query code, which implies the necessary operations. Thus we can divide this code generation process into two parts. The first part is to generate the code that needs to be specified by a user. The other part is to generate the routine code.

Regarding the code that needs to be specified by the users (i.e., domain experts), they need to specify two types of information. One is the type information such as doors, walls and columns. The other one is constrains on specific attributes.

In terms of routine code generation, a "pattern match" method can be used. For each query pattern, the corresponding routine code template is stored in a routine code pool. When certain scenarios are detected according to the input MVD, the matching routine code template is loaded from the pool and combined with customized code to generate the complete query code (FIG. 3).

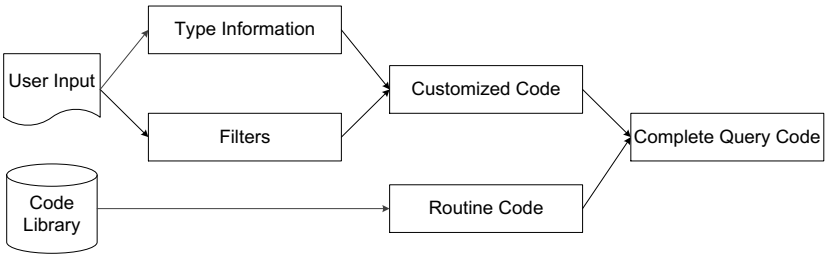

FIG. 3. Routine Code and Customized Code

It is noteworthy that the "pattern" here refers to the IFC/MVD object classification. For example, as shown in FIG. 4, almost all regular building components' IFC classes are a subclass of *IfcBuildingElement*. Additionally, they are in the same topology in the abstract building graph (FIG. 5), and thus they can share the same routine code. Different sets of routine code need to be developed for only those that have different IFC inheritance class hierarchies and different positions in an abstract building graph. It is important to note the differences between FIG. 4 and FIG. 5, which show two kinds of relationships. Specifically, FIG. 4 shows "is-a" relationship in IFC data schema. For example, *IfcDoor* "is an" *IfcBuildingElement*. FIG. 5 shows "has-a" relationship in a typical building structure. For example, *IfcBuildingStorey* "has a" or "have many" *IfcDoor(s)*. "Is-a" relationship and "has-a" relationship are two fundamental relationships of OO design, which are also the foundations of our "pattern match" approach.

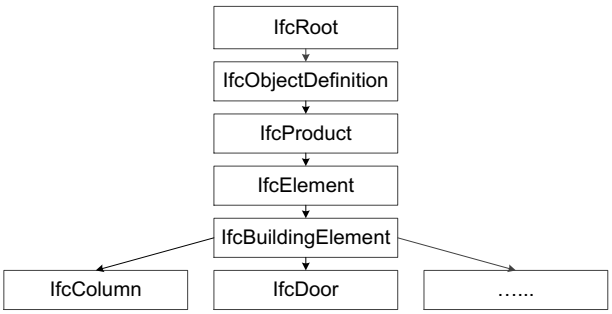

FIG. 4: IfcDoor Inheritance Class Hierarchy in IFC Data Schema

FIG. 5. A Typical Building Structure

3.4 Advanced Filters

In some cases, users may want to additionally select some objects from partial models based on the value of properties. For example, users may want to query the doors whose heights are greater than 2 meters, rather than just getting all of the doors in a building. In such a case, just specifying the object names and some coarse-grained constraints is not enough. Unlike using predicates ("x>2") to conduct conditional judgment in a programming language, in specification data schema, filters are also declared in a specification-based manner. FIG. 6 shows an example of a piece of query code written in Java. The code in bold is generated based on the filters in user input. We need to generate the conditional statements in a query language based on the value of some special properties in users' input data schema. In this subsection, we use JSON to demonstrate how it works.

```
@Override
public void query(IfcModelInterface model, PrintWriter out) {
  out.println("Running doors example");
  List<IfcBuildingStorey> stories = model.getAll(IfcBuildingStorey.class);
   Map<Double, IfcBuildingStorey> orderedStories = new TreeMap<Double, IfcBuildingStorey>();
   for (IfcBuildingStorey storey : stories) {
       orderedStories.put(storey.getElevation(), storey);
   }
  if (orderedStories.size() > 1) {
      IfcBuildingStorey firstFloor = stories.get(1);
      for (IfcRelContainedInSpatialStructure rel : firstFloor.getContainsElements()) {
      for (IfcProduct product : rel.getRelatedElements()) {
         if (product instanceof IfcDoor) {
            ifcDoor ifcDoor = (IfcDoor)product;
            if (ifcDoor.getOverallHeight() > 2) {
              out.println(ifcDoor.getName() + " " + ifcDoor.getOverallHeight());

}}}}}}
```

FIG. 6. A BIMserver Advanced Query Example

```
{"doors": [
        {"height": {
                "type": "integer",
                "unit": "meters",
                "minimum": 2
            }}]}
```

FIG. 7. JSON Input Example for Airflow Analysis

FIG. 7. shows a simplified IfcDoor data schema in JSON that specifies the minimum height of the door (shown in bold). For Integer type and other numeric attributes, “minimum”, “maximum”, “enum” and “multipleOf” are several examples of filter properties we can set. Regarding string type, available filters include “minLength”, “maxLength”, and “pattern”. mvdXML and other input formats work in a quite similar way.

