Integrating Relational Algebra into a Visual Code Checking Language for Information Retrieval from Building Information Models

Cornelius Preidel[1], André Borrmann[2]

1) Ph.D. Candidate, Chair of Computational Modeling and Simulation, Technische Universität München, Germany. Email: cornelius.preidel@tum.de
2) Dr.-Ing., Prof., Chair of Computational Modeling and Simulation, Technische Universität München, Germany. Email: andre.borrmann@tum.de

Abstract:

Standards and guidelines in the construction industry are used to standardize requirements to guarantee framework conditions, such as structural stability, reliability, usability and safety. Therefore, conformity checks of building designs relative to applicable rules and regulations are essential. Currently, these checks are primarily performed manually on the basis of two-dimensional technical drawings and textual documents. Because of the low level of automation, the conventional checking procedure is laborious, cumbersome and error-prone. As the building information modeling process has matured, the construction industry has gained the necessary technology to automate and thereby optimize the checking process in terms of time, effort and cost. To automate this process, we introduce a visual code checking language (VCCL), which uses visual elements to translate standards and guidelines into machine-readable language (Preidel & Borrmann, 2015). Similar to other conventional languages, the VCCL is subject to the rules of a logical system; however, it represents information as a visual flow.

To base the VCCL on a sound foundation, we focus on an essential automated compliance checking process, i.e. preparing information regarding a building model. Because of the enormous number of relational dependencies in a digital building model, we introduce a set of relational objects and operators, which enable the user to define precise information queries that consider relationships. Because these elements are based on the principles of relational algebra, they represent a powerful toolkit for the processing of information and thus provide a wider scope of action for the user.

Keywords: Building Information Modeling, Automated Code Compliance Checking, Visual Programming

1. INTRODUCTION

In today's construction industry, the preparation and processing of data from divergent scales has received increasing attention. According to the building information modelling (BIM) method, a central digital building model serves as a database and an information interface for all tasks performed during the execution of a building project (Eastman et al., 2011). Because of the many different disciplines and the large number of project participants, digital models must be considered from many different perspectives. Consequently, the data model can be divided into many layers. These layers must be maintained in the model and subsequently made accessible to all stakeholders.

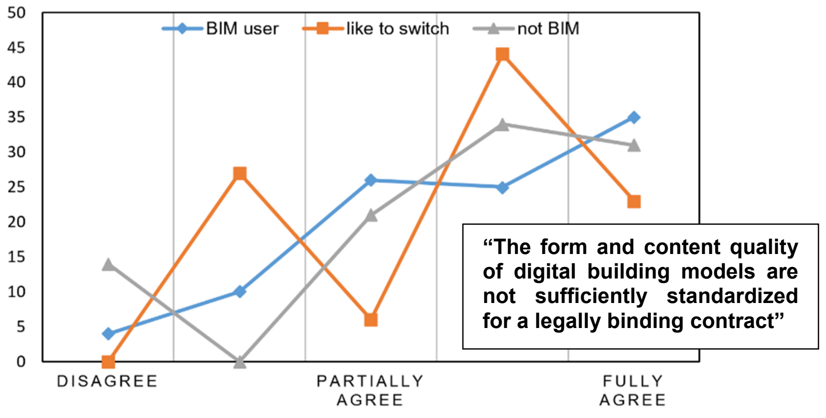

Figure 1. Result of Survey on Quality of Information in Construction Industry (Ebertshäuser & von Both, 2013)

Depending on the task to be performed, the data must be available in different forms, e.g. levels of detail or units. For example, the planning of escape routes has different requirements than a structural engineering task does in terms of the type and level of detail of the information. Therefore, the database of every task must be well defined at the beginning of every project and the quality of this information must be checked continuously throughout the project. Currently, requirements regarding the form and quality of the data are not met consistently because they are insufficiently standardized. Therefore, an average dissatisfaction regarding the quality of data is evident, as shown in Figure 1 (Ebertshäuser & von Both, 2013).

