Visualization in Engineering

RESEARCH Open Access

Data retrieval from building information models based on visual programming

Cornelius Preidel*, Simon Daum and André Borrmann

Abstract

Background: With the rising adoption of Building Information Modeling (BIM) in the AEC sector, computational models supersede traditional ways of information provision based on textual documents and two-dimensional drawings. The use of models enables the streamlining of workflows, and the included virtual construction increases the quality of the final product, the building. To create a comprehensive description of a planned building, information from different sources must be combined, specified and regularly updated by the project's stakeholders. The emerging models are highly structured, and instance files entail large amounts of data. However, in an unprocessed state, these models are of limited suitability for performing engineering tasks as the amount and structure does not match the domain-specific and purpose-oriented views.

Methods: Selection and filtering data for the user's needs is a well-understood task in computer science, and various approaches are available. A promising approach is the usage of formal query languages. In this paper, selected common query languages are examined and assessed for processing building model information. Based on the analysis, we come to the conclusion that textual query languages are too complex to be employed by typical end users in the construction industry such as architects and engineers.

Results: To overcome this issue, two Visual Programming Languages representing a new, more intuitive mechanism for data retrieval are introduced. The first one, QL4BIM, is designed for general filtering of IFC models, the second one, VCCL, has been developed for Code Compliance Checking. Both languages provide operators based on the Relational Algebra to allow handling of relations - a highly required feature of BIM QLs.

Conclusions: The paper concludes with a discussion of the strengths and limitations of visual programming languages in the BIM context.

Keywords: Building information modeling, Information retrieval, Relational algebra, Visual programming

Background

The application of Building Information Modeling (BIM) becomes an increasingly standard practice in the AEC sector. In a BIM environment, planning and construction decisions are increasingly made on the basis of a virtual building model which comprises geometric as well as semantic data. This model is used as the primary description of the to-be-built building. Due to the complexity of the considered field and a large number of involved stakeholders it is common practice to divide the overall model into several sub-models, each representing a domain specific view (Eastman et al. 2008; Preidel et al. 2016).

The benefit of BIM is not only its advantageous data representation but also the increased overall process quality since domain experts can perform their tasks more efficiently based on the information provided by the model. However, to enable architects and engineers using model information as a reliable starting point for their duties, it has to be prepared in such a way that it meets the individual requirements of the respective process. For this reason, BIM data has to be available in different forms, e.g. levels of development, model views or data formats. For example, for the quantity take-off, for the verification of the model quality and for the optimization of the design planning, different information must be extracted from one or more sub-models. For demanding tasks, such as the calculation of a

* Correspondence: cornelius.preidel@tum.de
Chair of Computational Modeling and Simulation, TUM Department of Civil, Geo and Environmental Engineering, Arcisstraße, 2180333 Munich, Germany

pedestrian evacuation scenario, the raw data provided by the building model is mostly not sufficient. At this point, further information must be derived to be able to make statements about the safety in evacuation situations. As a consequence, these pre-processing steps are a major challenge in the daily work of the domain experts as the available methods (textual query languages) do not enjoy a high level of user acceptance (Lee et al. 2015).

From the perspective of computer science, handling and analyzing extensive data sets has many applications and various methods are in use. Relevant approaches are information retrieval, information filtering, and knowledge discovery in databases (Belkin and Croft 1992; Liao et al. 2012). These methods deal primarily with unstructured data such as text documents. In contrast, building models are highly structured instances of a well-defined data schema. As a general approach for retrieving the desired subset of information, query languages are deployed for processing this kind of structured data.

In this contribution, three well-known general query languages are examined for their ability to filter BIM models. Also, a domain-specific language is discussed. To identify the unique characteristics of the different query languages, this comprises the implementation of a set of sample queries in each language. Doing so, conceptual and technical aspects are considered, especially the association of the language to the Relational Algebra, since it plays a prominent role in various approaches. Also, the effort to use the languages is considered. As engineers and architects execute BIM filtering, this analysis is performed from the viewpoint of a non-professional programmer.

To achieve a high level of acceptance, a BIM query language must be able to express comprehensive queries which are suitable for the various domains and be applicable for the end users. Taking the assessments of the reviewed query languages into account, two Visual Programming Languages (VPL) which were developed particularly for BIM data retrieval are presented: ^{v}QL4BIM and VCCL. The visual Query Language for 4D Building Models (^{v}QL4BIM) has been developed for the general filtering of IFC models. The Visual Code Checking Language (VCCL) is deployed for Code Compliance Checking purposes.

The aim of this paper is to demonstrate how the principle of Visual Programming (VP) can be used in various fields of application in the construction industry, to make everyday information retrieval tasks easily manageable for engineers and architects, even if they do not possess well-founded programming knowledge.

The paper is structured as follows. In Filtering BIM data with standard query languages section sample queries are introduced and their implementation in SQL, XQuery, SPARQL and the textual QL4BIM (^{t}QL4BIM) is presented. Methodology discusses the methodical basis of both, VP and Relational Algebra. To illustrate practical applications of the presented methodology, the sample queries are shown in ^{v}QL4BIM and VCCL in the following section. The paper concludes with a discussion of the benefits and the limitations of visual programming languages. To provide an overview, all query languages discussed in this article are listed in Table 1.

Filtering BIM data with standard query languages

Building Information Modeling (BIM) is based on the use of comprehensive digital representations of buildings. A BIM model includes typed entities with their attributes and referenced geometric shapes. Furthermore, the model comprises relationships between its entities in an explicit manner and thus describes structures and dependencies (see Fig. 1). Parts of the non-geometric information are considered as additional dimensions (4D and 5D BIM). The notion of a 4D BIM refers to the inclusion of temporal data for construction scheduling, whereas 5D models additionally include cost information for cost estimation and accounting.

As the Industry Foundation Classes (IFC) is the most established and most comprehensive vendor-neutral schema for building information models, we focus on this data model in our discussions regarding data analysis and filtering.

The IFC data model is defined using the object-oriented modeling language EXPRESS. IFC instance data is typically transported using the STEP Part 21 format. However, these technologies stem from the 1990s and provide only little support for querying and analyzing large data sets. For example, a STEP-based query language has never become established. To overcome this issue a number of researchers aimed at making the data provided by IFC available in other data formats. As a result, the IFC data model can be mapped to a relational database and converted to an Extensible Markup Language (XML) or Resource Description Framework (RDF) representation (BuildingSmart 2017; Beetz et al. 2009b). Each of these data schemas allows for filtering by a query language (QL): SQL in the case of relational databases, XQuery in the case of XML data and SPARQL in the case of RDF data. To examine whether these query languages are an appropriate tool for domain experts to filter IFC models, three sample queries are defined here and their implementation in the different languages is analyzed. In addition, the samples are realized in the domain specific (^{t}QL4BIM) language, which was tailored for processing IFC data. The sample queries are listed below; the corresponding short names used throughout the remainder of the paper are stated in brackets.

Table 1 Selected Query Languages, which can be used for the Data Retrieval of Building Information Models

Query Languages	User	Application	Programming Paradigm
Textual Query Languages			
SQL	Expert Users	Information Retrieval Queries	Declarative
SPARQL	-"-	-"-	Declarative
XQuery	-"-	-"-	Declarative
tQL4BIM	Engineers, Architects as Domain	ad-hoc Information Retrieval Queries	Imperative Procedural
Visual Query Languages			
vQL4BIM	Engineers, Architects	ad-hoc Information Retrieval Queries	Imperative Procedural
VCCL	Planning Consultants and Building Authority Officers	Information Derivation for Code Compliance Checking	Imperative

1. Find all walls (walls).
2. Find all wall-door pairs where the height of the door should not exceed 2000 mm (wall-door pairs).
3. Find all wall-ceiling pairs which are touching each other. In addition, the wall must be below the ceiling (spatial wall-ceiling pairs).