4. IMPLEMENTATION

To demonstrate the efficacy of our framework, we have developed a prototype called *QueryGenerator*. According to the literature review results listed in Section 2, many previous studies have focused on an implementation instance of BIM data hub: the BIMserver (Beetz et al. 2010). In addition, many users encounter difficulties when they try to conduct partial model query from the BIMserver. Therefore, we built *QueryGenerator* to demonstrate our approach and conducted a case study based on the BIMserver. But please note that our design and general approach is not locked into any specific vendors and implementations.

4.1 Query Mechanism of the BIMserver

Before we detail the implementation of each component, we review the BIMserver query mechanism first.

4.1.1 IFC Objects in Java Runtime

The primary input files of the BIMserver are the IFC and IFCXML files. The files are stored in the underlying Berkeley DB, which is a non-SQL-able database. It adopts the key-value-pair mechanism to store the objects. The BIMserver implements every IFC class into a corresponding Java class via Eclipse Modelling Framework (EMF) (Steinberg et al. 2008). When the BIMserver starts, it reads IFC objects data from the database and instantiates them into Java objects which are maintained in Java runtime. These Java objects that represent different building elements are in memory as nodes to construct a graph structure that is connected by different

types of predefined relationships. Such a graph structure is the BIM abstraction of the building. The query code of the BIMserver is written in Java. The main logic of the query code is to perform a graph search on a graph of building elements.

4.1.2 Query Compilation

The Java query code that the user uploads to the BIMserver is a Java class that implements the Query interface under the package name "org.bimserver.querycomplier". This query code is sent to the BIMserver to be compiled and executed with other classes in the same package. The names of the classes in the package are listed in Table 1.

TABLE 1. Java Classes in Package org.bimserver.querycompiler

Class name	Source
CompileException	Predefined in the BIMserver
Query	Submitted by a User
QueryCompiler	Predefined in the BIMserver
QueryInterface	Predefined in the BIMserver, Must Be Implemented in Query by a User
VirtualClassLoader	Predefined in the BIMserver
VirtualFile	Predefined in the BIMserver
VirtualFileManager	Predefined in the BIMserver

4.2 QueryGenerator

4.2.1 Overview

We implemented the BIMserver query generator in Java on top of the BIMserver (our tool has been open sourced and available at https://github.com/triangelfish/QueryGeneratorforBIMserver0.2.). FIG. 8 illustrates the implementation of the QueryGenerator. It consists of 5 components: the front end, the IR generator and optimizer, the target code generator, the missing data detector, and the results and resources integrator. Please note that our implementation is slightly different from our general conceptual architecture design. There are two more components in our implementation: missing data detector and results and resources integrator. We need them because in practice the BIMserver does not always have the data we want to query. In such a case, to facilitate the whole workflow, we add two more post-query components to help users double check the results returned by the BIMserver and give users a chance to manually fill out those missing data fields. Thus the final results can be instantly used by users' domain-specific analysis tools.

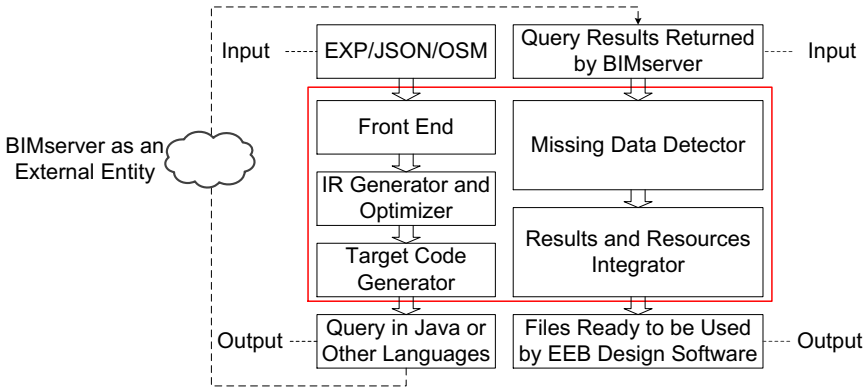

FIG. 8. Implementation Overview

4.2.2 Front End

The front end of the QueryGenerator consists of two parts: a usecase detector and a lexical parser.