It can be concluded that error-free data preparation is a base requirement for all downstream processes in the construction industry, e.g. automated code compliance checking. Because of the large number of regional, national and international codes, as well as different application scopes, there is an enormous number of guidelines that must be considered during the design planning of a building. Therefore, several approaches deal with the automation of compliance checking according to the applicable rules. As shown in Figure 2, the overall process of automated code checking can be divided into four process steps: translation of the rules into machine-readable language, execution of the checking process, building model preparation and reporting of the checking results (Eastman et. al., 2009). For a detailed description of the requirements, the reader is referred to (Preidel & Borrmann, 2015).

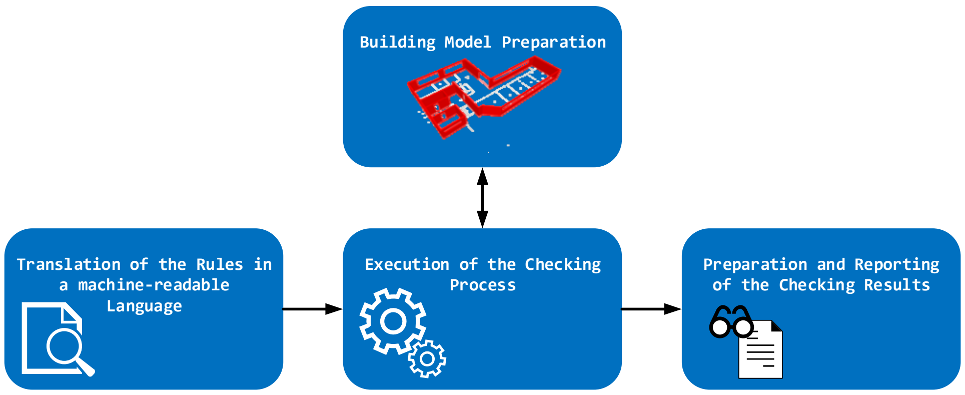
Building Model Preparation
Translation of the Rules in a machine-readable Language
Execution of the Checking Process
Preparation and Reporting of the Checking Results

Figure 2. Common structure of automated code compliance check process (Eastman et al., 2009)

Because of the complexity and special requirements of every checking process, translation to machine-readable language represents a major challenge. In addition, the preparation and pre-processing of the information are also challenging. Information queried from a data model must be converted to a form that allows subsequent checking processes to be executed seamlessly. Various investigations have shown that this step can be a significant source of error because of inconsistent, contradictory, false or non-existent information in the building model (Beetz et al., 2009). Furthermore, the correctness of a checking process is directly related to the quality of the building model; the process is useless if the information source contains errors. Therefore, many approaches have shown that an a priori process is preferable to direct use of the data model. At this point, tools are required to enable reliable and flexible access to the model data, while maintaining the highest possible level of data quality (Eastman et al., 2009).

2. RELATED WORK

As indicated in the introduction, information preparation, which is a basic requirement for many processes, is a major challenge in the BIM-based construction industry. Data preparation requirements are extremely diverse because of the many different types of guidelines and codes. Several approaches that deal with similar requirements use query languages (QLs) to extract data. QLs allow users to formulate common and complex queries for individual data models. Most of these approaches are based on a de facto standard, i.e. domain-independent Structured Query Language (SQL) (Codd, 1990). SQL has comparatively easy-to-use syntax and is based on relational algebra, which allows users to define complex queries. SQL provides relational operators that enable examination of structured data represented by relations. However, most of the data models used in the construction industry, such as the Industry Foundation Classes (IFC), are object-oriented. Because of the general incompatibility between object-oriented and relational data, i.e. the object-relational impedance mismatch problem, the extent to which such schemas can represent each other is limited and as a result not recommended for this particular application (Borrmann & Rank, 2009a; Copeland & Maier, 1984). Therefore, several approaches use basic SQL principles and adapt them according to the specific needs of the construction industry. Selected representative approaches are discussed in the following sections. In order to provide a better understanding of the different approaches, the following standard query shall be used:

"Find the corresponding building elements of the wall elements inside of the building model"