For the sake of completeness, it must be mentioned, that there are also several other domain-specific approaches such as PMQL (Adachi 2002), BERA (Lee 2011) or BIMQL (Mazairac and Beetz 2013; Mazairac 2015) which are not discussed in detail in this paper. Further approaches, which also relate to a textual-based data retrieval from BIM models, were introduced by Liu (Liu et al. 2017), Pauwels (Pauwels et al. 2016) or Zhang (Zhang and Beetz 2016).

Structured query language

The Structured Query Language (SQL) is the current de-facto standard for queries in relational databases (Codd 1991). The declarative language has been available as an ISO standard since 1989 and has been continuously developed further (Islam et al. 2015; ISO 2011). Despite the standardization of the language, relational databases usually use slightly different SQL dialects. Within this work, the Transact-SQL syntax of the Microsoft SQL Server 2014 is used (MSDN 2014).

To enable the application of SQL for querying IFC data, the schema has to be mapped to a relational model. This transformation is not standardized but has a direct impact on the performance and capability of the query mechanism. Especially, the representation of type hierarchies is expensive. For this contribution, we

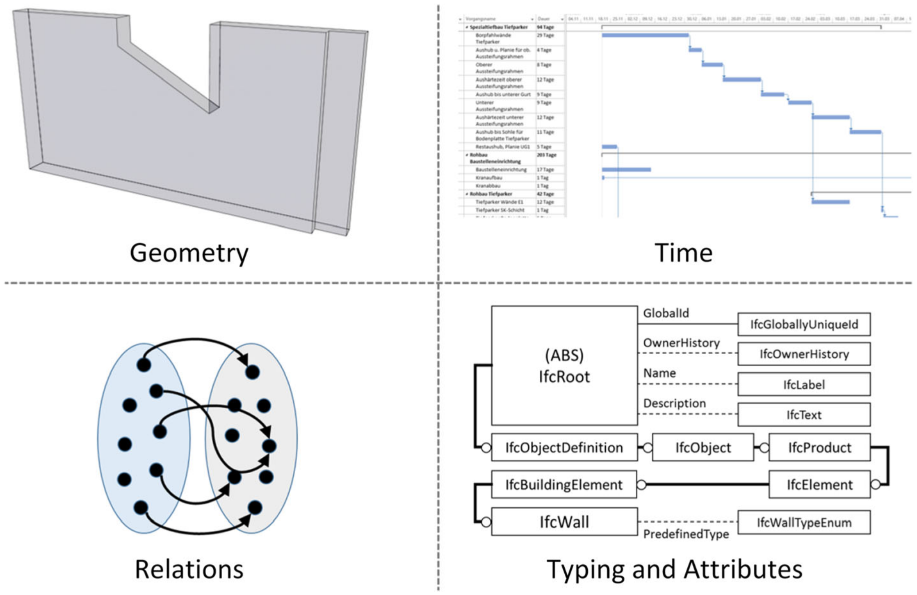

Fig. 1 Different data domains of a building information model

defined the mapping such that each non-abstract IFC class is represented in a single relation that combines all attributes from all super-classes. Other approaches use merged relations for several types or more fine-granular relations and on-thefly joining (Scott 2000). The translated sample queries are listed in Listing 1 and 2. Due to lack of spatial operators, sample query 3 (spatial wall-ceiling pairs) cannot be realized in SQL.

```
SELECT * FROM IfcWallStandardCase;
```

Listing 1: Sample query 1 (walls) in Transact-SQL

```
SELECT Wall.*, Door.* FROM IfcRelVoidsElement RelV
JOIN IfcWallStandardCase Wall ON RelV.RelatingBuildingElement =
    Wall.Id
JOIN IfcOpeningElement Op ON RelV.RelatedOpeningElement = Op.Id
JOIN IfcRelFillsElement RelF ON RelF.RelatingOpeningElement = Op.
    Id
JOIN IfcDoor Door ON RelF.RelatedBuildingElement = Door.Id
WHERE Door.OverallHeight >= 2000;
```

Listing 2: Sample query 2 (wall-door pairs) in Transact-SQL

The SQL language implements the Relational Algebra in a declarative manner by providing the SELECT-FROM-WHERE schema. While the SELECT statement defines a projection, the FROM clause specifies the relations the operations are applied on. The WHERE part is used to define a selection predicate. The JOIN statement is used to implement the formal join operation of the Relational Algebra.

SPARQL

The SPARQL Protocol and RDF Query Language (SPARQL) is a graph-based query language for the Resource Description Framework (RDF), which was developed in the course of the evolution of the Semantic Web. In 2013, SPARQL was adopted by the World Wide Web Consortium (Harris and Seaborn 2013; W3C 2015). The Web Ontology Language (OWL) is based on RDF and provides means to represent ontological data (Beetz et al. 2007). In general, ontologies are used to provide machine-readable and interpretable descriptions that can be defined as a formal, explicit description of a shared management concept (Gruber 1993). Ontological systems allow for computational reasoning, i.e. the inference of new facts from given knowledge and rules. Furthermore, ontological approaches can be used to link instances of different data models with each other (Vilgertshofer et al. 2017). For both querying RDF and OWL, the query language SPARQL can be applied.

An instance file in the RDF model is a set of triples, all of which are composed of a subject, a predicate, and an object. Because each triple is a member of a relation, the resulting structure is referred to as an RDF graph. The triples are termed statements.

SPARQL is a declarative language and uses an SELECT-WHERE expression in combination with triple-patterns. These are used as filters in the WHERE part and can include variables and constants for subjects, predicates, and objects. Using FILTERs, triples can be restricted in a sophisticated way, e.g., by applying arithmetic expressions. The basic data type of the language is a triple.

EXPRESS modeling represents classes, attributes, and relations by different expressions. In RDF, statements are the only pattern available. Therefore, IFC data with all its aspects has to be transferred to RDF statements. For the analysis undertaken in this contribution, the IFC-to-RDF conversion tool is used (Törmä et al. 2014). Alternatively, the ifcOWL representation may be used (Beetz et al. 2009a).

The sample queries expressed in SPARQL are listed in Listing 3 and 4. Sample query 3 (spatial wall-ceiling pairs) cannot be expressed by SPARQL.

```
PREFIX lbd: <http :// linkedbuildingdata .net/ schema / IFC4 #>
SELECT ? wall ? predicate ? wallprop
WHERE {? wall rdf: type lbd: IfcWallStandardCase ;
            ? predicate ? wallprop .}
```

Listing 3: Sample query 1 (walls) in SPARQL

```
SELECT ? wall ? door
WHERE { ? relV a lbd: IfcRelVoidsElement .
        ? wall a lbd: IfcWallStandardCase .
        ?op a lbd: IfcOpeningElement .
        ? relF a lbd: IfcRelFillsElement .
        ? door a lbd: IfcDoor .
        ? relV lbd : relatingBuildingElement ? wall;
        lbd: relatedOpeningElement ?op .
        ? relF lbd : relatingOpeningElement ?op;
        lbd : relatedBuildingElement ? door .
        ? door lbd : overallHeight ?he .
        FILTER (? he >= 2000) .}
```

Listing 4: Sample query 2 (wall-door pairs) in SPARQL

XQuery

XML is today's de-facto standard language for representing structured data. In most cases, XML-Schema is used to define an object-oriented data schema as basis for well-defined data exchange using XML instance files. XML and XML-Schema are also in widespread use in BIM environments. For the IFC schema, a standardized mapping to XML is available (ifcXML), and the concept is implemented in several BIM tools (Liebich 2001). It is also possible to convert regular IFC instance files (STEP Part 21) to ifcXML instances using corresponding tools.