The use case detector serves two purposes. First, it detects the input format and dispatches the parsing task to a specific lexical parser. Second, it detects the usecase of the input. Based on the usecase information, the following workflow may need to be adjusted. To process all input files in batch, the QueryGenerator guides users to specify a repository that contains all input files rather than letting users specify input files one by one. Domain experts can directly use the MVD documents[1] as input. They can also specify the properties of interest by customizing the input files by reviewing and interpreting the MVD documents. In the current implementation we support the parsing of the MVD in EXPRESS language or JSON format and other domain-specific scheme files. In EXPRESS language, each line of a file lists one MVD concept of interest. The QueryGenerator will map these MVD concepts to IFC objects by detaching the implantation instructions and validation rules from those MVD concepts. JavaScript Object Notation (JSON) format is a widely used platform-independent file format. The features such as lightweight file structure and human-readable content make JSON a suitable format of input. According to our surveys and interviews, many domain experts have already adopted JSON files to facilitate data exchange. Thus supporting JSON helps the QueryGenerator leverage existing assets in the AEC community. Domain-specific schema files serve a certain type of use case. Unlike the EXPRESS language and the JSON file format that offer a general solution for the query generation, a domain-specific input format has to be developed and supported application-by-application. However, domain-specific input formats offer a shortcut for designers. The files list the objects of interest in a domain-specific schema. Taking OpenStudio as an example, users query the data of building surfaces by listing the corresponding OpenStudio Model schema (OS:surface) in an .osm file. Such a filename suffix helps the input and use case detector identify that this data schema and query is a part of an OpenStudio use case and calls corresponding routines to fit the OpenStudio use case in the following interaction panels and steps. For instance, one of the routines must map "OS:surface" to the IFC object "IfcSurface". The output of a query, in such a case, will be .osm files containing formatted data that could be directly loaded by OpenStudio, rather than the JSON files for a more general purpose. We are aware that some other formats like mvdXML are also widely used. In the future, we will make more engineering efforts to support more formats. At this moment, we just use MVD in EXPRESS language or JSON format and several domain-specific formats to demonstrate the scientific contributions of our tool.

The lexical parser reads the character flow from the input and outputs a sequence of tokens and builds a symbol table. More specifically, the lexical parser must map each valid lexeme to a token based on the pattern. The terminologies, including token, pattern, and lexeme, are from the complier and language processor design literature (Aho et al. 1986). A token is an "abstract symbol representing a kind of lexical unit". A pattern describes the form that "the lexemes of a token may take". A pattern usually can be represented by a regular expression. A lexeme is a sequence of characters that "matches the pattern for a token". Table 2 shows some examples of IFC tokens and Lexemes.

TABLE 2. Examples of Tokens and Lexemes of IFC.

Token	Description	Regular Expression	Lexeme Examples
IFC Enumeration	The class name should start with "Ifc" and end with "Enum".	/^Ifc[a-zA-Z]+Enum$/	IfcActionTypeEnum IfcBeamTypeEnum
IFC Relationship	The class name should start with "IfcRel".	/^IfcRel[a-zA-Z]+$/	IfcRelConnectsPortToElement IfcRelAssignsToProduct

More specifically, when the lexical parser reads a sequence of characters, say *IfcBeamTypeEnum,* the output will be a matched token and an entry in the symbol table, which is shown below.

Token: <**IFC Enumeration**, a reference to symbol table entry for *IfcBeamTypeEnum* >.

Symbol Table Entry: [*IfcBeamTypeEnum* | IFC Enumeration | other information].

[1] The MVD documents may contain a set of documents including Definition Diagrams, Exchange Requirements, Worksheets, and other complementary materials. Here by the "MVD documents", we refer to the MVD data schema in a formal structural format (e.g. mvdXML).

The token information can be used to retrieve corresponding routine code. The symbol table retains the necessary information to fill in the customized part of a piece of routine code. This part is detailed in the following subsections.

4.2.3 Intermediate Representation (IR) Generator and Optimizer

The front end is followed by the IR generator and Optimizer, which generates and optimizes the IR based on the output of the lexical parser. We define a simplified and customized Java Context-Free Grammar (CFG) "QueryJava". We show QueryJava CFG in Backus-Naur Form (BNF) (Backus, J. W. 1959) in FIG. 9. A CFG consists of a set of production rules. Each rule consists of a head (the symbols listed on left-hand side) and the symbols that could be used to replace the head. A head contains at least one non-terminal symbol. A head could be replaced by a group of terminal symbols and non-terminal symbols. Terminal symbols are the elementary elements of a CFG such as Java keyword "void", "public". Nonterminal symbols could be replaced by a set of nonterminal and terminal symbols according to production rules. Terminal symbols, non-terminal symbols, and the production rules together construct a tree-like data structure which could represent a language grammar in different abstract level. Terminal symbols are leaves of the tree. In FIG. 9, terminal symbols are highlighted by bold font. We list the core part of QueryJava CFG in FIG. 9.

By defining QueryJava CFG, we customize and reduce the full Java grammar to fit the requirements of BIMserver query. From the customization side, the IR generator could treat IFC class names, which are passed from lexical analyzer, separately from other Java class names as a unique lexical element. Additionally, unlike a normal Java program starting from "MainClass" which contains a main method, from FIG. 9, we can see that the code we generate always has only one Java class "Query". The class "Query" comprises a set of fixed terminal symbols and four non-terminal symbols: identifier, statement, field declaration, and class declaration.". On the reduction side, QueryJava CFG helps the IR generator avoid driving the translation process cumbersomely by a full Java CFG. QueryJava CFG removes unused grammar elements from many aspects. For example, nonterminal symbol "modifier" defines the scale of visibility of a method.