2.1 BIMQL

The Building Information Model Query Language (BIMQL) was introduced in the bimserver.org project to represent an open-source implementation of an IFC-Server, which is based on a key-value database (Beetz et al., 2010). In this web-based framework, all IFC entities are provided as Java classes using an early-binding mechanism, which in turn can be used for queries by the BIMQL. Although the language is based on SQL, it currently supports only the SELECT and UPDATE operators (Mazairac & Beetz, 2013; Mazairac, 2015). However, modification of existing entities is supported and future developments, such as creation or deletion operations and various amendments, are planned. In Code 1, an example BIMQL query is shown, which performs two selections on the IFC data model to obtain the corresponding windows of basic walls.

Code 1. BIMQL query for the selection of IfcWalls and its corresponding IfcWindows

```
Select $wall
Where  $wall.EntityType = IfcWallStandardCase
Select $relvoid := $wall.Attribute.HasOpenings
Select $opening := $relvoid.Attribute.RelatedOpeningElement
Select $relfill := $opening.Attribute.HasFillings
Select $element := $relfill.Attribute.RelatedBuildingElement
Where  $element.EntityType = IfcWindow $todo
```

The BIMQL is a domain-specific query language with comparatively easy-to-use syntax. However, it does not provide the wealth of functionalities or operators that would enable a thorough evaluation of the potential of this tool. A current significant disadvantage is that the elements of the relational algebra are not completely available; therefore, this approach currently lacks a substantial component.

2.2 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, the SPARQL was recommended by the World Wide Web Consortium (Harris & Seaborne, 2013). Because the RDF schema can be used to model the information about a digital building (e.g. IFC schema) as entities as well as the relationships between objects, it serves as a basis for ontological knowledge (Beetz et al., 2007). In computer science, ontologies are used for machine-readable and interpretable descriptions of data that can be defined as a 'formal explicit description of a shared management concept' (Gruber, 1993). On the basis of this ontological database, the SPARQL can perform the query shown in in Code 2.

Code 2. SPARQL query for the selection of IfcWalls and its corresponding IfcWindows

```
PREFIX lbd:<http://linkedbuildingdata.net/schema/IFC2X3#>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
SELECT ? wallname ? windowname
WHERE {
   ?v a lbd:IfcRelVoidsElement.
   ?w a lbd:IfcWallStandardCase.
   ?o a lbd:IfcOpeningElement.
   ?f a lbd:IfcRelFillsElement.
   ?wi a lbd:IfcWindow.
   ?v lbd: relatingBuildingElement ?w.
   ?v lbd: relatedOpeningElement ?o.
   ?f lbd: relatingOpeningElement ?o.
   ?f lbd: relatedBuildingElement ?wi.
   ?w lbd: name ? wallname.
   ?wi lbd: name ? windowname.}
```

As shown in this example, the SPARQL query produces a projection such that the result is a simple data table that contains only the names of the corresponding walls and windows but not the objects. The RDF does not analyze actual entities; therefore, it is not possible to produce an actual entity from such a query. This limitation should always be considered when formulating SPARQL queries.

2.3 QL4BIM

The Query Language for Building Information Models (QL4BIM) is a domain-specific query language that can be applied to the analysis and processing of building information models. Therefore, it is an imperative and procedural language that offers easy-to-use syntax with 'carefully selected vocabulary to formulate queries at a high level of abstraction' (Daum & Borrmann, 2015). An example QL4BIM query that identifies the 'Touch' relation between the walls and windows of an IFC model is shown in Code 3.