For analyzing and filtering XML data, the XQuery language is well established and defined as W3C standard (W3C 2017). XQuery is a declarative, functional language. Its main pattern is the FLOWR clause which may comprise *For*, *Let*, *Where*, *Order by*, and *Return* expressions. The primary data type of the language is the sequence (ordered list).

The sample queries in XQuery are listed in Listing 5 and 6. Sample query 3 (spatial wall-ceiling pairs) is not supported by XQuery.

```
let $uos := db:open("sample","sample.ifcxml")/iso_10303_28/uos
for $wall in $uos/IfcWallStandardCase
return $wall
```

Listing 5: Sample query 1 (walls) in XQuery

```
for $relV in /iso_10303_28/uos/IfcRelVoidsElement,
     $wall in /iso_10303_28/uos/IfcWallStandardCase,
     $op in /iso_10303_28/uos/IfcOpeningElement,
     $relF in /iso_10303_28/uos/IfcRelFillsElement,
     $door in /iso_10303_28/uos/IfcDoor
where  $relV/RelatingBuildingElement/IfcWallStandardCase/@ref =
     $wall/@id and
          $relV/RelatedOpeningElement/IfcOpeningElement/@ref = $op/
               @id and
          $relF/RelatingOpeningElement/IfcOpeningElement/@ref = $op/
               @id and
          $relF/RelatedBuildingElement/IfcDoor/@ref = $door/@id and
          $door/OverallHeight >= 2200.
return <Pair>{$wall, $door}</Pair>
```

Listing 6: Sample query 2 (wall-door pairs) in XQuery

ᵗQL4BIM

The Query Language for Building Information Models (QL4BIM) is a domainspecific query language that can be applied to the analysis and processing of building information models (Daum and Borrmann 2014). It directly supports the IFC EXPRESS Schema and is available in a visual and a textual notation. A textual ᵗQL4BIM query is an imperative program consisting of a sequence of statements. A statement is typically composed of a variable assignment and an operation call including the handover of an ordered list of parameters. The high-level data types of the language are the relation and the set.

In contrast to the previously discussed languages, QL4BIM is is an imperative language, i.e. a QL4BIM program consists of a number of commands for the computer to perform sequentially. More details of QL4BIM are discussed in Visual Information Queries - vQL4BIM.

The sample queries are listed in Listing 7 and 8. As QL4BIM provides spatial operators for topological, directional and metric analysis, sample query 3 can be realized in a straight-forward manner (Listing 9).

```
entities = ImportModel("C:\institute.ifc")
walls = TypeFilter(entities, #IfcWall)
```

Listing 7: Sample query 1 (walls) in ᵗQL4BIM

```
entities = ImportModel("C:\institute.ifc")
entity-element = Deassociater(entities, .HasOpenings, .HasFilling)
wall-door = TypeFilter(entity-element, #IfcWall, #IfcDoor)
wall-highDoor = AttributeFilter(door, .OverallHeight >= 2000)
```

Listing 8: Sample query 2 (wall-door pairs) in ᵗQL4BIM

```
entities = ImportModel("C:\institute.ifc")
slabs = TypeFilter(entities, #IfcSlab)
walls = TypeFilter(entities, #IfcWall)
slab-tochingWall = Touch(slabs, walls)
```

Listing 9: Sample query 3 (spatial wall-ceiling pairs) in ᵗQL4BIM

Methods

In the previous chapter, four samples of textual query languages were listed and presented by their application for analysis and filtering BIM data. Despite their diverging paradigms and levels of abstraction, these languages have in common, that their users have to deal with textual programming principles. In many cases, this presents a major hurdle for architects and engineers, and adoption in practice has been very low in the past.

At the same time, we could observe in recent years that the concept of VP has become more and more popular in the architecture and engineering domain and its actual use is very widespread, especially in the context of BIM tools and projects. The most prominent examples are Dynamo for Revit and Grasshopper for Rhinoceros 3D.

We assume that the reason for this success is that the entry level is much lower for non-professional programmers, as VP is much more intuitive and error-tolerant than conventional textual programming languages. Furthermore, the visual formulation of queries is largely independent of the data source to which it relates since the environments for visual environments can be adapted very easily. This means that this paradigm can be applied on multiple sources of data, including relational databases, ontology-based ones, or others.

Consequently, we propose the use of a VPL for formulating BIM queries in order to overcome the low intuitiveness of existing query languages and make formal query functionalities accessible to the end users. In the following sections, the methodical principles of VP are discussed.

Visual programming languages

In general, a visual language is defined as a *formal language with visual syntax and semantics (*Myers 1990*;* Hils 1992*;* Erwig et al. 2017*)*. This means that such a language describes a system of signs and rules on the syntactic and semantic level with the help of visual elements. By the visual presentation of the elements, the language may be interpreted more quickly and easily. In the following, the different approaches of visual languages are presented. This categorization is based on (Schiffer 1998) and (Burnett and Baker 1994).

Control flow systems transfer the imperative programming paradigm to VP. The control flow within a program is expressed by explicit commands. For a visual representation, Nassi-Shneiderman diagrams or Petri nets can be used.

Data flow systems are related to functional systems and do not assign a strict sequence of operations. Instead, the availability of data triggers the execution of operations. This trait can be used to realize a parallel execution model for corresponding programs.

Functional systems are more strict in the dependency to the functional programming paradigm. They are based on higher-order and polymorphic functions and an implicit type system. For the visual representation, function diagrams can be applied.

Object-oriented systems act on a class library from which instances are created. Following the object-oriented paradigm, these instances communicate among themselves by sending and receiving messages. While these systems provide encapsulation, all other object oriented characteristics (inheritance, polymorphism) are typically not realized.

Constraint-based systems represent programs as constraints between variables and constants. The formal concept is known as Constraint Logic Programming. Variable assignments, procedures, and control structures are not used here. Instead, a constraint solver finds a valid configuration of all variables.

Transformation based systems uses graph grammars or graph rewriting systems. In this approach, programs are represented as graphs and transformations, which act on these graphs.

Example-based systems exist in two variants. The first is based on programming with examples. Programs are created by capturing and re-coding interactions of a user with an application. The well-known implementation of this approach is a macro recorder. In contrast, programming by examples generalizes the recorded user interactions based on their semantics. This requires components for artificial intelligence computing in these systems.

Form-based systems represent programs by forms and tables. Scalar and structured data together with formulas can be stored in the system's cells. A formula can reference other cells. In this way, the content of a cell acts as the input for a formula. If the content changes, the system automatically triggers a recalculation of dependent cells. As control structures are not used, a form-based system realizes a declarative programming style.

A VPL has a high degree of user-friendliness and typically lowers the hurdle for normal end-users to enter the field of programming. Thanks to its visual representation and means for intuitive interaction, VPL programs are simpler to create and to understand than its textual counterpart. The physiological background is that humans perceive images directly by their visual cortex while written text requires an additional level of perception and reasoning (Shu 1988). A study by Catarci and Santucci (Catarci and Santucci 1995) proves the user-friendliness of visual languages by discussing an example based on SQL.

Despite its advantages, the use of a VPL is discussed controversially. As a disadvantage it is usually stated that the programs created with a VPL often do not meet the high requirements for professional software development. In particular, more complex control flows, such as loops or recursions, can often not be implemented or are difficult to understand. A general argument is that users who are applying the visual language should also be able to describe the information process with a conventional programming language.