It is important to note that the IR design and implementation is completely hidden from the end users. In other words, users have no need to know IR to use our tool. Also, even we cut off some elements to make our CFG simplified, it has the same expressivity with the original one.

4.2.4 Target Code Generator

As we have discussed previously and shown in FIG. 3, information offered by users contains two aspects: object type information of interest and the filter information which is used to select qualified objects. Driven by the QueryJava CFG, Java generator generates statements or expressions related to variable declarations according to IFC objects type information in IR; meanwhile, filters information is mainly turned into conditional and loop statements. The target code generator then interleaves generated statements with pre-stored routine code to compose complete error-free compliant Java source code, or other target languages which become the query code to the BIMserver. The correctness of this interleave-and-fill process is also guaranteed by QueryJava CFG.

4.2.5 Post-Query Components

After the query code has been executed by the BIMserver, the results are exported. The results are JSON files which contain the actual data from the model query and the default values of those missing attributes of the model. The results can be freely used by users. For instance, they can directly conduct statistical analysis or run simulations by processing those results. However, even with results in hand, processing those data may still encounter problems like data missing or format mismatch. Since the goal of BIMserver Query Generator is to query IFC model in the BIMserver into the file format ready to be used by building design software applications, we implemented two more components (i.e., the missing data detector and the results and resources integrator) to help users handle missing data and organize the results for the target software application.

The missing data detector checks to see if the results cover all data attributes contained in the MVD, as in practice an IFC model from a designer in a specific software application may not cover all IFC-defined types and attributes. If some unforeseen analysis needs to be conducted, then some information might be missing from that model. By comparing the results from the BIMserver and data schema provided by the input file, the missing data detector examines if any data is missing. If it happens, the missing data detector notifies the user with an

interface panel so that the user can assign data to those empty fields. This interface panel needs to be developed for each use case. User-assigned data is saved as a resource file.

Goal	::=	Query <EOF>
Query	::=	"**class**" Identifier "{" "**@Override**" "**public**" "**void**" "**Query**" "(" "**IfcModelInterface**" Identifier "," "**PrintWriter**" identifier ")" "{" statement "}" {field_declaration} {class_declaration} "}"
statement	::=	variable_declaration | (expression ";") | (statement_block) | (if_statement) | (for_statement) | (try_statement) | ("return" [expression] ";") | ("throw" expression ";") | (identifier ":" statement) | ("break" [identifier] ";") | ("continue" [identifier] ";") | (";")
identifier	::=	"**a..z,$,_**" { "**a..z,$,_,0..9,unicode character over 00C0**" }
class_declaration	::=	{ modifier } "**class**" identifier ["**extends**" class_name] ["**implements**" interface_name { "," interface_name }] "{" { field_declaration } "}"
field_declaration	::=	((method_declaration | constructor_declaration | variable_declaration)) static_initializer | ";"
varaiable_declaration	::=	{modifier} type identifier { "[" "]" } ["=" variable_initializer] ";"
variable_initializer	::=	expression | ("{" [variable_initializer { "," variable_initializer } [","]] "}")
expression	::=	numeric_expression | testing_expression | logical_expression | string_expression | bit_expression | casting_expression | creating_expression | literal_expression | "**null**" | "**super**" | "**this**" | identifier | ("(" expression ")") | (expression (("(" [arglist] ")") | ("[" expression "]") | ("." expression) | ("," expression) | ("**instanceof**" (class_name | interface_name))))
statement_block	::=	"{" { statement } "}"
if_statement	::=	"**if**" "(" expression ")" statement ["**else**" statement]
for_statement	::=	**for**" "(" (variable_declaration | (expression ";") | ";") [expression] ";" [expression] ";" ")" statement
try_statement	::=	**"try"** statement **{ "catch" "("** parameter **")"** statement **} ["finally"** statement **]**
parameter	::=	type identifier { "[" "]" }
type	::=	type_specifier { "[" "]" }
type_specifier	::=	"**boolean**" | "**char**" | "**short**" | "**int**" | "**float**" | "**long**" | "**double**" | class_name | interface_name | ifc_objects_name
method_declaration	::=	{ modifier } type identifier "(" [parameter_list] ")" { "[" "]" } (statement_block |";")
modifier	::=	"**public**" | "**private**" | "**protected**" | "**static**" | "**final**" | "**transient**"
class_name	::=	identifier | (package_name "." identifier)
interface_name	::=	identifier | (package_name "." identifier)
ifc_objects_name	::=	"**Ifc**" Identifier | ("**org.bimserver.models.ifc2X**" **3**|**4** "." "**Ifc**" Identifier)

FIG. 9. QueryJava Context-free Grammar

The results and resources integrator combines query results and resource files into one well-formatted file according to the use case information preserved by the input and use case detector. This file is ready to be used by target EEB design software. For example, if our tool already knows that this query serves an OpenStudio use case, then the results and resources integrator combines all files into an .osm file format which is ready to be used by OpenStudio. Besides integrating the results and the user-complement data for model's missing attributes, the results and resources integrator also helps on integrating the results and the data out of the IFC scope. For example, when there are additional data needed to run a specific simulation, such as weather data that is required by an energy analysis, but cannot be queried from the model or inputted by the user, the user can specify the location of the data file (e.g., the location of the weather data file) so that the simulation application can access it. In this case, the file containing information like weather record is also considered as a resource file.