Code 3. QL4BIM query to identify the ‘Touch’ relation between IfcWalls and IfcWindows (Daum & Borrmann, 2015)

```
model = GetModel("C:\Institute.ifc")
allWalls = TypeFilter(model , "IfcWall")
allWindows = TypeFilter(model, "IfcWindow")
rel[wall, window] = Touch(allWalls, allWindows)
```

The authors have not implemented typical programming elements, such as those are usually provided in popular languages, to keep the language abstract and simple. For example, the language hides low-level data-handling operations, such as instantiations or cast operations, so that the number of patterns a user needs to understand to use the query language is minimized. For ease of use, the QL4BIM also provides a version based on visual programming principles. Therefore, the $_{t}$QL4BIM has a visual equivalent, i.e. the $_{v}$QL4BIM, that follows the paradigms of language design and uses graphical syntax to represent elements in the form of an abstract syntax tree (Daum & Borrmann, 2015). In summary, the QL4BIM demonstrates that complex queries can be formulated plainly; however, this approach is intentionally textual and not visual, even though a visual version is available.

3. METHOD

The approaches presented in Section 2 show that there are various ways to provide powerful tools for formulating queries for the construction industry. To make this functionality accessible and usable in terms of the automated code compliance checking process, it should be tailored to a given application in the VCCL. We demonstrate that code compliance checking can be automated by using a visual language and the BIM method. Thus, requirements that were not satisfied by previous methods can be fulfilled. In addition to imperative logic and formal structure, one of the major advantages of the VCCL is its usability by non-programmers. Compared to many existing approaches, the VCCL can be used directly by an engineer with professional knowledge and expertise who does not possess programming skills. Furthermore, the VCCL is not explicitly designed for a particular data model, but rather is intended to be applicable to various data models. For a detailed description of the principles, the reader is referred to the literature (Preidel & Borrmann, 2015).

To base the VCCL on a sound foundation, we focus on an essential process of automated compliance checking, i.e. the preparation of building model information. Because of the enormous number of relational dependencies in a BIM, we introduce a set of relational objects and operators that enable the user to define precise information queries. Because these elements are based on the principles of relational algebra, they represent a powerful toolkit for processing information and thus provide a wider scope of action for the user.

3.1 Theoretical Background

According to the mathematical definition, a relation R represents the subset of the Cartesian Product of n defined domains (Kemper & Eickler, 2011):

$$R \subseteq D_1 \times D_2 \times \cdots \times D_n$$

Thus, a relation assigns certain elements of a set to the elements of another set and, therefore, provides a well-defined mathematical relationship. The results can be presented graphically in different ways, as shown in Figure 3.

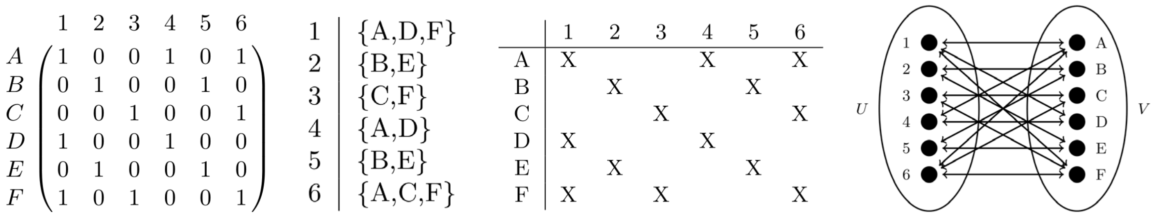

Figure 3. Different representation styles for relation between two sets (Kemper & Eickler, 2011)

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 Figure 4.

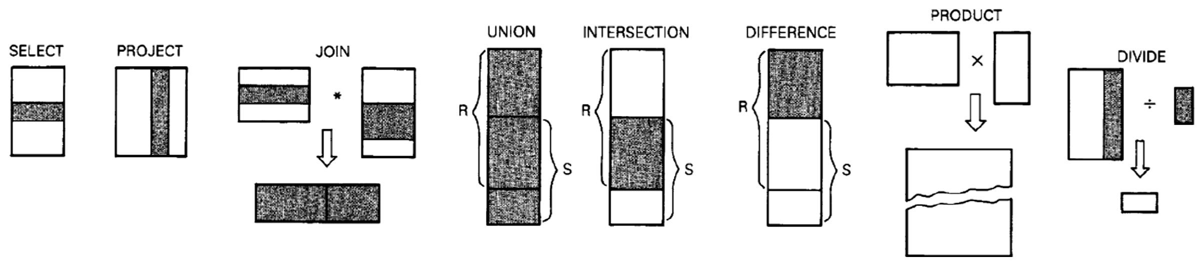

Figure 4. Relational algebra operators (Codd, 1990)