In digital construction, VPLs are mainly used in two application areas: (1) for generating geometric as well as semantic information or (2) for checking or querying existing models. However, for some VPL-based applications and environments, these boundaries are fuzzy and a precise classification is not possible. Most of the VP environments provide opportunities for developers to extend the libraries by user-defined functions and so the application area can be extended. Doing so, the design and functionality of a VPL can be decisively influenced by its application area. Current representatives of such VPLs are shown in Table 2.

Known software products in the context of building design are in particular the plug-in Grasshopper for Rhinoceros3D, Dynamo for Autodesk Revit (also available as a standalone application) and Marionette for Vectorworks. Although these representatives focused initially especially on 3D parametric modeling, they

Table 2 Selected VPL environments and libraries

Name	Application	Manufacturer	Programming Interface	URL
Dynamo	Standalone; Autodesk Revit Plug-In	Autodesk	C#, IronPython	http://dynamobim.org/
Google Blocky	Web-Based	Google	JavaScript	https://developers.google.com/blockly/
Grasshopper3D	Rhinoceros3D Plug-In	Open-Source	C++, C#	http://www.grasshopper3d.com/
Grasshopper3D ArchiCAD	ArchiCAD Plug-In	Graphisoft	C++, C#	https://www.graphisoft.com/archicad/rhino-grasshopper/
Marionette	Vectorworks Plug-In	Vectorworks	Python	http://www.vectorworks.net/training/marionette
Scratch	Web-based	MIT	JavaScript	https://scratch.mit.edu/
TUM.CMS. VplControl	Standalone	CMS Chair	C#	https://github.com/tumcms/TUM.CMS.VPLControl

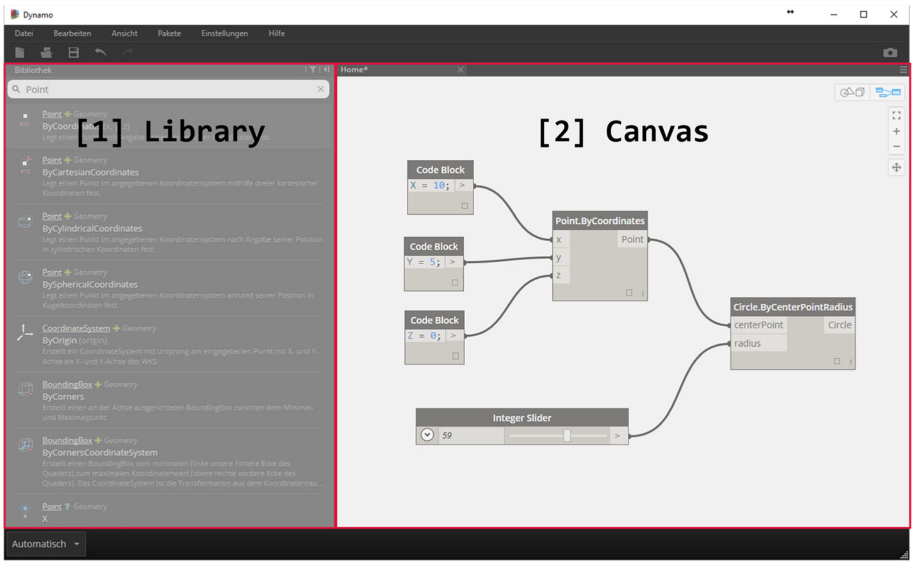

Fig. 2 Typical environment of a VPL: (Adachi 2002) Library containing the usable node elements and the (Beetz et al. 2009a) work-space canvas

were significantly extended by additional features through third-party communities supporting these projects. However, as the existing VPL systems in AEC industry focus on tasks like architectural and geometric modeling, the language is primarily designed to handle intuitive design tasks.

For the application of these visual languages usually a canvas is provided for the user as a basic work-space. On the canvas, the individual components (nodes) can be arranged and linked to each other by directed edges (also denoted as wires). The different functions are usually offered in a library and can be used for the composition of the intended information processing system (see Fig. 3). The resulting visual program can be externalized using a graph representation and passed on to other project participants or archived accordingly (Fig. 2).

Practical experiences in visual programming in teaching

Since the retrieval and extraction of BIM information is an important topic, it is a mandatory part of the BIM-related lectures at the Technical University of Munich. In theoretical as well as practical exercises the students learn how to use different query languages including VP tools. In the course of the lectures given 2011-2017, we have been observing that students from the fields of civil engineering, environmental engineering, and architecture can deal more easily with VPLs than with textual programming tools. This observation does not only apply to intuitive tasks like the geometric modeling of complex structures, but also for tasks related to the evaluation of semantic information, e.g. quantity take-off. Across various disciplines and applications, we observe that students finish tasks easier with the help of VP tools, especially if they have no or little programming skills.

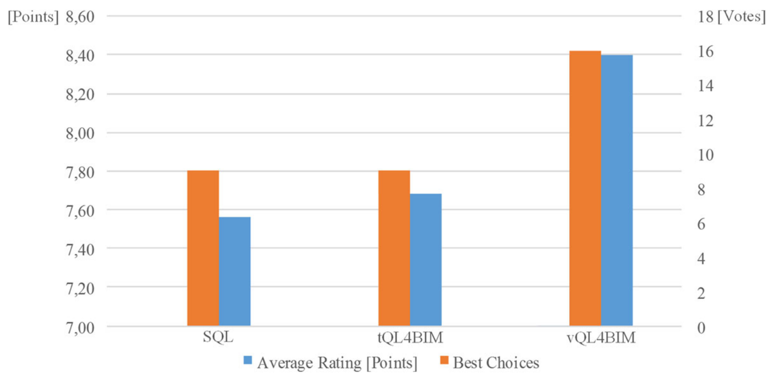

Fig. 3 Results of the survey with students on the use of VPLs for the data retrieval of building models. The main criteria for this survey are the "Average Rating [Points]" and the "Best Choice" of the different query candidates by the students

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline A & 1 & 0 & 0 & 1 & 0 & 1 \\ B & 0 & 1 & 0 & 0 & 1 & 0 \\ C & 0 & 0 & 1 & 0 & 0 & 1 \\ D & 1 & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 0 & 1 & 0 \\ F & 1 & 0 & 1 & 0 & 0 & 1 \end{array}$$

1	{A,D,F}
2	{B,E}
3	{C,F}
4	{A,D}
5	{B,E}
6	{A,C,F}

	1	2	3	4	5	6
A	X			X		X
B		X			X	
C			X			X
D	X			X		
E		X			X	
F	X		X			X

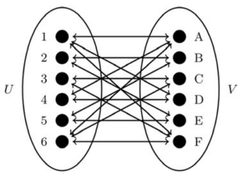

Fig. 4 Different schematic representations of a relation (Kemper and Eickler 2011)

So far, there are no reliable experience or survey results on the use of VPLs in digital construction industry. Therefore, to prove this observation, we conducted a study in the BIM lecture 2015/2016. For this purpose, we created a questionnaire which contains a series of sample information retrieval requests and the related translations encoded by (1) SQL, (2) textual as well as (3) visual QL4BIM. The students were asked to rate the different encoded queries according to their readability and intelligibility with a score from 1 to 10. The result of this survey is shown in Fig. 3.

In the survey, the students have rated the VPL as the best, however by a small margin only. At this point, the VPL can not be seen as clearly superior. However, if we consider how many students have chosen the visual query as a best choice, we can see a clear trend towards the VPL. This study is the first series of experiments which we intend to expand in future. In this way, we want to obtain further representative results and also document trends as well as changes in user behavior related to programming.