4.3 Query Generation Process Summary

In summary, the QueryGenerator performs the following functions. First, it takes structural MVD files as input. Second, it conducts lexical analysis on the input to generate a sequence of tokens and a symbol table. Then it loads routine code from the code library. This routine code is combined with the token and symbol information from the previous step to generate IR. Then, based upon IR, the component target code generator generates the query code in target languages. Depending on the specific usecases, some follow-up or application specific steps are needed.

5. CASE STUDY

5.1 Experiment Overview

A case study was performed to test if the QueryGenerator could facilitate data exchange for an airflow analysis workflow by automatically generating an input file for CONTAM (NIST 2013). Multi-zone airflow analysis is an important task for energy efficient building design. Airflow is one of the most important factors that affects the indoor concentration of oxygen, carbon dioxide and other gases. Indoor airflow also influences the efficiency of the HVAC system and the indoor temperature. Furthermore, the multi-zone airflow analysis can also be used to evaluate collective protection locations automatically (DeGraw and Bahnfleth 2010). CONTAM is one of the most widely used tools for airflow simulation. We tested the QueryGenerator on a simple building model called "GreenBuilding", which has two levels and is equipped with a simple HVAC system. The details are listed in Table 3. The GreenBuilding model was developed in Autodesk Revit^{TM} and then exported to the IFC format using the Revit IFC export function (see FIG. 10).

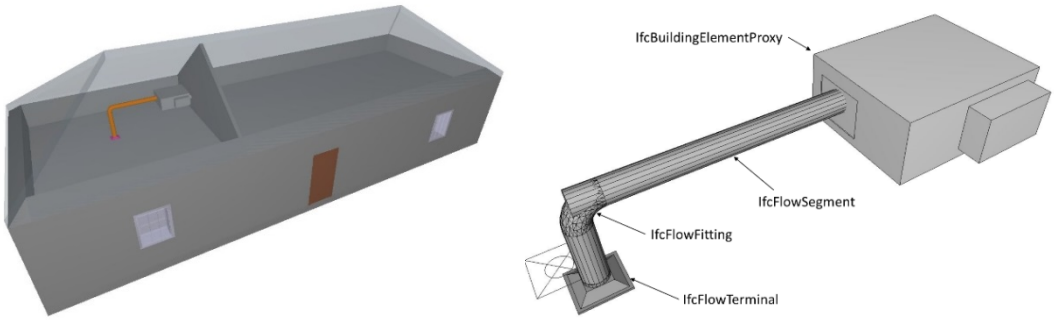

FIG. 10. GreenBuilding Model and details of its HVAC system

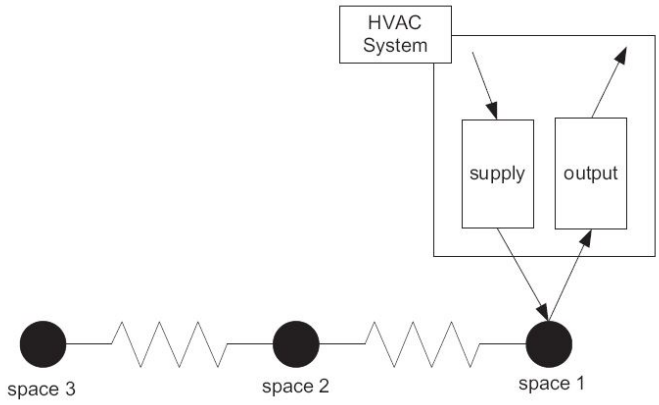

FIG. 11. Connectivity and the Nodes of GreenBuilding

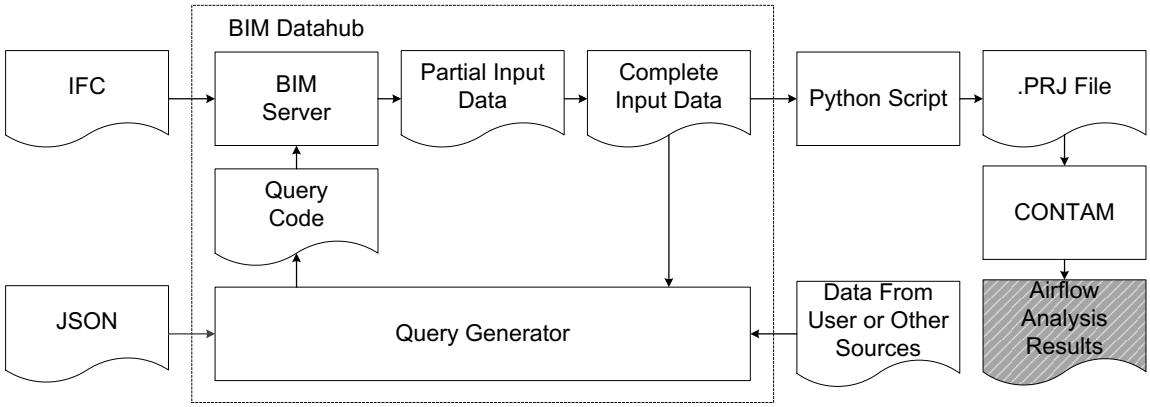

FIG. 12. Airflow Analysis Model Development Workflow Using BIMserver Query Code Generator