3.2 Visual Relational Objects and Operators

The principle of the relation and relational algebra facilitates the preparation and handling of data; thus, the VCCL has been expanded to include this feature. As a result, we introduce the relation as a node type that holds n-dimensional tuples and represents a relation. As shown in Figure 5, this object can be represented directly as part of a VCCL graph and, if needed, the content of the node can be displayed separately. The representation of the relation object inside the VCCL graph is carried out on the already available elements (Preidel & Borrmann, 2015). The process of listing the information only assists the user and may be omitted.

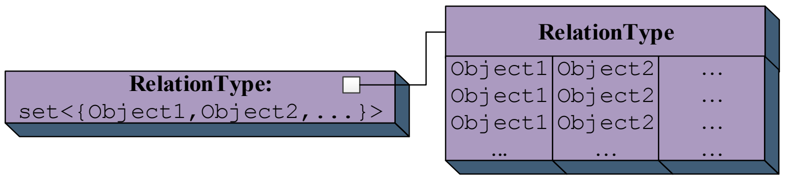

Figure 5. Representation of the Relation Object in the VCCL

An object is associated with a specified relation type to define the relationship. This relation type is described by an ordered list of element types of the corresponding tuple. The order of the objects within a tuple plays an essential role and should not be changed; otherwise, the contained value will be false. With this unique typecasting downstream operations can rely on the types that are encapsulated by the relation. This is particularly necessary because the VCCL is a strictly typed language. To create and process a VCCL relation, the framework provides a number of operators, which are described in the following sections.

(1) Relation Creator

To create a VCCL relation object, we introduce a relation creator. A simple example of the relation creator is shown in Figure 6. The creator can build a relation by checking whether the information provided by the incoming operands satisfies a certain criterion. If this criterion is satisfied, the elements have a relationship and can be mapped inside the relation as tuple pairs. Consequently, the mandatory checking process inside the creator must be implemented as an underlying algorithm. In the present example, the creator must check which building elements of the incoming sets are related by the criterion 'voids', which describes the type of the resulting relation. Depending on the data model, this information can be queried by the data model, or it must be processed by (e.g. geometric) algorithms. Another application is the processing of geometric–topological relationships because such information is typically not displayed in common data models (Borrmann & Rank, 2009b). Examples of the geometric–topological relations of 3D objects are shown in Figure 7.

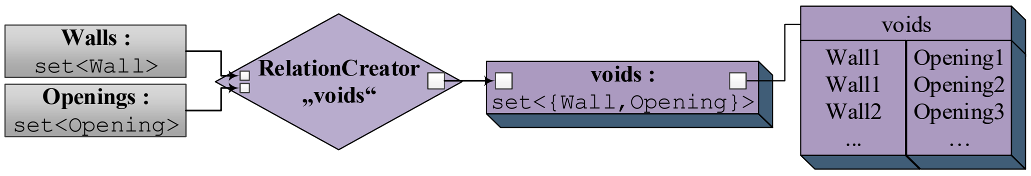

Figure 6. Application of relation creator

In addition, a user can create such a relation manually, which can be helpful if the data model does not provide the required information. In such a case, the algorithm cannot produce a relation. Thus, various types of creators can be provided, e.g. topology-based, geometry-based, cost-based or process-based.