Relational algebra

The mathematical concept of a relations plays an important role for representing and querying highly structured and inter-related data as typically provided by BIM models. In fact, the most wide-spread type of Database Management Systems (DBMS) are those that are based on relations and are denoted as Relational DBMS. But also semantic web technologies such as RDF and OWL are primarily based on relations. The basis for the querying relational data sets is the Relational Algebra. In terms of the mathematical definition, a relation R represents the subset of the Cartesian Product of n defined domains (Kemper and Eickler 2011):

$$R = D_1 \times D_2 \times \cdots \times D_n \tag{1}$$

More vividly, a relation can be described as a set of n-tuples where each tuple combines those objects that are in a particular relation with each other. Thus, a relation assigns individual elements of a set to the items in another set and, therefore, provides a definite mathematical relationship. The results can be presented graphically in different ways (see Fig. 4).

The Relational Algebra defined by (Codd 1970; Codd 1991) allows to operate on, process and analyze a set of relations. The concept is well-known from relational databases where the theory of Relational Algebra has been very successfully implemented in the widespread declarative query language SQL. Relational database management systems currently represent the de-facto standard database technology. Because of a strict and clean mathematical base, the main advantage of relational databases is error-free querying and processing of a large amount of information. The basis for such processing is provided by the relational operators, which are schematically shown in Fig. 5.

In many cases, query languages which work with relations or relational operators, follow the declarative

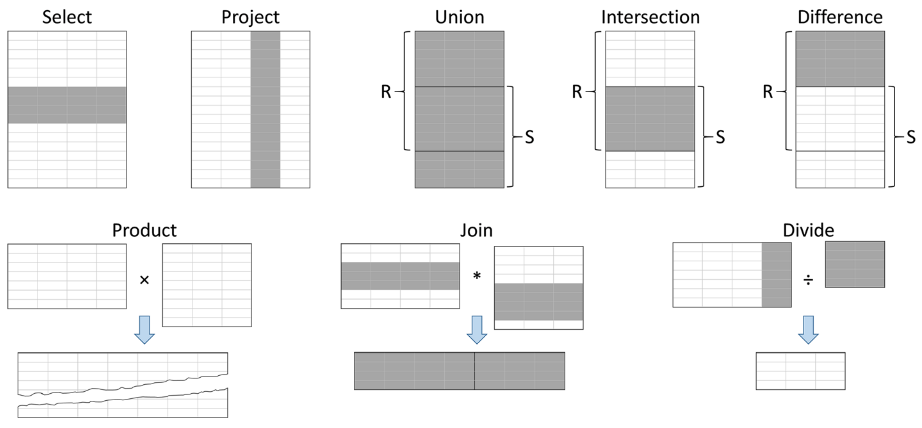

Fig. 5 Basic Operators of the Relational Algebra (Codd 1991)

programming paradigm, such as SQL. By contrast, VPLs are in most cases based on the flow of information, i.e. they implement the imperative procedural paradigm by sequentially executing individual information processing commands. There is a common misunderstanding that relational algebra can only applied in a declarative but not in a imperative programming context. As this is not the case, we will show how operators of the relational algebra can also be integrated in a VPL environment.

Results

In the following, two different VPLs will be presented which have been developed by the authors. Each of theses languages targets a different field of application in the construction industry. The first representative is a visual variant of QL4BIM, denoted as ^{v}QL4BIM, which is applicable for general IFC filtering and processing. To meet the individual requirements of Code Compliance Checking, the Visual Code Checking Language (VCCL) has been developed, a visual language that is suitable for translating the content of guidelines and standards in a digital format. In the following, both languages will be introduced, and their mode of operation is explained using the sample queries.

Visual information queries - ^{v}QL4BIM

The textual and visual notations of QL4BIM are based on the same grammatical foundation. A query is a sequence of statements. Each statement assigns a variable to the result of an operation. Variables are typed as sets and relations of IFC entities. Each operation is composed of an operator and parameter handovers. If variables are used as parameters, they represent the current data pool on which the operation operates on. Listing 10 shows the central part of the grammar of the language in Extended Backus-Naur Form (EBNF). For ^{v}QL4BIM, productions and terminal are mapped to corresponding visual elements. Table 3 shows the most important items.

Listing 10: Ql4BIM statement definition

QL4BIM ::= statement* definition* statement ::= variable '=' operation operation ::= operator '(' argument (',' argument)* ')' operator ::= [A-Z] (a-zA-Z0-9)* argument ::= variable | constant | predicate | externalLiteral variable ::= literal | complexLiteral literal ::= [a-z] (a-zA-Z0-9)* complexLiteral ::= literal ('-' literal)+ onstant ::= number | float | string /* 10 additional productions */

Table 3 Assignment of grammatical elements to visual representations

Production	Visual	Production	Visual
operator	[A-Z] (a-zA-Z0-9)*	literal	[a-z] (a-zA-Z0-9)*
comlexLiteral	literal literal	string	[^"]*

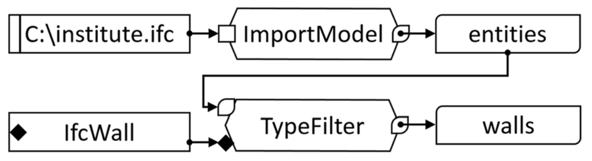

Fig. 6 Sample query 1 (walls) in vQL4BIM

The sample queries in ^{v}QL4BIM are shown in Figs. 6 and 7. As ^{v}QL4BIM uses the identical set of operators as ^{t}QL4BIM, spatial query 3 can be implemented (Fig. 8).

Visual code compliance checking - VCCL

For a building, there is a number of requirements that must be taken into account during the design as well as the construction phase. These requirements define the essential functions and the safety performance of the to-be-built product. The applicable regulations and rules are laid down in international, national and regional standards and guidelines. Their compliance regarding the design planning of the building must be checked continuously by planning consultants as well as building authority officers.

The Visual Code Checking Language (VCCL) is a domain-specific programming language designed for the formulation of checking and verification routines, which are contained in such standards or guidelines. Based on digital building information, the VCCL is intended to perform compliance checks semi or even fully automated, in order to increase the efficiency and quality of the overall process significantly. However, it must be noted that this scope of an application implies special requirements for the design of the VCCL.

As the query languages discussed above, the VCCL also provides functionalities for extracting information from a building model. Also, the data must be pre-processed to fit the requirements of the regulation under consideration. Furthermore, the rules of the regulation have to be represented. For this reason, several operators and methods were introduced for the VCCL, which distinguish the visual

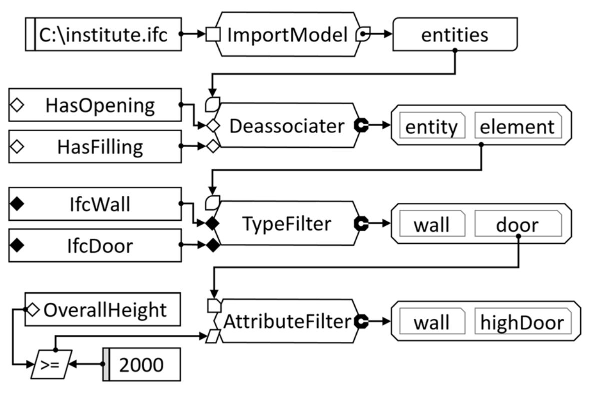

Fig. 7 Sample query 2 (wall-door pairs) in vQL4BIM

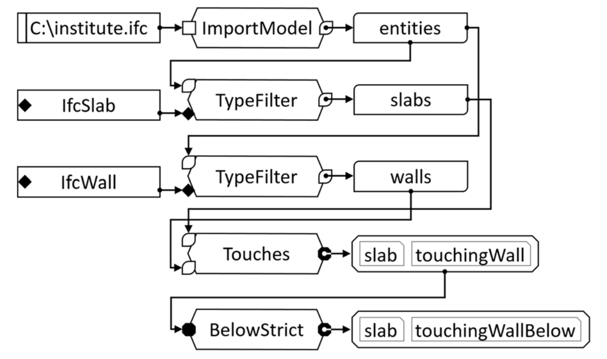

Fig. 8 Sample query 3 (spatial wall-ceiling pairs) in vQL4BIM

language from other VPL representatives. For the description of information processes, the VCCL provides essential elements: methods (rectangularly shaped nodes) representing well-defined operations, ports (input- and output circles forming part of the method shapes) representing the incoming and outgoing data type and directed edges, which connect the methods by linking the ports.