TABLE 3. GreenBuilding Facts

Property	**Number**
Number of Levels (1st floor and plenum)	2
Number of Spaces	6
The Number of the Spaces Connected to HVAC System	1

```
{"levels":[{"name": "", "elevation": ,"height": }],
        "ahs":    [{"name": "", "outdoorAirPercent":  }],
        "zones":[{"name": "", "volume": ,"level": "",
                        "supplyAir":  [{"system": "",  "flow":
                                        {"value": ,"units": ""}}],
                        "returnAir": [{"system": "","flow":
                                        {"value": ,"units": ""}}]}],
        "walls":  [{"name": "", "area": ,"elevation": ,
                        "zones":[""],
                        "azimuth": }]}
```

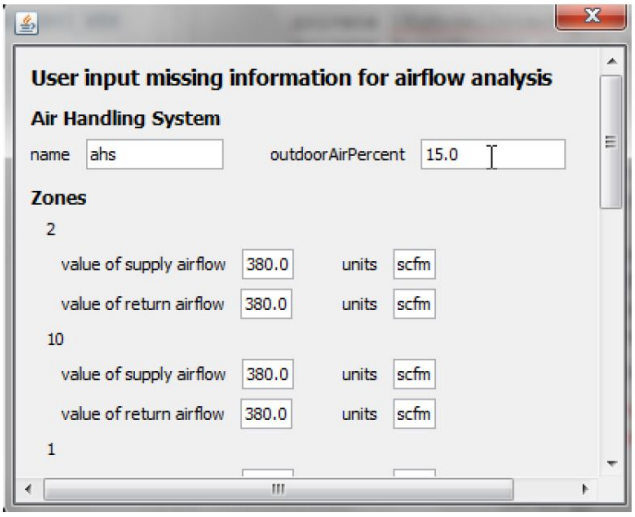

FIG. 14. Custom User Input Panel to Collect Missing Data

The information needed by airflow analysis is more about the topology of the building. FIG. 11 shows the topology of the first floor of the GreenBuilding. The nodes in FIG. 11 represent the spaces or HVAC system; the edges represent the air leakage between adjacent spaces and the airflow between spaces and HVAC system.

5.2 Overview of Airflow Analysis Workflow Using QueryGenerator

FIG. 12 demonstrates a complete airflow analysis workflow using the QueryGenerator. The workflow consists of 4 steps: 1) query the partial model of GreenBuilding from the BIMserver; 2) assign missing data (optional); 3) combine the partial model data and missing data to make a complete input file; and 4) load the input file (.PRJ file) into CONTAM.

The QueryGenerator guides the user to finish the whole workflow step by step. The user needs to provide the folder name of input files and the folder in which QueryGenerator creates the Java query code. In this case study, the input file format we use is JSON. FIG. 13 shows an example of input file for querying necessary data for running a simulation in CONTAM. Because the current version of the generator is separate from the BIMserver code, the user needs to copy and paste the generated code into the BIMserver. The code drives the BIMserver to export one or two files in the output folder specified. The first file contains the query results from the BIMserver (file name ends with .queried.json). If there is any missing data, the second file will contain the default value of those missing data (file name ends with .missing.json).

```
{"walls":[{"area":22.360693,"azimuth":0,"elevation":1.5,"name":
         "Basic Wall:Interior - 4 7/8 Partition (1-hr):158353","zones":["1","2"]},
        {"area":20.10463,"azimuth":90,"elevation":1.5,"name":
         "Basic Wall:Exterior - Brick on CMU:157673","zones":["3"]},
         {"area":60.96375,"azimuth":180,"elevation":1.5,"name":
          "Basic Wall:Exterior - Brick on CMU:157571","zones":["1","2"]},
         {"area":22.360693,"azimuth":0,"elevation":1.5,"name":
          "Basic Wall:Interior - 4 7/8 Partition (1-hr):158433","zones":["2","3"]},
         {"area":54.890507,"azimuth":270,"elevation":1.5,"name":
          "Basic Wall:Exterior - Brick on CMU:157723","zones":["1","2"]},
         {"area":20.10463,"azimuth":0,"elevation":1.5,"name":
          "Basic Wall:Exterior - Brick on CMU:157953","zones":["1"]}],
 "levels":[{"elevation":3,"height":0,"name":"Level 2"},
         {"elevation":0,"height":3,"name":"Level 1"}],
 "description":null,
 "name":"Project Number",
 "ahs":[{"name":"ahs","outdoorAirPercent":35}],
 "zones":[{"level":"Level 1","name":"3",
          "returnAir":[{"flow":{"units":"scfm","value":280},"system":"ahs"}],
          "supplyAir":[{"flow":{"units":"scfm","value":180},"system":"ahs"}],
"volume":78.13969},
         {"level":"Level 1","name":"2",
          "returnAir":[{"flow":{"units":"scfm","value":380},"system":"ahs"}],

"supplyAir":[{"flow":{"units":"scfm","value":380},"system":"ahs"}],"volume":80.64787},
        {"level":"Level 1","name":"1",
         "returnAir":[{"flow":{"units":"scfm","value":380},"system":"ahs"}],

"supplyAir":[{"flow":{"units":"scfm","value":380},"system":"ahs"}],"volume":78.13969}]}
```