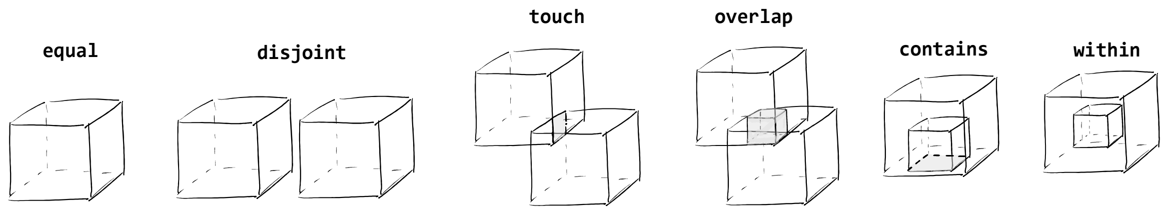

Figure 7. Geometric-topological relations (Borrmann & Rank, 2009b)

(2) Basic Relational Operators

To process the created relations, the SQL relational algebra operators (Figure 4) can be applied to the VCCL relational elements. Relational algebra queries can be formulated according to a relational scheme that allows linking relations to each other to decrease the relations or to derive more complex information. For this purpose, the following basic operators have been implemented (Kemper & Eickler, 2011):

- Set Operators: Union, Difference, Intersection
- Cartesian Product
- Projection
- Selection
- Join Operators: Equi-Join, Natural Join, Semi Join, Outer Join, Left Join, Right Join

(3) Aggregate Operators

Another class of operators can be found within the relational algebra, i.e. aggregate operators or aggregation functions (Grabisch et al., 2009). These are based on the principle of relational completeness, which means that any relational algebra operation can be implemented in the query language by (at least) one expression. In principle, an aggregate operator is a function f with a relation R as an argument that returns a single value in a defined range. For example, several functions for relations can be defined to determine the sum, maximum, minimum, arithmetic mean or number of elements of a relation. A basic example of such an aggregate operator is shown in Figure 8. The ‘Sum’ operator accesses a certain column of the relation that is defined by the additional string information ‘Height’. With this information, the standard routine of summing all elements stored inside the column of this relation can be executed and transferred to the resulting VCCL node.

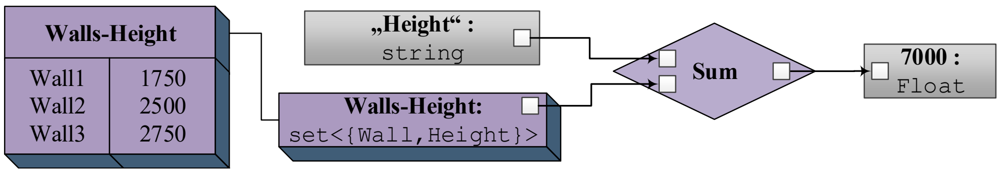

Figure 8. Application of VCCL aggregate operator

4. RESULTS

A new category of queries, which were shown in the various code examples in Section 2, can be expressed as a VCCL graph. The graph in Figure 9 describes a VCCL process. Initially, this process filters all window, opening and building elements of the corresponding data model. Then, two relations are generated to define the corresponding relationships between the walls and openings and between the opening and building elements. These relations can be processed by the relational operators. Next, the two relations are merged by a join operation so that the relationships among the three components are represented as a new relation object. The final result can be produced by a projection operation. The resulting relation contains the relationship between the wall elements and the corresponding building elements, such as doors and windows.