As a starting point, basic methods are provided for the user, which he can use to build up VCCL programs. The provided basic methods describe fundamental operations, whose semantics are unambiguously well defined. These methods are called atomic methods as their implementation is hidden from the user and realized by standard programming languages. Atomic methods include, among others mathematical operators, geometric operators, Relational Algebra operators, and access operators for extracting information from building models. These methods represent a black-box level, from which we assume that it is acceptable for a given application area, where deeper control and insight is neither desired nor necessary, and too much effort would be created for implementing the provided functionality using VCCL instead of a standard programming language (Preidel and Borrmann 2016b). As an example, an acceptable black-box atomic method is the evaluation of geometric information such as the computation of the shortest distance route for a given floor plan. We assume that the end-user does not require to have an insight in the

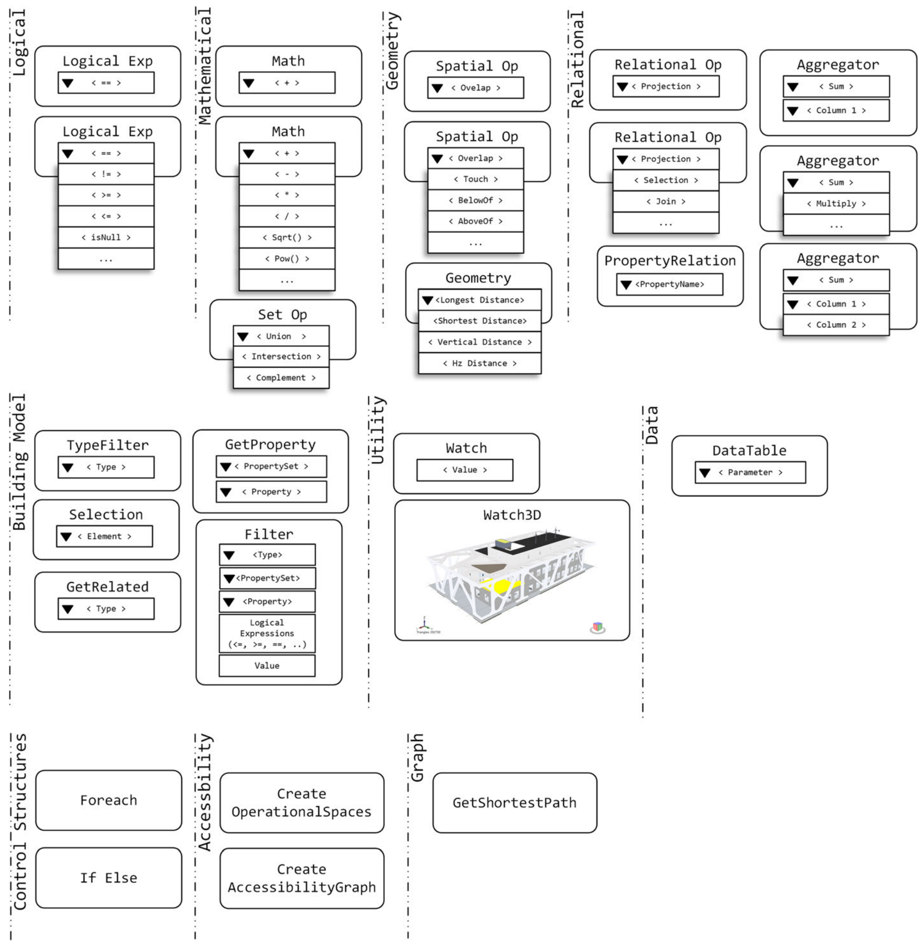

Fig. 9 VCCL atomic methods

algorithms for realizing this functionality, whereas the result of the computation forms an essential part of many international and national guidelines and regulations. The atomic methods of the VCCL are shown in Fig. 9.

Since this paper focuses on the area of data retrieval, in the following, not all operators that can be used for the representation of rule knowledge, are described in detail. For a complete description and listing of all existing operators of the VCCL, the reader is referred to (Preidel and Borrmann 2016b).

The mathematical concept of Relational Algebra has shown to be extremely useful in the context of algorithmic code compliance checking. To check the contents of guidelines, often information and data must not only be extracted from the building model but derived, which means that it takes an intermediate step to process information (Solihin and Eastman 2015). At this point, the relational operators are a valuable tool, because they allow the processing of interrelated elements from different sets of objects. A typical example is a Wall-Void-Window relation which represents objects inherently connected to each other in a straight and mathematically profound way.

The VCCL integrates the possibility to work with relations by providing the fundamental data type *Relation* and providing the operators of the Relational Al-gebra (see Methodology) as dedicated operator nodes of the VCCL (Preidel and Borrmann 2016a). In this respect, it has to be noted that, by contrast to SQL which is declarative language implementing the Relational Algebra, VCCL provides an imperative implementation of the Relational Algebra, i.e. the operators are applied in an procedural manner. This, however, does not at all restrict the applicability of the Relational Algebra. To prove the applicability of the VCCL. Selected passages from various standards and guidelines have already successfully been translated into visual checking programs (Preidel and Borrmann 2015; Preidel and Borrmann 2016a; Preidel and Borrmann 2016b; Preidel and Borrmann 2017). The main objective of the integration of relational operators into the visual code checking environment is that users can use these operators to process information according to their requirements and thus also to check them.

Various methods of the VCCL produce relational output results. Usually, such a relation is created out of at least two input objects by checking if given criteria of the elements of these input sets are met. If the criteria are fulfilled, the items have a relationship and are added to the relation as an *n*-tuple (pair, triple, etc.). The necessary checking processes of such a method must be implemented using a dedicated algorithm, which is typically formulated using a textual notation.

In the following Figs. 10, 11 and 12, the VCCL translations of the sample queries are shown.

The shown sample queries translated with the VCCL, are modularized as systems, which means that they have a clear start and end point. Since the VCCL graph is composed of individual methods and instructions, the user can have an insight at any intermediate step, which is represented by shown support nodes, visualizing geometrical or textual intermediate results. In this way, the overall process is not described as a black box but transparent.

With the help of the VCCL, several test cases have already been implemented, and the results have been promising so far (Preidel and Borrmann 2015; Preidel and Borrmann 2016a; Preidel and Borrmann 2016b). However, there is a variety of regulations which contain different forms of knowledge and rule representation, which must also be covered by the VCCL. The operators of Relational Algebra are extremely helpful in this context, but it requires further development of VCCL elements as well as test scenarios to guarantee a holistic approach.