FIG. 15 Data in JSON Format

In this case study, the IFC model of the GreenBuilding does not contain several types of information that are needed. The BIMserver query code generator detected this fact and pops up a panel which guides the user to assign values to those missing attributes (FIG. 14).

The next step is to combine the partial input data through the query and the file that contains user-assigned data into one final input file. This step is also automated by the QueryGenerator. The complete input file is in JSON format. Part of the final query result of GreenBuilding is shown in FIG. 15. This JSON file is translated into a legitimate CONTAM input file using a Python script.

6. LIMITATIONS AND FUTURE WORK

Our QueryGenerator prototype bears several limitations regarding the implantation at this stage. First, our approach proposed several possible input formats. Some of them are not implemented yet in our prototype. Second, our tool cannot support advanced filters now. Both of them require additional engineering efforts.

The automatic generation of BIMserver query code contributes to the integration of fragmented software applications and multiple participants in the EEB design area at the data level. Our work can be extended horizontally by improving our data level integration, and vertically by archiving application levels of integration. At the data level, proposed future work includes the following promising directions. First, more design scenarios (e.g. daylighting analysis and energy consumption analysis) can be developed and explored. Second, more data formats such as gbXML and STEP Physical File can be supported. Third, to simplify and unify the code generation process, the IR could be further tuned. At application level, besides query code automatic generation, automatic query code importing and query result exporting would additionally facilitate the building design process. Launching the functions of current BIM data server and other applications as a web service may solve the inter-application function call challenges. Service-Oriented Architecture (SOA), Simple Object Access Protocol (SOAP), and Representational State Transfer (RESTful) style architecture are possible options.

7. CONCLUSIONS AND FUTURE WORK

By surveying the open data standards and BIM data hub implementations, we find that current partial model query mechanisms become a barrier that prevents building designers and analysts from fully taking advantage of BIM. To address this problem, we make the following scientific contributions. First, we propose a query generation approach, which makes the query language details and BIM data hub internal data structure implementation transparent to the end users. Second, we define a customized and simplified IR to drive the code generation process. Third, we build a QueryGenerator prototype and demonstrate our method on a real-world multi-zone airflow analysis. Our contributions may have impact on reducing overhead cost in EEB design workflows. We will conduct a user study oriented research to do more quantitative evaluation in the future.

8. ACKNOWLEDGEMENTS

This material is based upon work supported by the Energy Efficient Buildings Hub (EEB Hub), an energy innovation hub sponsored by the U.S. Department of Energy under Award Number DE-EE0004261. Any opinions, findings, conclusions, or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the EEB Hub, an energy innovation hub and the Department of Energy.

9. REFERENCES

Adachi, Y. (2003). Overview of partial model query language. In *ISPE CE* (pp. 549-555).

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). Compilers, Principles, Techniques. Addison wesley.

Backus, J. W. (1959). The syntax and semantics of the proposed international algebraic language of the Zurich ACM-GAMM conference. *Proceedings of the International Comference on Information Processing*, 1959.

Bazjanac, V. (2001). Acquisition of building geometry in the simulation of energy performance. *Proceedings of the 2001 Building Simulation Conference*, August 2001.

Beetz, J., van Berlo, L., de Laat, R., & van den Helm, P. (2010). BIMserver. org–An open source IFC model server. In *Proceedings of the CIP W78 conference*.

Berlo, L. v. (2012). First release candidate version 1.2. <http://bimserver.org/2012/12/20/first-release-candidate-version-1-2/> (Jan. 20, 2014).

Berlo, L. v., and Laat, R. d. (2010). Integration of BIM and GIS: The development of the CityGML GeoBIM extension. *Proceedings of the 5th International 3D GeoInfo Conference*.

Buddrus, F., & Schödel, J. (1998). Cappuccino—A C++ to Java translator. In *Proceedings of the 1998 ACM symposium on Applied Computing* (pp. 660-665). ACM.

buildingSMART International Ltd. (2014a). "Industry Foundation Classes (IFC) data model." < http://www.buildingsmart.org/standards/ifc> (Apr. 14, 2014).

buildingSMART International Ltd. (2014b). "Model View Definitions (MVD)" < http://www.buildingsmart.org/standards/mvd/model-view-definitions-mvd> (Apr. 14, 2014).