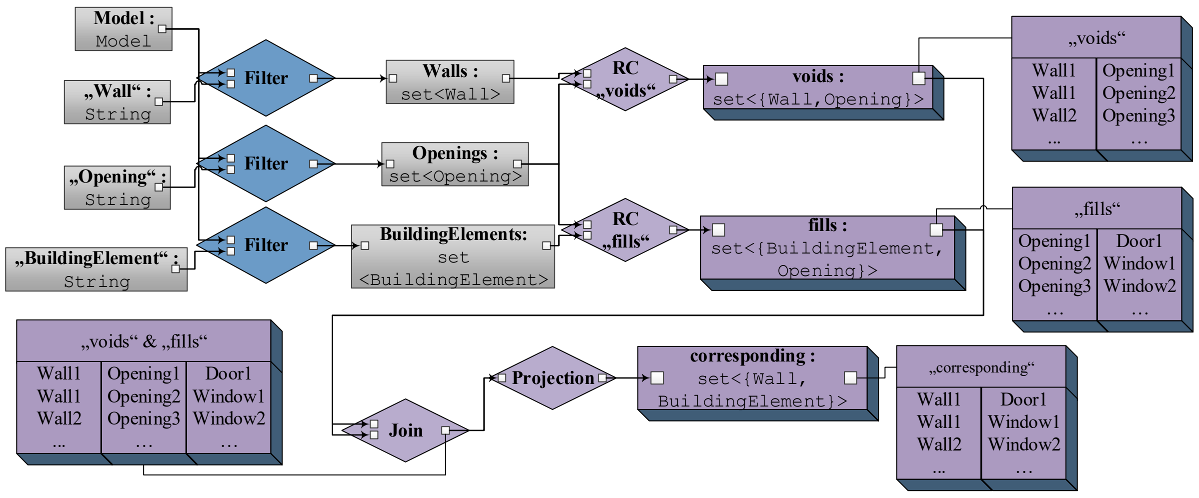

Figure 9. VCCL query for selection of walls and corresponding building elements

By adding the relation objects and operators, the VCCL becomes a powerful toolkit for the preparation of building model data and the subsequent checking processes. With this tool, the basis for a performant and high-quality checking process is created. As a result, the general VCCL design is adapted, as shown in Figure 10. By introducing relation creators, a new source of information for elements of the VCCL library is provided; however, such elements should not be unnecessarily mixed with the data model. Therefore, the VCCL data model and the relation creator instance are introduced as separate modules. These modules receive the data from common BIMs via an interpreter instance and are used as intermediate quality buffers such that the information is never directly transferred to the elements of the VCCL library. This decoupling allows preparation and ensures that the data is not transferred without an intermediate quality checking.

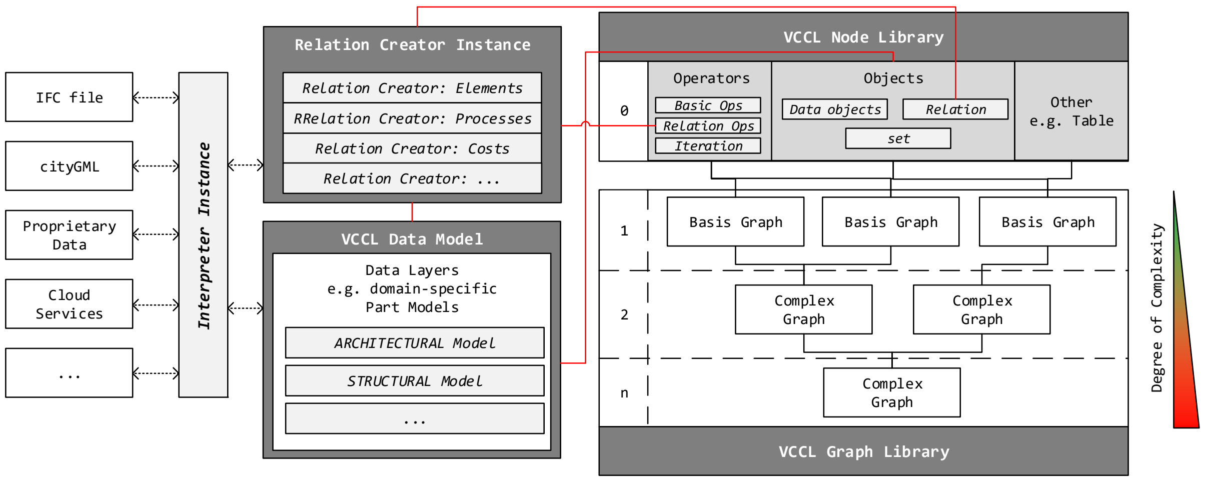

Figure 10. Expansion of VCCL library to relational elements

5. CONCLUSIONS

We demonstrated that the BIM method and its underlying processes are largely dependent upon the model used to prepare the data. The presented approaches introduce different solutions for seamless and high-performance extraction of digital building model information. However, there are still gaps in our analysis and data handling, particularly in terms of the formal examination of models and the derivation of sub-models.