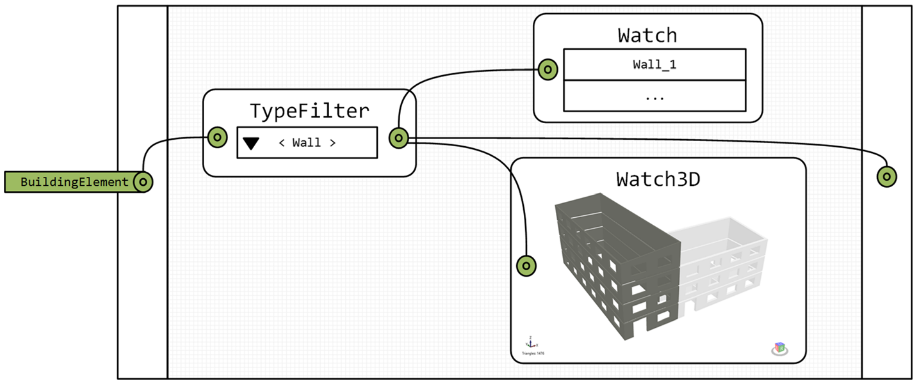

Fig. 10 Encoded sample query 1 (walls) as VCCL graph

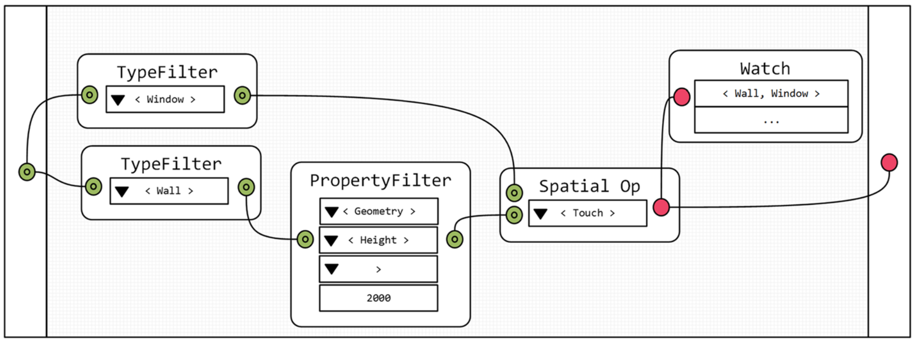

Fig. 11 Encoded sample query 2 (wall-door pairs) as VCCL graph

Conclusions

The retrieval of information from digital building models plays an essential role in the digitization of the construction industry. Although there are already several proposals, which aim to provide tools as domain-specific and simplified query languages, most of them either lack transparency or are too complicated for endusers such as architects and engineers to apply. At this point, VPLs lower the entry hurdle thanks to their intuitive interaction mechanisms. In this paper, we consequently discussed the significant capabilities of visual languages in particular concerning ease-of-use and expressivity. Using the provided example queries we could illustrate the advantages of visual programming over conventional text-based query languages.

We believe visual languages have great potential for data analysis and processing tasks in the context of Building Information Modeling. To prove this, we presented two visual languages we developed for two different use cases in the BIM context: A visual query language ($^{\mathrm{V}}$QL4BIM) and a visual code checking language (VCCL). Both languages have been carefully designed to meet the specific demands of both application domains.

However, significant challenges and limitations of visual languages lie in the representation of more complex queries. More complex control flow schemes, such as iteration loops, as well as the handling of programming errors, are still challenges which we intend to address in future research by various application cases.

It is very likely that the importance of Data Retrieval features will continue to grow, as the increased use of BIM in practice results in more and more engineers and architects applying this method. Furthermore, greater use will make the projects, the digital models and project structures more complex. For this reason, the demand for tools architects and engineers can easily use to extract appropriate information from the models will be higher.

We expect that the use of VP in the various areas of digital construction will increase in next years. To reaffirm these first impressions which point in this direction and since there are no experience results on the use of VPLs so far, we are planning to expand our studies and to receive further reliable survey results.

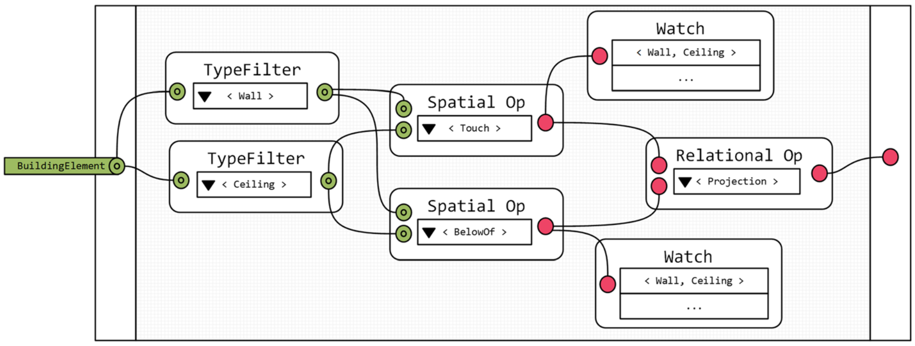

Fig. 12 Encoded sample query 3 (spatial wall-ceiling pairs) as VCCL graph

Abbreviations

AEC: Architecture, Engineering and Construction; BIM: Building Information Modeling; DBMS: Database Management System; IFC: Industry Foundation Classes; OWL: Ontology Web Language; QL: Query Language; QL4BIM: Query Language for Building Information Modeling; RDF: Resource Description Framework; SPARQL: SPARQL Protocol And RDF Query Language; SQL: Structured Query Langauge; STEP: Standard for the exchange of product model data; VCCL: Visual Code Checking Language; VP: Visual Programming; VPL: Visual Programming Language; XQuery: XML Query Language; XML: Extensible Markup Language

Acknowledgements

The authors gratefully acknowledge the support by the Nemetschek Group as well as ALLPLAN GmbH for the research project presented in this paper.

Funding

The research presented was partially funded by the German Research Foundation under grant BO3575/1–1 and by ALLPLAN GmbH, Germany. Their support is gratefully acknowledged. The views and opinions expressed in this article are those of the authors and do not necessarily reflect the views of the funders.

Authors' contributions

CP and SD have cooperated in the literature research. CP is the creator of the VCCL and SD of the QL4BIM system, which are described in this paper. AB has guided the research, contributed to the conception of the languages and, in particular, to the integration of the Relational Algebra. All authors read and approved the final manuscript.

Competing interests

The authors declare that they have no competing interests.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Received: 13 December 2016 Accepted: 14 August 2017
Published online: 11 October 2017

References

Adachi, Y. 2002: Overview of partial model query language . URL http://cic.vtt.fi/projects/ifcsvr/tec/VTT-TEC-ADA-12.pdf.

Beetz, J., de Vries, B., & van Leeuwen, J. P. (2007). RFD-based distributed functional part specifications for the facilitation of service-based architecture. In D. Rebolj (Ed.), *Bringing ITC knowledge to work - 24th W78 conference Maribor* 26.-29.6.200. URL http://itc.scix.net/data/works/att/w78-2007-028-087-beetz.pdf.

Beetz, J., Van Leeuwen, J., & De Vries, B. (2009a). Ifcowl: A case of transforming express schemas into ontologies. *Ai Edam, 23*(1), 89–101.

Beetz, J., van Leeuwen, J., & de Vries, B. (2009b). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing, 23*(01), 89. doi:10.1017/s0890060409000122.

Belkin, N. J., & Croft, W. B. (1992). Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM, 35*(12), 29–38. doi:10.1145/138859.138861.

BuildingSmart (2017): Ifcxml overview. URL http://www.buildingsmart-tech.org/specifications/ifcxml-releases.

Burnett, M. M., & Baker, M. J. (1994). A classification system for visual programming languages. *Journal of Visual Languages & Computing, 5*(3), 287–300. doi:10.1006/jvlc.1994.1015.

Catarci, T., & Santucci, G. (1995). Are visual query languages easier to use than traditional ones? : An experimental proof. In M. A. R. Kirby (Ed.), *People and computers X, Cambridge programme on human-computer interaction*. Cambridge: Cambridge Univ. Pr.