Cagle, K. (2010). "Why Data as a Service Will Reshape EAI." <http://www.devx.com/enterprise/article/44245> (Jul. 09, 2012).

Chen, Y. (2011). A conceptual framework for a BIM-based collaboration platform supported by mobile computing. *Applied Mechanics and Materials* (Vol.94–96).

Dan, A., Johnson, R., and Arsanjani, A. (2007). Information as a service: Modeling and realization. *Proceedings of the 2007 International Workshop on Systems Development in SOA Environments* (SDSOA'07).

Daum, S., & Borrmann, A. (2014). Processing of Topological BIM Queries using Boundary Representation Based Methods. Advanced Engineering Informatics, 28(4), 272-286.

DeGraw, J. W. and Bahnfleth, W. P. (2010). Automated Evaluation of Potential Collective Protection Locations Using Multizone Modeling. *Indoor Air 2011*, Austin, Texas.

Dwivedi, V. and Kulkarni, N. (2008). Information as a service in a data analytics scenario: A case study. *Proceedings of the 2008 IEEE International Conference on Web Services*.

gbXML.org. (2014). "About gbXML" < http://www.gbxml.org/aboutgbxml.php> (Aug. 27, 2015).

Hijazi, I., Ehlers, M., Zlatanova, S., Becker, T., and Berlo, L. v. (2010). Initial investigations for modeling interior utilities within 3D Geo context: Transforming IFC-interior utility to CityGML/UtilityNetworkADE. *Proceedings of the 5th 3D GeoInfo Conference*.

Jiang, Y., Ming, J., Wu, D., Yen, J., Mitra, P., Messner, J. I., and Leicht, R. M. (2012). BIM server requirements to support the energy efficient building lifecycle. *Proceedings of 2012 ASCE International Conference on Computing in Civil Engineering*, ASCE, Reston, VA, 365-372.

Kim, S.-A., Choe, Y., Jang, M., and Seol, W. (2011). Design process visualization system integrating BIM data and performance-oriented design information. *Proceedings of the 28th International Symposium on Automation and Robotics in Construction*, 2011.

Lee, S., Liu, Y., Chunduri, S., Solnosky, R. L., Messner, J. I., Leicht, R. M., and Anumba, C. J. (2012). Development of a process map to support integrated design for energy efficient buildings. *Proceedings of 2012 ASCE International Conference on Computing in Civil Engineering*, ASCE, Reston, VA, 261-268.

Liu, F., Jallow, A. K., Anumba, C. J., and Wu, D. (2013). Building knowledge modeling: Integrating knowledge in BIM. *Proceedings of the 30th International Conference on Applications of IT in the AEC Industry (CIB W78 2013)*, Beijing, China.

Liu, F., Jallow, A. K., Anumba, C. J., and Wu, D. (2014). A framework for integrating change management with building information modeling. *Proceedings of the 15th International Conference on Computing in Civil and Building Engineering,* Orlando, FL.

Mazairac, W., & Beetz, J. (2013). BIMQL–An open query language for building information models. Advanced Engineering Informatics, 27(4), 444-456.

National Institute of Building Sciences. (2014). "Frequently Asked Questions About the national BIM Standard-United StatesTM." < http://www.nationalbimstandard.org/faq.php> (Apr. 14, 2014).

NIST, (2013). NIST Multizone Modeling Website. <http://www.bfrl.nist.gov/iaqanalysis/> (Jul. 09, 2012).

Olson, J. A. (2009). Data as a service: Are we in the clouds? *Journal of Map & Geography Libraries: Advances in Geospatial Information*, 6(1), 76–78.

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

Qi, J., Issa, R. R., Hinze, J., and Olbina, S. (2012). Fall hazard checking tool based on BIMserver. *Proceedings of 2012 ASCE International Conference on Computing in Civil Engineering*, ASCE, Reston, VA, 357-364.

Singh, V., Gu, N., and Wang, X. (2011). A theoretical framework of a BIM-based multi-disciplinary collaboration platform. *Automation in Construction*, 20(2), 134–144.

Steinberg, D., Budinsky, F., Paternostro, M., and Merks, E. (2008). EMF: Eclipse Modeling Framework. 2nd ed. Addison-Wesley Professional.

Truong, H. L., and Dustdar, S. (2009). On analyzing and specifying concerns for data as a service. *Proceedings of the 2009 IEEE Asia-Pacific Services Computing Conference*.

Yu, N., Jiang, Y., Luo, L., Lee, S., Jallow, A. K., Wu, D., Messner, J. I., Leicht, R. M., and Yen, J. (2013). Integrating BIMserver and OpenStudio for energy efficient building. *Proceedings of 2013 ASCE International Conference on Computing in Civil Engineering*, ASCE, Reston, VA, 516-523.

Zakai, A. (2011). Emscripten: an LLVM-to-JavaScript compiler. In *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (pp. 301-312). ACM.