The VCCL is intended to provide an easy-to-use, reliable and powerful library of elements for the formulation and execution of checking processes. By introducing relational objects into the VCCL, we facilitated the data preparation process and created a sound foundation for further developments. We also integrated operators based on relational algebra that are normally represented in declarative languages (such as SQL) into imperative visual language. This extension provides a variety of new possibilities that can be used to formulate more complex preparation or checking processes without jeopardizing the correctness of the checking process because of errors in information sources. For example, the relational elements can be used to link different data layers of a building model. A digital building model contains both geometric and alphanumeric information from different domain-specific models and such information must be linked. In principle, this linkage must occur whenever a change is made to effectively update the model. The expanded VCCL can map such relationships; thus, it can be used to define processes to link such information automatically.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support by ALLPLAN GmbH and Nemetschek Group for the presented research.

REFERENCES

Beetz, J., de Vries, B., & van Leeuwen, J. (2009). IfcOWL: A case of transforming EXPRESS schemas into ontologies. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, *23*(1), 89–101. doi:10.1017/S0890060409000122

Beetz, J., van Berlo, L., de Latt, R., & van den Helm, P. (2010). bimserver.org – An Open Source IFC Model Server. In *Proc. CIB W78 conference*.

Beetz, J., Vries, B., & van Leeuwen, J. (2007). RDF-based distributed functional part specifications for the facilitation of service-based architectures. In *Proc. of the 24th CIB-W78 Conf. on Information Technology in Construction*.

Borrmann, A., & Rank, E. (2009a). Query Support for BIMs using Semantic and Spatial Conditions. In *Handbook of research on building information modeling and construction informatics: Concepts and technologies* (pp. 405–450). IGI Global.

Borrmann, A., & Rank, E. (2009b). Topological analysis of 3D building models using a spatial query language. *Advanced Engineering Informatics*, *23*(4), 370–385. doi:10.1016/j.aei.2009.06.001

Codd, E. (1990). *The relational model for database management: version 2*. Addison-Wesley Longman Publishing Co., Inc.

Copeland, G., & Maier, D. (1984). Making smalltalk a database system. *ACM SIGMOD Record*, *14*(2), 316. doi:10.1145/971697.602300

Daum, S., & Borrmann, A. (2015). Simplifying the Analysis of Building Information Models Using tQL4BIM and vQL4BIM. In *Proc. of the EG-ICE 2015*. Eindhoven, The Netherlands.

Eastman, C., Lee, J. M., Jeong, Y. S., & Lee, J. K. (2009). Automatic rule-based checking of building designs. *Automation in Construction*, *18*(8), 1011–1033. doi:10.1016/j.autcon.2009.07.002

Eastman, C., Teicholz, P., Sacks, R., & Liston, K. (2011). *BIM Handbook* (Second Edition). John Wiley & Sons, Inc.

Ebertshäuser, S., & von Both, P. (2013). ifcModelCheck - A tool for configurable rule-based model checking. In *Computation and Performance - Proceedings of the 31st eCAADe Conference* (Vol. 2).

Grabisch, M., Marichal, J.-L., Mesiar, R., & Pap, E. (2009). Aggregation functions. In *Encyclopedia of Mathematics and its Applications*. Cambridge: Cambridge University Press.

Gruber, T. (1993). A translation approach to portable ontology specifications, *5*(2), 199–220.

Harris, S., & Seaborne, A. (2013). SPARQL 1.1 Query Language. Retrieved from http://www.w3.org/TR/sparql11-query/

Kemper, A., & Eickler, A. (2011). *Datenbanksysteme - Eine Einführung* (8th Edition).

Mazairac, W. (2015). BimQL. Retrieved December 8, 2015, from http://bimql.org/

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

Preidel, C., & Borrmann, A. (2015). Automated Code Compliance Checking Based on a Visual Language and Building Information Modeling. In *Proceedings of the International Symposium of Automation and Robotics in Construction*. Oulu, Finland. doi:10.13140/RG.2.1.1542.2805