Codd, E. F. (1970). A relational model of data for large shared data banks. *Communications of the ACM, 13*(6), 377–387. doi:10.1145/362384.362685.

Codd, E. F. (1991). *The relational model for database management: Version 2, reprinted with corr edn*. Reading: Addison-Wesley.

Daum, S., & Borrmann, A. (2014). Processing of topological BIM queries using boundary representation based methods. *Advanced Engineering Informatics, 28*(4), 272–286. doi:10.1016/j.aei.2014.06.001.

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2008). *BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors, 1 aufl. Edn*. New York: Wiley.

Erwig, M., Smeltzer, K., & Wang, X. (2017). What is a visual language? *Journal of Visual Languages & Computing, 38*, 9–17. doi:10.1016/j.jvlc.2016.10.005.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition, 5*(2), 199–220. doi:10.1006/knac.1993.1008.

Harris, S., Seaborn, A. (2013): SPARQL 1.1 query language: W3C recommendation. URL http://www.w3.org/TR/sparql11-query/.

Hils, D. D. (1992). Visual languages and computing survey: Data flow visual programming languages. *Journal of Visual Languages & Computing, 3*(1), 69–101. doi:10.1016/1045-926x(92)90034-j.

Islam, M. S., Kuzu, M., & Kantarcioglu, M. (2015). A dynamic approach to detect anomalous queries on relational databases. In *Proceedings of the 5th ACM conference on data and application security and privacy, CODASPY '15* (pp. 245–252). New York: ACM URL http://doi.acm.org/10.1145/2699026.2699120.

ISO (2011). ISO/IEC 9075–1:2011 information technology –database languages – SQL – Part 1: Framework (SQL/framework). International Organization for Standardization.

Kemper, A., & Eickler, A. (2011). *Datenbanksysteme: Eine Einfu¨hrung, 8., aktualisierte und erw. aufl. edn*. Mu¨nchen: Oldenbourg.

Lee, J.K. (2011): Building environment rule and analysis (BERA) language. Ph.D. thesis, Georgia Institute of Technology.

Lee, S., Yu, J., & Jeong, D. (2015). BIM acceptance model in construction organizations. *Journal of Management in Engineering, 31*(3), 04014,048. doi:10.1061/(ASCE)ME.1943-5479.0000252.

Liao, S. H., Chu, P. H., & Hsiao, P. Y. (2012). Data mining techniques and applications – A decade review from 2000 to 2011. *Expert Systems with Applications, 39*(12), 11,303–11,311. doi:10.1016/j.eswa.2012.02.063.

Liebich, T. (2001): Xml schema language binding of express for ifcxml. International alliance for interoperability.

Liu, H., Liu, Y. S., Pauwels, P., Guo, H., & Gu, M. (2017). *Enhanced explicit semantic analysis for product model retrieval in construction industry. IEEE transactions on industrial informatics* (p. 1). doi:10.1109/TII.2017.2708727.

Mazairac, W. (2015): BimQL. URL http://bimserver.org/2012/08/28/new-query-language-bimql/.

Mazairac, W., & Beetz, J. (2013). BIMQL – An open query language for building information models. *Advanced Engineering Informatics, 27*(4), 444–456. doi:10.1016/j.aei.2013.06.001.

MSDN (2014): Transact-sql reference (database engine). URL https://msdn.microsoft.com/en-us/library/bb510741.aspx.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing, 1*(1), 97–123. doi:10.1016/s1045-926x(05)80036-9.

Pauwels, P., de Farias, T. M., Zhang, C., Roxin, A., Beetz, J., & de Roo, J. (2016). Querying and reasoning over large scale building data sets. In: Sven Groppe und Le Gruenwald (Hg.): Proceedings of the International Workshop on Semantic Big Data (SBD 2016). In *conjunction with the 2016 ACM SIGMOD/PODS Conference in San Francisco, USA, July 1, 2016. the International Workshop. San Francisco, California. ACM-Sigmod International Conference on Management of Data; ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems. New York*, New York: The Association for Computing Machinery, S. 1–6.

Preidel, C., & Borrmann, A. (2015). Automated code compliance checking based on a visual language and building information modeling. In *Connected to the future - 32nd international symposium on automation and robotics in construction and mining (ISARC 2015): Oulu, Finland, 15–18 June 2015*. Red Hook: Curran Associates Inc. doi:10.13140/RG.2.1.1542.2805.

Preidel, C., & Borrmann, A. (2016a). Integrating relational algebra into a visual code checking language for information retrieval from building information models. In N. Yabuki & K. Makanae (Eds.), *Proceedings of the 16th international conference on computing in civil and building engineering*. Osaka: ICCCBE. doi:10.13140/RG.2.1.4618.5201.

Preidel, C., & Borrmann, A. (2016b). Towards code compliance checking on the basis of a visual programming language, ITcon Vol. 21, *Special issue CIB W78 2015 Special track on Compliance Checking*, pg. 402-421, http://www.itcon.org/2016/25.

Preidel, C., & Borrmann, A. (2017). Refinement of the Visual Code Checking Language for an Automated Checking of Building Information Models Regarding Applicable Regulations. In: Ken-Yu Lin, Nora El-Gohary und Pingbo Tang (Hg.): Computing in Civil Engineering 2017. *ASCE International Workshop on Computing in Civil Engineering 2017. Seattle*, Washington, June 25-27, 2017. Reston, VA: American Society of Civil Engineers, S. 157–165.

Preidel, C., Borrmann, A., Oberender, C. H., & Tretheway, M. (2016). Seamless integration of common data environment access into BIM authoring applications: The BIM integration framework. In *E-work and E-business in architecture, engineering and construction* (Vol. 11). CRC Pr I Llc. doi:10.13140/RG.2.1.4487.4488.

Schiffer, S. (1998). *Visuelle Programmierung: Grundlagen und Einsatzm¨oglichkeiten*. Bonn: Addison Wesley.

Scott, W.A. (2000): Mapping objects to relational databases. R mapping in detail, practice leader, agile development, IBM, software group.

Shu, N. C. (1988). *Visual programming*. New York: Van Nostrand Reinhold.

Solihin, W., & Eastman, C. (2015). Classification of rules for automated BIM rule checking development. *Automation in Construction, 53*, 69–82. doi:10.1016/j.autcon.2015.03.003.

Törmä, S, VuHoang, N, Pauwels, P (2014). Ifc-to-rdf conversion tool. URL http://linkedbuildingdata.net/tools/tool-ifc-to-rdf-conversion-tool/.

Vilgertshofer, S.; Amann, J.; Willenborg, B.; Borrmann, A.; Kolbe, T. H. (2017): Linking BIM and GIS Models in Infrastructure by Example of IFC and CityGML. In: Ken-Yu Lin, Nora El-Gohary und Pingbo Tang (Hg.): *Computing in Civil Engineering 2017*. ASCE International Workshop on Computing in Civil Engineering 2017. Seattle, Washington, June 25-27, 2017. Reston, VA: American Society of Civil Engineers, S. 133–140.

W3C (2015): Sparql for rdf. URL https://www.w3.org/TR/rdf-sparql-query/.

W3C (2017): World wide web consortium. URL https://www.w3.org/.

Zhang, C., & Beetz, J. (2016): Querying linked building data using SPARQL with functional extensions. In: Symeon Christodoulou (Hg.): eWork and eBusiness in Architecture. ECPPM 2016: *Proceedings of the 11th European Conference on Product and Process Modelling (ECPPM 2016)*, Limassol, Cyprus, 7-9 September 2016: CRC Press.